TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zpracování a vizualizace PQ a PFC záznamů na tenkých klientech

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: B2646 – Informační technologie

Autor práce: **Sanzhar Sangispayev.**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zpracování a vizualizace PQ a PFC záznamů na tenkých klientech

Bakalářská práce

Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie

Autor práce: **Sanzhar Sangispayev**
Vedoucí práce: Ing. Jan Kraus, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Zpracování a vizualizace PQ a PFC záznamů na tenkých klientech

Jméno a příjmení: **Sanzhar Sangispayev**
Osobní číslo: M15000051
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Seznamte se s veličinami, uloženými v souborech obvyklého archivu monitoru kvality anebo regulátoru jalového výkonu, a se základními způsoby jejich vyhodnocování a prezentace výsledků uživatelům.
2. S využitím archivů nebo dat načtených z online služby navrhněte aplikaci pro prohlížeč resp. mobilní telefon, která záznamy z jednoho či více odběrných míst vhodným způsobem agreguje a zobrazí.
3. Implementujte funkcionalitu pro přihlašování uživatelů platformy, mechanismy sdílení dat a výsledků, funkce pro uživatelskou zpětnou vazbu a vhodné způsoby definice alarmů.
4. V závěru shrňte dosažené výsledky a diskutujte další možnosti rozvoje tématu.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] KURTZ, Jamie, 2013. ASP.NET MVC 4 and the Web API: building a REST service from start to finish. Berkeley, CA: Apress. Expert's voice in ASP.NET.

[2] KRAUS, Jan a Martin BLÍŽKOVSKÝ. Uživatelská příručka aplikace ENVIS v. 1.2 [online]. 2015. [cit. 2015-1- 08]. Dostupné z: http://www.kmb.cz/

[3] BOJINOV, Valentin. RESTful Web API Design with Node. js. Packt Publishing Ltd, 2016.

[4] CASCIARO, Mario; MAMMINO, Luciano. Node. js Design Patterns. Packt Publishing Ltd, 2016.

Vedoucí práce:	Ing. Jan Kraus, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	10. října 2019
Předpokládaný termín odevzdání:	18. května 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2019

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

3. září 2020 Sanzhar Sangispayev

Zpracování a vizualizace PQ a PFC záznamů na tenkých klientech

Abstrakt

Tato bakalářská práce se zabývá návrhem klientské mobilní aplikaci v oboru hospodaření s energií. Navržená aplikace umožňuje sledování archivních dat z monitorů kvality s cílem pomoci uživatelům optimalizovat spotřebu elektronické energie. Rešeršní část této práce se skládá z literární a technologické částí. Literární rešerše seznamuje čtenáře s problematikou hospodaření s energií, kvalitativními charakteristikami elektronické energie a současnými existujícími řešeními. Ve druhé rešeršní části práce jsou popsáné mobilní platformy a porovnávané přístupy k vývoji mobilních aplikací. Výsledkem praktické části této práce je víceplatformní mobilní aplikace, která je schopna načítat a zobrazovat vybraná historická data o spotřebě a kvalitě elektronické energie. Po přihlášení emailem a heslem, uživateli bude nabídnuta odlišná funkcionalita v závislosti na jeho role. V základní verzi jsou tři uživatelské role inspirované normou ISO 50001. Při vývoji byl použit framework Flutter, umožňující rychlejší vývoj.

Klíčová slova: energetický management, PQ monitor, mobilní aplikace, Dart, Flutter.

Abstract

This bachelor thesis aims the design of a client mobile application in the field of energy management. The proposed application allows monitoring of archived data from quality monitors to help users optimize power consumption. The research part of this work consists of literary and technological parts. The literature research acquaints the reader with the issues of energy management, electric power quality, and currently existing software solutions in the field. The second research part of the thesis describes mobile platforms and compares approaches to the development of mobile applications. The result of the practical part of this work is a cross-platform mobile application that is able to load and display selected historical data on the consumption and quality of electric energy. After logging in with email and password, the user will be offered different functionality depending on his role. In the basic version, there are three user roles inspired by the ISO 50001 standard. The Flutter framework was used in the development, making development faster.

Keywords: energy management, PQ monitor, mobile application, Dart, Flutter.

Poděkování

Rad bych poděkoval vedoucímu této bakalářské práce panu Ing. Janu Krausovi, Ph.D. za trpělivost a rady při psaní této práce.

Obsah

Seznam zkratek

EnMS	Systém managementu hospodaření s energií
BCPM	Branch Circuit Power Meter
kvar	Voltampér reaktanční. Jednotka jalového elektrického výkonu
kVA	Kilovoltampér. Jednotka zdánlivého elektrického výkonu
SQL	Standardizovaný strukturovaný dotazovací jazyk
PQ	Kvalita energie
API	Rozhraní pro programování aplikací
SDK	Soubor nástrojů pro vývoj software
VM	Virtuální stroj
UI	Uživatelské rozhraní
REST	Architektura rozhraní, navržená pro distribuované prostředí
JSON	JavaScript Object Notation je způsob zápisu dat, který není závislý na počítačové platformě
HTTP	Hypertext Transfer Protocol
ORM	Objektově relační mapování. Programovací technika, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem.
DBS	Databázový systém
APNs	Apple Push Notification service
FCM	Firebase Cloud Messaging
MQ	Fronta zpráv
AMQP	Advanced Message Queuing Protocol
Stomp	Streaming Text Oriented Messaging Protocol
MQTT	MQ Telemetry Transport
TCP	Protokol transportní vrstvy
FIFO	Zkratka anglického sousloví First In, First Out, což se do češtiny zpravidla překládá jako první dovnitř, první ven.
CRUD	Vytvářet (create), číst (read), upravovat (update) a mazat (delete)

1 Úvod

S rozvojem technologie a růstem ekonomiky stoupá i spotřeba energie. Jedním z nejzávažnějších problémů dnešního světa je nutnost snižovat spotřebu energie. To má jak hospodářské, tak i environmentální důvody. Zvýšení spotřeby energie vede k rychlejšímu vyčerpání nejen neobnovitelných, ale také obnovitelných přírodních zdrojů, což má a bude mít negativní vliv na životní prostředí. Nízká kvalita energie může vést k oteplování motorů, transformátorů a vedení, což má negativní vliv na životnost těchto zařízení, náhodným výpadkům počítačových systémů, poškozování elektroniky a blikání osvětlení [29]. Z těchto důvodů je energetický management aktivně realizovaným trendem. Úroveň rozvoje cloudových, mobilních a vestavěných technologií, a rostoucí poptávka nám umožňuje vytvořit technologické řešení, které bude schopné pomoci uživatelům držet se moderních standardů hospodaření s energií.

Hospodaření s energii je aplikovatelné jak pro větší podniky tak i pro malé. Každý podnik muže mít vlastní cíle a důvody k zavedení systému hospodaření s energii. Proto toto řešení musí být snadno přizpůsobitelné požadavkům širokého spektra zákazníků. Také vývoj vlastního softwarového řešení může být finančně nezvladatelnou úlohou pro menší organizaci [17]. S každým rokem vznikají technologie a programovací nástroje za použití kterých lze dosáhnout zrychlení vývoje. Z těchto důvodu, důraz při zpracování této práce bude kladen na hledání moderních technologií, které přispějí k nejrychlejšímu a snadnějšímu vývoji.

Cílem této bakalářské práce je vývoj mobilní aplikace, která zobrazuje historická data z měřících přístrojů uživatelsky přívětivým způsobem. Výsledná aplikace musí nabízet funkcionalitu pro přihlašování, správu uživatelů, správu měřicích zařízení a definici kritických hodnot uživatelem.

První část této bakalářské práce se zabývá problematikou energetického managementu a seznámení se s měřícími zařízeními pro sběr dat. Dále se zabývá prozkoumáním dat, která jsou tato zařízení schopna sbírat. S cílem stanovení požadavku na funkčnost aplikace se pojí činnost energetických manažerů, kteří jsou koncovými uživateli navrhované aplikace. Na konci bude uveden krátký průzkum existujících softwarových řešení pro nalezení uživatelsky přívětivého způsobu prezentace výstupů z měřících zařízení. Teoretická část této práce končí definicí požadavků, které navrhovaná aplikace musí splňovat.

Druhá část této práce se bude věnovat problematice mobilních serverových aplikací. V této části bude popsána krátká charakteristika nejpoužívanějších operačních systémů běžících na mobilních zařízeních. Dále se popíšou moder-

ní technologie používaná při realizaci mobilních aplikaci, a serverových služeb. Cílem této části je volba technologií pro realizaci tohoto projektu na základně současných přístupů.

Následuje návrh a implementace serverové části aplikace. Uvádí se mechanismy přihlašování uživatelů, mechanismus práv a roli a zabezpečení aplikace. V kapitole o implementaci klientské části je popsána funkčnost mobilní aplikace s realizačními detaily včetně architektury, způsobu komunikaci se serverovou části. V této části práce bude uveden způsob přizpůsobování vizualizačních schopností aplikace, včetně definice uživatelem kritických hodnot, upravování grafů a spravování měřicích zařízení. Na konci se ukazuje výsledný způsob prezentace archivovaných dat uživatelům. V závěrečné části práce, kromě shrnutí dosažených výsledků, jsou uvedené navrhované možnosti rozvoje tohoto tématu.

2 Literární rešerše

2.1 Hospodaření s energií

Hospodaření s energií je proaktivní, organizované a systematické řízení zadávání, přeměny, distribuce a využívání energie s cílem splnit požadavky. To vše s ohledem na environmentální a ekonomické cíle. Hlavními cíli je zachování zdrojů, ochrana klimatu a úspora nákladů, zatímco uživatelé mají trvalý přístup k potřebným zdrojům energie.

Požadavky na EnMS (Systém hospodaření s energií) pro organizace stanovuje norma ČSN EN ISO 50001. Cílem této normy je umožnit organizacím vytvářet systémy potřebné pro neustálé zlepšování energetické hospodárnosti, včetně zlepšování energetické účinnosti, užití a spotřeby energie. Tento standard může být realizován jakoukoliv organizací bez ohledu na její velikost, geografickou polohu, zdroje energie, aktivitu, nebo její sociální či kulturní podmínky [10]. Obecné požadavky normy jsou: zavedení, dokumentace, implementace, údržba a neustálé zlepšování EnMS, definice rozsahu a hranic EnMS, Určit, jakým způsobem bude organizace tuto normu splňovat. Zavedení systému zahrnuje: cíle a cílové hodnoty v oblasti energie, akční plány týkající se energetické účinnost systému, použití energie a spotřeby energie při současném plnění příslušných právních předpisů a jiných požadavků.

EnMS je založen na přístupu neustálého zlepšování – *Plánuj-Dělej-Kontroluj-Jednej*. V průběhu plánovací fáze cyklu musí být určen energy officer a energy tým. Organizace musí zformulovat vlastní energetickou politiku v písemné formě. Energy tým řídicí tento cyklus reprezentuje spojení mezi zaměstnancem a vedení společnosti. Energetická politika musí být jasně vysvětlena uvnitř organizace. V této fázi musí organizace identifikovat významné směry využití energie a stanovit priority pro zlepšení kvality. Ve fázi *Dělej* probíhají zavedené procesy pro splnění stanovených cílů. Vedení zajišťuje dostupnost prostředků a kontroluje, zda jsou odpovědnosti správně delegovány. Odpovědní zaměstnanci musí vykonávat své povinnosti, týkající se energy managementu. Realizace EnMS je zahájená. Ve fázi *Kontroluj* probíhá interní audit, který určuje, jestli systém funguje správně a generuje očekávané výsledky. Výsledky auditu jsou zdokumentované a reportují se vedení firmy. V poslední fázi cyklu vedení připravuje písemné hodnocení na základě interního auditu. Tento dokument se nazývá management review. V případě potřeby je možně zahájit nápravná nebo preventivní opatření. Energeticky významné procesy se optimalizuji

a odvozují se nové strategické cíle [10].

