FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Webová aplikace pro sběr a zpracování dat IoT
Student:	Bc. Jan Prášil
Vedoucí:	Ing. Jan Duba
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Cílem práce je vytvořit webovou aplikaci, která bude prezentovat výstupy naměřených hodnot firemních měřidel využívajících internetu věcí (IoT).
Implementujte aplikaci tak, aby obsahovala:
1) rozhraní pro správce portálu – pro schvalování žádostí firem o registraci a přidělování přístupů k API rozhraním,
2) rozhraní pro společnosti – registrace, schvalování a správu zaměstnanců, nastavení limitních hodnot jednotlivých měřidel pro notifikace,
3) rozhraní pro zaměstnance – sledování naměřených hodnot pomocí grafů a nastavení notifikací při překročení limitních hodnot.
Data poskytovaná měřidly mohou mít libovolný formát a být přístupná různými způsoby, proto je třeba implementovat moduly, které získaná data znormalizují do lokální databáze. Implementujte tyto moduly pro předem dodaný teploměr a vlhkoměr, které v čase sbírají data o teplotě, resp. vlhkosti.
Aplikaci implementujte v JavaScriptu. Serverovou část v Node.JS, klientskou část s použitím technologie React.JS nebo Angular.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 7. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Webová aplikace pro sběr a zpracování dat IoT

Bc. Jan Prášil

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jan Duba

7. května 2018

Poděkování

Rád bych poděkoval mé rodině a přátelům, kteří mé v průběhu studia podporovali. Nesmím opomenout poděkování vedoucímu této práce, panu Ing. Janu Dubovi, který se ujal toho, aby tato diplomová práce mohla vzniknout.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. § 46 odst. 6 tohoto zákona 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 7. května 2018 .....

Odkaz na tuto práci

Prášil, Jan. *Webová aplikace pro sběr a zpracování dat IoT*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato diplomová práce se zabývá vytvořením webové aplikace pro správu zařízení internetu věcí. Hlavním účelem této aplikace je systematický sběr hodnot měřené veličiny (dat) a jejich následné vyhodnocení, včetně notifikace při překročení mezních hodnot. Produktem této diplomové práce je funkční webová aplikace.

Klíčová slova Webová aplikace, IoT, React, Node, JavaScript

Abstract

This Master's Thesis deals with a creation of an Internet of Things devices management web application. The main purpose of this application is the systematic collection of measured values and their evaluation, including notifications when limit values are exceeded. The product of this diploma thesis is a functional web application.

Keywords Web application, IoT, React, Node, JavaScript

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Slovní spojení internet věcí, neboli IoT, je v poslední době velmi skloňovaným tématem. Jedná se o síť fyzických zařízení, která jsou vybavena síťovou konektivitou, díky které je možné vyměňovat data a zároveň zařízení kontrolovat a spravovat[1].

V současné době je tak možné pomocí internetu věcí spravovat nejrůznější zařízení, které můžeme rozdělit na dva typy.

- **Podnikové**, které slouží k automatizaci procesů a sběru dat.
- **Spotřebitelské**, známé jako *chytrá zařízení*, díky kterým je možné ovládat domácí spotřebiče. Dále také pomáhají zdravotně postiženým lidem například následujícími funkcemi: ovládání hlasem, kontrola krizových situací, např. pády nebo záchvaty.

V rámci této práce se budu zabývat zařízeními s podnikovým využitím, které periodicky získávají data o svém okolí. Tyto hodnoty pak zpracuje server a bude uživatele informovat podle jeho zadaných pravidel.

Zadavatel si je vědom jiných řešení na trhu, které v současné době využívá. Z důvodu lepšího propojení s jeho současnou infrastrukturou, a s tím souvisejícím pozdějším rozšířením aplikace, se rozhodl pro implementaci nového produktu, který budu dále označovat pod zkratkou IOTCP.

Dalšími důvody pro realizaci nové aplikace bylo navrhnuté jednodušší a přímočaré ovládání celé webové aplikace a použití jednotného zdroje dat.

Celý proces správy zařízení bude začínat registrací společnosti na webovém portále a následném schválení administrátorem navrhované webové aplikace. Ve chvíli, kdy je společnost autorizována, bude moci přidávat zařízení, které definuje názvem, strukturou vrácených dat a poskytnutým modulem, který data ze zařízení IoT získá a transformuje do definované struktury, a frekvence jeho spouštění.

Každé takto přidané zařízení, především zdrojový kód modulu, bude zkontrolováno administrátorem systému jakožto prevence před prováděním kódu,

který by mohl narušovat, či dokonce ohrožovat funkčnost webové aplikace. Po úspěšném schválení bude spuštěn plánovač úloh pro získávání hodnot měřené veličiny.

Schválená společnost bude umožňovat přístup zaměstnanců ke svým datům a jednotliví uživatelé si budou moci nastavit pravidla pro notifikace v případě porušení definovaných podmínek daných pravidel.

Kapitola 1

Cíle práce

Cílem této diplomové práce je vytvořit webovou aplikaci, která uživateli zobrazuje výstupy naměřených hodnot firemních měřidel využívajících internetu věcí. Dle požadavků bude jako programovací jazyk použit JavaScript. Aplikace by měla implementovat serverovou část využívající technologie Node.JS a poskytující REST API[1] pro přístup z klientské strany, která má být implementována technologií React nebo Angular.

Mým úkolem je navrhnout, vysvětlit a použít vhodné nástroje tak, aby produkt odpovídal požadavkům zadavatele.

Důležitým rozhodnutím pro inicializace webové aplikace je vybrat správný framework pro uživatelské rozhraní na klientské části a výběr databáze, která bude spuštěna na straně serveru. Dalším podstatným krokem je provést dobrý návrh uživatelského rozhraní pro jednotlivé uživatelské role, které budou aplikace používat - *administrátor*, *správce* a *zaměstnanec* společnosti.

[1] Typ architektury pro webové rozhraní.

Kapitola 2

Současné řešení

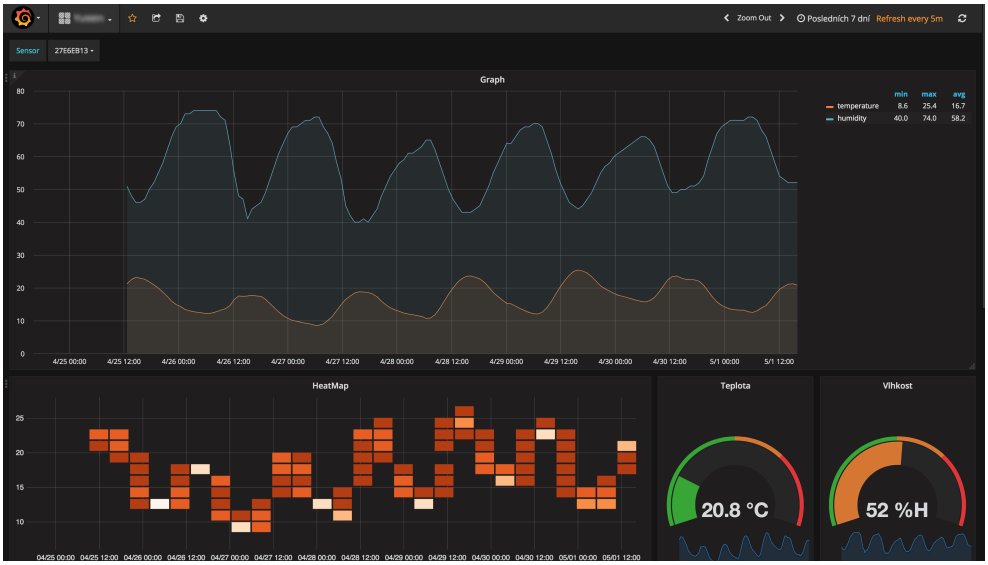
Momentálně, pro správu dat, používá zadavatel služby společnosti, která pro něj zároveň dodává IoT zařízení. Z tohoto důvodu tak může využívat pouze sortiment, který tato společnost nabízí. Pro všechna zařízení zajišťuje dodavatel úložiště dat v databázi MySQL a dále poskytuje portál, na kterém je možné kontrolovat či konfigurovat IoT zařízení či jejich jednotlivé senzory. Pro vizualizaci získaných dat je poskytnuta webová stránka nástroje **Grafana**.

2.1 Nástroj Grafana

Grafana je open-source platforma pro analýzu a monitoring dat v časových intervalech s možností notifikací uživatelů dle přesně zadaných pravidel. Data mohou pocházet z různých zdrojů, mezi které patří například rozsáhlé analytické systémy Elasticsearch či Graphite, ale také běžné databázové systémy typu MySQL nebo PostgreSQL. Existuje však mnoho dalších pluginů pro jiné typy datových úložišť[2].

Velkou přednostní nástroje Grafana je pak jeho otevřenost vůči vývojářům, kteří mohou vyvíjet různé pluginy - *dashboardy*, *panely* či *mini aplikace*, a publikovat je veřejně pro ostatní uživatele tohoto nástroje.

Ukázku stránky zařízení s grafy hodnot můžete spatřit na obrázku 2.1.



Obrázek 2.1: Ukázka detailu zařízení s grafy v systému Grafana.

2.2 Motivace pro nové řešení

Webová aplikace, která vzniká v rámci této diplomové práce má v podstatě za úkol nahradit nástroj Grafana. Může se zdát, že z důvodu velmi vysoké úrovně a rozšíření nástroje Grafana se nejedná o dobrý krok. Nicméně zadavatel objevil několik nedostatků, které rozhodly o tom, že bude vyvíjena nová aplikace.

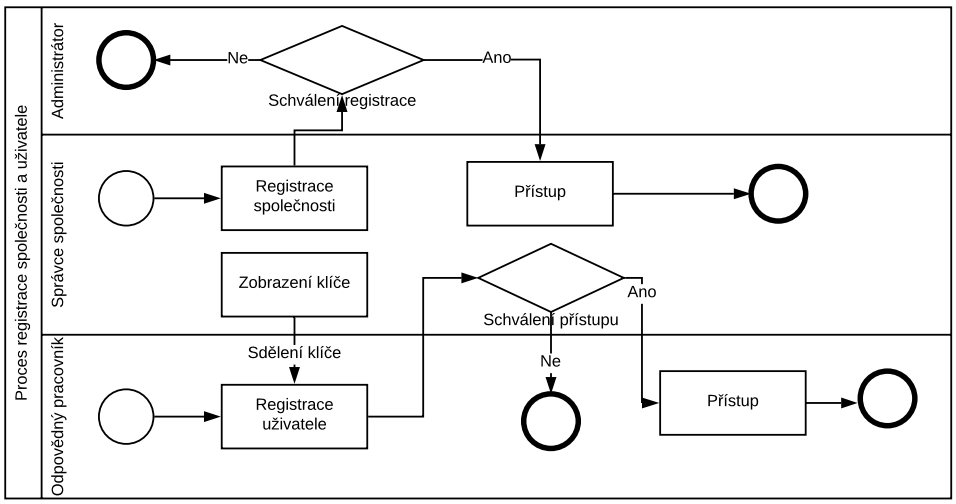
Jedná se o nemožnost provádět náročnější výpočty nad konkrétními zařízeními a hlubší propojení s jinými produkty, které zadavatel používá - především pak se službou Microsoft SharePoint Server 2013.

Dalším důvod je připojení k datovým zdrojům, ze kterých jsou data čerpána. V nové webové aplikaci by mělo být umožněno, pokud to dané zařízení bude podporovat, zasílat data ze zařízení přímo do systému s použitím vlastní databáze.

Kapitola 3

Analýza

V této části budu reprezentovat výsledky analýzy při přípravě nového řešení webové aplikace. Důležitou částí je akceptovat a strukturovat požadavky zadavatele, rozpoznat typy uživatelů nové webové aplikace a navrhnout jednotlivé případy užití.



Obrázek 3.1: Diagram procesu registrace nového uživatele.

3.1 Funkční požadavky

3.1.1 Funkční požadavky uživatele

Tato sekce se věnuje funkčním požadavkům jednotlivých typů uživatelů webové aplikace. Schéma procesu registrace uživatelů můžete vidět na diagramu 3.1

F1.1 **Přihlášení**

Registrovanému, nepřihlášenému uživateli je umožněno přihlásit se. Přihlášení probíhá zadáním kombinace *emailové adresy* a *hesla* zadaných při registraci.

F1.2 **Registrace**

Nepřihlášený uživatel má možnost se registrovat do systému pomocí jednoduchého formuláře, ve kterém vyplní svůj *email*, *telefonní číslo*, zvolí si své osobní *heslo* a *roli*, dle svého důvodu pro registraci: *správce společnosti* či *zaměstnanec*.

F1.2.1 **Registrace zaměstnance**

Pokud si uživatel zvolí jako typ uživatelského účtu *zaměstnance*, musí pro přihlášení do své společnosti zadat společností dodaný *ověřovací kód*. Po úspěšném zadání kódu pak čeká na schválení danou společností.

F1.2.2 **Registrace společnosti**

V případě zvolení uživatelského účtu typu *společnost*, je žádost předána administrátorovi webové aplikace, který musí nově vytvořenou společnost nejdříve schválit.

F1.3 **Změna zapomenutého hesla**

Zapomene-li uživatel své heslo, má možnost ho změnit pomocí formuláře, ve kterém vyplní svoji registrační emailovou adresu, na kterou mu je odeslán email s odkazem na stránku pro změnu hesla.

F1.4 **Změna hesla**

Je-li uživatel přihlášený, může změnit své heslo. Pro změnu hesla je pak ve formuláři nutné zadat současné heslo pro ověření uživatele.

F1.5 **Odhlášení se**

Přihlášený uživatel má možnost se odhlásit z aplikace.

F1.6 **Schválení zaměstnance**

Přihlášený uživatel typu *společnost* schvaluje žádosti o přístup uživatelů typu *zaměstnanec*.

F1.7 **Schválení společnosti**

Přihlášený uživatel typu *administrátor* může schválit žádost o vytvoření nové společnosti.

F1.8 **Zobrazení přístupového klíče**

Přihlášený uživatel typu *společnost* má možnost zobrazit přístupový klíč poskytovaný *zaměstnancům* pro registraci do společnosti.

3.1.2 Funkční požadavky pro správu zařízení

V této podkapitole se věnuji funkčním požadavkům na správu zařízení internetu věcí.

F2.1 **Přidat zařízení**

Uživatel typu *společnost* může přidat nové zařízení internetu věcí. Ve formuláři vyplní *název*, nahraje *komprimovaný modul pro získávání dat zařízení*, pro který vyplní strukturu výstupních dat ve formátu JSON, a zvolí *interval* spouštění modulu pro získávání dat. Během procesu odesílání nového zařízení je spuštěna automatická kontrola modulu, která spočívá v ověření struktury zdrojového kódu modulu a vrácených dat. V případě úspěchu kontroly je zařízení uloženo a čeká na ruční schválení správcem systému (viz F2.3).

F2.2 **Odebrat zařízení**

Uživatel typu *společnost* může odebrat zařízení. Veškeré úlohy provádějící získávání dat jsou zastaveny a notifikace související s daným zařízením smazány. Notifikační hlášky jednotlivých pravidel zůstanou zobrazeny, dokud je uživatel nepotvrdí.

F2.3 **Schválit zařízení**

Uživatel typu správce aplikace může schválit zařízení přidané společností. Schválení probíhá po manuálním ověřením funkčnosti modulu, kdy se kontroluje především jeho nezávadnost, která by mohla způsobit přetížení, či dokonce napadení systému.

F2.4 **Spustit získávání dat**

Správce společnosti může schválené zařízení spustit za účelem získávání dat. Data jsou získávána v intervalu zadaným při vytváření zařízení.

F2.5 **Zastavit získávání dat**

Správce společnosti má možnost zastavit získávání dat.

F2.6 **Zobrazit detail zařízení**

Uživatel typu zaměstnanec nebo správce společnosti má možnost zobrazit detail zařízení, na kterém vidí grafické výstupy, které si může specifikovat (viz F2.7).

F2.7 **Přidat graf výstupních hodnot**

Zaměstnanec či správce společnosti může přidat graf zobrazující hodnoty naměřené zařízením internetu věcí. Je možné si vybrat sloupcový či čárový graf a dále definovat veličinu na ose X a hodnoty na ose Y.

3.1.3 Funkční požadavky pro notifikace

Následující seznam funkčních požadavků reflektuje možnosti pro nastavení notifikace uživatele dle výsledných naměřených hodnot zařízeními internetu věcí. Nastavení notifikací platí pro uživatele uživatelské role správce společnosti a zaměstnance.

F3.1 **Vytvořit pravidlo notifikace**

Uživatel vytváří pravidla notifikací, která se vyhodnocují při každém spuštění získávání dat ze zařízení.

Uživatel zadá název notifikace a vybere zařízení, pro které se bude pravidlo vyhodnocovat. Následně vytváří tzv. porovnávače. Ty mohou nést buď statické hodnoty nebo hodnoty vytvořené z výsledku měření zařízení.

F3.2 **Upravit pravidlo notifikace**

Uživatel má možnost upravit své pravidlo notifikace podle stejného formuláře jako bylo vytvoření dané notifikace.

F3.3 **Odebrat notifikaci**

Uživatel má možnost odebrat své pravidlo notifikace a odebrat tak všechna upozornění.

F3.4 **Zobrazit seznam aktuálních notifikačních hlášek**

Výstupní hodnoty ze zařízení, které neodpovídají zadaným kritériím notifikací, jsou zobrazeny uživateli. Pokud jsou hodnoty porušeny při několika měřeních za sebou, nejsou vytvářeny nové notifikace. Pokud uživatel notifikaci potvrdí, pak se další notifikace zobrazí jako nový záznam.

Tabulka 3.1: Vlastnosti serveru pro běh webového systému

Vlastnost	Celkové prostředky server	Prostředky virtuálního stroje
Operační systém	Windows Server 2012 R2 Standard	Linux Ubuntu
Operační paměť	42 GB	4 GB
Procesor	Intel Xeon (celkem 24 jader, 2.79 GHz)	2 virtuální procesory

3.1.4 Funkční požadavky pro nastavení

Následující seznam funkčních požadavků reflektuje možnosti pro obecné nastavení webové aplikace.

F4.1 **Vytvořit notifikaci**

Společnost má možnost přidat globální notifikaci pro své zaměstnance.

F4.2 **Upravit notifikaci**

Společnost má možnost upravit globální notifikaci pro své zaměstnance.

F4.3 **Odebrat notifikaci**

Společnost má možnost smazat globální notifikaci pro své zaměstnance.

F4.4 **Zobrazit seznam aktuálních notifikačních hlášek**

Zaměstnanci a společnost mohou zobrazit globální notifikační hlášky.

3.2 Nefunkční požadavky

N1 **Systém je webová aplikace**

Aplikace IOTCP je webová aplikace, jež bude veřejně přístupná pod doménou iot.kalibrace.cz.