Podle normy ČSN EN ISO 50001 vrcholový management musí prokázat svůj závazek, podporovat systém energetického managementu a neustále zlepšovat jeho efektivitu. Manažeři musí definovat energetickou politiku a stanovit cíle, stanovit reprezentanta, zajistit stanovení energetického týmu, poskytovat zdroje (finanční, lidské, vybavení, ...), zajistit správnost indikátorů, zohlednit energie při plánování strategií do budoucna, monitorovat energetický výkon a provádět revize. Pro tuto práci jsou relevantní odpovědnosti manažerů stanovení a schválení indikátorů, monitorování výkonu, schválení týmů, stanovení reprezentanta a činnosti v období revize.

Energetický reprezentant je člověk, který koordinuje aktivity související s hospodařením s energií. Je zodpovědný za zajištění implementace systému, jeho údržby a zlepšování. Dále vytváří a řídí energetický tým, vytváří reporty pro vedení a zajišťuje efektivitu kontroly systému. Z těchto odpovědností vyplývají role a zároveň požadavky na funkčnost pro každou roli v aplikaci.

2.2 Kvalita elektrické energie

Pod pojmem kvalita elektrické energie se rozumí míra, do které charakteristiky elektronické energie v daném bodě sítě odpovídají platným normám. Požadované hodnoty pro této veličiny v distribučních sítích stanovuje norma ČSN EN 50160 [24]. Nízká kvalita energie může vést k následujícím následkům: oteplování motorů, transformátorů a vedení, což má negativní vliv na životnost těchto zařízení, náhodným výpadkům počítačových systémů, poškozování elektroniky a blikání osvětlení.

V ideálním stavu dodávaná elektřina by měla mít stabilní frekvence, která se rovná nominální hodnotě, stabilní velikost napětí a harmonický průběh v sinusoidálním tvaru. Třífázová soustava by měla mít vyváženou amplitudu a fázový posun [29]. V reálném světě žádná elektrická síť není ideální, proto normou EN 50160 jsou definovány akceptovatelné podmínky pro provoz sítě (viz tab. 2.1).

2.2.1 Přerušení napájení

Přerušení napájení je problém s kvalitou energie který nastává, když dojde k výpadku napájení dodávaného k elektrické zátěži, což se nazývá „přerušení napájení“. Různé typy přerušení napájení jsou klasifikovány podle jejich trvání. *Krátkodobé přerušení* je úplná ztráta napětí na jednom nebo více fázových vodičích po dobu mezi 0,5 cykly a 3 sekundami. *Dočasné přerušení* je úplná ztráta napětí na jednom nebo více fázových vodičích po dobu mezi 3 sekundami a 1 minutou. *Trvalé přerušení* je úplná ztráta napětí na jednom nebo více fázových vodičích po dobu delší než 1 minuta.

Přerušení napájení jsou způsobena mnoha různými zdroji, jako jsou operace přepínání obslužných programů osvětlení, fyzické poškození elektrického

Limity charakteristik akceptovatelné normou ČSN EN 50160				
Charakteristika	Akceptovatelné limity	Měřící interval	Perioda monitorování	Akceptovatelný výskyt
Frekvence	$49.5 - 50.5Hz$ $47 - 52Hz$	$10s$	1 týden	$95 - 100\%$
Pomále změny napětí	$230 \pm 10\%$	$10min$	1 týden	95%
Překmity a poklesy napětí (<= 1 min.)	$10 - 1000$krát ročně pod 85%	$10ms$	1 rok	100%
Krátká přerušení (<= 3 min.)	$10-100$krát ročně pod 1%	$10ms$	1 rok	100%
Poruchová dlouhá přerušení (> 3 min.)	$10-100$krát ročně pod 1%	$10ms$	1 rok	100%
Přechodná napětí (L-N)	$< 1.5kV$	$10ms$	1 rok	100%
Transienty (L-N)	$< 6kV$	Neurčen	Neurčen	100%
Napěťové nevyvážení (L-N)	$2\% - 3\%$	$10min$	1 týden	95%
Harmonické složky napětí (L-N)	8% THD	$10min$	1 týden	95%

Tabulka 2.1: Limity charakteristik akceptovatelné normou [24]

vedení nebo lidské chyby. Okamžité přerušení napájení může mít vážné nebo dokonce nebezpečné výsledky v závislosti na připojené zátěži, například na mikroprocesorovém nebo nemocničním zařízení[29].

2.2.2 Podpětí, přepětí, poklesy a otoky

Ke druhému typu problémů dochází, když napětí na zátěži klesne pod minimální jmenovité napětí nebo stoupne nad maximální jmenovité napětí po určitou dobu. V závislosti na tom, jak dlouho tyto podmínky trvají, mohou být označovány jako *podpětí* nebo *přepětí* a *poklesy* nebo *otoky*.

Podpětí nastane, když efektivní napětí klesne pod 90% jmenovitého účinného napětí a zůstane na této úrovni déle než jednu minutu.

Přepětí je událost, kdy efektivní napětí stoupá nad 110% jmenovitého účinného napětí a zůstává tam déle než jednu minutu.

K poklesům napětí dochází, když napětí rms klesá mezi 10% a 90% po dobu půl cyklu na jednu minutu. V napájecím systému 50 Hz trvá kompletní sinusová vlna přibližně 20 milisekund, poloviční cyklus je přibližně 10 milisekund.

Zvyšení napětí jsou definovány jako zvýšení efektivní hodnoty napětí na více než 110% po dobu půl cyklu na jednu minutu[29].

2.2.3 Flikry, transienty a šum

Opakované snižování napětí v obvodech osvětlení, které může být detekováno lidským okem se nazývá Flikr. Termín flikr označuje velmi specifický problém související s lidským vnímáním světla produkovaného žárovkami. Mezi běžné zdroje blikání patří: svářečky oblouků, elektrické kotle, průmyslové motory, lasery, kopírovací stroje, pily a rentgenové stroje.

Transienty nastávají, když jsou hroty superponovány na napěťové nebo proudové sinusové vlně, v rozsahu amplitudy od několika voltů po několik tisíc voltů. Osvětlení a přepínání obsluhy obvykle způsobují krátkodobě krátkodobé impulsy s vysokou energií, zatímco elektronická zařízení, VFD a spínání indukčních zátěží obvykle způsobují nepřetržitě nízkou energii. Impulzivní transienty trvají od 50 nanosekund do 1 milisekundy. Oscilační transienty trvají od 0,3 milisekundy do 5 mikrosekund.

Šum označuje nežádoucí vysokofrekvenční oscilace, které jsou superponovány na střídavé napěťové nebo proudové sinusové vlně. Tento jev je obvykle zesílen nesprávným uzemněním a je schopen narušit elektronická zařízení, jako jsou počítače a programovatelné ovladače[29].

2.2.4 Účiník, nevyváženost a harmonické

Elektrické zátěže jsou často složeny z více než čistého odporu, kombinace odporu a reaktance v AC systému se nazývá impedance. Reaktivita má dvě formy: indukční a kapacitní, které nepřispívají k „užitečné“ práci na energetickém systému.

Účiník(Power Factor) je způsob, jak charakterizovat, kolik elektrické energie směřuje k produkci užitečné práce, jako je světlo, topení nebo strojní zařízení. Nízký účiník znamená, že v systému dochází ke ztrátě velkého množství energie ve formě zbytečného tepla, což obecně odpovídá vyššímu vyúčtování energie a degradaci zařízení.

K nevyváženosti dochází v trojfázových energetických systémech, když jednofázové zátěže (osvětlení, kancelářské vybavení atd.) Nevytahují stejné množství proudu v každé fázi, což vede k většímu namáhání neutrálního vodiče. Ideální stav nastane, když jsou zátěže vyvážené, což znamená, že napěťová a proudová fáze jsou od sebe přesně 120 stupňů, ačkoli proudy nemusí být ve fázích s napětími. Vyvážený třífázový čtyřvodičový systém bude mít nulový proud na neutrálním vodiči. Množství proudu na neutrálním vodiči v nevyváženém systému se bude zvyšovat se zvyšující se nevyvážeností, mohlo by to vést k přehřátí a riziku požáru. Motory poháněné nevyváženým napětím budou mít za následek malý točivý moment motoru pracující v opačném směru od otáčení motoru, což je fenomén známý jako protahovací moment. Když nastane tento stav, část energie dodané do motoru bude fungovat proti sobě.

Harmonické jsou formou zkreslení tvaru vlny, ke kterému dochází v obvodech obsahujících polovodičovou elektroniku, jako je LED osvětlení, spínané napájecí zdroje, elektronické předřadníky, počítače, robotika, zkušební zařízení atd. Tato „nelineární“ zátěž způsobuje sinusové vlny s vyšší frekvencí systém, který má za následek větší ztrátu energie ve formě zbytečného tepla. Přebytečné teplo produkované harmonickými může mít škodlivé účinky na energetický systém. Transformátory jsou obzvláště náchylné k poškození způsobenému harmonickými v důsledku bludných „vířivých proudů“, které cirkulují v železném jádru a vytvářejí přebytečné teplo.

Harmonické složky jsou identifikovány podle své frekvence v násobcích „základní“ nebo hlavní frekvence (50 Hz v České republice). Například třetí harmonická v systému 50 Hz by byla 150 Hz (50 × 3 = 180) a pátá harmonická by byla 250 Hz (50 x 5 = 300). Velikost každé harmonické frekvence lze měřit pomocí měřičů kvality energie a obvykle se zobrazují ve formě harmonického spektra. Celkové harmonické zkreslení (THD) a celkové zkreslení poptávky (TDD) se někdy používají u měřičů kvality energie, aby se zjednodušilo harmonické zkreslení jako jediné měření namísto celého spektra[29].

2.3 PQ monitory a analyzátory kvality

2.3.1 Požadavky na analyzátory kvality

Analyzátory kvality napětí v bodech přenosu mezi přenosovým systémem EPS a distribučními společnostmi musí, podle [19], být třídy A a musí být schopny současně měřit tyto parametry kvality. v třífázové síti: kmitočet sítě, velikost napájecího napětí a jeho odchylky, rychlé změny napětí, flikr, poklesy a zvýšení napájecího napětí, přerušení napájecího napětí, nesymetrie napětí, harmonické napětí, meziharmonické napětí, signály v napájecím napětí.

Kromě těchto parametrů kvality musí analyzátor umožňovat měření velikosti proud a z nich odvozených (podle přiřazených napětí) i dalších veličin: činný výkon, zdánlivý výkon, jalový výkon, zpětnou složku proudu a její úhlový vztah k referenčnímu napětí (nebo výkon), harmonické proudy a jejich úhlový vztah k referenčnímu napětí (nebo výkon).

Pro analyzátory kvality napětí v předávacích místech z DS a společných napájecích bodech s regionálními výrobci, podle [19], obvykle postačí třída B, v případě sporů se pro kontrolní měření kvality použijí analyzátory třídy A.

2.4 Zařízení pro sběr dat

Pro monitorování, porovnávaní, plánovaní a stanovení indikátoru výkonu energie je potřeba sbírat podstatná data. Data jsou sebrána pomocí vysoce přesných měřících zařízení od společnosti KMB systems. Dále jsou popsané zařízení, data ze kterých jsou k dispozici.

- BCPM je systém měřících přístrojů pro efektivní a přesné monitorování většího počtu vývodů. Plná instalace tohoto systému může měřit 3 napětí a až 60 proudů, což je 20 třífázových vývodů. Kromě napětí, proudu a energie je tento systém schopen měřit výkony (kW, kvar, kVA), harmonické zkreslení napětí i proudů nebo harmonické napětí až do 50. Sebraná data se mohou šířit pomocí Modbus nebo mohou být uložena do vnitřní paměti [8].
- NOVAR 2600 je kombinací multifunkčního analyzátoru kvality třífázové sítě s pokročilým regulátorem jalového výkonu. Pro sledování stavu sítě a funkce regulace v reálném čase mohou být přístroje vybaveny dálkovým komunikačním rozhraním [8].
- Analyzátor kvality ARTIQ 233 má vysokou přesnost a vzorkovací frekvenci. Umožňuje komunikaci použitím široké množiny rozhraní. Naměřená data se ukládají do operační paměti, jejíž kapacita je 512 megabajtů [8].

2.5 Aplikace ENVIS

ENVIS je software pro přenos dat z měřících přístrojů do počítače a jejich následné zpracování. Tento program je dodáván včetně podporovaných zařízení. Po načtení dat z přístrojů data jsou archivována do SQL databáze nebo do binárních souborů. ENVIS poskytuje možnost vzdáleného on-line sledování stavu zařízení a obsahuje základní nástroje pro vizualizaci a analytické zpracování archivovaných dat. Data mohou být exportovaná do formátů jako CSV, XML, PDF HTML a dalších [1]. Každý přístroj v závislosti na své konfiguraci a volitelných rozšíření podporuje následující typy archivů:

- Hlavní archiv – hodnoty všech měřených veličin ukládá po předem zvoleném časovém intervalu.
- Archiv S a M profilů – minutové průměry nejdůležitějších veličin pro jeden vybraný den v roce a pro den maximálního zatížení.
- LOG – informace o událostech přístroje jako změna stavu, výpadky, změna konfigurace a další.
- PQ hlavní archiv a archiv PQ událostí – měření potřebná pro vyhodnocování kvality energie podle EN 50160.
- Archiv odečtu elektroměru – obsahuje po předvolených časových krocích ukládaná měření z automatického elektroměru.
- Archiv zaznamenaných maximálních průměrných činných výkonů Pmax – měsíční hodnoty PavgElmerMax pro posledních 12 měsíců.

- PQ oscilogram a PQ průběh událostí – obsahují údaje o jednorázových přechodových dějích. Jsou podporované jenom některými zařízeními [1].

2.6 Nejpodstatnější data

Nejpodstatnější veličiny jsou, podle [24], uložené v hlavním archivu. To jsou: fázová napětí (u1, u2, u3), sdružená napětí (uLL...), napětí mezi středním a ochranným vodičem (uN), proud fázemi (i1, i2, i3), Činný výkon (P) jednotlivých fází, činný výkon první harmonické, jalový výkon (Q) jednotlivých fází, jalový výkon první harmonické, zdánlivý výkon (S) jednotlivých fází, zdánlivý výkon první harmonické, THD napětí a proudu (uTHD, iTHD), poměrná úroveň harmonických a meziharmonických frekvencí až do řádu 50, flikr, deformační výkon (D), účiník (cos(fi)). Této veličiny budou prezentována uživateli formou sloupcových a čárových grafů.

2.7 Existující řešení

Na trhu softwarů pro hospodaření energie je velký počet řešení. Této řešení jsou vyvíjené jak velkými firmami tak i startupy. Většina těchto řešení mají docela totožnou funkčnost, v této kapitole jsou popsaná vybraná řešení pro znázornění situace na tomto trhu.

2.7.1 Energy Elephant

Energy Elephant je cloudová služba, která umožňuje sledování spotřeby energie a nákladů. Je použitelná pro více druhů budov a organizací. Funkce správy faktur automaticky kontroluje ceny tarifů a pomáhá uživatelům snížit odhadované hodnoty faktur za energii. Energy Elephant sbírá data z faktur, analyzuje je a zobrazuje je ve formě grafů. Při zobrazení dat je kladen důraz na ty nejpoužitelnější pro uživatele, který není odborníkem v energetice. Jsou to data jako: cena, spotřeba a uhlíková emise. Pro detailnější monitoring dat lze porovnávat historická měsíční data. Energy Elephant umí provádět audity za účelem určení způsobu potenciálního snížení nákladů na energie. Energy Elephant nabízí svým uživatelům možnost zjistit, zda jiný dodavatel energie může poskytnout nižší cenu při aktuální spotřebě. Aplikace umí spočítat pětileté náklady při aktuální spotřebě a pomoct najít vhodnou alternativní cestu. Mezi zajímavé funkce tohoto softwaru patří možnost porovnat výsledky organizace s výsledky organizací které spravují energie nejlépe. Je taky poskytována mobilní aplikace, která umí nahrávat na cloud fotografie faktur a pozorovat vizualizovaná data. Větším firmám je nabízena funkce nahrávání dat ze senzorů [15].

2.7.2 SkySpark

SkySpark je platforma pro analýzu dat ze senzoru, systémů automatizace, měřicích zařízení, historických dat z databáze či csv souborů nebo webové služby. V nasbíraných datech vyhledává poruchy, závislosti, trendy a možnosti pro vylepšení konzumace energetických zdrojů a redukce nákladů. Na nasbíraná data aplikují rozpoznávací techniky pro vyhledávání nejpodstatnějších problémů a vizualizuje výsledky. Vizualizace zahrnuje: frekvence vzniku, časovou délku, magnitudu a náklady, které jsou touto událostí způsobeny. SkySpark je programovatelný, proto je přizpůsobitelný pro potřeby širokého spektra organizací [16].

2.7.3 Wattics

Wattics je webová aplikace pro energetický management. Používá data z měřících zařízení a senzorů pro analýzu energetické efektivity organizace. Funkcionalita aplikace je rozdělena na tři sekce: Dashboard, Data a Administrace. Domovní stránka dashboardu je konfigurovatelná sada grafických prvků, pomocí které uživatel je schopen rychle seznámit s daty, které považuje za podstatné. V dashboardu lze zvolit způsob notifikace to jsou logy, emaily nebo sms. Pro detailnější zobrazení dat se používají grafy, které zobrazují celkovou nebo obdobnou spotřebu energie a náklady pro každou budovu. Pro zobrazení nákladů a detekce vysoce spotřebních období se používají heatmapy. Podstatné události se detekují použitím algoritmů strojového učení. V aplikaci lze registrovat akce pro zlepšení energetické efektivity pro následující sledování jejich účinnosti. V sekci "Data" jsou funkce pro import a export dat. Sekce "administrace" obsahuje funkcionalitu pro správu uživatele a pro konfigurace pravidelně generování reportů [26].

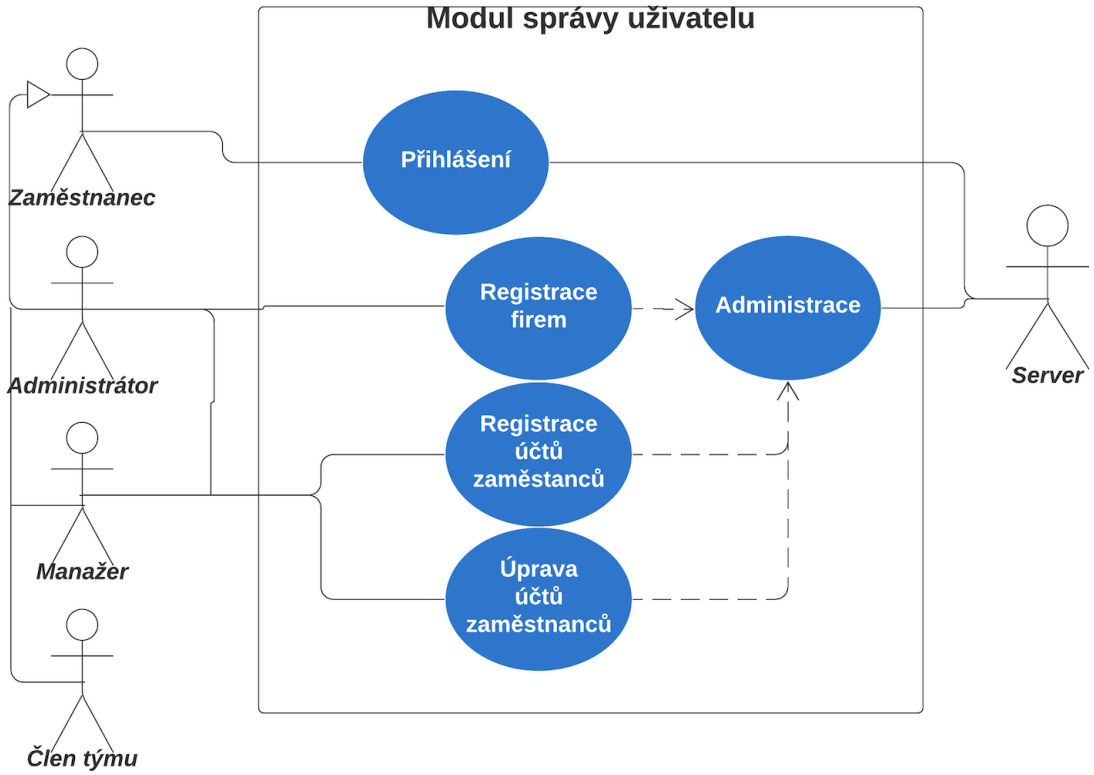
2.8 Shrnutí

I přesto, že se ve fázi průzkumu existujících řešení ukázalo, že existuje dostatečný počet řešení od větších firem, tato práce stejné může být přínosná. S každým rokem vznikají nové technologie a nástroje pro programátory. V případě když jde o mobilní aplikaci firmy preferují programovat je nativně. To znamená, že je potřeba vyvíjet aplikaci pro každou platformu zvlášť. I když to má hodně přínosů, ne každý podnik si toto může dovolit. V případě když jde o funkční prototyp řešení, muže dojít k nedostatku času na nativní realizaci projektu. Proto při realizace této mobilní aplikaci důraz bude kladen na použití novějších mobilních technologie, které můžou výrazně zjednodušit finanční a časovou náročnost realizace projektu.

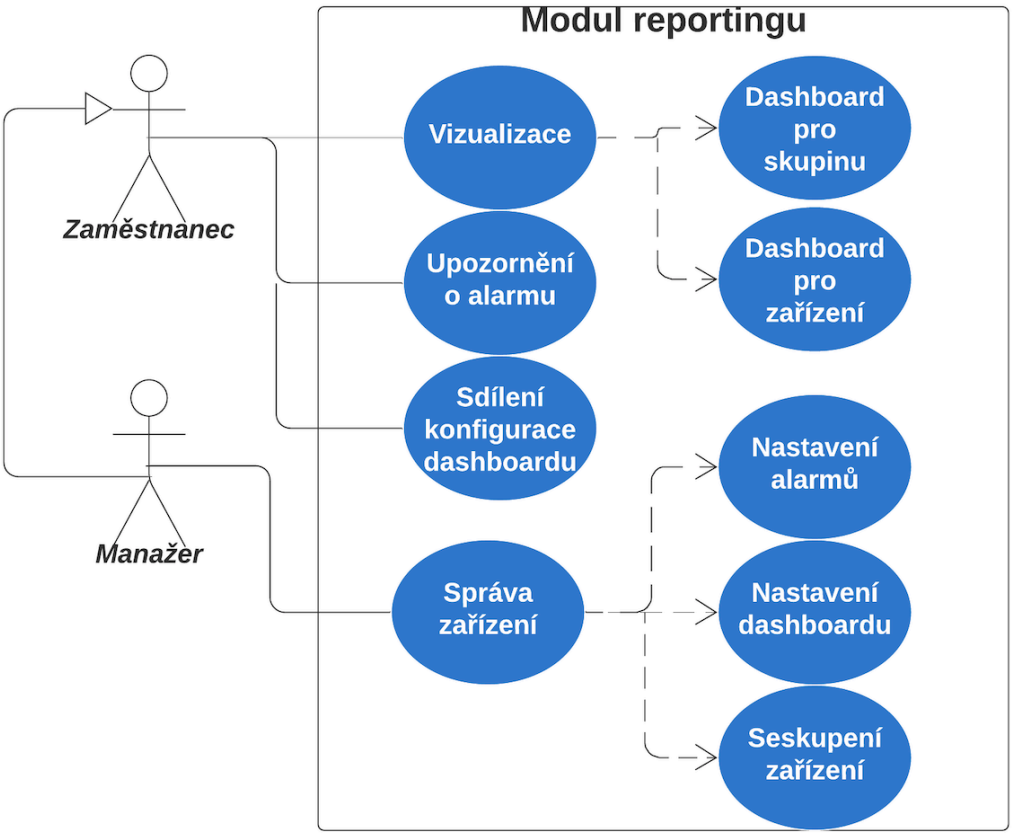
Z teoretické části této práce vyplývají následující požadavky na mobilní aplikaci. Aplikace musí nabídnout uživatelům možnost konfigurovatelné vizualizace dat ze zařízení. Je nutné, aby aplikace umožnila vytvoření skupin zařízení a seskupit jednotlivé skupiny do nadřazenějších skupin. Uživatelům

aplikace musí být nabídnuta možnost vizualizace dat z konkrétního zařízení nebo skupiny zařízení. V případě, že jde o skupinu, je potřeba vizualizovat data pro celou skupinu jako celek, tak i možnost porovnat data mezi jednotlivými zařízeními v této skupině. Konfigurovatelnost musí být realizovaná tak, že si uživatel u konkrétního zařízení, nebo skupiny zařízení, zvolí seznam dat, o který má zájem. Do možností nastavení také patří konfigurace alarmů následujícím způsobem: uživatel si specifikuje podmínky, po jejichž splnění mu bude odeslána notifikace. Do této aplikace se bude nutné přihlásit. Po přihlášení bude uživatelům nabídnuta funkcionalita v závislosti na jejich roli. Administrátoři mohou registrovat, mazat a upravovat firmy i uživatele. Team leader má plný přístup k funkcím aplikace, ale pouze v rámci své firmy. Mezi jeho funkce patří registrace zařízení a uživatelů, seřazení několika zařízení do skupin, úpravy nastavení pro alarmy a nastavení vizualizace dat pro zařízení nebo skupinu zařízení. Team member a manager mají pouze monitorovací.