N2 **Serverové prostředí pro spuštění systému**

Aplikace bude spuštěna na serveru zadavatele. V tabulce 3.1 můžete nalézt celkové prostředky serveru a prostředky uvolněné pro běh webové aplikace.

N3 **Databázový server**

Serverová část aplikace se bude připojovat k databázi, která je k dispozici zdarma a měla by umožňovat práci s big data[2].

N4 **Technologie serverové strany**

Serverová část bude implementována v jazyce JavaScript pomocí systému Node.JS tak, aby podporoval nový protokol HTTP/2.

[2]Zpracování velkého množství dat v rozumném čase.

N5 **Technologie klientské strany**

Klientská část bude implementována v jazyce JavaScript pomocí frameworku Angular nebo React dle výběru vývojáře.

N5 **Podporované prohlížeče**

Aplikace bude podporována nejnovějšími verzemi následujících prohlížečů:

B1 Internet Explorer

B2 Google Chrome

B3 Mozilla Firefox

B4 Safari

N6 **Podoba externích modulů**

Každý modul, který bude do aplikace přidán, by měl splňovat následující podmínky:

1. modul je komprimován v archivu ZIP,
2. modul je implementován v jazyce JavaScript,
3. při implementaci modulu je použit Node.JS,
4. funkce volaná pro získání dat je výchozí funkce v souboru `index.js` v adresářovém kořeni modulu,
5. výsledek volání modulu je pole objektů, které splňují datovou strukturu definovanou při vytváření zařízení.

3.3 Uživatelské role

V této části definuji autorizační role přihlášených uživatelů. Každý typ uživatele má své specifické možnosti používání aplikace.

3.3.1 Neautorizovaný uživatel

Neautorizovaný uživatel je typ uživatele aplikace, který není přihlášený. Tento uživatel se může do aplikace registrovat, požádat o zaslání zapomenutého hesla nebo se může přihlásit do aplikace se svými přihlašovacími údaji.

3.3.2 Administrátor aplikace

Administrátor je typ uživatele nejvyšší úrovně. Schvaluje žádosti společností o registraci. Dále kontroluje a schvaluje moduly pro komunikaci se zařízeními internetu věcí.

Administrátor nemá přístup k soukromým datům jednotlivých společností a jejich zaměstnanců.

3.3.3 Společnost

Společnost je typ uživatele, který schvaluje žádosti uživatelů o přidání. Může přidávat nová zařízení, odstraňovat je a spouštět získávání dat nad těmito zařízeními. Tato role je nadřazena roli *Zaměstnanec*.

3.3.4 Zaměstnanec

Zaměstnanec je role uživatele, která umožňuje přidávat, upravovat či mazat vlastní notifikační pravidla. Uživatel je pak notifikován podle výsledků těchto měření.

Zaměstnanec má možnost zobrazit detail zařízení a v detailu zařízení vytvářet grafy nad získanými hodnotami.

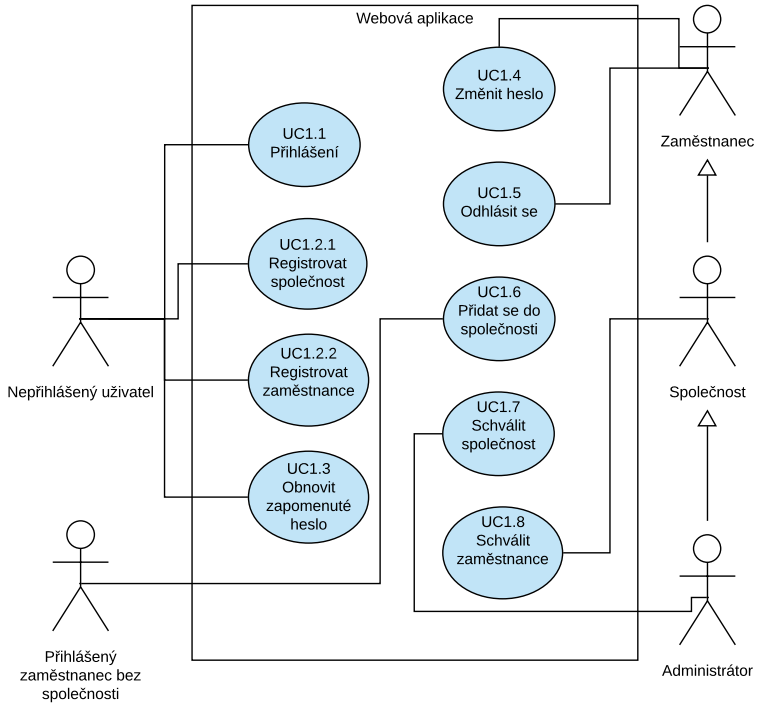
3.4 Případy užití

V této kapitole vypíši seznam případů užití, které lze v aplikaci provádět.

3.4.1 Případy užití uživatele

Případy užití v této části můžete vidět na diagramu 3.2. V této kapitole jsem dále podrobněji rozepsal případ užití registrace *UC1.2*.

UC1.1	Přihlásit se
UC1.2	Registrovat se
UC1.3	Obnovit zapomenuté heslo
UC1.4	Změnit heslo
UC1.5	Odhlásit se
UC1.6	Přidat se do společnosti
UC1.7	Schválit společnost
UC1.8	Schválit zaměstnance



Obrázek 3.2: Diagram případů užití pro uživatele.

3.4.1.1 UC1.2 Registrovat se

Popis: Nepřihlášený uživatel zaregistruje nový účet a vybere účel své registrace, tedy svou budoucí roli.
Frekvence užití: Nízká frekvence.
Předpoklady:

- Uživatel není přihlášený,
- V případě, že se chce uživatel stát zaměstnancem, zná registrační kód společnosti.

Společné kroky:

1. Uživatel se nachází na domovské stránce.
2. Uživatel vyplní registrační formulář.

Kroky pro registraci společnosti:

2.a) Společnost vyplní na místě typu uživatele hodnotu *Společnost*.

3. Společnost je registrována a čeká na schvalovací stránce.

Kroky pro registraci zaměstnance:

2.b) Zaměstnanec vyplní typ uživatele hodnotou *Zaměstnanec*.

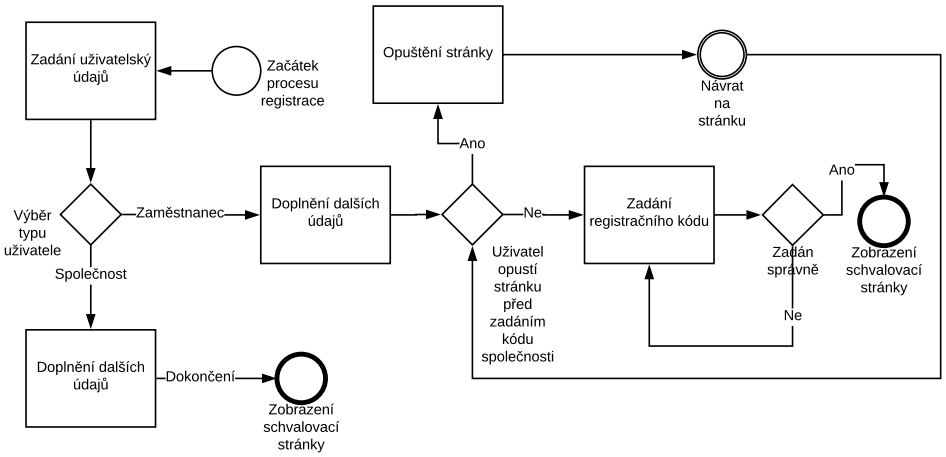
3. Zaměstnanec je přesměrován na stránku, kde vyplní a odešle registrační kód společnosti.

4. Zaměstnanec je registrován a čeká na schvalovací stránce.

Problémy:

- **Zaměstnanec nezná svůj registrační kód?** Požádat společnost o zaslání kódu a vyčkat na stránce nebo se odhlásit a postupovat podle následujícího kroku.
- **Zaměstnanec se odhlásí před zadáním registračního kódu?** Zaměstnanec bude po přihlášení přesměrován na stránku s formulářem pro zadání registračního kódu.

Diagram tohoto případu užití naleznete na obrázku 3.3.



Obrázek 3.3: Podrobný diagram případu užití UC1.2.

3.4.2 Případy užití správy zařízení

Případy užití v této části můžete vidět na diagramu 3.4. V této kapitole jsem dále podrobněji rozepsal případ užití *UC2.2*.

UC2.1 Zobrazit seznam zařízení

UC2.2 Vytvořit zařízení

UC2.3 Schválit zařízení

UC2.4 Odebrat zařízení

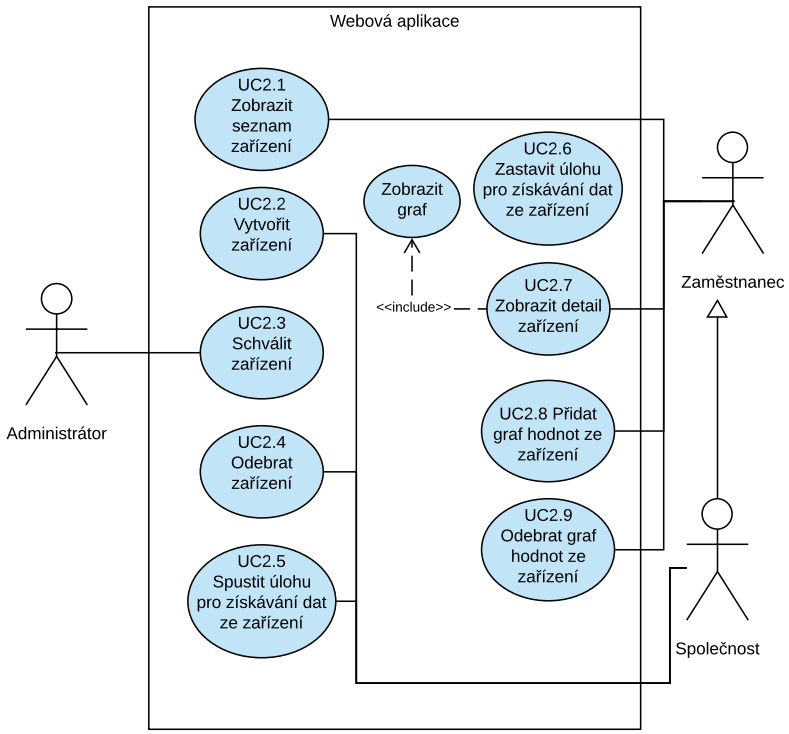
UC2.5 Spustit úlohu pro získávání dat ze zařízení

UC2.6 Zastavit úlohu pro získávání dat ze zařízení

UC2.7 Zobrazit detail zařízení

UC2.8 Přidat graf hodnot ze zařízení

UC2.9 Odebrat graf hodnot ze zařízení



Obrázek 3.4: Diagram případů užití pro správu zařízení.

3.4.2.1 UC2.2 Vytvořit zařízení

Popis: Správce společnosti přidává nové zařízení internetu věcí s modulem, který předává systému data.
Frekvence užití: Několikrát za měsíc.
Předpoklady:

- Uživatel je přihlášený a typu správce společnosti,
- společnost je schválená administrátorem systému,
- společnost disponuje implementovaným modulem komprimovaným v ZIP archivu.

Kroky:

1. Uživatel se nachází v sekci *Devices*.
2. Uživatel klikne na tlačítko *Create new device* v pravém horním rohu obrazovky.
3. Zobrazí se stránka s formulářem pro vytvoření nového zařízení.
4. Uživatel zapíše název zařízení, nahraje soubor s modulem, vyplní frekvenci spouštění modulu a zapíše strukturu modulem vrácených hodnot ve formátu JSON.
5. Uživatel klikne na tlačítko uložit a je přesměrován na stránku se zařízeními.

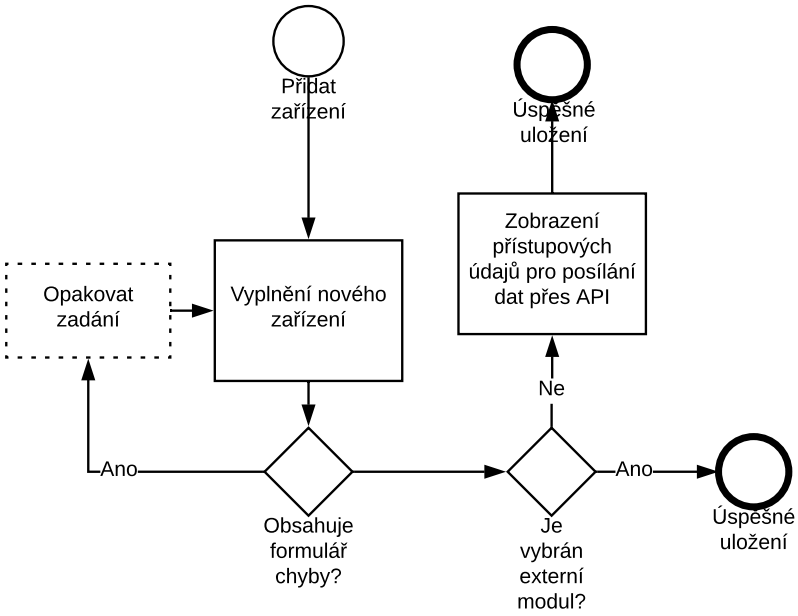
Varianty:

- Varianta kroku **1**: Uživatel se nachází na domovské stránce.
- Varianta kroku **4**: Uživatel zapíše pouze název zařízení (hodnoty se budou posílat přes API požadavky z externě spuštěného programu nebo zařízení).

Problémy:

- V případě nevyplnění zařízení je potřeba co nejdříve zobrazit API klíč a adresu pro zápis hodnot.

Diagram tohoto případu užití naleznete na obrázku 3.5.



Obrázek 3.5: Podrobný diagram případu užití UC2.2.

3.4.3 Případy užití správy notifikací

Případy užití v této části můžete vidět na diagramu 3.6.

UC3.1 Zobrazit seznam notifikačních pravidel

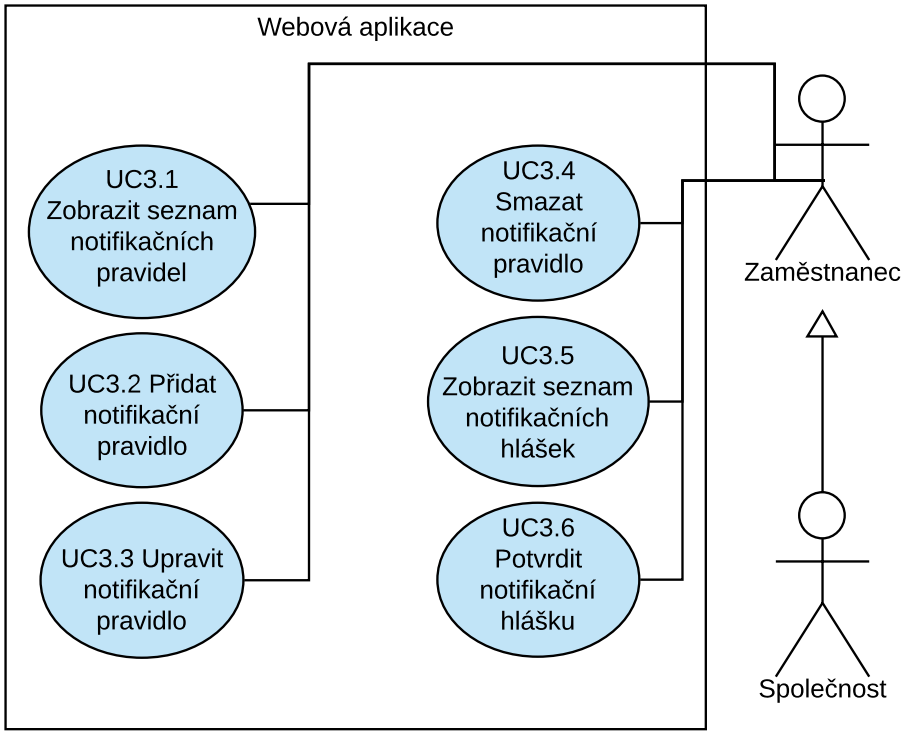
UC3.2 Přidat notifikační pravidlo

UC3.3 Upravit notifikační pravidlo

UC3.4 Smazat notifikační pravidlo

UC3.5 Zobrazit seznam notifikačních hlášek

UC3.6 Potvrdit notifikační hlášku



Obrázek 3.6: Diagram případů užití pro správu notifikací.

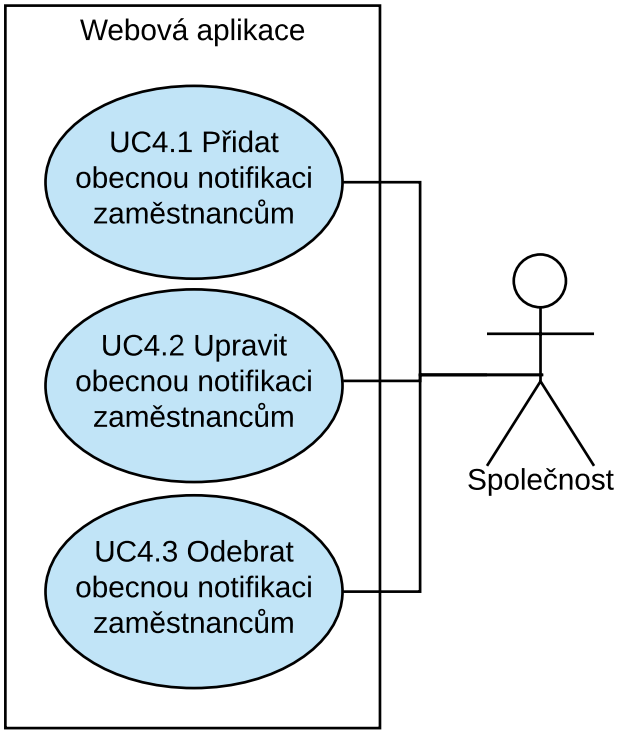
3.4.4 Případy užití pro nastavení

Případy užití v této části můžete vidět na diagramu 3.7.

UC4.1 Přidat obecnou notifikaci zaměstnancům

UC4.2 Upravit obecnou notifikaci zaměstnancům

UC4.3 Odebrat obecnou notifikaci zaměstnancům



Obrázek 3.7: Diagram případů užití pro nastavení.

3.5 Doménová analýza

Aby bylo možné zjistit a pochopit hlubší vztahy mezi jednotlivými entitami a procesy aplikace, vypracoval jsem doménovou analýzu jednotlivých částí dle funkčních požadavků.