Požadovánou funkcionalitu lze rozdělit do dvou modulů: spává uživatelů a reporting. Každý modul má svoje případy použití, které pro přehlednost znázorněné na obrázcích 2.1 a 2.2.



Obrázek 2.1: Případy použití modulu pro správu uživatelů



Obrázek 2.2: Případy použití modulu pro reporting

3 Technologická rešerše

3.1 Mobilní aplikace

Mobilní aplikace je software, který je navržen, aby běžel na mobilním zařízení jako mobil, tablet nebo chytré hodinky. V současné době jsou dva velké mobilní operační systémy IOS, od společnosti Apple a Android od Google. V dubnu 2020 Android běží na 70.68 % zařízení a IOS na - 28.79 % [14].

3.1.1 IOS

Mobilní operační systém IPhone OS byl vydán v roce 2007 a v roce 2010 byl přejmenován na IOS. IOS je systémem UNIXového typu. Má architekturu postavenou ze čtyřech základních vrstev: Cocoa Touch, Media, Core Services, Core OS. IOS Aplikace komunikují s hardwarem pomocí množiny konkrétních rozhraní. Tato rozhraní zjednodušují proces vývoje aplikací, které se chovají stejně na zařízeních s různými hardwarovými charakteristikami.

Cocoa touch poskytuje abstrakce pro vývoj uživatelského rozhraní a interakci s uživatelem, zpřístupňuje hlavní hardwarové funkce jako kontakty na kartě SIM, kameru, notifikace, uživatelská gesta apod. Media vrstva obsahuje knihovny pro tvorbu grafiky, přehrávání zvuků, videí a animací. Vrstva Core Services dává přístup k prostředkům potřebným pro vývoj aplikace jako volání, SMS, networking, GPS, databáze SQLite a šifrování.

Core OS zpřístupňuje operace jádra a další nízkoúrovňové API jako USB a Bluetooth[30].

3.1.2 Android

Android je mobilní operační systém, jehož jádrem je modifikovaný Linux. Vývoj androidu je veden firmou Google. Architektura tohoto operačního systému je rozdělena na 5 vrstev. Nejnižší vrstva je jádro, které tvoří abstrakci mezi hardwarem a softwarem ve vyšších vrstvách. Další vrstvy jsou přístupné vývojářům prostřednictvím Aplikačního Frameworku Android. Jsou napsány v C nebo C++ a využívají různé komponenty systému. Příklady takových knihoven jsou: Media Libraries, LibWebCore, Libc, SQLite, OpenSSL apod. Služby poskytované Aplikačním Frameworkem Android mohou být použity vývojářem přímo v aplikacích.

Základní sada služeb zahrnuje: *Views* – grafické prvky pro vývoj uživatelského rozhraní aplikace, *Content Providers* – poskytují přístup k obsahu jiných aplikaci, *Resource Manager* – zpřístupňuje přidané soubory jako text, obrázky apod., *Notification Manager* – umožňuje aplikacím zobrazit vlastní upozornění ve stavovém řádku. *Activity Manager* – řídí životní cyklus aplikace a umožňuje orientaci se zásobníkem aplikací. Nejvyšší vrstvou systému je aplikace, kterou využívá koncový uživatel [11].

3.1.3 Typy mobilních aplikací

Existují tři hlavní druhy mobilních aplikací – nativní, hybridní a webové. Aplikace napsané pouze pomocí prostředků a jazyku platformy jsou nativní aplikace. Jsou cíleny na konkrétní mobilní platformy. Aplikace, která je určena pro Android, se nikdy nespustí na IOS. Aplikace tohoto typu jsou vysoce výkonné, mají plný přístup k nativním API zařízení (kamera, senzory apod.) a díky konzistentnosti mohou poskytovat lepší uživatelskou zkušenost. Nativními jazyky pro Android jsou Java a kotlin (od května 2017), pro IOS jsou to Objective C a Swift.

Nevýhodou nativních aplikací je, fixace na platformu. Aby aplikace podporovala více platforem, je potřeba ji napsat zvlášť pro každou cílenou platformu. Kvůli tomu vzniká potřeba zaměstnat odborníky na každou platformu. Tím se zvyšuje časová náročnost, což přináší i růst finanční náročnosti celého projektu. Proto hodně firem dává přednost crossplaftormním řešením. Crossplatformní mobilní aplikace jsou mobilní aplikace, které jsou kompatibilní s několika mobilními operačními systémy.

Webové aplikace se píší pomocí technologií HTML, CSS a JavaScript. Uživatel přistupuje k webovým aplikacím použitím prohlížeče, proto mohou fungovat na více platformách. Pro použití této skupiny aplikací je nutná přítomnost stabilního internetového připojení. Hlavními nedostatky jsou, že webové aplikace mají omezený přístup k nativním API. Webové aplikace nejsou konzistentní a nejsou vyvíjeny přímo pro platformu, a proto poskytují horší uživatelskou zkušenost. Výhodou je, že využívají nejméně místa na disku.

Hybridní aplikace vznikají použitím jak nativních, tak webových technologií. Tím se dosahuje šiřitelnost kódu mezi platformy. Píší se pomocí frameworků jako je ReactNative, Cordova apod. Tyto aplikace mají horší výkon a na některých platformách mohou pro uživatele působit nepřívětivě. Mají také omezený přístup k nativním API [9].

3.1.4 Flutter

Flutter je mobilní SDK na vytváření vysoce výkonných mobilních aplikací použitím jediné báze kódu. Cílem je umožnit vývojářům poskytovat vysoce výkonné aplikace, které se budou tvářit jako nativní na různých platformách. Výrobcem je společnost Google. Dne 4. prosince 2018 bylo představeno stabilní vydání Flutter 1.0. Flutter má následující výhody:

- Rychlost vývoje – jediná báze kódu, Hot reload.
- Bohatá sada grafických widgetů pro MaterialDesign a Cuppertino (pro IOS).
- Jediný programovací jazyk pro GUI a logickou část – Dart.
- Zachování nativního look and feel pro každou platformu.
- Flutter také umožňuje perfektní ovládání pixelů po celém plátně, což umožňuje vytvářet prvotřídní designy, které jsou identické na obou platformách a stejné pro různé verze.
- Flutter je realizován tak, aby vývojář mohl udělat více práce, ale napsat méně kódu.

Flutter obsahuje reaktivní framework, 2D renderovací engine, předem připravené grafické komponenty a vývojářské nástroje. Architektura flutteru je rozdělena do několika vrstev. Po každé změně se widgety renderují do Skia canvasů. Pro renderování se většinou používá grafická karta zařízení. Platforma zobrazuje canvas a posílá zpět události. Aplikace zkompilovaná ahead-of-time běží na platformě nativně.

Flutter na úrovni platformy poskytuje shell, který hostuje Dart VM. Tento shell je specifický pro platformu. Shell umožňuje přístup k nativním API, životnímu cyklu aplikace, IME, a vytváří canvas, který je specifický pro platformu. Flutter se dá použít jako knihovna pomocí Embedder API.

Ve Flutteru jsou veškeré komponenty uživatelského rozhraní widgety. Widgety mohou být složeny z jiných widgetů. Takovým vnořením vzniká struktura, která se nazývá strom widgetů (widget tree). Objekty třídy BuildContext, obsahující informace o lokaci widgetu ve stromu, jsou předávány renderovací funkci build jako parametr.

Flutter je deklarativní framework. Deklarativní programování je založeno na myšlence programování pomocí definic. To znamená, že narozdíl od imperativního programování se zabývá otázkou, co se má udělat, něž jak se to má udělat. Funkce build, která je povinná pro všechny widgety, deklaruje, jak se má tento widget vyrenderovat. Podle deklarativní ideologie ve Flutteru jsou prvky uživatelského rozhraní odrazem aktuálního stavu aplikace. Při změně stavu se UI prvky nemění, ale rendrují znova. Widgety se dělí na dva typy, stavové a nestavové. U stavových widgetů je funkce build závislá na stavu [7].

3.2 Řízení stavu

Po změně stavu widgetu se musí vyrenderovat znovu nejen widget, který se změnil, ale i celý podstrom, který je pod ním. I když je tento proces ve Flutteru dobře optimalizován, tak se vývojář musí starat o správné řízení stavu, aby nedošlo ke ztrátě výkonu aplikace.

V nejširším možném smyslu je stav aplikace všechno, co existuje v paměti, když aplikace běží. Toto zahrnuje stav animace, textur, písmo atd. Některé stavy nemusí být řízené ručně, protože jsou řízené frameworkem. Ta část, o kterou se stará vývojář, se dělí do dvou typů, UI stav a šířený stav. UI stav je stav konkrétního widgetu, který určuje, jak se tento widget má zobrazit, např. aktuální stav běžící animace. Tento stav je uzavřen uvnitř widgetu, protože nepotřebujeme k tomuto stavu přistupovat z venku. K řízení UI stavu nejsou nutné žádné speciální techniky. Aplikační stav je ten stav, který je potřeba šířit mezi mnoha části aplikace, např. informace o aktuálním uživateli aplikace. Pro tento typ stavu vzniká potřeba použití speciálních technik řízení stavu.

Žádná ideální technika řízení stavu neexistuje. Proto se volba dělá na základě projektu a zkušenosti vývojáře. Mezi nejznámější techniky patří Redux, MobX a BLoC [6].

3.2.1 Redux

Redux původně vznikl v jazyce JavaScript v roce 2015. Autoři jsou Dan Abramov a Andrew Clark. Stav Aplikace (App State) je Neměnný Objekt, který se nachází na vrcholu hierarchie widgetů ve skladu (Store). Sklad je zpřístupněn potomkům pomocí dědičného widgetu (Inherited Widget) - Poskytovatel skladu (Store Provider). Pro vytvoření nového stavu je nutné vyřídit akce (Action). Reducer je čistá funkce, která přijímá akci, vytváří a vrací nový stav vznikající na základě předchozího stavu a zpracované akce. Kontejnery (Container) jsou widgety, které používají Store Konektor. Kontejnery odpovídají pouze za konverzi nového stavu aplikace do view modelu. Po změně stavu se všechny widgety připojené ke store konektoru aktualizují. Prezentační (Presentation) widgety jsou widgety, které zobrazují pouze data z view modelu. Pro načtení dat z databáze, nebo Web API se používá Middleware. Redux přináší do aplikace kruhový jednosměrný tok dat. Diky immutabilitě stavu nedojde nikdy ke změně stavu mimo datový tok. V případech, kde dochází k asynchronnímu programování, Redux zajišťuje předpověditelné chování aplikace, což výrazně usnadňuje zachycení chyb. Mezi nevýhody použití reduxu patří nutnost napsání velkého množství boilerplate kódu. Při asynchronním programováním dochází k vedlejším efektům, které mohou být kritické[25].

3.2.2 BLoC

Zkratka BLoC znamená Business Logic Component. Vzor vznikl ve společnosti Google a byl představen veřejnosti na konferenci Google IO v roce 2018. BLoC používá reaktivní programování k řízení toku dat v aplikaci. BLoC odpovídá za komunikaci mezi zdrojem dat jako databáze nebo Web API a widgetem, který tyto data používá. BLoC čerpá data ze zdroje, provádí potřebnou byznys logiku a publikuje změny dat pomocí datového toku (event stream). Widgety, které mají o data zájem, se podepisují na datový tok potřebného bloku. Po změně v datovém toku se související widgety aktualizují. BLoC má

v sobě dvě základní komponenty: sink a stream. Sink je používán jako vstup pro data. Stream je využíván jako výstup, umožňuje sledování změn v těchto datech[27]. Jako jedna z hlavních výhod se jeví, že při použití není nutnost instalace knihoven, protože Dart a Flutter podporují práci se streamy. Další pozitivní vlastností je dobrá separace uživatelského rozhraní a aplikační logiky. Bloc lze snadno pokrýt unit testy, protože je nezávislý na prostředí, ve kterém je spuštěn. Za podmínky dobrého použití tohoto přístupu, je znovupoužití komponent v jiných projektech snadné. Díky optimalizované implementaci streamů, nemá použití tohoto vzoru záporný vliv na výkon aplikace. Potřeba použití dvou streamů pro oba směry vytváří nutnost psaní většího množství boilerplate kódu. Tomuto je možné se vyhnout použitím funkcí místo vstupního sinku. Také existují knihovny, které dokáží usnadnit implementaci a tento nedostatek eliminují.

3.2.3 Reaktivní programování

Reaktivní programování je paradigma programování orientované kolem asynchronních datových toků a šíření změn, které je současným trendem vývoje moderních aplikací. Datový tok je sekvence událostí seřazena podle času, která při přidání nové události šíří data mezi pozorovatele. Datové toky jsou výpočetně levné a mohou být vytvořeny z čehokoliv. Tok může být vytvořen z proměnných, uživatelských vstupů, vlastností, datových struktur atd.

ReactiveX nebo Rx je nejpopulárnější rozhraní pro reaktivní programování, které je postaveno s respektováním ideologie návrhových vzorů jako Pozorovatel (Observable), Iterátor a funkčního programování. Pozorovaný objekt zpřístupňuje datový tok, pozorovatel odposlouchává a reaguje na změny. Rx také přináší Operátory. Operátory umožňují transformování, kombinování a manipulování s datovými toky. Výsledkem těchto operací jsou také datové toky [3].

3.3 Přihlašovací mechanismus

JWT (JSON Web Token) je způsob pro bezpečnou výměnu informací mezi klientem a serverem. Server vytváří přístupový token, který je podepsán privátním klíčem serveru a je odeslán klientovi po jeho autentizaci kombinací uživatelského jména a hesla. Token se skládá ze tří částí. První část nese informace o typu tokenu a algoritmu, pomocí kterého byl token vytvořen. Druhá část je datová. Do datové části tokenu se zapíše identifikátor uživatele. Poslední část slouží k verifikaci podpisu tokenu. Pomocí této části se dá na serveru ověřit integrita tokenu. Klient tento token ukládá a posílá ho s každým požadavkem pro autentizace uživatele. Token se dá z klienta odesílat v hlavičce *Authorization*, nebo v URL adrese pomocí parametru. Vzhledem k bezpečnosti aplikace je vhodné volit zasílání v hlavičce. Každý koncový bod serveru má seznam práv. Tato oprávnění jsou vyžadována pro jeho použití [2].

3.4 Serverová část

Před realizaci serverové části bylo nutně zvolit programovací jazyk, za použití kterého bude tato část realizovana.

3.4.1 Volba programovacího jazyku

PHP je populární programovací jazyk pomocí kterého vytvořeno velké množství systémů. Původně tento jazyk byl vytvořen na řešení jednoduchých úloh. S časem PHP začal získávat velkou popularitu a vznikla poptávka programátorů, které dříve používali jiné programovací jazyky. V důsledku toho PHP začal růst: vznikl manažer balíčku Composer, do jazyku byly přidané jmenné prostor, standardy a velké množství knihoven[5].PHP se stal profesionálním nástrojem, pomocí kterého se dá implementovat nejen jednoduché úkoly, ale také plnohodnotné projekty a velké portály.Dále v ekosystému PHP vznik velký počet CMS a frameworků. PHP získal vlastní mechanismus typování, který se ale liší od absolutní silné typizace.

Mimo PHP na realizace serverových části jsou používány i jiné programovací jazyky jako Ruby, Python, NodeJS, Go a další. Ruby byl populární v roce 2013 a 2014. V průběhu času, éra Ruby On Rails začala končit a jazyk Ruby zmizel z dohledu. To ale vůbec neznamená, že za použití Ruby nelze vyvíjet dobré projekty. Python je programovací jazyk, který je široce používaný ve vývoji modelů pro strojové učení. Pro tento jazyk jsou napsané velké množství knihoven na výpočty a statistiku, které umožňuji snadnější implementaci programů pro práci s daty. Přestože Python lze použít v mnoha oblastech programování tento jazyk má také několik nevýhod, jako: nízká rychlost a dynamická typizace kódu. Java patří mezi nejlepší volby jazyku pro Enterprise řešení za použití objektově orientovaného paradigma. Java má silnou statickou typizaci což je jednou z výhod tohoto jazyku.

Node nebo Node.js — je programovací platforma, založená na enginu V8, který převádí JavaScript do strojového kódu. Node.js transformuje JavaScript z úzce specializovaného jazyku do univerzálního jazyku. Pomocí NodeJS lze mikroslužby, web sokety, daemonické programy. Daemonické jsou programy, které jsou spuštěné jednou a dále dlouhodobě běží na pozadí[22]. Nicméně existuje problém, na který se mnoho vývojářů v posledních letech stěžovalo. Problém je v úniku paměti. Je nutně denně restartovat skripty, protože NodeJS spotřebovává více a více pamětí[23].

Go vznikl v roce 2009 a získal popularitu díky dobrému marketingu. Jazyk Go je nízkoúrovňový a kompilovaný. Tento jazyk, velmi dobře pracuje s vlákny. Syntaxe jazyku je odlišná tím, že obsahuje kolem 20 syntaktických konstrukci, což dovoluje programovat hned po přečtení dokumentace. Také mezi výhody tohoto jazyku patří “goroutine”, umožňující snadné psání asynchronního kódu. Další výhodou Go je kompilace do .exe nebo binárního souboru[18].

Každý programovací jazyk má řadu výhod a nevýhod. Každý jazyk je navržen s cílem řešit určité úlohy ve vlastním kontextu. Pro tento projekt byl zvolen

jazyk PHP protože je dobrou volbou pro implementace tohoto zadání.

3.4.2 Volba frameworku

Pro zrychlení vývojového procesu serverové části aplikace bude využit framework. Framework – je větší struktura, předem přepravených tříd a metod, knihoven a také komplikovaných programovacích řešení. Může obsahovat návrhové vzory nebo doporučené postupy při vývoji projektů. Každý framework muže mít zajímavé a užitečné vlastnosti. Frameworky se liší od knihoven komplexnějším přístupem. S technického pohledu jsou to mechanismy, které umožňuji usnadnění implementace určitého systému tím, že poskytují hotový kód pro nejčastější případy nebo standardní řešení. Znát filozofii konkretního frameworku je nutně pro programátora pro pochopení případu použíti frameworku a pochopení způsobu interakci s frameworkem.

Stejně jako u volby programovacího jazyku pří výběru frameworku je důležité volit na základě vaši konkretní úlohy. Mezi největší frameworky v PHP patří Laravel a Symfony. Existují také průměrně velké, ale rychlé platformy jako YII Framework. Jsou mikro platformy jako: Slim, Phalcon, Lumen. Podstatním rozdílem mezi nimi je v architektuře a případech použití.

Existují kritéria pro volbu platformy jsou to: monolitnost, modularita, kompatibilita, interní architektura, škálovatelnost, podpora, přístup autorů řešení, vzory a principy návrhu. Pokud nástroj je monolitní, migrace na jiné řešení bude složitější. Například, pokud vznikne potřeba migrovat projekt, který je realizován pomocí WordPress na jinou technologii, hned se zjistí, že migrace z WordPressu je hodně složitá. V tomto případě nepomůže i to, že WordPress má několik tisíc pluginů. Hned se zjistí, že jednoduše bude předělat celý projekt znova. Proto je lépe použíti modulárních frameworku.

Také je důležité zkontrolovat jestli framework je kompatibilní s moderními standardy. Ve světě PHP jsou stanovené standardy jako: PSR 7 pro požadavky a odpovědí serverů, PSR 15 pro controllery a middleware, PSR 16 pro cachování. Odpovídání standardům přináší flexibilitu. Pokud systém je navržen s dodržením principů OOP a návrhových vzoru tento projekt se dá jednoduše rozšiřovat přidáním veškeré potřebné funkčnosti. I když framework je rozšiřovatelný je důležité zkontrolovat jestli tento framework je stále udržován. Jsou případy když používaná knihovna je napsaná jedním jediným člověkem, který už během několika let nepodporuje vlastní projekt. Podobné knihovny můžou být nebezpečné, použití takových knihoven je velmi riskantní. Proto je potřeba dávat pozor na udržovatelnost používaných knihoven a frameworků. Autoři knihoven a frameworků vyvíjí svoje nástroje buď ve volném čase nebo na plný úvazek jako například Taylor Otwell, autor Laravel. Údržba, podpora a vývoj Laravel je pro Taylora jako zaměstnání. Podobné vývojáře se snaží dbát o kvalitu svým výrobků a neustále zdokonalovat svoje nástroje. Také ne nejlépe použití frameworků které jsou brané vážně a používané ve větších firmách.

Symfony je framework cílený na větší korporativní projekty. Tento framework dodržuje standardy vývoje a vyžaduje větší formalitu při psání kódu programá-

torem, což pozitivně působí na výslednou kvalitu produktu. Ale to má i svoje nevýhody, protože dodržování všech standardů zpomaluje rychlost vývoje. Pro menší projekty použití tohoto frameworku je redundantní [28].

Laravel je jedním z populárních frameworků PHP a má větší ekosystém komponent. Laravel je modulární a podporuje architekturu mikroslužeb. Laravel obsahuje hotové řešení pro běžné úlohy jako: autorizace, zpracování dotazů,posílání emailů, stránkovaní, ORM atd. Má také své vlastní doplňkové služby jako: Vapor, Forge, Vyslanec, Horizon, Nova, Echo, Lumen, Homestead, Spark, Valet, Mix, Cashier, Dusk, Passport, Scout, Socialite, Telescope, Tinker [20]. Jiné platformy takové služby nemají. Některé ze služeb laravelů jsou placené, což je zdůvodněný krok od vývojářů tohoto softwarového produktu, protože této služby můžou běžet na velmi drahých serverech, a také vyžadují podporu a aktualizace. Komponenty jsou psáné podle principu SOLID, všechny závislosti jsou předávané za použití techniky vkládání závislostí. Nastavení lze nakonfigurovat prostřednictvím kontejneru pro injektáž závislostí.

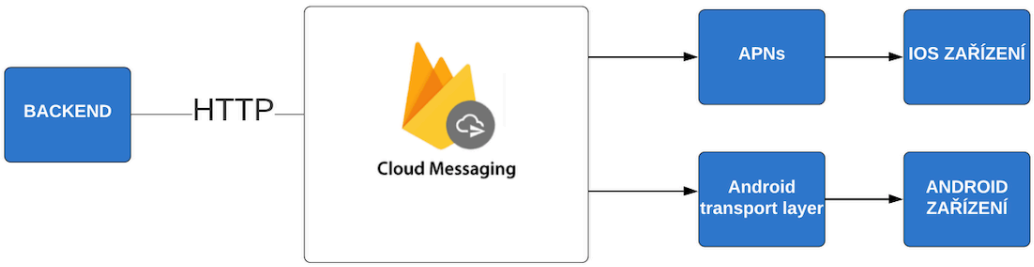
3.5 Notifikační služba

3.5.1 Apple Push Notification service

Na zařízeních s operačním systémem IOS je APNs jedinou možností vzdáleného zasílání push notifikací uživatelům. Veškeré notifikační služby třetích stran jako Firebase Cloud Messaging nebo One Signal tuto službu využívají. Při prvním spuštění aplikace systém automaticky vytvoří trvalé šifrování IP a spojení mezi aplikací a notifikační službou. Toto připojení umožňuje obdržení vzdálených notifikací pro danou aplikaci. Další nutná část pro využití této služby je vytvoření trvalého a bezpečného kanálu mezi serverem poskytovatele aplikace a APNs. Pro toto připojení jsou vyžadovány kryptografické certifikáty od společnosti Apple, které vývojář aplikace může získat po konfiguraci vývojářského účtu Apple. Dále server poskytovatele musí získat od APNs token a další užitečná data. Token je unikátní pro instanci aplikace a zařízení. Díky tomu poskytovatel ví o každé běžící instanci aplikace a směřuje notifikace cílovým uživatelům. V případě, že cílové zařízení není připojeno k internetu, APNs ukládá notifikaci na omezenou dobu. Pokud je toto zařízení znovu dostupné, pokus o odeslání notifikace se opakuje. Tato funkce ukládá pouze nejnovější žádost o notifikace pro instanci aplikace, pokud je zařízení offline, odeslání požadavku na oznámení s cílením na toto zařízení způsobí, že předchozí žádost bude zrušena. Služba APNs je bezplatná, ale je nezbytně nutně mít vývojářský účet Apple [5].