3.5.1 Uživatelé

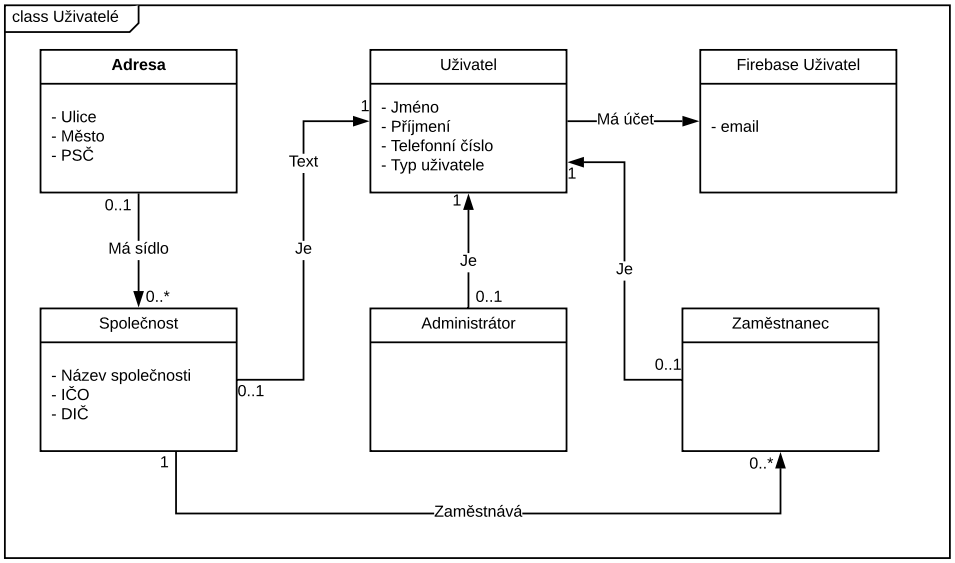
V této části popisuji propojení jednotlivých entit, které dohromady skládají informace o uživateli, jeho roli a případně dalších detailů.

Tato sekce obsahuje následující entity:

- **Uživatel Firebase** je uživatel registrovaný pomocí autentizační infrastruktury Google Firebase.
- **Uživatel** obsahuje základní informace o uživateli, propojení k uživatelskému účtu na Google Firebase a jeho roli.

- **Společnost** je typ uživatelské role, která obsahuje informace o společnosti a propojení k entitě nesoucí informace o adrese společnosti a zařízeních, které má uložené.
- **Zaměstnanec** je typ uživatelské role, která je propojená s danou společností a uživatelem systému.
- **Administrátor** je typ uživatele s nejvyšším oprávněním.

Diagram těchto entit je graficky popsán na obrázku 3.8.



Obrázek 3.8: Diagram entit sekce Uživatelé.

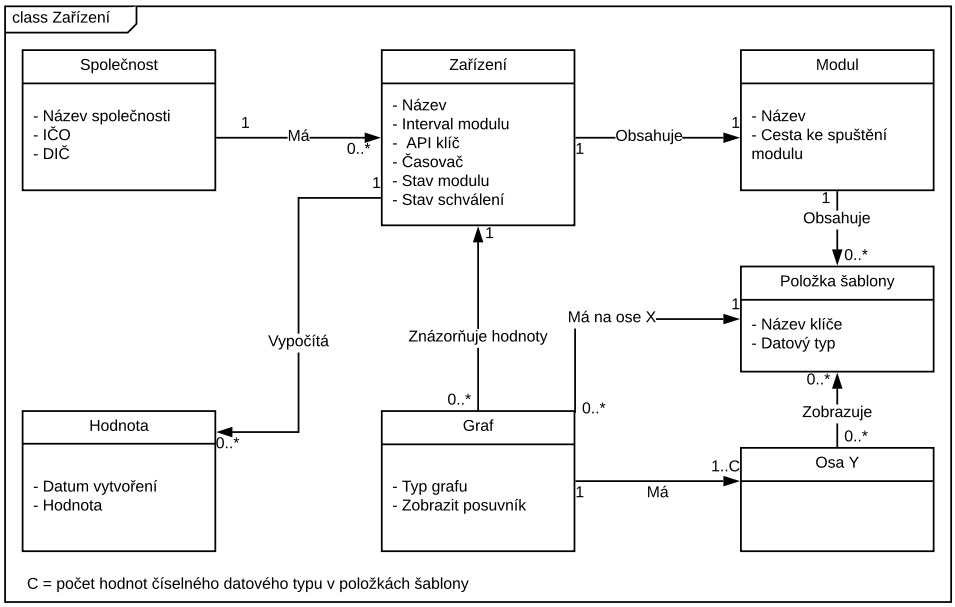
3.5.2 Zařízení

Další částí je analýza v rámci zařízení, které je složeno z následujících entit:

- **Zařízení** je definice zařízení společně s informací o modulu, který data tohoto zařízení získává
- **Hodnoty** jsou výstupy z daného modulu zařízení

- **Grafy** jsou definice grafů, které jsou z hodnot daných zařízení generovány

Diagram těchto entit je graficky popsán na obrázku 3.9.



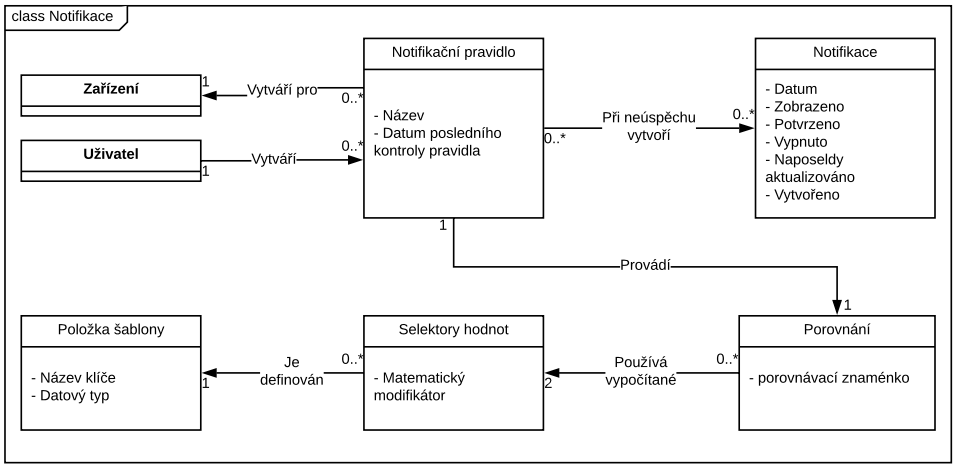
Obrázek 3.9: Diagram entit sekce Zařízení.

3.5.3 Notifikace

Sekce **Notifikace** obsahuje entity o notifikačních pravidlech a samotných notifikacích.

- **Notifikační pravidla** jsou matematické definice mezních hodnot zařízení
- **Notifikace** je výsledek notifikačního pravidla, které bylo porušeno.

Diagram těchto entit je graficky popsán na obrázku 3.10.



Obrázek 3.10: Diagram entit sekce Notifikace.

Kapitola 4

Návrh

V této kapitole se věnuji nejdříve výběru vhodných technologií, které jsem při implementaci webové aplikace využil. Dále do této kapitoly zahrnuji návrh databázového schématu a uživatelského rozhraní.

4.1 Obecné použité technologie

Vzhledem k požadavkům zadavatele na použité technologie je celá práce napsána v jazyce JavaScript. Zadavatel se k použití tohoto jazyku rozhodl z důvodu velké podpory balíčků třetích stran umístěných v balíčkovacím systému NPM[3], který je podrobněji popsán níže.

4.1.1 ECMAScript a Babel.JS

V této sekci bych se rád vrátil k základům JavaScriptu. JavaScript vznikl v roce 1995 a od té doby prošel velkými změnami, které zároveň znamenaly velké množství nezávislých implementací.

Po převzetí JavaScriptu pod organizaci Ecma International roce 1997 vznikl **ECMAScript** standard *ECMA-262*. Ten byl implementován jazykem JavaScript (společností Netscape Communications) a jazykem JScript (společností Microsoft)[3].

Od svého vzniku se ECMAScript stále vyvíjí a rozšiřuje, vydané edice si můžete prohlédnout v tabulce 4.1.

Nicméně prohlížeče nejsou jednotné a každý může podporovat jinou verzi. Jistotu podpory ve všech prohlížečích má pouze ECMAScript verze 3, absolutní většina prohlížečů však podporuje verzi 5[4].

Nicméně i tato verze obsahuje zastaralou syntaxi, kvůli které by zdrojový kód byl nepřehledný a aplikaci by tak bylo náročné spravovat. Velký průlom přinesla až právě verze ECMAScript 6, označovaná též jako ES2015.

[3]Node package manager.

Tabulka 4.1: Verze ECMAScript[4]

1997	ECMAScript 1	První edice
1998	ECMAScript 2	Pouze ediční změny.
1999	ECMAScript 3	Přidány regulární výrazy a syntax *try/catch*
	ECMAScript 4	Nebylo vydáno
2009	ECMAScript 5	Přidán striktní mód a podpora JSON
2011	ECMAScript 5.1	Pouze ediční změny
2015	ECMAScript 6	Přidány třídy a moduly
2016	ECMAScript 7	Přidán exponenciální operátor (**) a `Array.prototype.includes`

Aby mohli vývojáři psát kód v přívětivé verzi ES6, vznikl JavaScript kompilátor **Babel.JS**. Ten nejenže umožňuje používat ES6, který následně kompiluje do ECMAScript 5 pro podporu kompatibilních prohlížečů, ale obsahuje tzv. *Polyfill* obsahující funkcionalitu, která v JavaScriptu buď úplně chybí nebo je špatně implementována, a proto ji rozšiřuje[5].

Na obrázku 4.1 můžete vidět rozdíl mezi syntaxí v ECMAScript 6 a zkompilovanou verzí pro podporu prohlížečů ve verzi ECMAScript 5.

```
const foo = array =>
    array.map(x => x**x + first);

const array = [1,2,3];
foo([...array]);

const bar = ({ key1, key2 }) => {
    const x = key1 + key2;
    return x * 100;
}

const object = { key1: 1, key2: 2 };
const { key1, key2 } = object;
```

```
"use strict";

var foo = function foo(array) {
    return array.map(function (x) {
        return Math.pow(x, x) + first;
    });
};

var array = [1, 2, 3];
foo([].concat(array));

var bar = function bar(_ref) {
    var key1 = _ref.key1,
        key2 = _ref.key2;

    var x = key1 + key2;
    return x * 100;
};

var object = { key1: 1, key2: 2 };
var key1 = object.key1,
    key2 = object.key2;
```

Obrázek 4.1: Ukázka kompilace JavaScript kódu z ECMAScript 6 na ECMAScript 5.

```
1 ~/git/mi-dip-iot-reporting-server npm install express
```

Zdrojový kód 4.1: Příklad instalace balíčku Express pomocí NPM

4.1.2 Node.JS a NPM

Node.JS je open-source prostředí pro spouštění aplikacích napsaných v JavaScriptu, který je podporován většinou operačních systémů. Node.JS spouští asynchronní, jedno-vláknové a neblokující aplikace, a díky tomu se jedná o paměťově nenáročný systém. Vývojář má také přístup ke komunikaci s operačním systémem, a proto se jedná o vhodný nástroj především pro psaní serverových aplikací[6].

Balíčkovací systém **NPM** je navržen pro použití v prostředí Node.JS, jehož je zároveň součástí při instalaci[7]. NPM nabízí mnoho modulů, které uživatel při tvorbě aplikace může použít. Díky tomu, že většina balíčků jsou vyvíjeny pod otevřenou licencí, může je díky tomu programátor modifikovat a použít pro vlastní potřeby.

Instalace balíčku probíhá v konzoli z adresáře daného projektu, na kterém uživatel pracuje. Na ukázce výpisu z konzole 4.1 můžete vidět příkaz pro instalaci balíčku Express.JS, o němž se budu zmiňovat níže. NPM moduly jsou instalovány do složky `node_modules` v kořeni adresáře, ve kterém se projekt nachází.

Existují balíčky, které není vhodné svazovat přímo k danému projektu, ale instalovat je globálně. Týká se to především pomocných CLI[4] nástrojů, které slouží k vývoji či spuštění aplikace, ale nejsou její přímou součástí. Příkladem je balíček Gulp, který je popsán v následující kapitole.

4.1.3 Gulp

Bez použití pomocných nástrojů by bylo velmi obtížně sestavit pracovní postup pro sestavení Node.JS aplikace; vývojář by musel všechny konfigurační proměnné či úlohy související se sestavením aplikace spouštět ručně z konzole nebo je přímo náročně implementovat.

Dva nejpoužívanější open-source nástroje, které tento proces automatizují, se jmenují **Gulp** a **Grunt**. Z důvodu mé osobní zkušenosti, ale především kvůli vyšší rychlosti[8], jsem se rozhodl pro použití balíčku **Gulp**.

Jednotlivé úlohy jsou implementovány v Node.JS a nejčastěji se jedná o proces vytvoření produkčního buildu[5] aplikace nebo spouštění aplikace ve vývojovém prostředí. Gulp může být také použit pro operace, které je nutné vykonat před spuštěním, či po skončení práce s aplikací, jedná se například o kopírování obrázků do adresáře s produkční aplikací či minifikaci[6] JavaScript nebo CSS, kterou můžete vidět v ukázce zdrojového kódu 4.2.

[4] Command Line Interface; příkazový řádek.

[5] Kompilované verze vyvíjeného software.

[6] Komprese zdrojového kódu do hůře čitelné podoby pro produkční účely.

```
1 const gulp = require('gulp')
2 const minifyCss = require('gulp-minify-css');
3
4 gulp.task('min-css', () => gulp.src('./css/stlye.css')
5     .pipe(minifyCss())
6     .pipe(gulp.dest('./dest'));
7 });
```

Zdrojový kód 4.2: Příklad zápisu úlohy psané v *Gulp*

4.2 Použité technologie serverové části

Serverová strana aplikace je postavena, jak je již popsáno výše, na technologii Node.JS, která obsahuje vestavěný modul *http* pro práci s daty nad protokolem HTTP.

Tento modul je však velmi obecný a vystavit nad ním celou aplikaci by byla časově náročná záležitost. Proto jsem se rozhodl použít volně dostupný framework **Express.js**, který poskytuje funkce pro jednoduché a efektivní vybudování serveru, a stal se tak v této oblasti de facto standardem.

4.2.1 Express.JS

Framework **Express.JS** vznikl v roce 2010 pod licencí MIT a bývá porovnáván s frameworkem *Sinatra* pro jazyk Ruby[9].

Mezi hlavní výhodami a důvody obliby toho frameworku jsou především rozsáhlé možnosti nastavení směrování a zaměření na maximální výkonnost.

Důležitou vlastností je také podpora *middleware* funkcí. Jedná se o obecné funkce volané před provedením konkrétní akce definované směrováním, které mají plný přístup k objektům požadavku, odpovědi a obsahují funkci zpětného volání, tzv. *callback* pro volání navazujících funkcí.

Výhodou je možnost skládání těchto funkcí za sebe, díky čemuž lze dosáhnout znovupoužitelnosti kódu. Ve zdrojovém kódu 4.3 se nachází ukázka autentizace uživatele prostřednictvím middleware. V ukázce si také můžete prohlédnout způsob vytváření směrování, v tomto případě koncového bodu `GET /me` za předpokladu, že je konstanta `app` kořenovým směrováním.

4.2.2 Použité knihovny

V následujících podsekcích vyjmenuji a popíši několik, dle mého názoru nejužitečnějších, NPM balíčků, kterém jsem při implementaci serverové strany webové aplikace použil.

```
1  const app = require('express')();
2
3  const checkAuthentication = (req, res, next) => {
4      if (/* user is loggedin */) {
5          req.user = /* current user information */;
6          next(); return;
7      }
8
9      res.status(401).json({Error: 'User unauthorized'});
10 }
11
12 const getContent = (req, res) =>
13     res.json({user: req.user});
14
15 app.get('/me', checkAuthentication, getContent);
```

Zdrojový kód 4.3: Příklad použití *middleware* v *Express.JS*

4.2.2.1 body-parser

Body-parser je middleware, který analyzuje, zpracuje a převede do objektové podoby obsah těla HTTP požadavku. Díky tomu je možné pracovat s tělem zprávy jako s klasickým JavaScript objektem[10].

4.2.2.2 cookie-dough

Cookie-dough je knihovna pro manipulaci s HTTP cookies v požadavku a odpovědi[11].

4.2.2.3 cookie-parser

Cookie-parser je middleware, který slouží pro zpracování a převedení HTTP cookies do objektové podoby[12].

4.2.2.4 express-fileupload

Express-fileupload je middleware pro práci se soubory, které jsou na server nahrány prostřednictví protokolu HTTP[13].

4.2.2.5 firebase a firebase-admin

Knihovny **firebase** a **firebase-admin** poskytují nástroje pro operaci s infrastrukturou Google Firebase[14].

4.2.2.6 joi

Joi je balíček pro popis objektového schématu (například datového modelu či datové struktury zaslané uživatelem v HTTP požadavku) a jeho validaci[15].

4.2.2.7 lru-cache

Lru-cache je balíček poskytující cache[16], která používá algoritmus **least-recently-used**, který vyřazuje nejdéle nepoužívané prvky při jejím zaplnění[17].

4.2.2.8 mathjs

Math.JS je knihovna, která dokáže zpracovat text do matematických formulí a následně je vyhodnocovat[18].

4.2.2.9 mongoose

Pro modelování dat nad databázovým systémem **MongoDB** je použita knihovna **mongoose**. Ta v sobě poskytuje možnost vytváření *modelů* a *dotazů*, *validace*, *bussiness logiku* a další funkce pro usnadnění práce s databází[19].

4.3 Použité technologie klientské části

Klientská část webové aplikace je typově SPA[7]. Výhodou tohoto typu aplikací je především rychlá odezva, protože při uživatelské interakci není třeba stahovat celý obsah HTML, ale pouze data, která je potřeba aktualizovat[20].

Existují dva největší frameworky pro implementaci webových aplikací typu SPA: **React** a **Angular.JS**. Z důvodu mé znalosti knihovny React jsem se rozhodl ji pro vytvoření IOTCP použít.

Klientská strana se skládá ze dvou složek:

- serverové strany psané v Node.JS s použitím Express.js, která slouží pro renderování obsahu do HTML dokumentu,
- sestavené aplikace, jejíž výsledný kompilovaný kód je vložen jako JavaScript do HTML stránky.

4.3.1 React

Knihovna React vznikla jako open-source projekt v roce 2013 pod společností Facebook, která React před jeho oficiálním vydáním již několik let používala[21].

React zastupuje funkci pohledové části architektury MVC a jeho největším přínosem je odstínění vývojáře od DOM[8].

4.3.1.1 Komponenty

Základním kamenem Reactu jsou komponenty, díky kterým je možné rozdělit uživatelské rozhraní do nezávislých a znovu použitelných izolovaných částí.

[7] Single page application.