3.5.2 Firebase Cloud Messaging

Pro mobilní aplikace je výchozí notifikační službou Firebase Cloud Messaging. Na rozdíl od APNs je FCM crossplatformní. Firebase podporuje zasílání notifikací na zařízení Android, IOS a web, což je výhoda z hlediska údržby a jednoduchosti implementace. Na úrovni platform-level message transport firebase komunikuje s APNs, když je odeslaná notifikace cílena na IOS zařízení, viz obr. 3.1.



Obrázek 3.1: Infrastruktura doručení notifikací firebase messaging

Firebase má 2 typy notifikací: notification message a data message. Oba typy mají kapacitu 4 kilobajty. Notifikace obsahují množinu párů klič-hodnota. Notification message může obsahovat data, která jsou viditelná pro uživatele aplikace, tak i užitečná data pro zpracování aplikací, na rozdíl od data messages, které obsahují jenom data. Po rozkliknutí notification message se aplikace otevře, nebo provede určitou činnost. Pro to, aby aplikace mohla dostávat notifikace od služby firebase, je potřeba mít aplikaci zaregistrovanou ve Firebase Cloud Console, kde se vygeneruje konfigurační soubor, který je potřeba přidat do sestavení aplikace. V tomto souboru se nachází klíč, kterým se aplikace identifikuje. Poté už aplikace může posílat notifikace pro všechny uživatele. Aby uživatelé mohli dostávat personalizované zprávy pro konkrétního uživatele, je potřeba ze spuštěné aplikace odeslat požadavek na získání tokenu. Na zprávy pro určitý segment uživatelů se používají topiky. Ze spuštěné aplikace se posílá požadavek na registraci, který obsahuje název topiku.

Do navrhovaného systému se notifikační služba integruje následujícím způsobem. Při registraci uživatele na serveru, se pro něj vygeneruje název topiku, který se uloží do databáze. Při úspěšném požadavku o autentizaci uživatele bude vrácen název topiku, na který se instance mobilní aplikace podepíše. Pokud dojde k potřebě odeslání notifikace, vyberou se z databáze uživatelské topiky, vygeneruje se zpráva a pošle se požadavek o odeslání notifikace do FCM. Samotná notifikační služba Firebase Cloud Messaging patří mezi bezplatné produkty rodiny Firebase [12].

3.5.3 Apache Kafka a RabbitMQ

Firebase a APNs nezaručují uspořádání doručovaných zpráv dle času odeslání. Pro garanci doručení a uspořádání zpráv ve větších aplikacích reálného času a big data projektů, existují platformy Apache Kafka, nebo RabbitMQ. Obě platformy nabízejí tyto záruky, ze kterých je možné si vybrat „maximálně jednorázového doručení“ a „minimálně jednorázového doručení“, nebo „přísně jednorázového doručení“. Poslední záruka v Kafce funguje dle velmi omezeného scénáře.

RabbitMQ je distribuovaný systém řízení fronty zpráv. Pracuje v klastrech uzlů pro odolnost vůči chybám a vysokou dostupnost. Implementuje protokoly AMQP 0.9.1, Stomp, MQTT. Při použití dalších modulů je k dispozici implementace HTTP. Publishers posílají zprávy na exchange.

Exchange odesílá zprávy ve frontách a na jiné exchange. RabbitMQ pošle potvrzení vydavatelům, jakmile obdrží zprávu. Spotřebitelé udržují trvalá připojení TCP k RabbitMQ a oznamují, které fronty nebo více front obdrží. RabbitMQ odesílá zprávy příjemcům. Příjemci pošlou potvrzení úspěchu nebo chyby. Po úspěšném přijetí budou zprávy z front odstraněny. Úkolem programátora je vytvořit správnou architekturu z komponentů RabbitMQ. Například pokud tři příjemci přijímají zprávy z jedné fronty, pak každý příjemce obdrží třetinu zpráv z fronty, které lze použít k řízení zátěže. Toto je důležité pro projekty, kde všichni příjemci potřebují přijímat veškeré zprávy. Nedostatkem škálování je, že jedna fronta není schopna doručovat více zpráv najednou, ale pouze jednotlivě[13].

Kafka je distribuovaná streamovací platforma, která umožňuje publikovat a přihlásit se k odběru záznamových toků, ukládat je a zpracovávat nahrávky, jakmile dojde k jejich vzniku. Používá se k výměně datových toků mezi aplikacemi s garantovaným doručením. Každá položka představuje klíč, hodnotu a čas vytvoření. Příspěvky jsou publikovány podle témat. Každá položka je uložena na přizpůsobitelné časové období od okamžiku zveřejnění. Položky v tématu jsou uloženy v uspořádané FIFO sekvenci. Témata s položkami jsou uložena ve skupině serverů, aby se zjednodušilo škálování a zajistilo se, že nedochází k chybám. Cluster má vedoucí server, který přijímá všechny požadavky na zápis a čtení dat, zatímco zbytek serveru opakuje akce vedoucího. Pokud se akce vedoucího serveru nezdaří, převezme druhý server automaticky vedoucí funkce. Každý server v clusteru funguje jako vedoucí pro určitá témata pro vyrovnávání zatížení. Rychlost čtení není závislá na množství dat. Na rozdíl od RabbitMQ za realizaci logiky čtení odpovídá spotřebitel Kafky. Spotřebitel může například rozhodnout, zda obdrží první nepřijatou zprávu v daném topiku, nebo nové, které vznikly po zahájení odposlechu datového topiku [4].

3.5.4 Shrnutí

V mobilní aplikaci na platformách iOS a Android nelze provádět doručování nativních push notifikací vzdáleně bez použití cloudových služeb APNs a Firebase. V běžící aplikaci pro aktualizaci dat v reálném čase, je ale možné použít Apache Kafka nebo RabbitMQ. RabbitMQ nabízí široké spektrum šablon na výměnu zpráv mezi aplikacemi díky velkému počtu funkcí. Příjemce nemusí provádět načítání, deserializaci a kontrolu jednotlivých zpráv, pokud potřebuje pouze podmnožinu. Je snadné s ním pracovat, škálování nahoru a dolů se provádí jednoduchým přidáváním a odebíráním příjemců. Jeho plug-in architektura umožňuje podporovat další protokoly a přidávat nové funkce. Tyto dvě technologie se však používají k vytvoření aplikační infrastruktury, proto bude v rámci tohoto projektu pro snadnou implementaci použit firebase cloud messaging.

4 Implementace serverové části

4.1 Modely

Modely jsou třídy, které jednoznačně popisují data a pravidla pro tyto data. To jsou entity, které budeme ukládat v databázi. Veškeré modely v tomto projektu dědí třídu Model z komponenty Eqolent ORM, což umožňuje serializaci dat pro stále ukládaní do databáze. Výhodou použití ORM je to, že skutečné úložiště lze změnit bez žádných změn v kódu, stačí upravit konfigurační soubor *config/database.php*.

4.2 Kontrollery

Kontrollery popisují rozhraní web api a mají na starosti získávaní dotazu uživatele a posílání odpovědí. Konverze mezi složitými datovými typy jako modely a typy pro přenost dat ve formátu JSON. V tomto projektu kontrollery jsou pouze vstupní bránou serveru. Skutečná aplikační logika se pak dělá ve službách, které jsou závislostmi kontrollerů. Po obdržení dotazu kontroller předává data příslušné službě. Po vracení výsledku ze službou, kontroller složí z tohoto výsledku serverovou odpověď.

4.3 Validace Dat

Validace dat je realizovaná ve vlastních třídách, rozšiřujících třídu *Request* z Laravel. Validace se dělá hned až požadavek přijde na server. Proto pokud vstupní data nejsou validní, požadavek se ani nedostane do controlleru a klientovi bude vracena odpověď s kódem *400* a popisem nesprávných dat. Ve funkci *rules* třídy *Request* se popisují pravidla pro validaci jednotlivého dotazu.

```
POST api/auth/
POST api/auth/ refresh
```

Obrázek 4.1: Seznam koncových bodů pro přihlašování

```
GET api/users/me
GET api/users/
POST api/users/
GET api/users/{id}
PATCH api/users/{id}
DELETE api/users/{id}


GET api/companies/
POST api/companies/
GET api/companies/{id}
PATCH api/companies/{id}
DELETE api/companies/{id}
```

Obrázek 4.2: Seznam koncových bodů pro správu uživatelů

```
GET api/devices/
POST api/devices/
GET api/devices/{id}
PATCH api/devices/{id}
DELETE api/devices/{id}

GET api/devices_groups/
POST api/devices_groups/
GET api/devices_groups/{id}
PATCH api/devices_groups/{id}
DELETE api/devices_groups/{id}

GET api/dashboards/
POST api/dashboards/
GET api/dashboards/{id}
PATCH api/dashboards/{id}
DELETE api/dashboards/{id}
```

Obrázek 4.3: Seznam koncových bodů pro správu zařízení

4.4 Přihlašovací mechanismus

Pro autentizace uživatele v serverové části se používá token přístupu. Do tohoto tokenu se vkládají id uživatele a jeho přístupová práva. Platnost přístupového tokenu je omezená na deset minut. Po vypršení platnosti přístupového tokenu uživatel může získat nový token přístupu pomocí refresh tokenu.

Refresh token je platný 90 dní. Viz obr. 4.4.

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
"refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.e...",
"access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b..."
}
```

Obrázek 4.4: Ukázka odpovědi serveru po úspěšné autorizaci

Pro použití zabezpečených koncových bodu hlavička požadavku musí obsahovat klíč *Authentication* s hodnotou *Bearer {token}* . Koncové body, které jsou zabezpečené nebo vyžadují určitá přístupová práva budou získávat tuto informaci z tokenu. Pokud token není platný, odpověď serveru bude mít status kód 401. Pokud v tokenu chybí vyžadovaná práva, server odpověď serveru bude mít status 403.

Pro získání obou tokenu je vytvořen autorizační koncový bod *POST auth/*. Jako vstup je nutně v těle POST požadavku poslat kombinaci emailu a hesla. Pokud kombinace emailu a hesla je správná, server vrátí odpověď se status kódem 200, jejíž tělo obsahuje refresh token a přístupový token zakódované do formátu JSON. Jinak bude vracená odpověď s kódem 401 a chybovou hláškou. Pro získaní nového přístupového tokenu byl vytvořen endpoint *POST auth/refresh/*.

Správnost kombinaci emailu a hesla se kontroluje na základě přítomnosti v databázi uživatele s získaným emailem. Pokud tento uživatel existuje, heslo z požadavku bude hashovano s použitím tajemného klíče a porovnáno s hashem uloženým v databází. Tento mechanismus je implementovan komponentou Laravel Passport. Konfigurace pro autentizaci lze upravit v souboru *config/auth.php*. Koncové body pro přihlašování jsou implementované v třídě AuthController. Seznam koncových bodů pro přihlašování je na obr. 4.1.

4.5 Správa uživatelů

Správa uživatelů je realizovaná použitím systému práv a rolí. K provedení jakékoliv činnosti v aplikace je potřeba aby uživatel měl určité přístupové právo. Každá role je definována množinou přístupových práv. Logika pro správu přístupových práv a roli je implementovaná v jmenným prostoru *UserAccess*. Tato logika umožňuje vytvoření, mazaní, přiřazení jednotlivých práv rolím. Dále se vytvořená role může být přiřazena konkretnímu uživateli. K realizaci této lo-

giky byla použita komponenta *Permission* z Laravel. *PermissionController* a *RoleController* definují koncové body pro vytvoření, mazání a úpravu pro práv a rolí. V kontrolleru *RoleManagerController* je realizovaná funkčnost přiřazení jednotlivých práv rolím. *PermissionManagerController* implementuje logiku přiřazení roli uživatelům.

4.6 Nastavení dashboardů

Pro vytvoření dashboardu s vybranými uživatelem grafy pro každé zařízení bylo potřeba navrhnout datovou strukturu, která by uchovávala data o grafu. Táto struktura se při ukládání posílá na server a ukládá do databáze. Pro tento účel byla vytvořená třída *Chart*, která ukládá identifikátor zdroje dat, typ grafu a seznam veličin pro zobrazení. Dashboard je složen ze seznamu instanci této třídy. Proto není potřeba ukládat dashboard.

Pro získávaní dat pro jeden graf, stačí poslat na server identifikátor grafu, časový interval mezi měření, počáteční datum měření pro vyhledávaní a počet měření. Výsledkem dotazu na data je datová struktura, která obsahuje identifikátor zdroje měření a seznam veličin s daty. Každý sloupec obsahuje název veličiny, měrnou jednotku, seznam limitů a seznam jednotlivých měření a času.

5 Implementace klientské části

5.1 Architektura aplikace

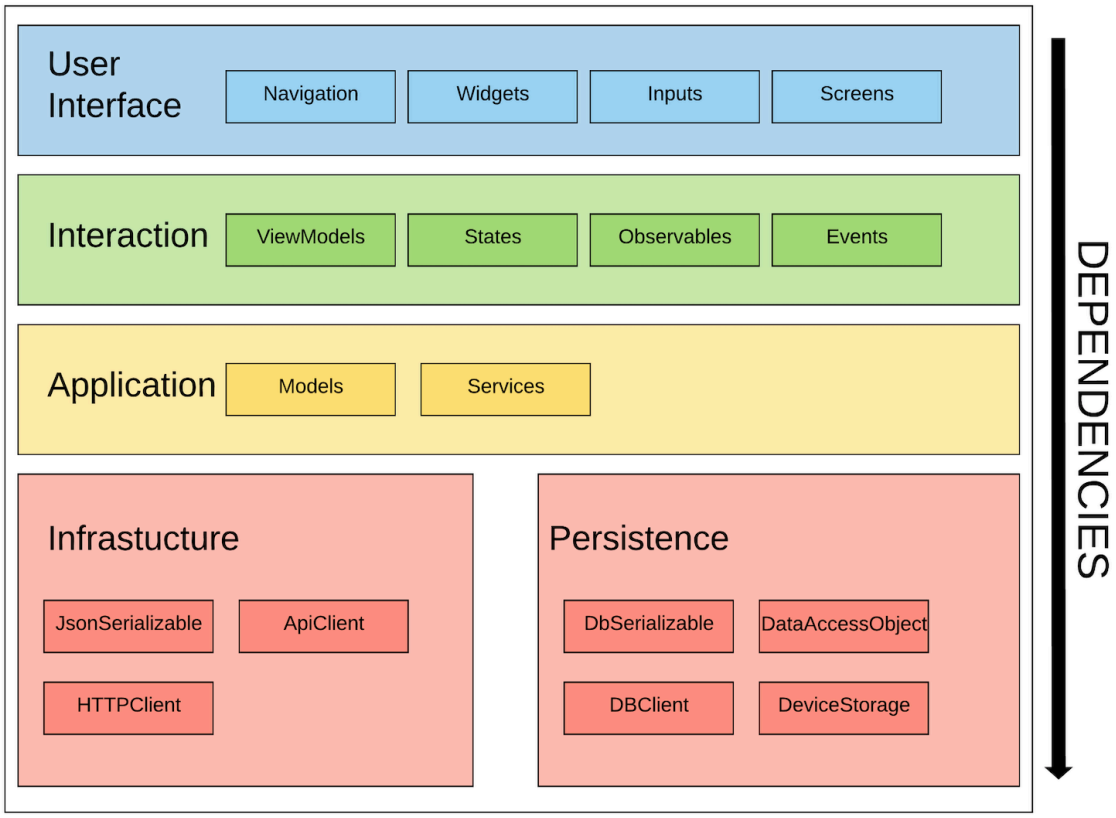
Architektura aplikace nemá zásadní vliv na funkčnost aplikace, ale má kritický vliv na její složitost, udržovatelnost a rozšiřitelnost. Cílem softwarové architektury je minimalizace mzdových nákladů potřebných k vytvoření a údržbě systému [21]. Flutter implicitně nenabízí návrhový vzor pro architekturu. To znamená, že programátor je zodpovědný za výběr a implementaci architektury podle potřeb každé konkrétní aplikace.

Aplikace je rozdělena do pěti vrstev: prezentační, interakční, infrastrukturní, perzistentní a aplikační, viz obr. 5.1. Infrastrukturní vrstva odpovídá za přenos dat mezi aplikacemi, například komunikace se serverem. Perzistentní vrstva řeší získávání a ukládání dat do stálé pamětí zařízení. Aplikační vrstva řeší logiku a stav aplikace. Je to vrstva, která definuje účel aplikace. Interakční vrstva je složená z poskytovatele pro view model. To jsou třídy, které mají za úkol poskytovat relevantní ViewModel. Tyto třídy mají na starosti transformace dat z aplikačních služeb do tvaru, který je vhodný pro zobrazení konkrétním View. Dalšími starostmi jsou získávání a validace vstupů z View, ošetření chyb a notifikace View o změnách v UI stavu. View zobrazuje data z ViewModelu a překresluje se při jeho změně, reaguje na vstup uživatele voláním funkcí provideru.

5.1.1 Aplikační vrstva

Funkcionalitu této aplikace lze seřadit do tří služeb: Authentication Service, User Management Service a Device Management Service. Authentication Service:

- Má funkci pro přihlášení uživatele uživatelským jménem a heslem.
- Ukládá a zpřístupňuje instanci naposled úspěšně přihlášeného uživatele, práva, která tento uživatel má, a flag, jestli přihlášení tohoto uživatele je platné.
- Má funkci pro zjištění, jestli je tento uživatel pracovníkem určité firmy.
- Má funkci pro obnovení přihlašovacího tokenu s vypršenou dobou platnosti.



Obrázek 5.1: Případy použití modulu pro správu uživatelů

- Má funkci pro odhlášení uživatele.

User Management Service:

- CRUD operace pro firmy.
- CRUD operace pro uživatele.

Device Management Service:

- CRUD operace pro měřící zařízení.
- CRUD operace pro skupinu měřících zařízení.
- CRUD operace pro datové záznamy měřících zařízení.
- CRUD operace pro konfigurace dashboardů a definice alarmů.

Notification service odpovídá za odesílání žádosti o obdržení zpráv z určitého topiku. Přijímá callback funkce, které se zavolají po obdržení zprávy.

5.1.2 Infrastrukturní vrstva

Třídy infrastrukturní vrstvy v této aplikaci jsou zastoupeny třídami, které komunikují s REST API použitím HTTP protokolu. Další zodpovědností těchto tříd je serializace a deserializace mezi objekty jazyka Dart a objekty ve formátu JSON.

V této aplikace třídy této vrstvy interně používají singleton instanci http klienta z knihovny *dio* na komunikaci se serverem. Což umožňuje centralizované nastavení http klienta jako: změna konfigurace za běhu aplikace, nastavení mechanismu pro ošetřováni chyb a synchronizaci dotazů.

Samotné volání a parsování odpovědí ze serveru jsou implementovány generátorem kódu pomocí knihoven *retrofit* a *json_serializable*. Pro generací kódu knihovnou *retrofit* je nutné jenom popsat rozhání pro komunikaci se serverem a popsat koncové body pomocí anotací.

5.1.3 Perzistenční vrstva

Třídy perzistentní vrstvy mají podobné rozhraní jako třídy infrastrukturní vrstvy, ale roli datového úložiště představuje soubor ve filesystému zařízení. Pro zajištění bezpečnosti a konzistence přístupu k souboru, mohou být použity knihovny které zapouzdří přístup k souborům. Po zapouzdření přístupu k datovým souborům, může takový systém působit jako přístup k SQL nebo NoSQL databázi. Bezpečnost lze posílit zavedením šifrování.

5.1.4 Interakční vrstva

Interakční vrstva je složená z tříd typu Provider, které schovávají všechno, co je mimo zájem prezentační vrstvy. Třídy typu Provider dostávají na vstupu pouze objekty typu Event a na výstup posílají Stream s objektem ViewModel. Objekty typu Event obsahují informace o typu události a data. Například na vstupu třídy AuthProvider máme event typu "PokusOAutentizace" užitečnými daty tohoto eventu bude uživatelské heslo a jméno. Po zpracování tohoto eventu bude vytvořen nový objekt typu AuthErrorViewModel, který obsahuje informace o tom, že došlo k chybovému stavu a jeho popis. Dále tento objekt bude odeslán do streamu, odkud ho obdrží View, který tento stream odposlouchává. Tento View se přerenderuje a vypíše se text, že došlo k chybě a bude zobrazen popis z AuthErrorViewModel.

5.1.5 Vrstva uživatelského rozhaní

Uživatelské rozhání je složeno z tříd frameworku Flutter *StatefulWidget a StatelessWidget*. Této třídy mapujou vstupní parametry a vnitřní stav do vzhledu. V této aplikace se používají widget *Consumer* z knihovny *Provider*, které umožňuji párovaní widgetu na ViewModel pro překreslování části widgetu při změně v instanci ViewModelu. Také se používá widget *StreamBuilder*, který umožňuje překreslovaní částí widgetu při vzniku nových dat ve streamu.

Mezi zvláštními případy widgetu patří *Navigator*, který umožňuje navigaci. *Navigator* ukládá navigační cesty do zásobníku, což umožňuje navigaci zpět. *Navigator* je *Inherited Widget* což znamená, že přístup k instanci lze získat z instanci *BuildContext* v jakémkoliv podřazeném widgetu.

5.2 Přihlašovací mechanismus

Rozhraní *IAuthApi* viz obr. 5.2 implementuje volání koncových bodů pro přihlašování a obnovení přihlašovacích tokenu ze Serveru. Po obdržení přístupového tokenu implementace tohoto rozhraní nastaví autorizační hlavičku http klienta. Refresh token bude uložen do bezpečného úložiště v stále pamětí zařízení. Implementace třídy IAuthApi také má na starosti nastavení http klienta takovým způsobem, aby po neúspěšném požadavku s odpovědí serveru s kódem *401*, proběhl pokus o obnovení přístupového tokenu. Po úspěšném obnovení tokenu požadavek bude jednou opakován. Pokud refresh token není validní, nevalidní token bude smazán, následně se zavolá funkce *logoutListener*. Funkce *logoutListener* je předena nadřazeným objektem pro ošetření odhlášení.