[8] Document object model; stromová reprezentace obsahu HTML stránky.

V Reactu rozlišujeme dva základní typy komponent podle své funkčnosti: *hloupé* a *chytré*.

Hloupé komponenty jsou prezentační komponenty, jejichž jediným úkolem je předávat informaci do DOMu. Tyto komponenty dostávají data, která následně vykreslí. Nemají žádný interní stav a nemůžou ovlivňovat jiné komponenty. Nejčastěji se hloupé komponenty vyskytují na listech stromu reprezentující zanořování komponent v rámci aplikace[22].

Zodpovědností chytrých komponent je provádět akce, které pracují se stavem aplikace a spojují aplikaci kombinováním komponent v jednotný celek[22].

Existují dva možné způsoby zápisu nové komponenty:

- bezstavová komponenta, která je zapsána jako *čistá funkce*[9], ve které nelze použít interní stav, a proto je možné ji zároveň označit jako hloupou komponentu,

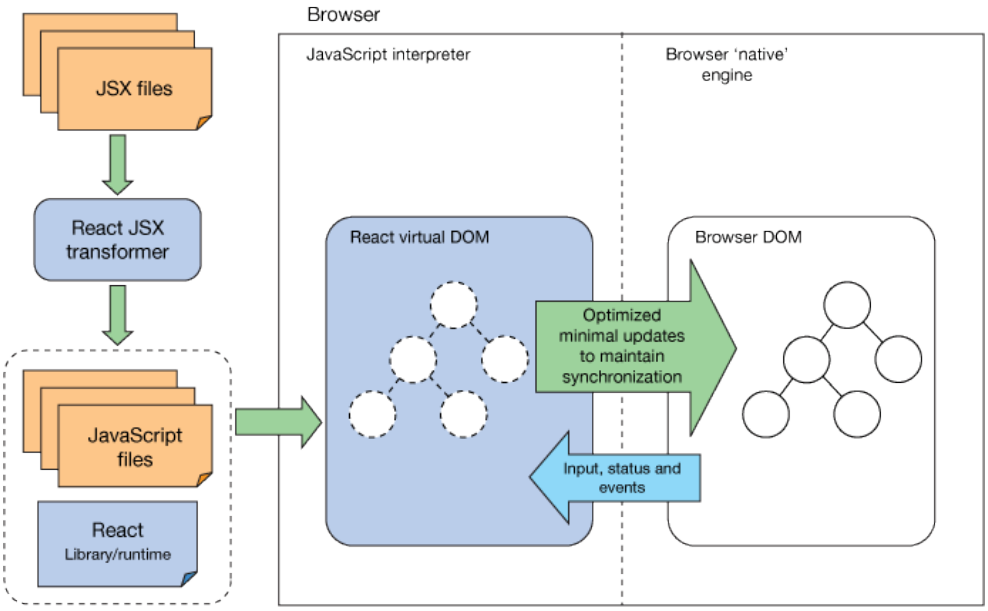
- stavová komponenta, která umí uchovat svůj stav a pracovat s ním. Je vytvořena jako třída, která rozšiřuje základní React komponentu obsahující jedinou povinnou metodu k implementaci `render`[22].

4.3.1.2 JSX

Elementy, které popisují, co má být zobrazeno, se zapisují syntaxí zvanou **JSX**, která je velmi podobná značkovacímu jazyku HTML. Jedná se o tzv. *syntaktický cukr*, tedy syntaxi, jejímž cílem je vývojáři usnadnit zápis kódu. Při kompilaci se tento zápis transformuje pomocí kompilátoru Babel.JS do klasických JavaScript funkcí[23]. React následně vytváří vlastní virtuální DOM, který s pomocí algoritmů detekují změny a aktualizuje skutečný DOM[24].

Přehledný diagram této transformace můžete vidět na obrázku 4.2.

[9] Funkce, které nemění hodnotu argumentů, neprovádí vedlejší operace a vrací nové objekty.



Obrázek 4.2: Diagram transformace React JSX do JavaScriptu s ovlivněním DOM prohlížeče[25].

4.3.1.3 Základní syntaxe

Při použití Reactu je potřeba nejdříve importovat do zdrojového kódu, balíček React a to kvůli následné kompilaci *JSX* tagů.

Při zápisu komponent je možné přiřadit atributy. U bezstavových komponent se tyto atributy objeví v prvním argumentu funkce jako objekt všech přiřazených atributů. Ve třídách stavových komponent jsou tyto parametry dostupné v objektu `this.props`.

Objektem, který reprezentuje stav u stavové komponenty, je `this.state`. Do tohoto objektu je možné ukládat jakkoliv strukturovaná data. Stav komponenty je možné měnit pomocí metody `this.setState`, která obsahuje dva parametry. Prvním parametrem je objekt, který se slučuje se současným stavem komponenty, druhý parametr pak umožňuje definovat *callback* po úspěšném provedení změny stavu.

Při změně stavu metodou `this.setState` nebo změně hodnot vstupních parametrů v `this.props` dochází k novému vykreslení dané komponenty.

Příklad implementace jednoduché stavové komponenty společně se způsobem vykreslení do HTML stránky můžete vidět v ukázkovém kódu 4.4.

```
1  import React from 'react';
2
3  class Hello extends React.Component {
4    render() {
5      return (
6        <div>
7          Hello {this.props.name}
8        </div>
9      );
10   }
11 }
12
13 ReactDOM.render(
14   <Hello name="Diploma" />,
15   document.getElementById('app')
16 );
```

Zdrojový kód 4.4: Příklad React komponenty s vykreslením v DOM

4.3.1.4 Životní cyklus komponenty

Každá komponenta obsahuje přepisovatelné metody, které se týkají jejího životního cyklu. Metody s prefixem `Will` jsou vykonávány těsně předtím, než nastane nějaká událost. Metody s prefixem `Did` jsou volány hned poté, co událost nastala.

V následujícím seznamu naleznete metody seřazené podle jejich pozice v životním cyklu komponenty[26].

- Metody volané při vytvoření komponenty:
  - metoda `constructor(props)` je volána před načtením komponenty,
  - metoda `componentWillMount()` je volána před načtením komponenty po zavolání konstruktoru,
  - metoda `render()` vykreslí komponentu,
  - metoda `componentDidMount()` je volána po úplně prvním vykreslení komponenty.
- Metody volané při změně stavu (*state*) nebo parametrů (*props*):
  - metoda `componentWillReceiveProps(nextProps)` je volána při změně *props* a tyto nové parametry jsou obsaženy v argumentu funkce `nextProps`.
  - metoda `shouldComponentUpdate()` říká, zda-li je při změně stavu nutné komponentu překreslit. Implementací této funkce je možné dosáhnout obrovské efektivity.
  - metoda `render()` vykreslí komponentu,

  - metoda `componentDidUpdate()` je volána po úspěšném překreslení komponenty.

- Metody volané při rušení komponenty:

  - metoda `componentWillUnmount()` je volána při odstranění komponenty z DOM.

- Metody volané při zachycení chyby:

  - metoda `componentDidCatch()` je volána v případě zachycené chyby.

4.3.2 Redux

V době největšího rozvoje SPA bylo zjištěno, že ve zdrojovém kódu je potřeba daleko větší kontrola stavů aplikace. Mezi tyto stavy patří odpovědi serverů na HTTP požadavky, *kešovaná* data, data čekající k odeslání na server, ale i stavy udržující podobu uživatelského rozhraní.

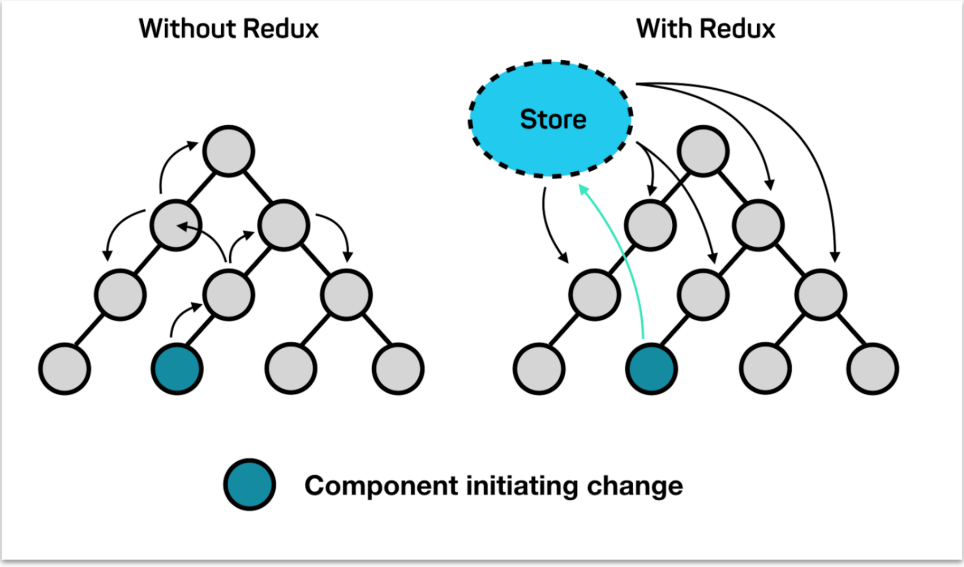
Správa těchto stavů, které se mohou velmi často měnit, není jednoduchá záležitost. Nacházejí se totiž na různých místech v aplikaci a jejich aktualizace mohou být vyvolány nejrůznějšími událostmi. Zároveň každá aktualizace stavu ovlivňuje spouštění překreslení stávajících komponent. Při mnoha různě navazujících změnách pak vývojář lehce ztratí kontrolu, jak aplikace vlastně funguje.

Poté vznikají další požadavky na vývoj SPA. Mezi ty patří například SSR[10], získávání dat před vykreslením pohledu (tzv. *prefetching*) a podobně. Komplexnost se pak dále zvyšuje z důvodu mísení dvou konceptů psaní JavaScript aplikací: *mutace* a *asynchronicita*, které sice React řeší, ale pouze na úrovni pohledů, resp. komponent, a nikoliv při vzájemných vztazích v rámci aplikace[27].

Podíváte-li se na strom umístěný vlevo na obrázku 4.3, můžete spatřit klasické oznamování změn komponent v Reactu. V případě, že bychom chtěli ovlivnit dvě nejvzdálenější komponenty, musely by téměř nad všemi komponentami tohoto stromu existovat callback[11] funkce, které by o změnách informovaly.

Tento problém inicioval vytvoření stavového kontejneru **Redux**, který používá globální úložiště, nad kterým mohou jednotlivé komponenty komunikovat tak, jak je znázorněno v pravé části obrázku 4.3.

[10] Server-side rendering; renderování aplikace na straně serveru.
[11] Funkce zpětného volání.



Obrázek 4.3: Rozdíly komunikace komponent při užití, resp. neužití vzoru Redux[28].

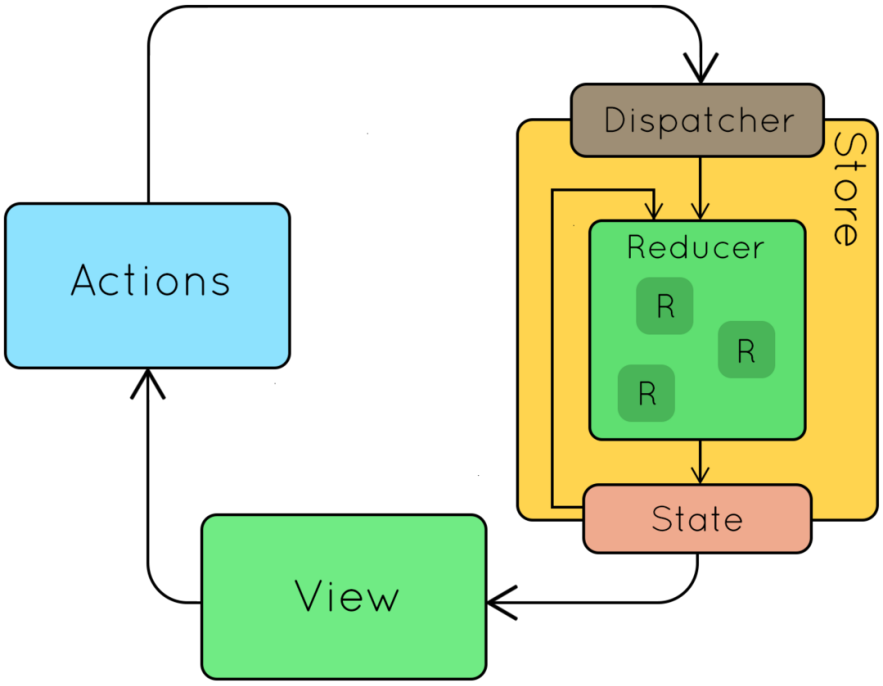
V Reduxu platí tři základní principy:

- **Jediný zdroj pravdy**. Stav aplikace je uložen ve stromové struktuře na jediném místě aplikace, tzv. *store*, na které všechna data a ostatní komponenty nahlíží a reagují na ně[29].
- **Stav je pouze pro čtení**. Stav aplikace je určen pouze pro čtení a jediná možnost, jak ho změnit, je vyvolat akci obsahující objekt, který popisuje změnu stavu[29].
- **Změny provádí čisté funkce**, které data přicházející z daných akcí mohou transformovat[29].

Data vyjadřují aktuální stav aplikace a stejná data pak musí vždy aplikaci uvést do stejného stavu, který je označován též jako *předvídatelný stav*.

Díky informacím o akcích a datech, která skrz ně protékají, je velmi snadné aplikaci ladit. Další výhodou je možnost vytvořit funkci *undo*, která vrací zpět jednotlivé akce, ale také možnost *serializovat* stav celé aplikace.

Na obrázku 4.4 se nachází diagram pro zpracování akcí ve vzoru Redux. Proces začíná zavoláním akce, nejčastěji z pohledu nebo z jiné akce. Tato akce je odeslaná (tzv. *dispatch*) a v případě, že má dojít k uložení dat do úložiště, je zachycena *reducerem*, který obsah zpracuje a uloží do stavu aplikace. Při změně stavu přijde pohledu aktualizace a dochází k jeho překreslení.



Obrázek 4.4: Diagram zpracování akcí vzoru Redux[30].

Redux je možné použít nejen s frameworkem React, ale i s jakoukoliv jinou pohledovou knihovnou napsanou v JavaScriptu[31].

4.3.3 Webpack

Frontend webové aplikace, na které pracuji, se snaží dodržovat principy modularity. Jednotlivé soubory jsou tedy rozděleny logicky do adresářů a v projektu se zároveň nachází několik desítek NPM modulů.

Pro mé potřeby je také velmi důležité rozlišovat vývojovou a produkční verzi aplikace, především kvůli logům a chybovým výpisům, které chci vidět pouze při vývoji.

Pokud by zdrojové soubory nebyly pro produkční použití optimalizovány ani kompilovány, vyskytlo by se několik závažných problémů.

Největší problém by způsobilo nízké rozšíření standartu ES6 v prohlížečích. Kód by tedy nebyl často vůbec spuštěn. A dále vysoké množství provedených HTTP požadavků pro získání veškerých implementovaných souborů.

Webpack je nástroj, který zdrojový kód rozdělí do jednotlivých balíčků složených pouze z na sobě závislých zdrojových souborů, určených velmi často podle směrovacích pravidel v aplikaci. Optimalizován je tak, aby byl vhodný

pro přenos k uživateli aplikace prostřednictvím internetu - např. *minifikací*, která optimalizuje velikost výsledného souboru, či převodem z ES6 do ES5. Základním stavebním kamenem jsou *loadery*, které dle typu importovaných souborů ve zdrojovém kódu provádí s daným souborem různé operace[32].

4.3.4 Použité knihovny

Na klientské straně aplikace jsem použil další NPM balíčky, které mi usnadnily implementaci. Níže jmenuji jen ty nejpodstatnější:

4.3.4.1 codemirror

Codemirror je knihovna, která poskytuje rozšiřitelný textový editor[33].

4.3.4.2 immutable

Immutable je knihovna, která poskytuje perzistentní nezměnitelné datové struktury, mezi které patří *list*, *mapa*, *zásobník*, *množina* a *záznam*. Data v daných strukturách nemohou být nikdy přímo změněna. Při změně pomocí definovaných metod se převezme, duplikuje aktuální objekt a pozmění data[34]. Tato knihovna je velmi často používána k implementaci úložiště Redux, který neměnnost přímo ze své definice vyžaduje.

4.3.4.3 isomorphic-fetch

Isomorphic-fetch je implementací **fetch API** z prohlížeče pro Node.JS, který umožňuje načítat zdroje ve webovém prohlížeči[35].

4.3.4.4 lodash

Lodash je knihovna, která poskytuje pomocné funkce pro běžné programátorské úlohy, například pro práci s poli, čísly a řetězci[36].

4.3.4.5 material-ui

Material-ui je balíček uživatelského rozhraní, který obsahuje sadu React komponent, jež implementují principy *Material Design* od společnosti Google[37].

4.3.4.6 moment

Knihovna **Moment** usnadňuje manipulaci s časy a daty.

4.3.4.7 prop-types

Prop-types je nyní oddělená část knihovny React, která slouží pro definování datových typů atributů jednotlivých komponent[38].

4.3.4.8 query-string

Balíček **query-string** slouží ke zpracování dotazů v URL adresách.

4.4 Databázový model

Pro tvorbu webové aplikace jsem se rozhodl použít dokumentovou, tedy nerelační databázi, které bývají označené také pod pojmem NoSQL. Výhodou tohoto typu databáze je rychlé a efektivní zpracování velkých kolekcí dat se zaměřením na výkon, spolehlivost a agilnost, což prospívá především oblasti big data a real-time webům[39].

Obecně k databázovým transakcím existují dva přístupy: *ACID* a *BASE*. Pravidly přístupu ACID jsou:

- **atomicita** - transakce buď proběhne celá, nebo vůbec,
- **konzistence** - transakce převede databázi z jednoho konzistentního stavu do jiného konzistentního stavu a je zaručeno neporušení integritních pravidel,
- **izolovanost** - změny transakce před jejím dokončením nejsou pro ostatní transakce viditelné.
- **trvalost** - změny úspěšně dokončené transakce jsou v databázi uloženy trvale[40].

Mezi pravidla přístupu BASE patří:

- **V základu k dispozici** - garance dostupnosti podle pravidel *CAP* teorému,
- **měkký stav** - stav systému nemusí být celou dobu konzistentní,
- **eventuální konzistence** - systém bude mít po nějaké době konzistentní stav[40].

Na rozdíl od relačních databází, které se vzhledem k transakcím řídí principem ACID, NoSQL databáze používají pro transakce přístup BASE. Ten není tak striktní jako ACID. Díky tomu je dovolena dočasná nekonzistence dat a neblokování zápisu, aby se zvýšila dostupnost, výkon a zrychlila odezva[40].