```
abstract class IAuthApi {
        Future<bool> auth(String email, String password);
        Future<bool> refresh();
        set logoutListener(void Function() logoutListener);
        Future<void> logOut();
}
```

Obrázek 5.2: Třída IAuthApi

V interakční vrstvě za přihlašovaní odpovídá instanci třídy ILoginViewModel viz obr. 5.3. Táto třída zpřístupňuje funkci pro přihlašovaní, informaci jestli uživatel je přihlášen, instanci modelu uživatele a informaci o chybovém stavu. Funkce *login* inkapsuluje volání přihlašovací logiky prostřednictvím třídy IAuthService. Pokud při přihlašování došlo k chybě, informace o chybě se uloží do proměnné.

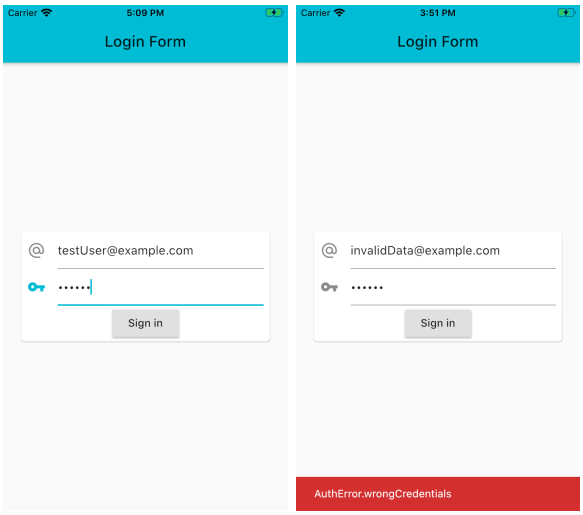
V UI vrstvě jsou tři třídy které se týkají přihlašování: *LoginWidget, HomeDrawer* a *LoginNavigationWidget*.

LoginWidget je vzhled pro přihlašovaní uživatele, kde uživatel může zadat jméno a heslo. Pokud dojde k chybě tento widget vykresli chybovou hlášku. Viz obr. 5.4.

LoginNavigationWidget odposlouchává stav *ILoginViewModel*.Pokud je uživatel úspěšně přihlášen tento widget přesměruje *HomeWidget*, pokud není, uživatel bude přesměrován na *LoginWidget*.

```
abstract class ILoginViewModel {
        User get current;
        bool get busy;
        bool get active;
        AuthError get error;
        Future<bool> login(String email, String password);
}
```

Obrázek 5.3: Třída ILoginViewModel

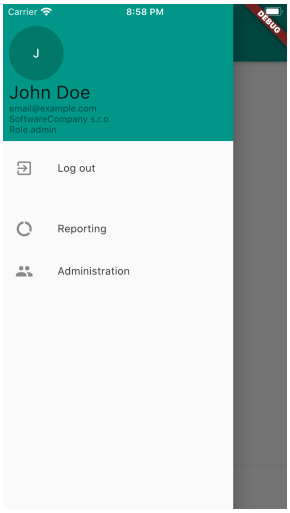


Obrázek 5.4: Ukázka vzhledu obrazovky pro přihlašovaní

HomeDrawer je boční menu hlavní obrazovky aplikace. Které se používá pro navigaci v aplikaci, zobrazuje informace o přihlášení uživateli a umožňuje odhlášení. Viz obr. 5.5.

5.3 Správa uživatelů

Základním kamenem správy uživatelů v této aplikaci je systém rolí. Na základě rolí uživatelům jsou přiřazené přístupová práva k prostředkům. Nejvyšší role je ”administrátor”, toto je jediná role, která umožňuje přístup k prostředkům, které patří jiné firmě. Uživatele, které mají menší roli než administrátor mají přístup pouze k údajům svoje firmy. Administrátor může vytvářet a upravovat firmy, uživatele, zařízení, skupiny zařízení, nastavovat dashboardy a přidávat grafy. Druhá role podle dostupnosti práv je ”energy manager”. Uživatele z této roli jsou administrátory ale mají práva pouze k prostředkům v rámci svoje firmy. Můžou přidávat a mazat uživatele s menšími rolí. Můžou registrovat zařízení, seskupovat zařízení do skupin, nastavovat dashboardy a přidávat do dashboardů grafy. Nejmenší roli je ”member”. Uživatele s této roli nemají pří-



Obrázek 5.5: Ukázka vzhledu bočního menu

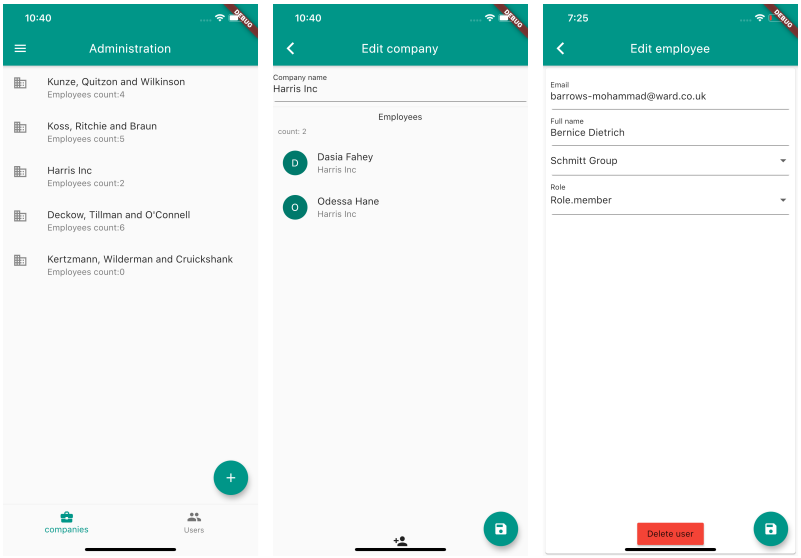
stup správě jiných uživatelů. Můžou pouze spravovat zařízení a nastavovat dashboardy.

Modul pro zprávu uživatelů je rozdělen do dvou sekci: správa firem a správa uživatelů. Sekce pro správu firem slouží je dostupný pouze uživatelům s roli ”administrátor”. Táto sekce je určená pro administrování zákaznických firem. Základní obrazovka této sekci je seznam již zaregistrovaných firem. Dále, pomocí tlačítka ”přidat” administrátor může zaregistrovat další firmu. Při rozkřiknutí firmy ze seznamu administrátor bude navigován na obrazovku s detaily firmy kde je zobrazen seznam uživatelů z této firmy. Na této obrazovce administrátor muže vybranou firmu přejmenovat nebo upravit údaje nebo roli uživatele z této firmy. Po jakékoliv úpravě v pravém dolním rohu obrazovky se objeví tlačítko uložit, po stisknutí kterého bude proveden pokus o uložení dat na server. Pro použití druhé sekci, sekci správy uživatelů, je nutně mít nejméně roli ”energy manager”. Základní obrazovkou této sekci je seznam uživatelů. Administrátor v tomto seznamu vidí veškeré uživatele ze všech firem, ale ”energy manager” vidí jenom uživatele z jeho firmy. Po stisknutí položky ze seznamů, energy manager bude navigován na obrazovku s detaily uživatele, kde může upravit údaje uživatele nebo smazat ho. Při stisknutí tlačítka přidat objeví se dialog s formulářem pro přidání nového uživatele.

5.4 Reporting

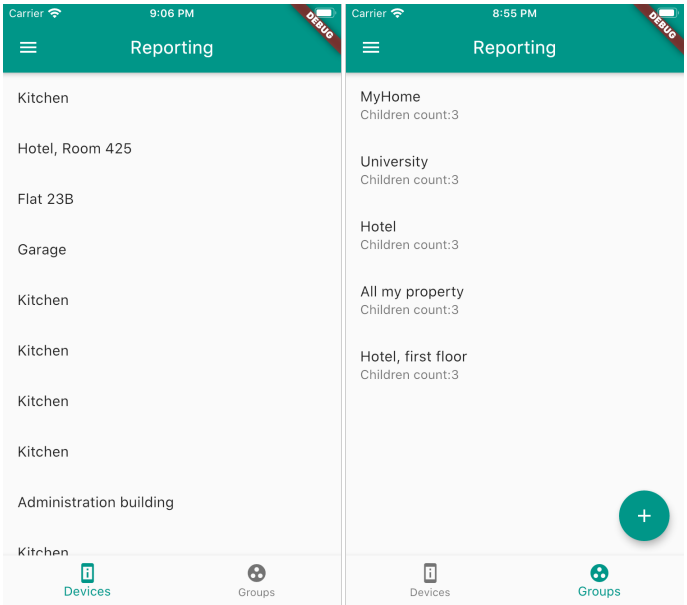
Modul reporting je rozdělen do dvou sekci: správa zařízení a správa skupin zařízení. Navigace mezi moduly je možná z dolního navigačního panelu nebo z bočního menu .

První obrazovkou sekci pro správu zařízení je seznam zařízení (viz obr. 5.4). Po výběru zařízení uživatel bude přesměrován na obrazovku s detaily zařízení kde se mu zobrazí dashboard zvoleného zařízení. V navigací detai-



Obrázek 5.6: Ukázka widgetů pro správu uživatelů

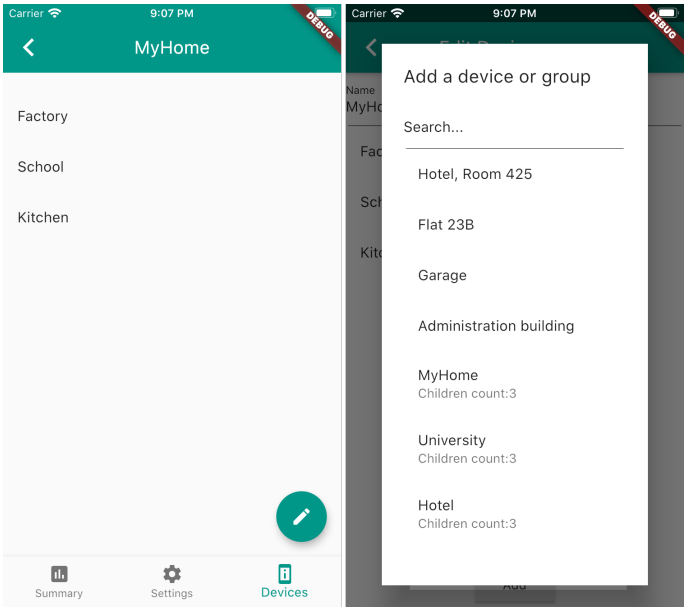
lu zařízení je možnost jít do nastavení zařízení, kde jsou zobrazené nastavené kritické hodnoty pro jednotlivou veličinu (viz obr. 5.4).



Obrázek 5.7: Seznam zařízení a seznam skupin

Stisknutím tlačítka ”přidat” v dolní části obrazovky uživatel muže přidat další kritickou hodnotu. Při nastavení kritické hodnoty uživatel musí zvolit veličinu, typ, hodnotu a typ časového intervalu. Jsou dva typy kritických hodnot: dolní limita a horní limita. Z důvodu toho, že měřených veličin muže být více, byla přidána možnost vyhledávaní mezi veličiny (viz 5.4). Po stisknutí tlačítka ”uložit” přidaná veličina se uloží. Při dalším vykreslení grafu, vzorky,

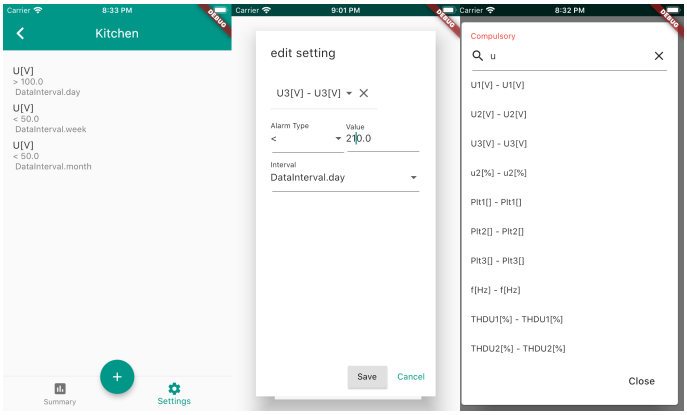
které překročily nastavené limity na sledovaném časovém intervalu, budou barevně znázorněné (viz obr. 5.12). Kritické hodnoty lze nastavit i pro skupinu zařízení.



Obrázek 5.8: Zařízení v skupině a přidání dalšího zařízení do skupiny

Sekce pro správu skupin začíná seznamem skupin (viz obr.5.4). Na této obrazovce uživatel muže si zvolit zařízení pro další práci nebo vytvořit novou skupinu. Do skupin zařízení je možné přidávat jak jednotlivá zařízení tak i již existující skupiny zařízení. Po volbě zařízení uživatel bude přemístěn na obrazovky s detaily zařízení. V dolní části je navigační panel s navigaci na se seznam zařízení ve skupině a na nastavení kritických hodnot. Na obrazovce se seznamem zařízení uživatel může přidávat nebo další zařízení nebo odebrat zařízení, které v této skupině už mít nepotřebuje.