NoSQL databáze byly dříve pravým opakem relačních databází - dodržoval se přesný rozdíl mezi výše zmíněnými přístupy. Tyto databáze byly zaměřené na dostupnost s co nejkratší odezvou, bez nutnosti definovat strukturu dat, bez podpory vztahů jednotlivých záznamů a možnosti provádět nad nimi spojovací operace (např. JOIN).

V současné době však není rozdělení přístupů *ACID* a *BASE* striktní v závislosti na druhu databázového modelu a některé databázové přístupy se

prolínají. U relačních databází je tak možné nastavit stupeň izolace, naopak u NoSQL databází je možné provádět transakce s vlastnostmi ACID[40].

V platnosti by však stále mělo zůstat rozlišení podle CAP teorému - věty, která říká, že distribuovaný systém může splňovat maximálně dvě ze tří vlastností:

`C` – **konzistence** – každé čtení vrací výsledek poslední zápisu nebo skončí chybou.

`A` – **dostupnost** – každý požadavek dostává nechybovou odpověď.

`P` – **odolnost vůči přerušení** – systém pokračuje v činnosti i za předpokladu, že dojde ke zdržení či ztrátě dat v rámci sítě[40].

V důsledku tak lze říci, že relační databáze by měly být typu `CA`, zatímco NoSQL databáze typu `AP`.

Automatickou konverzi dat mezi NoSQL databází a programovacím jazykem zajišťuje ODM[12]. Jedná se o techniku mapování paměťových struktur daného programovacího jazyka na dokumentové formáty, zejména JSON[41].

4.4.1 MongoDB

Pro implementaci webové aplikace jsem použil databázový systém MongoDB. Jedná se o dokumentovou databázi, která používá dokumenty formátu BSON podobný klasickému formátu JSON, ale optimalizovaný pro tuto databázi. MongoDB je nejpopulárnější NoSQL open-source software[42].

Mezi hlavní výhody tohoto systému patří:

- **Agregační funkce**

  Agregační funkce slouží pro efektivní zpracování dat. Patří mezi ně například funkce **MapReduce** pro zpracování velkých množin dat paralelním zpracováním. Skládá se z implementací dvou funkcí: *map* a *reduce*. Implementací funkce *map* je z dvojic `klíč` a `hodnota` vygenerována pracovní dvojice `klíč2` a `hodnota2`. Tato pracovní data jsou poté spojena funkcí *reduce* podle stejného klíče[43].

- **Formát BSON**

  BSON[13] je formát vytvořený speciálně pro MongoDB. Data jsou serializovaná v binární formě, díky čemuž je možná podpora datumů a binárních dat, které JSON nepodporuje. Formát BSON používá jako primární klíč hodnotu `_id`, který slouží jako identifikátor a je automaticky generovaný databázovou službou[43].

[12] Objektově-dokumentové mapování.
[13] Binary JSON.

- **Horizontální rozdělení databáze do více serverů**

  MongoDB škáluje horizontálně pomocí shardingu[14], který se používá především k podpoře velkých datový skladů s vysokou propustností[43][44].

- **AdHoc dotazy**

  MondoDB podporuje rozsahové dotazy, hledání podle pole nebo regulárních výrazů. Dotazy následně vrací specifická pole dokumentů, která mohou obsahovat JavaScript funkce[43].

- **Bezschémová databáze**

  MongoDB je databáze bez schématu. Díky tomu je mnohem flexibilnější než tabulky relačních databází. Ukládány jsou pouze serializované objekty bez potřeby mapování datových typů[43].

- **Indexace**

  Každé pole může být indexované, podobně jako v relačních databázích, pro zlepšení výkonu vyhledávání. Je možné používat jak primární, tak i sekundární index[43].

- **Ukládání souborů**

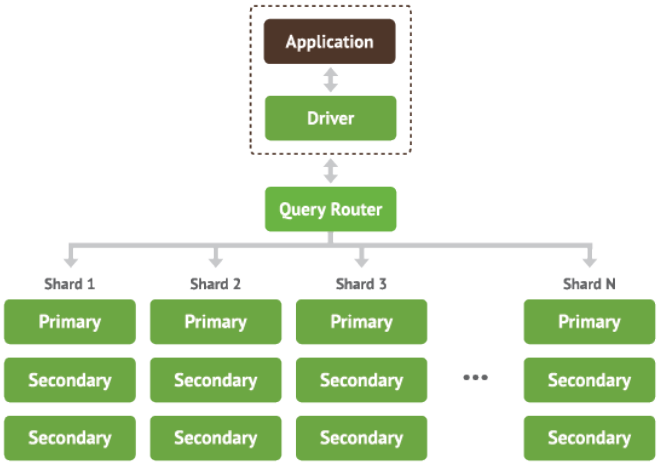
  MongoDB je možné použít jako souborový systém (zvaný *GridFS*), který se zároveň stará o vyvažování zátěže a replikaci dat přes více serverů[43].

- **Replikace**

  Aby se zamezilo ztrátě dat, podporuje MongoDB sadu replik obsahující dvě či více kopií dat. Primární replikace implicitně provádí čtecí a zapisovací operace. Ostatní repliky udržují konzistentní kopii dat primární repliky. V případě selhání primární repliky dochází k výběru náhradní repliky[43].

V diagramu na obrázku 4.5 je znázorněno použití metody horizontálního rozdělení pomocí *shardingu* společně s replikací každého stroje.

[14] Metoda pro distribuci dat na více strojích.



Obrázek 4.5: Diagram horizontálního rozdělení s použitím shardingu společně s replikací dat[45].

Pro implementaci techniky ODM jsem použil NPM knihovnu mongoose popsanou v kapitole 4.2.2.9.

4.4.2 Databázové kolekce webové aplikace

V aplikaci evidujeme následující kolekce:

- **Users** – pro ukládání uživatelů.
- **Companies** – pro ukládání společností.
- **Employees** – pro ukládání zaměstnanců.
- **Devices** – pro ukládání IoT zařízení.
- **Graphs** – pro ukládání grafů jednotlivých zařízení.
- **Values** – pro ukládání hodnot z jednotlivých zařízení.
- **NotificationRules** – pro ukládání notifikačních pravidel.
- **Notifications** – pro ukládání notifikací z notifikačních pravidel.

4.5 Uživatelské grafické rozhraní

Při zadání této práce jsem dostal prostor pro vytvoření vlastního uživatelského rozhraní aplikace. Vzhledem k její povaze jsem nemusel do celkového návrhu zahrnovat následující části:

- **Responzivní design**

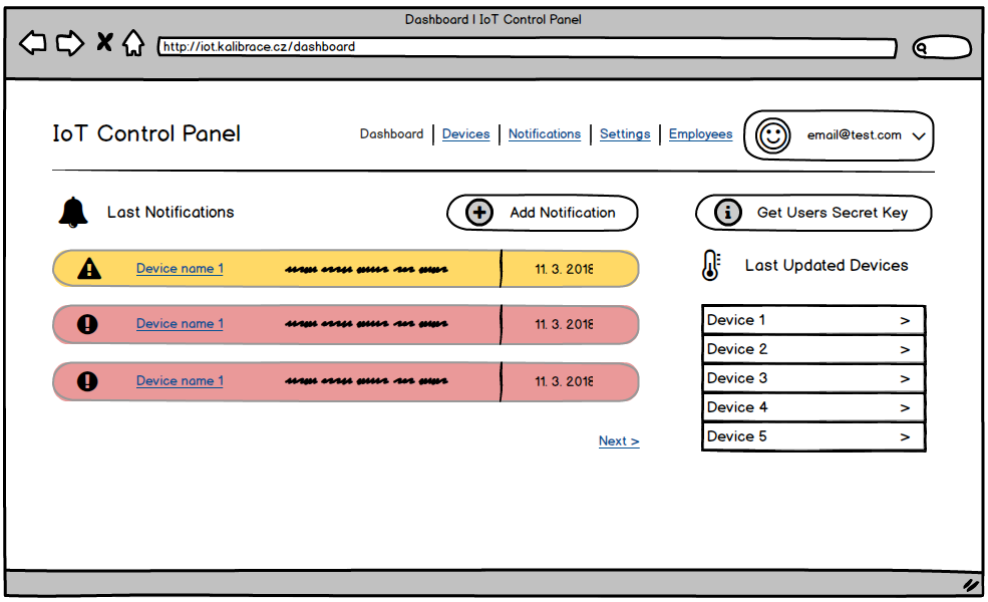
  Není potřeba, aby aplikace podporovala responzivní zobrazení – tedy zobrazení, které se automaticky přizpůsobuje danému typu zařízení, na kterém je obsah zobrazen.

- **Grafický design**

  Aplikace nepotřebuje perfektní a líbivý webový design. Typickému uživateli této webové aplikace by neměla být perfektní vizualizace zásadní.

Důležitou částí je naopak funkčnost a přehlednost aplikace. Pro lepší pochopení fungování výsledné webové aplikace jsem vytvořil drátěné modely (tzv. *wireframe*), jejichž úkolem je definovat funkce a rozložení funkčních prvků na stránce, nikoliv grafický návrh.

Drátěné modely jsem vytvořil v aplikaci Balsamiq Mockup. Poté jsem je použil v prototypu, díky kterému je možné simulovat průchod celou aplikací.



Obrázek 4.6: Drátěný model dashboardu správce společnosti.

Na základě připomínek zadavatele byl domluven finální návrh webové aplikace. Ukázku drátěného modelu můžete vidět na obrázku 4.6. PDF dokument s možností průchodů celou aplikací pak naleznete na přiloženém DVD.

Aplikace obsahuje dvě šablony: pro *přihlášené* a *nepřihlášené* uživatele.

Nepřihlášeným uživatelům je obsah zobrazen ve středu obrazovky. Ve vrchní části se nachází logo aplikace a pod ním box s přihlašovacím formulářem,

který dále obsahuje odkazy na stránku registrací a zapomenutým heslem. Tyto stránky pak mají podobné grafické uspořádání a liší se pouze obsah formulářů.

Přihlášeným uživatelům je v horní části aplikace zobrazen jednotný panel pro všechny stránky, který se skládá z loga umístěného vlevo a menu, které se nachází vpravo. Pod horním panelem se nachází pro každou stránku unikátní obsah.

4.5.1 Material Design

Pro implementaci drátěných modelů jsem použil vizuální jazyk **Material Design**, který vyvíjí společnost Google. Jedná se o jednotný systém, který vytváří sjednocené grafické nástroje napříč platformami a velikostmi jednotlivých zařízeních, který je postavený na základu teorie o dobrém grafickém uspořádání s inovacemi a uplatněním technologie a vědy.

Material Design využívá rozvržení stránek založené na mřížkách, animace, přechody, osvětlení a stínování jednotlivých komponent a zakládá se na devíti hlavních principech:

- **Materiál je metafora**. Teorie racionalizovaného prostoru a pohybového systému zvaná *Materiální metafora* je inspirována studiem papíru, inkoustu a hmatatelnou realitou[46].
- **Povrchy jsou intuitivní a přirozené**. Povrchy a hrany poskytují vizuální prvky, které jsou spojeny s realitou a lidskými zkušenostmi[46].
- **Hloubka prostoru poskytuje interakci**. Světlo, povrch a pohyb jsou klíčem k informování o interakci mezi objekty[46].
- **Jeden přizpůsobivý design**. Každé zařízení, jakéhokoliv typu či velikosti, používá stejný základní systém. Ten bývá vhodně přizpůsoben danému typu zařízení. Barvy a prostorové vztahy zůstávají stejné[46].
- **Obsah je výrazný, vizuální a cílený**. Výrazný design přináší hierarchii se specifickým významem a zaměřením[46].
- **Barva, povrch, a ikony zdůrazňují akce**. Uživatelské akce jsou podstatou *experience design*[15]. Základními akcemi jsou prvky, které transformují celý návrh. Jejich zdůraznění tak zjednodušuje základní funkcionalitu a poskytuje uživatelům záchytné body[46].
- **Uživatelé zahajují změny**. Vyvolá-li uživatel akci, pak změny probíhají právě z místa, ze kterého byla akce inicializována[46].

[15]Přístup ve snaze vytvářet smysluplnou zkušenost, který kombinuje všech 5 smyslů, interaktivitu a emocionální kontext.

- **Animace probíhají synchronně**. Všechny akce se odehrávají v jediném prostředí. Objekty jsou uživateli zobrazeny bez narušení zážitku i přesto, že se mohou transformovat či měnit své pořadí[46].
- **Pohyb je smysluplný a vhodný**. Slouží k soustředění pozornosti a udržení kontinuity. Zpětná vazba je jemná, ale jasná. Přechody účinné a zároveň souvislé[46].

Pro implementaci Material Design v Reactu existuje několik knihoven, které poskytují implementované a rozsáhle modifikovatelné komponenty. Jak už bylo uvedeno v kapitole 4.3.4, použil jsem NPM knihovnu Material-UI, o které si myslím, že patří mezi nejzdařilejší implementace Material Design pro React.

Kapitola 5

Architektura

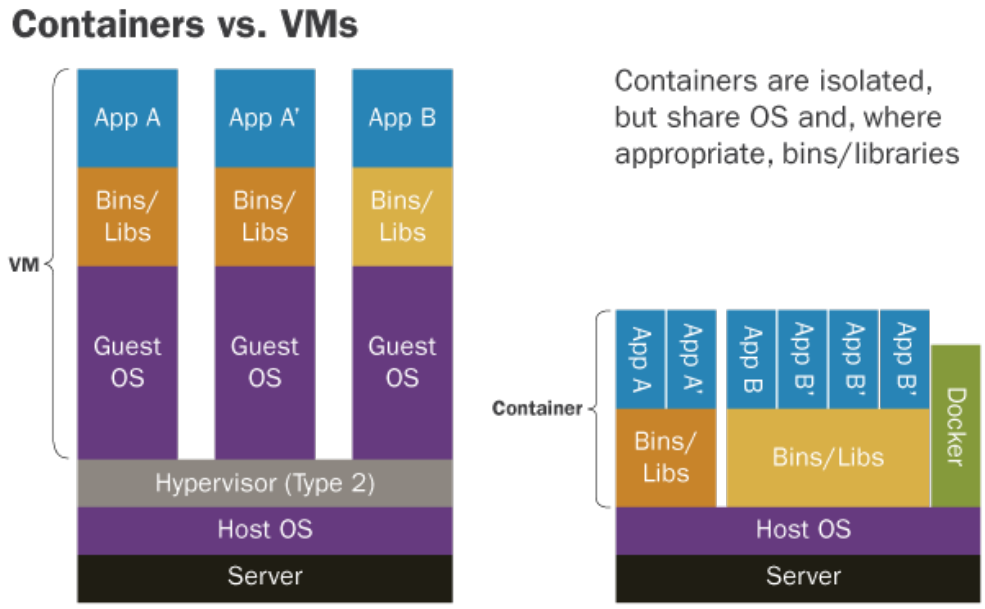
V této kapitole nejdříve popíši jednotlivé technologie, které byly použity pro nasazení do produkčního prostředí. Dále uvedu model nasazení a rozeberu strukturu modulů, které lze v rámci webové aplikace přidávat k zařízením internetu věcí. Nakonec uvedu v jakém prostředí byla aplikace vyvíjena.

5.1 Docker

Docker je open-source systém publikovaný v roce 2013, jehož cílem je poskytnout virtualizaci na úrovni operačního systému. Umožní tak spouštět více na sobě nezávislých vzájemně izolovaných virtuálních počítačů, které se označují pojmem *kontejnery*[47].

Docker byl primárně vytvořen pro operačním systém Linux, aby mohl využívat součásti linuxového jádra, tzv. kernelu, například knihovny *namespace* pro oddělení síťové komunikace či *cgroups* pro sledování využití zdrojů procesy. Díky tomu jsou jednotlivé kontejnery spouštěny v rámci jedné instance systému, nad kterou běží vrstva Dockeru a nad ní již virtualizované aplikace. Oproti klasickým virtuálním strojům, které obsahují vrstvu operačního systému daného stroje navíc, je tak výrazně snížena režie a náklady na provoz[48].

Grafické znázornění rozdílu virtuálního stroje oproti Dockeru můžete vidět na obrázku 5.1



Obrázek 5.1: Schéma reprezentace kontejnerů Dockeru v operačním systému v porovnání s virtuálními stroji[49].

Další výhodou kontejnerů je velmi malá velikost a s tím související lehká přenositelnost.

Systém Docker je obecně složen ze tří částí:

- **Software**. Rozlišujeme dvě instance softwaru Docker:
  - **Démon**[16], který spravuje kontejnery a jejich objekty a naslouchá požadavkům pocházejícím z Docker Engine API.
  - **Klient**, který umožňuje komunikovat s démonem Docker, skrze CLI[17][50].
- **Objekty**. Objekty Dockeru se vztahují k entitám, které se používají k sestavení aplikace. Mezi hlavní objekty Dockeru patří kontejner, obraz systému a služba:
  - **Kontejner** je izolované prostředí, ve kterém běží aplikace.
  - **Obraz** je šablona systému pro sestavení kontejneru.
  - **Docker služba** umožňuje škálování více démony, které spolu komunikují[50].

[16] Dlouhodobě spuštěný program v pozadí bez přímého kontaktu s uživatelem.
[17] Příkazový řádek.

```
services:
  pm2app:
    build:
      context: ..
      dockerfile: ./mi-dip-iot-reporting-docker/Dockerfile
    image: iotcp:latest
    environment:
      - MONGO_HOST=mongodb://mongo/iot-reporting-server
      - JWT_SECRET=0a6b944d-d2fb-46fc-a85e-0295c986cd9f
    links:
      - mongo
    ports:
      - "8000:8000"
  mongo:
    image: "mongo:3.4.10"
    volumes:
      - ~/mongodb:/data/db
    ports:
      - "27017:27017"
```

Zdrojový kód 5.1: Konfigurace Docker kontejnerů pomocí nástroje Docker Compose ve formátu YAML

- **Registry**. Registr Docker je online repozitář pro obrazy. Repozitáře mohou být buď veřejné a poskytované zdarma, určené především pro open-source projekty, nebo soukromé a placené nejčastěji pro firemní řešení[50].

Platforma Docker také poskytuje nástroje pro automatizaci procesů. Pro tuto diplomovou práci jsem využil nástroj Docker Compose, který pomocí konfiguračního souboru ve formátu YAML umožňuje definovat operace pro vytvoření a start procesů všech závislých kontejnerů[51].

V ukázce 5.1 je znázorněna YAML konfigurace webové aplikace, která vytváří dva kontejnery. Prvním kontejner spouští webovou aplikace na platformě PM2 a připojuje se k druhému kontejneru, na kterém je spuštěna databáze MongoDB.

Pro webovou aplikaci IOTCP jsem vytvořil dva Docker kontejnery. První kontejner je použit pro běh databáze MongoDB a je postaven na volně dostupném obrazu `mongo`. Pro umožnění přístupu k databázi jsem zadefinoval a směroval port 27017, který naslouchá databázovým požadavkům.

Druhý kontejner je aplikační a spouští klientskou a serverovou část webové aplikace, které jsou spuštěny pod nástrojem PM2, který popíši v následující kapitole. Pro umožnění přístupu k aplikaci směruji z toho kontejneru port 8000.

5.2 PM2

PM2 je nástroj pro správu Node.JS procesů spuštěných v produkčním prostředí s podporou vyvažování zátěže (load balancer). Mezi základní přednosti tohoto nástroje patří:

- jednoduchá, přesto efektivní správa prostřednictvím CLI,
- automatické restarty v případě výpadku,
- clusterování bez nutnosti zásahu do zdrojového kódu aplikace,
- hot reloading[18][52].

Proces vytvoření ekosystému v PM2 je ulehčen díky možnosti jej definovat v konfiguračním souboru. Konfiguraci, která je použitá pro webovou aplikaci IOTCP naleznete na ukázce 5.2.

Atribut *apps* je pole, ve kterém jsou jednotlivé aplikace přidávány. V našem případě se jedná o klientskou a serverovou část. V konfiguraci jednotlivých aplikací je pak definován název (*name*), spouštěcí soubor (*script*), adresář, který se považuje za výchozí pro danou aplikaci (*cwd*), počet instancí (*instances*) a konfigurace prostředí, ve kterém aplikace běží (*env* a *env_production*).

5.3 Windows Server

Jak bylo zmíněno výše, webová aplikace bude spuštěna na serveru s operačním systémem Microsoft Windows Server 2012 R2 Standard. Systém Docker podporuje pouze 2 verze operačních systémů od společnosti Microsoft:

- Windows 10,
- Windows Server 2016,

díky podpoře kontejnerizace jádra systému, kterou předešlé operační systémy od společnosti Microsoft nedisponují[53].

Z tohoto důvodu bylo nutné nejdříve vytvořit virtuální stroj pod manažerem virtuálních počítačů Microsoft Hyper-V, na kterém běží Dockerem plně podporovaný systém Linux Ubuntu.

O následné síťové přesměrování webové aplikace v rámci virtualizace se stará Microsoft Hyper-V společně s nastavením DHCP server, který zafixuje IP adresy Linux serveru, aby nedocházelo k jejich změnám.

Pro přesměrování webové aplikace do internetu používám webový systém Microsoft Internet Information Services, na kterém byl konfigurován proxy server, který aplikaci z lokální sítě přesměruje do internetu pod doménou http://iot.kalibrace.cz.

Příklad této konfigurace si můžete prohlédnout v ukázce 5.3.

[18]Metoda, při které je aplikace po změně kódu načtena se zachováním aktuálního stavu.

```
{
  "apps" : [{
    "name": "iotcp-server",
    "script": "dist/index.js",
    "cwd" : "./mi-dip-iot-reporting-server",
    "instances": "1",
    "env": {
      "NODE_ENV": "development"
    },
    "env_production" : {
      "NODE_ENV": "production"
    }
  },
  {
    "name": "iotcp-frontend",
    "script": "src/server",
    "cwd" : "./mi-dip-iot-reporting-frontend",
    "instances": "1",
    "env": {
      "NODE_ENV": "development",
    },
    "env_production" : {
      "NODE_ENV": "production",
      "APP_ENV": "production"
    }
  }]
}
```

Zdrojový kód 5.2: Konfigurace webové aplikace pro manažera Node.JS procesů PM2

5.4 Model nasazení

Webová aplikace bude spuštěna na fyzickém serveru s podporou virtualizace, který disponuje operačním systémem Windows Server 2012 R2 Standard.

Požadavkem zadavatele bylo moduly aplikace spouštět izolovaně v jednom kontejneru v rámci systému Docker. Databáze, ke které se připojuje serverová část, by měla být spuštěna v jiném Docker kontejneru.

Diagram celkového modelu nasazení se nachází na obrázku 5.2 umístěného na konci této kapitoly. V následujících kapitolách popíši technologie, které byly pro nasazení použité.

5.5 Moduly pro zařízení

Webová aplikace se neskládá pouze z částí, které jsou propojené staticky, obsahuje i dynamicky uložené moduly, které uživatel aplikace vytváří a přiřazuje k danému zařízení internetu věcí.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
 <system.webServer>
  <rewrite>
   <outboundRules>
    <rule name="ReverseProxyOutboundRule1"
        preCondition="ResponseIsHtml1"
    >
     <match filterByTags="A, Form, Img"
        pattern="^http(s)?://10.0.0.12:8000/(.*)" />
     <action type="Rewrite"
        value="http{R:1}://iot.kalibrace.cz/{R:2}" />
    </rule>
    <preConditions>
     <preCondition name="ResponseIsHtml1">
      <add input="{RESPONSE_CONTENT_TYPE}" pattern="^text/html"
          />
     </preCondition>
    </preConditions>
   </outboundRules>
   <rules>
    <rule name="ReverseProxyInboundRule1" stopProcessing="true">
     <match url="(.*)" />
     <action type="Rewrite" url="http://10.0.0.12:8000/{R:1}" />
    </rule>
   </rules>
  </rewrite>
 </system.webServer>
</configuration>
```

Zdrojový kód 5.3: Konfigurace proxy serveru v Microsoft Internet Information Services

Aby bylo možné spouštět a spravovat tyto moduly, musí být požadováno dodržení základního aplikačního rozhraní. Bylo rozhodnuto, že moduly budou muset splňovat následující požadavky:

- Podpora pouze implementace JavaScriptu, resp. Node.JS,
- projekt je inicializován pomocí NPM. Obsahuje tedy soubor `package.json`, ve kterém se nachází použité balíčky třetích stran,
- vstupní soubor je `index.js`, který obsahuje výchozí funkci,
- výchozí funkce z předešlého bodu vrací pole objektů, které dodržují strukturu definovanou uživatelem při vytvoření zařízení,
- moduly jsou komprimované v ZIP archivu.

```
1 ~/git/mi-dip-iot-reporting-server: yarn start:devel
2 ~/git/mi-dip-iot-reporting-frontend: NODE_TLS_REJECT_UNAUTHORIZED
    =0 yarn start
```

Zdrojový kód 5.4: Spuštění vývojové verze webové serveru a klientské části

Po načtení modulu na server dochází k jeho extrahování do složky a automatické kontrole. Projde-li zařízení automatickou kontrolou, dostane administrátor webu upozornění k ruční kontrole modulu.

Proběhne-li tato kontrola úspěšně, pak administrátor zařízení schválí. V rámci toho se provede instalace všech závislých balíčků daného modulu a následně proběhne *build* tohoto modulu.

5.6 Vývojové prostředí

Vývoj této webové aplikace bych rád rozdělil do několika částí, ve kterých byly použity různé nástroje.

Rozhodl jsem se používat verzovací nástroj Git pro ukládání zdrojového kódu. Jako online repozitář pro zdrojový kód jsem použil GitHub. Zde jsem vytvořil tři projekty pro *webový server*, *klientskou část aplikace* a *konfiguraci systému Docker*.

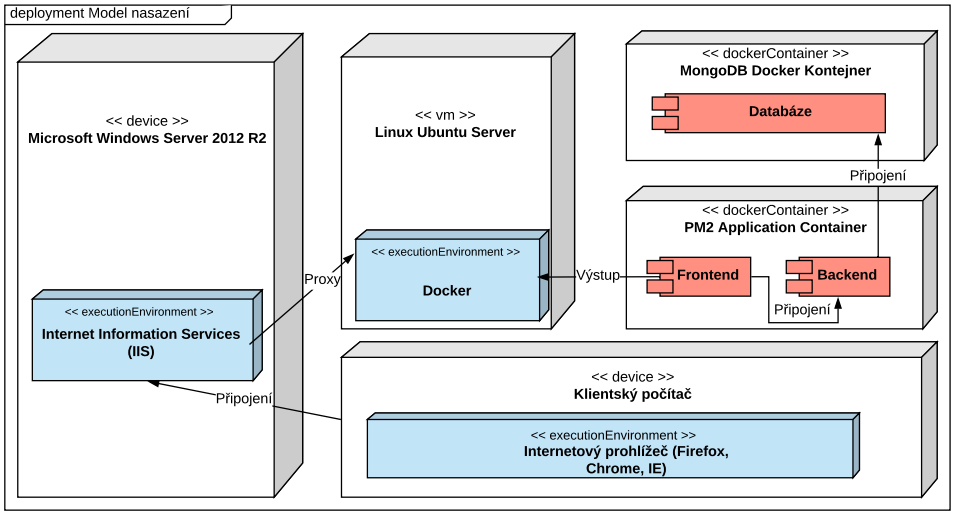
Projekt jsem implementoval v editoru zdrojového kódu Microsoft Visual Code, který byl dle ankety na portále `https://stackoverflow.com` označen jako nejpoužívanější editor zdrojového kódu[54].

Před spuštěním vývojové verze webové serveru a klientské části je nejdříve nutné spustit ze stejného systému databázi MongoDB na portu *27017*.

Na ukázce 5.4 jsou uvedeny konzolové příkazy pro spuštění webového serveru a klientské části podle adresáře, ve kterém se nachází.

V klientské části spouštěcího příkazu můžete vidět proměnnou prostředí `NODE_TLS_REJECT_UNAUTHORIZED`, která je zde kvůli nastavení proxy přístupu ze zabezpečeného serveru na nezabezpečený.

Po spuštění obou příkazů a počkání na sestavení aplikace je možné otevřít webovou stránku `https://localhost:8000`, na které je webová aplikace spuštěna.



Obrázek 5.2: Návrh modelu nasazení.

Kapitola 6

Realizace

V této kapitole popíši způsob realizace a rozdělení projektu do jednotlivých částí.

Projekt diplomové práce se skládá ze tří základních modulů:

- `mi-dip-iot-reporting-server`, ve které je implementována serverová část aplikace, která poskytuje *API* pro klientskou část,
- `mi-dip-iot-reporting-frontend`, která implementuje klientskou část,
- `mi-dip-iot-reporting-docker`, který obsahuje definice pro vytvoření Docker kontejnerů a základní konfiguraci PM2.

Další části dodávají do aplikace společnosti, které chtějí připojit své IoT zařízení, v rámci dynamických modulů, které jsou popsané v kapitole 5.5.

Všechny moduly jsou uskupeny do určité adresářové struktury, která by měla usnadnit případné změny v aplikaci. Hlavním a základním aspektem projektu založeného systémem NPM je soubor package.json, který se nachází v koření struktury.

Tento soubor, strukturovaný ve formátu JSON, obsahuje základní informace o projektu jako jsou *název projektu*, *aktuální verze*, *popisek projektu*, *autora či autory projektu*, *klíčová slova* a *licence*. Tyto atributy slouží především pro repozitáře, ve kterých jsou zdrojové kódy ukládány.

Mezi další základní atributy souboru package.json patří definice cesty k souboru, který slouží pro spuštění aplikace systémem NPM, resp. příkazem `npm start`. V package.json jsou dále definovány skripty, tedy aliasy pro provedení složených operací.

Nejpodstatnější částí je výčet závislostí, které jsou v daném projektu použity. Ty jsou dle typu použití roztříděny do čtyř základních skupin:

- **dependencies**, který obsahuje klasické závislé balíčky pro běh aplikace;

- **devDependencies**, který obsahuje závislé balíčky potřebné při vývoji aplikace, nikoliv již v produkci;
- **peerDependencies**, který obsahuje seznam požadovaných balíčků v nadřazeném projektu. Tento typ je používán v knihovnách, nikoliv samotných aplikací, které nadřazený projekt nemají;
- **optionalDependencies**, který v případě neúspěchu při instalaci závislých modulů neskončí chybou. Tento přístup je určen především pro systémové knihovny, které jsou v některých operačních systémech jejich součástí, pro jiné operační systémy je však nutná jejich lokální instalace.

6.1 Realizace serverové části

Úkolem serverové části je vytvořit webové API pro klientskou část. Z pohledu třívrstvé architektury MVC[19] se jedná o část **MC**. Pro vytvoření této části jsem použil boilerplate[20] `express-mongoose-es6-rest-api` dostupný jako open-source projekt na GitHubu[21].

Tuto základní struktura aplikace jsem použil díky podpoře technologií, které byly navrženy pro serverovou část. Patří mezi ně použití modulu Express.js pro implementaci serveru a modelování databáze v MongoDB pomocí balíčku Mongoose.

Kromě souboru `package.json`, definovaného výše, se nacházejí v kořeni projektu důležité soubory a adresáře pro spuštění aplikace:

- Soubor `gulpfile.babel.js` obsahuje definici úloh psaných pro Gulp. V případě webové aplikace IOTCP se jedná o úlohy pro kompilaci projektu, spuštění verze pro vývojáře s podporou *hot reload* a pro kopírování zkompilované aplikace do složky s produkční verzí.
- Soubor `index.js`, který je výchozím bodem pro spuštění.
- Adresář `modules`, který slouží jako adresář pro moduly k zařízením internetu věcí.

6.1.1 Konfigurační část

Definice chování aplikace a konfigurační proměnné, které se v aplikaci používají, se nachází v adresáři `config`. V následujícím seznamu vyjmenuji jednotlivé součásti tohoto adresáře a jejich zodpovědnosti:

[19] Model-View-Controller.

[20] Označení pro základní dodanou strukturu projektu, ve které není potřeba dělat jiné změny.

[21] Webová služba pro vývoj softwaru pomocí verzovacího nástroje Git.

- Adresář `validations` obsahuje soubory pro validaci struktury dat pro jednotlivé entity. V rámci dodržení struktury je název každého validačního souboru zakončen příponou `.validate.js`. Jednotlivé validační soubory jsou pak exportovány ze souboru `param-validation.js`, který se nachází v kořeni adresáře `config`, aby se nemusely importovat jednotlivě v případě potřeby souběžného použití více validačních pravidel.
- Soubory `cache.js`, `constants.js` a `config.js` obsahují konfigurace, mezi které patří nastavení kešování, přístupové údaje k databázi či systému Firebase, ale také typ prostředí, ve kterém aplikace běží a port, na kterém naslouchá.
- Nejdůležitějším souborem v tomto adresáři je `express.js`, pomocí kterého je konfigurován server Express.JS. Nejdříve jsou nastaveny middleware funkce, které se starají například o transformaci těla (viz kapitola 4.2.2.1) a cookies (viz kapitola 4.2.2.3) HTTP požadavku objektové podoby. Dále jsou zde uplatněny zabezpečovací mechanismy, například nastavení CORS[22]. Zbytek zdrojového kódu tohoto souboru se stará o přesměrování požadavku dle URL adresy a HTTP metody ke zpracování a zajištění odpovědi dle výsledku dané operace.

6.1.2 Implementační část

Samotná implementace serverové části webové aplikace probíhá v adresáři `server`. Tento adresář je dále rozdělen do podadresářů obsahující zdrojové soubory pojmenované podle významu.

6.1.2.1 Adresář Routes

V adresáři `routes` se nachází definice směrování aplikace. Názvy jednotlivých souborů značí kořenové části adres v URL adresách koncových bodů.

6.1.2.2 Adresář Controllers

V adresáři `controllers` se nachází kontrolery. Ty jsou použity ve chvíli, kdy je zavolán konkrétní přístupový API bod na základě směrování aplikace.

6.1.2.3 Adresář Models

Adresář `models` obsahuje soubory s definicemi schémat kolekcí v databázi MongoDB, které jsou vytvářeny pomocí knihovny mongoose. Každé schéma nese informace o přijímaných vlastnostech a jejich datových typech. Ke schématu je také možnost nad schématem implementovat metody a následně je volat z kontrolerů aplikace.

[22] Cross Origin Resource Sharing; mechanismus umožňující sdílení zdrojů aplikace na jiné doméně[55].

6.1.2.4 Adresář Middlewares

V adresáři `middlewares` se nachází middleware funkce, jejichž volání může být inicializováno ve směrovacích pravidlech před zavoláním kontrolerů. Aplikace obsahuje tři hlavní middleware funkce:

- **Autentizační middleware** pro kontrolu přihlášení uživatele,
- **autorizační middleware** pro kontrolu akcích umožněných danému uživateli,
- **načítací middleware** pro načtení položek z databáze podle identifikátoru, aby bylo dosaženo vyšší přehlednosti v kontrolerech.

6.1.2.5 Adresář Helpers

Adresář `helpers` obsahuje podpůrné funkce a třídy, které mohou být použity v rámci celé aplikace. Například v souboru `APIError.js` se vyskytuje třída pro formátování a výpis API chyb. V souboru `runModule.js` obsahuje logiku pro spuštění získávání dat z modulů IoT zařízení a výpočet jejich hodnot.

V ukázce zdrojového kódu 6.1 je znázorněna část pro vytvoření pracovní úlohy pro získávání dat v určitém intervalu. Na řádku 22 je vytvořena kešovaná (pro případ výpadku serveru) úloha s parametry pro interval spouštění a funkce, která je v těchto intervalech volána. Na řádku 2 se nachází implementace této funkce. Ta zavolá daný funkci v daném modulu, poté zkontroluje, zda-li vrací výsledek v poli a dále toto pole filtruje tak, aby obsahovalo validní schéma s datem od posledního uloženého výsledku měření.

6.1.2.6 Adresář Tests

Adresář `tests` obsahuje jednotkové testy, o které se stará NPM knihovna Mocha.

6.2 Realizace klientské části

Klientská část webové aplikace používá serverovou část pro získávání dat, reprezentuje tak pohledovou vrstvu třívrstvé architektury MVC.

6.2.1 Adresáře pro vývoj

Klientská část aplikace může být spouštěna v různých prostředích. V této webové aplikaci rozlišujeme především mezi dvěma prostředími: vývojovým a produkčním. Tyto prostředí se od sebe liší tím, že pro produkci chceme maximálně minimalizovat velikost sestavovaných souborů a skrýt všechny logy a chybové hlášky z konzole.

```
...
function onTick() {
  const date = new Date();
  date.setSeconds(date.getSeconds() - getSecondsInterval(repeat))
      ;
  const result = func()

  if (!Array.isArray(result)) {
    return reject('Incorrect returned format. Array needed');
  }

  valueCallback(result
    .filter(item =>
      item &&
      typeof (item.date || {}).getMonth === 'function'
      && item.date >= date
      &&  checkSchema  (item, template)
    )
  );
}
...
const key = crypto.randomBytes(30).toString('hex');
cache.set(key, new CronJob({
  cronTime:  getSecondsIntervalText(repeat),
  onTick,
}));
...
```

Zdrojový kód 6.1: Ukázka implementace automatického načítaní dat z IoT zařízení prostřednictvím dodaného modulu

V adresáři `webpack` se nachází konfigurace balíčku Webpack, který slouží pro sestavení aplikace. Adresář `env` obsahuje soubor s podporovanými systémovými proměnnými v daném prostředí.

6.2.1.1 Automatické generování API

Pro zjednodušení volání jednotlivých přístupových API bodů z aplikace jsem implementoval automatické generování *akcí* a *reducerů*, které získané odpovědi ukládají do stavu aplikace, na základě definovaného pole příslušných metod a URL adres.

Implementace generátoru je provedena ve složce `generate_api` a soubory *akce* a *reduceru* jsou generovány do podsložky `output`. Pro zahájení generování jsem poté přidal příkaz `generate:api` do konfiguračního souboru NPM `package.json`.

6.2.2 Implementační část

Samotná implementace klientské části probíhá v adresáři `src`, který obsahuje tři podadresáře:

- Adresář `browser`, ve kterém se nachází implementace Reactu pro webový prohlížeč.
- Adresář `server`, ve kterém je implementován server klientské části webové aplikace.
- Adresář `common`, ve kterém je implementována společná logika sdílená mezi složkami `browser` a `server` a logika pro architekturu Redux.

6.2.2.1 Implementační část pro vykreslení na klientské straně

V této kapitole popíši čistě klientskou část JavaScript aplikace, ze které je po kompilaci vytvořen artefakt, který se vkládá do HTML webové stránky pomocí tagu `script`. Tento přístup je také nazýván pod pojmem *client-side rendering*.

Implementační adresář `browser` obsahuje v jednotlivých podadresářích React komponenty webové aplikace, podle kterých jsou logicky pojmenovány. Obecné komponenty, které mohou být použity různými ostatními komponentami, jsou uloženy ve složce `components`.

V adresáři `browser` se dále nachází tyto podstatné soubory:

- Soubor `routesList.js`, který obsahuje seznam cest pro směrování aplikace.
- Soubor `connect.js` poskytuje *dekorátory*, které se aplikují na komponenty a předávají jim data z globálního úložiště.
- Soubor `index.js`, který slouží jako vstupní soubor.
- Soubor `main.js` volaný z `index.js`. Slouží k vytvoření kořenového aplikačního kontejneru s podporou všech technologií, které požadují obalení aplikace pro přístup a řízení, například Redux jako poskytovatel úložiště, Material-UI pro poskytování vizuálních stylů a React-Router pro směrování v aplikaci.
- Soubor `createRoutes.js`, který slouží k vytvoření hierarchie směrování v aplikaci.

6.2.2.2 Implementační část pro vykreslení na straně serveru

Pokud bychom používali pouze vykreslení na straně klienta, tedy *client-side rendering*, museli bychom na serveru vytvořit jednoduchý HTML dokument pouze se základní strukturou, ve kterém bychom aplikaci načetli.

Tento přístup však neodpovídá požadavkům moderních dynamických aplikací, a proto se, v souvislosti s použitím JavaScriptu, začala používat metoda *server-side rendering* – tedy vykreslování obsahu na straně serveru.

Snaha o použití tohoto principu vzešla především z jeho výhod: zrychlení načtení první stránky a jejího vizuálního zobrazení, ale především lepší vlastnosti pro SEO optimalizace.

Pro optimalizaci SEO v aplikacích, které využívají přístup *client-side rendering*, existovala metoda *Escaped fragment* podporovaná vyhledavačem Google, při které serverová část, po zavolání parametru `_escaped_fragment_` v adrese URL, vracela informace o dané stránce přímo v obsahu odpovědi. Dnes již tato metoda není podporována[56].

Pro implementaci klientské části aplikace IOTCP jsem také proto použil server s podporou *server-side renderingu*. Vstupním souborem při spouštění serveru je `index.js`, ve kterém se definuje podpora modulu Webpack, který při sestavení aplikace dodává názvy a typy všech jejich částí pro automatické vložení do HTML dokumentu.

V souboru `main.js` je vytvořen server Express.JS a načteny balíčky pro bezpečnost aplikace (například pro prevenci útoku *XSS*). Poté jsou inicializovány middleware funkce a samotné funkce pro vykreslení aplikace, které při zavolání dané stránky vykonají následující proces:

- inicializace stavu aplikace,
- získání konkrétní URL směrování v rámci aplikace,
- ověření, zda-li stránka nemá být přesměrována,
- provedení dotazů klientské části,
- vykreslení webové stránky do HTML,
- vrácení odpovědi uživateli.

6.2.2.3 Společná implementační část

Společná implementační část, která se nachází v adresáři `common`, je věnována především implementaci *akcí* a *reducerů* z architektury Redux.

Kromě toho se v této části nachází adresář `libs`, který částečně zastupuje roli NPM balíčků – obsahuje mnou používané pomocné implementace, které nejsou k dispozici z veřejného repozitáře. Jedná se o funkce, které mi usnadňují vytvořit a konfigurovat úložiště aplikace nebo pomáhají s HTTP požadavky.

6.3 Řízení přístupu

Ve webové aplikaci bylo třeba zajistit, aby se k různé funkcionalitě dostali pouze uživatelé s daným oprávněním. Toho jsem dosáhl pomocí implementace autentizace a autorizace.

Autentizace je činnost ověření identity daného uživatele, při které je možné použít následující metody:

- Co uživatel zná - kombinace uživatelského jména a hesla[57].
- Co uživatel má - použití technického prostředku, např. hardwarový klíč[57].
- Čím uživatel je - použití biometrických vlastností, např. otisk prstu[57].
- Co uživatel umí - odpověď na náhodně vygenerovaný kontrolní dotaz[57].

Autorizace je proces, při které se ověřuje oprávnění autentifikovaného uživatele k provádění činností na základě předem daných pravidel[58].

V případě této webové aplikace dochází ke kontrole *autentifikace* i *autorizace* jak na straně serveru, tak na klientské straně.

6.3.1 Proces ověření autentizace uživatele

Pro správu uživatelských účtů je v tomto projektu použita služba Google Firebase, která nabízí NPM knihovnu pro komunikaci s webovou aplikací.

Pokud se chce uživatel autentifikovat, přistoupí na webovou stránku s přihlašovacím formulářem a zadá přihlašovací údaje. Po potvrzení je zavolán přístupový API bod `login` pro přihlášení, v jehož těle jsou odeslány přihlašovací údaje.

Webový server tyto údaje ověří oproti systému Firebase a v případě úspěchů získá token s omezenou dobou platnosti. Tento token je uživateli v odpovědi nastaven do obsahu cookies.

Při volání přístupových API bodů s nutností ověření přístupu, je na serveru získán uložený token uživatele, který je následně ověřen na Firebase.

V případě chyby autentifikace z důvodu expirace tokenu, dochází k zaslání chybové odpovědi, ve které je odebrán token z hlavičky cookies.

Ověření autorizace uživatele probíhá na serveru pomocí použití funkcí middleware, které se nachází u jednotlivých definic směrování s požadavkem na řízení přístupu.

6.3.2 Proces ověření autorizace uživatele

I přesto, že je uživatel přihlášený, stále nemusí mít dostatečná oprávnění k volání jednotlivých cílových bodů.

K tomuto účelu slouží *autorizace*. Ve webové aplikaci jsem definoval tři základní typy uživatelů podle možnosti provádět jednotlivé úkony:

- **Administrátor**

  Administrátor je nejvyšší role webové aplikace. Má oprávnění schvalovat, odebírat a zobrazovat seznam společností. Má také na starosti schvalování modulů pro zařízení internetu věcí. Administrátor však nemůže nahlížet do interních informací jednotlivých společností a zaměstnanců, např. tedy výsledků měření jednotlivých zařízení a notifikací.

- **Společnost**

  Společnost je druhá nejvyšší role, která má možnost přidávat nová zařízení a schvalovat zaměstnance. Nemůže však zaměstnancům měnit či odebírat jejich vlastní nastavení notifikací.

- **Zaměstnanec společnosti**

  Zaměstnanec společnosti má nejnižší možnost konfigurace. Může pouze vytvářet a měnit notifikace.

Každý uživatel má v databázi uvedenou svoji roli a podle toho je možné daného uživatele autorizovat. Dalším typem je autorizace podle uživatele, který záznam vytvořil nebo je mu povoleno záznam modifikovat.

Pro autorizace se stejně jako při autentizaci používají middleware funkce, které porovnávají daný záznam s aktuálně přihlášeným uživatelem.

Kapitola 7

Testování

Poté, co implementační část byla z větší části hotova, jsem se rozhodl vytvořit několik ukázkových testů různých typů, které se zde pokusím především vysvětlit, částečně implementovat a později dokončit tak, aby byla aplikace co nejvíce pokryta různými testovacími scénáři.

7.1 Testování serverové části

Na straně serveru jsem se rozhodl testovat jednotlivé přístupové body REST API pomocí jednotkového testování (tzv. unit testing). Cílem jednotkového testování je ověřit vytvořením testovacích scénářů, zda-li je kód v pořádku implementován a aplikace vrací správné výsledky.

Jednotkový test by měl obsahovat testy pouze pro danou funkční část aplikace s pokud možno co největší izolací od ostatních funkčních částí. Z tohoto důvodu nahrazují skutečné objekty uměle vytvořené zástupné objekty, označované jako *test doubles*. Existují čtyři typy zástupných objektů sepsaných od nejjednodušších až po nejsložitější:

- **Dummy** slouží pouze pro splnění kontraktu testované třídy, nemusí nic obsahovat ani implementovat[59].
- **Fake** na rozdíl od *dummy* obsahuje velice prostou implementaci požadovaných rozhraní[59].
- **Stub** je poměrně zdařilý dvojník, který by měl být automaticky generovaný a vrací již věrně napodobené předpřipravené výsledky[59].
- **Spies** jsou *stuby*, které dokáží zaznamenat informace o svých voláních[59].
- **Mock** jsou naprogramované objekty s očekávanými (expectations) výsledky podle specifikace volání, které očekávají[59].

```
1 chai.should();
2 foo.should.be.a('string');
3 const expect = chai.expect;
4 expect(foo).to.have.lengthOf(3);
5 const assert = chai.assert;
6 assert.property(tea, 'flavors');
```

Zdrojový kód 7.1: Porovnání očekávaných hodnot v knihovně Chai

Pro testování jsem se rozhodl použít testovací framework **Mocha**, který slouží pro definice jednotlivých testovacích pravidel. Obsahuje dvě základní funkce:

- `describe`, která třídí testy dle dané funkční části,
- `it`, která slouží pro jednotlivé testovací případy.

Společně s testovacím frameworkem Mocha jsem použil *assertovací* knihovnu[23] **Chai**. Ve zdrojovém kódu 7.1 pak můžete vidět užití této knihovny.

7.2 Testování klientské části

Pro testování na klientské části webové aplikace se kombinují nejčastěji tři hlavní techniky:

- **Jednotkové testování**, které jsem již popsal výše, a které v případě tohoto projektu slouží pro testování *akcí* a *reducerů* z Reduxu.
- **End-To-End testování**, které testuje, zda-li je možné provést průchod aplikací od počátku testovaného procesu až do jeho konce.
- **Uživatelské testování**, které otestuje budoucí uživatele portálu podle způsobu užití.

7.2.1 End-To-End testování

End-to-End, také možno E2E, testování ve webové aplikaci bývá nejčastěji provedeno simulací webového prohlížeče, ve kterém je automatizován průchod podle daného testovacího případu.

Pro JavaScript existuje velké množství End-to-End frameworků. Mezi nejznámější a nejpoužívanější patří: **Nightwatch.js** a **Cypress**.

Tyto frameworky nejčastěji pracují se *stub* objekty, které reprezentují odpovědi API serveru a to z důvodu řízení jednotlivých odpovědí.

[23] Knihovna, která porovnává výsledek volání s předem zadanou porovnávací hodnotou.

7.2.2 Uživatelské testování použitelnosti

Metody uživatelského testování jsem se rozhodl provést ve fázi, kdy na aplikaci budou probíhat další úpravy, abych zjistil, jakým směrem by se návrh webové aplikace měl ubírat.

Pro testování jsem vybral tři uživatele, kteří s vysokou pravděpodobností přijdou s webovou aplikací do kontaktu a budou reprezentovat odlišné typy uživatele.

7.2.2.1 Scénáře

Pro každý typ uživatele jsem poté vytvořil testovací scénáře:

Administrátor

- **Scénář č. 1 - Registrace administrátora na portál**. Prvním úkolem bylo provést registraci na portál, následně se odhlásit a znovu přihlásit. Uživateli byl sdělen registrační klíč a adresa pro registraci administrátora, která v tomto scénáři sloužila jako výchozí bod.
- **Scénář č. 2 - Schválení společnosti**. Druhým úkolem bylo schválit registrovanou, ale neschválenou společnost.

Správce společnosti

- **Scénář č. 3 - Registrace společnosti na portál**. Prvním úkolem pro zástupce společnosti bylo provést registraci na portál.
- **Scénář č. 4 - Přidání nového zařízení**. Uživatel obdržel soubor s modulem v archivu ZIP a dostal za úkol vytvořit nové zařízení na portálu. V přiloženém postupu byly definovány následující hodnoty:
  - **Název**. Teploměr t.1.
  - **Soubor**. Archiv `teplomer.zip`, který se nachází na ploše testovacího PC.
  - **Frekvence spouštění**. Jednou za minutu.
  - **JSON šablona vrácených dat**. Obsahuje číselnou hodnotu `value`.

Zaměstnanec

- **Scénář č. 5**. Prvním úkolem zaměstnance bylo registrovat se do aplikace pomocí registračního kódu společnosti, který byl zaměstnanci přidělen v pomocném dokumentu.

– **Scénář č. 6**. Druhým úkolem zaměstnance bylo vytvořit notifikační pravidlo pro zařízení *Teploměr t.1*. Dle přiloženého souboru mělo pravidlo obsahovat průměr posledních 20 měření složky `value` a notifikace měla být vyvolána, pokud tato hodnota přesáhla číselnou hodnotou 20.

7.2.2.2 Průběh uživatelského testování

Uživatelské testování bylo provedeno v kanceláři zadavatele na počítači s operačním systémem Windows 10 v prohlížeči Google Chrome.

Před samotným testováním dostali jednotliví uživatelé anonymní vstupní dotazník, který vyplnili. Ten obsahoval otázky ohledně věku, pohlaví a vzdělání uživatele a dále otázky týkající se schopnosti práce s počítačem.

Samotné testování probíhalo za mé přítomnosti, kdy jsem seděl vedle uživatele, sledoval pracovní postup a v případě nejasností se snažil dávat nápovědy, nikoliv postup za uživatele vyplnit. Ke každému testovacímu scénáři jsem se snažil dělat poznámky, abych mohl následně analyzovat problémy.

Po skončení samotného testování jsem jednotlivým uživatelům rozdal výstupní dotazníky týkající se práce s webovou aplikací.

7.2.2.3 Výsledky uživatelského testování

V dotazníku uvedli téměř všichni uživatelé pokročilou znalost PC, o čemž jsem se přesvědčil z běžně prováděných scénářů, mezi které patřila registrace všech typů uživatelů.

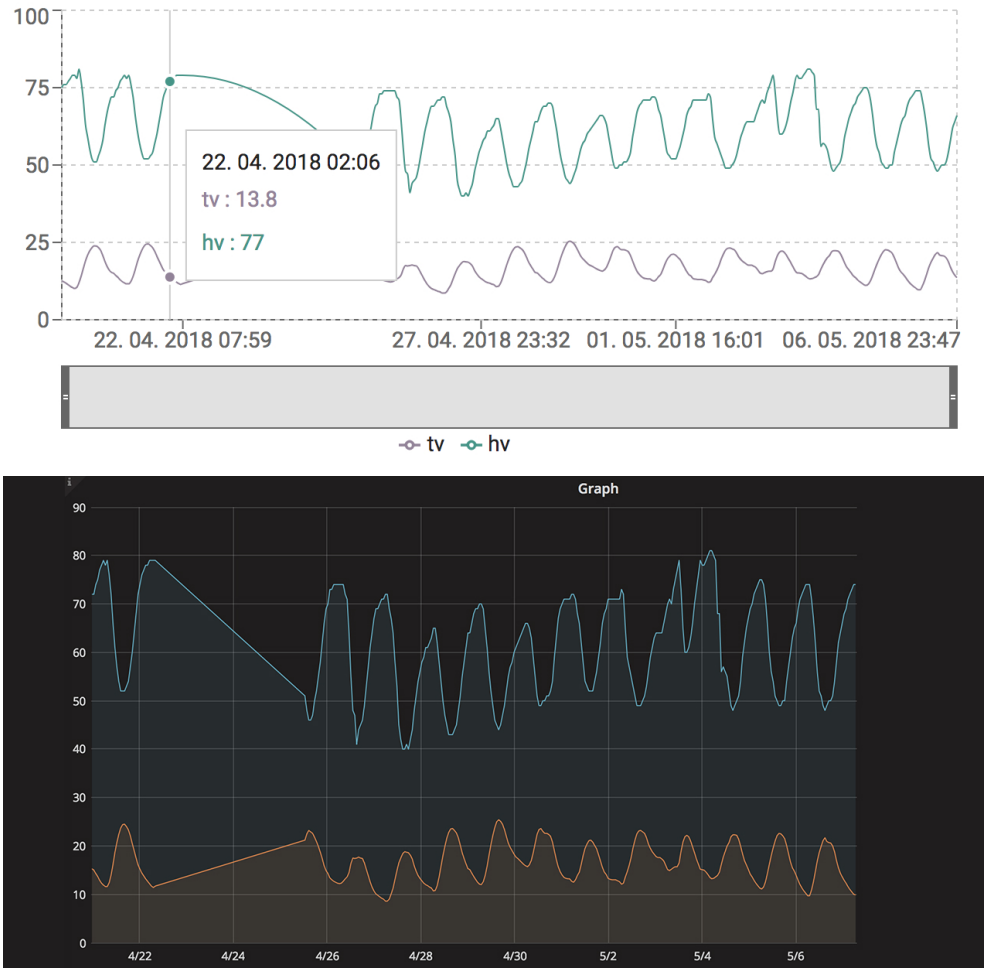
Scénář č. 2 byl administrátorem také dobře splněn bez jakýchkoliv dotazů. U scénáře č. 4 došlo k mírnému zaváhání při vytváření nového zařízení. Největší problém způsobila definice šablony hodnot vrácených zařízením, která má být uvedena ve formátu JSON. Pro nápravu tohoto problému bude později aplikace obsahovat lepší editor pro úpravu těchto vstupů.

Scénář č. 6 byl ještě těžším problémem kvůli definici tzv. porovnávačů. Uživatel nevěděl, jakým způsobem má danou hodnotu zapsat do políčka určeného pro daný porovnávač. Z tohoto důvodu budeme společně se zadavatelem přemýšlet, jakým způsobem by měla být tato část aplikace pozměněna tak, aby bylo snadnější vytvořit nové notifikační pravidlo.

Při testování jsem dále postřehl, že by aplikace potřebovala u jednotlivých formulářových vstupů nápovědy. Daleko vhodnějším řešením by pak bylo provedení školení tét webové aplikace.

7.3 Testování s nástrojem Grafana

Posledním krokem v rámci testování aplikace bylo použít stejné datové zdroje a zkontrolovat, zda-li jsou výsledky vycházející z nástroje Grafana stejné jako výsledky v nové webové aplikaci.



Obrázek 7.1: Porovnání grafu hodnot se současným řešením, nástrojem Grafana.

Pro tento test jsem použil zařízení pro měření teploty a vlhkosti zobrazené ve společném grafu. Na obrázku 7.1 si můžete prohlédnout grafy těchto hodnot. V horní části obrázku je graf z nové webové aplikace, ve spodní části se nachází graf z nástroje Grafana. Můžeme tak potvrdit, že v porovnání s tímto nástrojem, ukládá a zobrazuje naše nová aplikace data v pořádku.

Další částí ověření funkčnosti aplikace bylo vytvoření notifikačního pravidla, u kterého jsem věděl, že kontrolou neprojde a provede notifikaci uživatele (obrázek C.1). Jednalo se o teplotu měřenou teploměrem, kde se průměrné hodnoty pohybují v jarních měsících okolo 13 až 15 stupni Celsia. Na obrázku C.2 pak můžete vidět, že notifikace uživatele skutečně proběhla.

Závěr

Hlavním cílem diplomové práce bylo vytvoření webové aplikace, která splňuje požadavky zadavatele a bude nabízena jak stávajícím, tak i novým klientům zadavatele.

Aplikace IOTCP ještě není úplně dokončená, ani odladěná. K tomu by mělo dojít během léta až podzimu roku 2018. K tomu se váže několik kroků, které má zadavatel v plánu: pořízení a přechod na servery s operačním systémem Linux nebo použití služeb v cloudu, vytvoření předem definovaných modulů pro častá řešení (například připojení k databázím) a propojení aplikace s aktuálními systémy zadavatele.

Především to bylo důvodem k vytvoření této aplikace. Společnost využívá k uchování dat zákazníků systém Microsoft SharePoint Server, v tomto systému se ukládají informace o firemních zařízeních a úkonech, které nad nimi probíhají. Díky propojení tak bude možné sledovat závislosti mezi těmito zařízeními a IoT přístroji.

Především díky návrhu dobré architektury si myslím, že aplikace bude snadno rozšiřitelná o další funkcionalitu. Serverová část je snadno rozšiřitelná díky efektivnímu vytváření nových směrovacích pravidel a oddělení kontrolerů, validačních podmínek a middleware funkcí. Frontend je taktéž jednoduše rozšiřitelný díky základnímu rozdělení směrovacích pravidel a použití architektury Redux.

Doufám, že zadavatel diplomové práce bude s fungováním IOTCP spokojen a během další spolupráce dokončíme základní funkcionalitu této webové aplikace a rozšíříme ji o další možnosti.

Literatura

[1] Y.2060 : Overview of the Internet of things. 2012. Dostupné z: https://www.itu.int/rec/T-REC-Y.2060-201206-I

[2] Grafana Features. Dostupné z: https://grafana.com/grafana

[3] *ECMAScript: A general purpose, cross-platform programming language.* Geneva, 16 vydání, 1997.

[4] JavaScript Versions. Dostupné z: https://www.w3schools.com/js/js_versions.asp

[5] *Learning JavaScript.* O'Reilly Media, Inc., 2016, ISBN 9781491914939, str. 25.

[6] Tilkov, S.; Vinoski, S.: Node.js. *IEEE Internet Computing*, ročník 14, č. 6, 2010: s. 80–83, ISSN 1089-7801, doi:10.1109/MIC.2010.145. Dostupné z: http://ieeexplore.ieee.org/document/5617064/

[7] *Building APIs with Node.JS.* New York, NY: Springer Science Business Media, 2016, ISBN 978-1484224410, str. 9.

[8] Pellegrom, G.: Delicious Brains. 2018. Dostupné z: https://deliciousbrains.com/grunt-vs-gulp-battle-build-tools/

[9] *Web development with Node and Express.* Sebastopol, CA: O'Reilly, 2014, ISBN 1491949309, str. 4.

[10] Body-parser. Dostupné z: https://www.npmjs.com/package/body-parser

[11] Cookie-dough. Dostupné z: https://www.npmjs.com/package/cookie-dough

[12] Cookie-parser. Dostupné z: https://www.npmjs.com/package/cookie-parser

[13] Express-fileupload. Dostupné z: https://www.npmjs.com/package/express-fileupload

[14] Add the Firebase Admin SDK to Your Server. Dostupné z: https://firebase.google.com/docs/admin/setup

[15] Joi. Dostupné z: https://www.npmjs.com/package/joi

[16] Lru-cache. Dostupné z: https://www.npmjs.com/package/lru-cache

[17] *Proceeding*. New York: Association for Computing Machinery, 1999, ISBN 158113083X, s. 134–143.

[18] Math.js. c2013-2017. Dostupné z: http://mathjs.org/

[19] Mongoose. c2011. Dostupné z: http://mongoosejs.com/

[20] *JavaScript the definitive guide*. Sebastopol, Calif: O'Reilly, páté vydání, 2006, ISBN 9780596554477, str. 497.

[21] Papp, A.: The History of React.js on a Timeline. c2017. Dostupné z: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

[22] Abramov, D.: Presentational and Container Components. Dostupné z: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

[23] JSX In Depth. c2018. Dostupné z: https://reactjs.org/docs/jsx-in-depth.html

[24] Rendering Elements. c2018. Dostupné z: https://reactjs.org/docs/rendering-elements.html

[25] Sing, L.: React: Create maintainable, high-performance UI components. Dostupné z: https://www.ibm.com/developerworks/library/wa-react-intro/index.html

[26] React.Component. c2018. Dostupné z: https://reactjs.org/docs/react-component.html

[27] Motivation. Dostupné z: https://redux.js.org/introduction/motivation

[28] Weck, S.: Developing modern offline apps with ReactJS, Redux and Electron – Part 3 – ReactJS + Redux. Dostupné z: https://blog.codecentric.de/en/2017/12/developing-modern-offline-apps-reactjs-redux-electron-part-3-reactjs-redux-basics/

[29] Three principles. Dostupné z: https://redux.js.org/introduction/three-principles

[30] Gupta, A.: Introduction To Redux(using React Native). Dostupné z: https://medium.com/@abhayg772/introduction-to-redux-using-react-native-a8f1e8778333

[31] Redux. Dostupné z: https://redux.js.org/

[32] *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, Inc., 2017, ISBN 9781491954577, s. 93–95.

[33] CodeMirror. Dostupné z: https://codemirror.net/

[34] Immutable.js. Dostupné z: https://facebook.github.io/immutable-js/

[35] Isomorphic-fetch. Dostupné z: https://www.npmjs.com/package/isomorphic-fetch

[36] Lodash. Dostupné z: https://lodash.com/

[37] Material-UI. Dostupné z: http://www.material-ui.com/

[38] Typechecking With PropTypes. Dostupné z: https://reactjs.org/docs/typechecking-with-proptypes.html

[39] Grolinger, K.; Higashino, W. A.; Tiwari, A.; aj.: Data management in cloud environments. *Journal of Cloud Computing: Advances, Systems and Applications*, ročník 2, č. 1, 2013: s. 22–, ISSN 2192-113X, doi:10.1186/2192-113X-2-22. Dostupné z: http://journalofcloudcomputing.springeropen.com/articles/10.1186/2192-113X-2-22

[40] McCreary, D.; Kelly, A.: *Making sense of NoSQL*. Shelter Island: Manning, [2014], ISBN 978-161-7291-074.

[41] *Big Data a NoSQL databáze*. Grada Publishing a.s., 2015, ISBN 9788024759388, str. 110.

[42] DB-Engines Ranking. c2018. Dostupné z: https://db-engines.com/en/ranking

[43] Arora, M.: WHAT ARE THE KEY FEATURES OF MONGODB? c2017. Dostupné z: https://www.tutorialsjar.com/key-features-of-mongodb/

[44] Sharding. c2018. Dostupné z: https://docs.mongodb.com/manual/sharding/

[45] Sharding in MongoDB. Dostupné z: http://blog.optimal.io/sharding-in-mongodb/

[46] Summers, N.: 9 principles Google created for its colorful 'Material Design' UI refresh. c2006-2018. Dostupné z: https://thenextweb.com/google/2014/06/26/google-explains-principles-material-design-language-android-chrome-web/

[47] What is Docker? c2018. Dostupné z: https://www.docker.com/what-docker

[48] Kernel Requirements. 2014. Dostupné z: https://web.archive.org/web/20140821065734/http://docker.readthedocs.org/en/v0.7.3/installation/kernel/

[49] Docker vs VMs. 2016. Dostupné z: https://hub.docker.com/r/tplcom/docker-java/

[50] Docker overview. c2018. Dostupné z: https://docs.docker.com/engine/docker-overview/

[51] Overview of Docker Compose. c2018. Dostupné z: https://docs.docker.com/compose/overview/

[52] PM2. c2018. Dostupné z: https://github.com/Unitech/pm2

[53] Friis, M.: BUILD AND RUN YOUR FIRST DOCKER WINDOWS SERVER CONTAINER. c2017. Dostupné z: https://blog.docker.com/2016/09/build-your-first-docker-windows-server-container/

[54] Stack Overflow. 2018. Dostupné z: https://insights.stackoverflow.com/survey/2018/

[55] MDN Web Docs. 2018. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[56] Deprecating our AJAX crawling scheme. Dostupné z: https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html

[57] Gibson, D.: Understanding the Three Factors of Authentication. 2018. Dostupné z: http://www.pearsonitcertification.com/articles/article.aspx?p=1718488

[58] The Economic Times. 2018. Dostupné z: https://economictimes.indiatimes.com/definition/authorization

[59] Fowler, M.: martinfowler.com. 2018. Dostupné z: https://martinfowler.com/articles/mocksArentStubs.html

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

BSON Binary JSON

CLI Command Line Interface

CORS Cross-origin resource sharing

CSS Cascading Style Sheets

DOM Document Object Model

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IoT Internet of Things

IOTCP Webová aplikace IOTCP

JSON JavaScript Object Notation

MVC Model-view-controller

NPM Node Package Manager

ODM Object-Document Mapper

PDF Portable Document Format

REST Representational State Transfer

SEO Search Engine Optimization

SPA Single-page application

SQL Structured Query Language

URL Uniform Resource Locator

XSS Cross-site scripting

Příloha B

Instalace webového serveru

B.1 Instalační požadavky

Pro spuštění webové aplikace IOTCP je potřebné použít jeden z následujících operačních systémů, které jsou podporovány systémem Docker:

- Linux,
- MacOS,
- Windows 10,
- Windows Server 2016,

nebo použít IaaS platformu AWS nebo Microsoft Azure. Na použitém systému pak potřeba Docker nainstalovat nejlépe z oficiálního zdroje `https://www.docker.com/community-edition`.

Dále je potřeba, aby systém obsahoval nástroj Docker-compose. Pro jeho instalaci postupujte podle návodu na oficiálních stránkách Dockeru: `https://docs.docker.com/compose/install/`.

B.2 Spuštění webového serveru

Po provedení instalace všech potřebných nástrojů zkopírujte projekt z přiloženého CD na lokální disk a přesuňte se do složky `impl/mi-dip-iot-reporting-docker`.

Ujistěte se, že systém nepoužívá port `8000`.

Instalaci a spuštění webového serveru proveďte z příkazového řádku.

Používáte-li systém MacOS nebo Linux, ujistěte, že soubor `run.sh` má právo spouštění a zadejte příkaz `./run.sh`.

Používáte-li systém Windows, zadejte následující příkaz: `docker-compose up --build --abort-on-container-exit`.

Instalace všech závislých modulů Dockeru a následná instalace balíčků v rámci webové aplikace trvá zhruba 5 - 10 minut.

Po úspěšném spuštění uvidíte výpisy webového serveru v konzoli, ze které byla aplikace spuštěna.

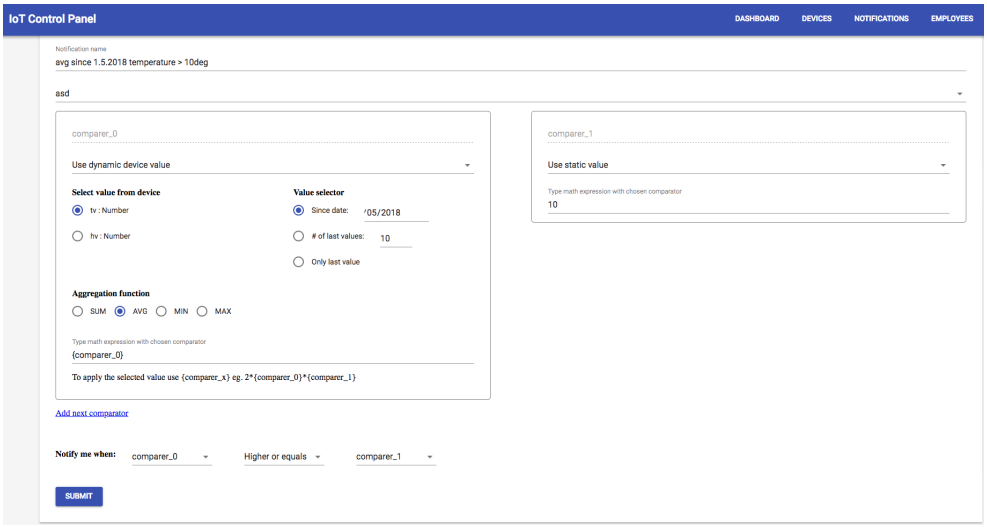
B.3 Spuštění webové aplikace v prohlížeči

Pro otevření aplikace použijte prohlížeč dle zadaných požadavků a ujistěte se, že umožňuje spuštění JavaScriptu.

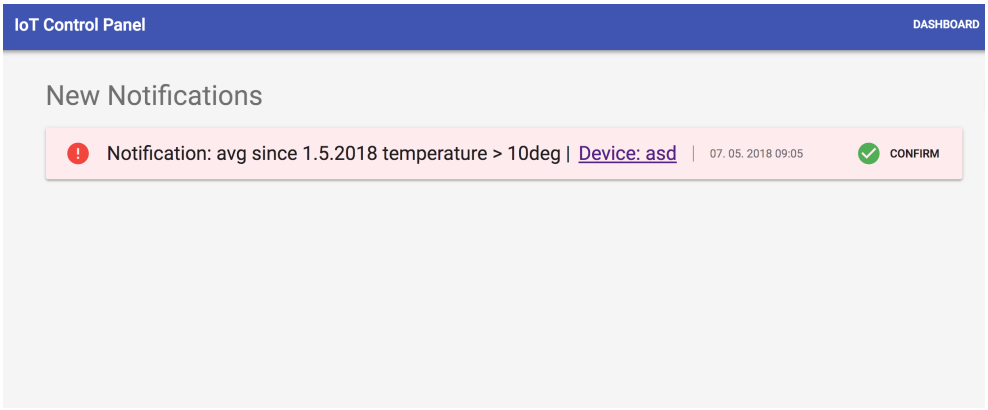
Následně zadejte webovou adresu `http://localhost:8000` a webová aplikace by se měla načíst.

Příloha C

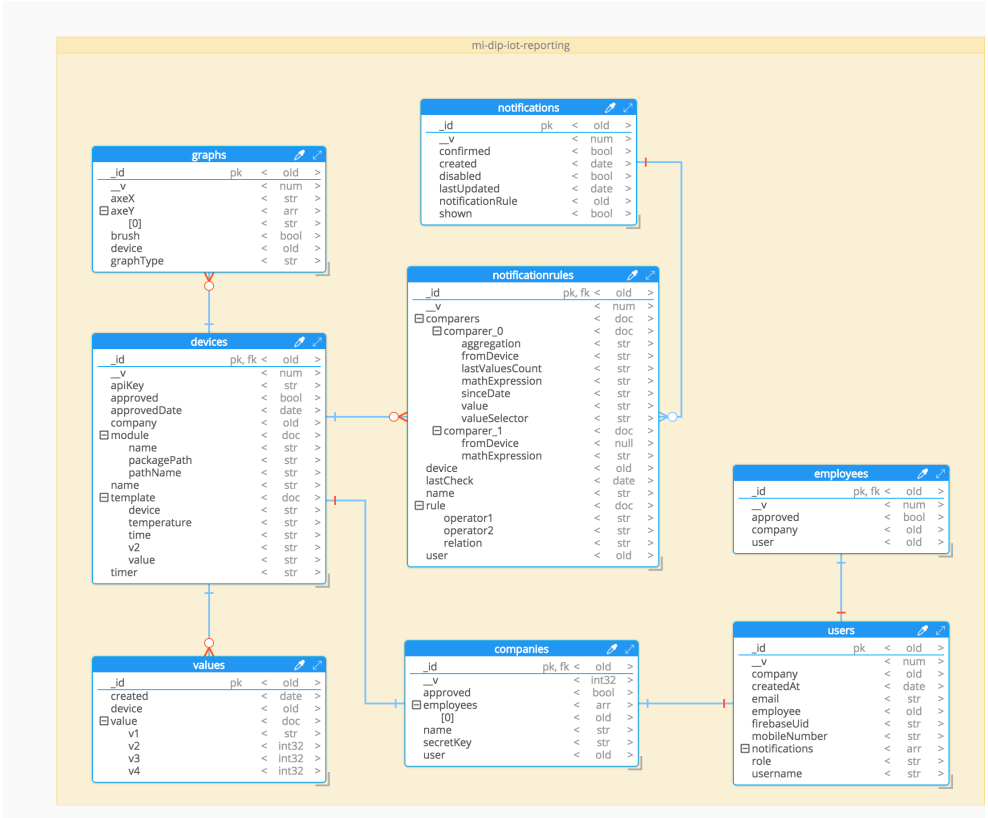
Přílohy



Obrázek C.1: Obrazovka přidání nového notifikačního pravidla.



Obrázek C.2: Obrazovka hlavní uživatelské stránky zobrazující aktuální notifikaci.



Obrázek C.3: Databázový diagram IOTCP. Pro lepší čitelnost doporučuji verzi na CD.

Příloha D

Obsah přiloženého CD

Níže je popsána adresářová struktura na přiloženém CD.

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `mi-dip-iot-reporting-server` ..... Implementace serverové části
    - `mi-dip-iot-reporting-frontend` ... Implementace klientské části
    - `mi-dip-iot-reporting-docker` ..... Sestava pro Docker
    - `modules` ..... Ukázkové moduly pro aplikaci
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `src` ..... zdrojové obrázky a diagramy použité v bakalářské práci
  - `other` ..... ostatní přidané dokumenty
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF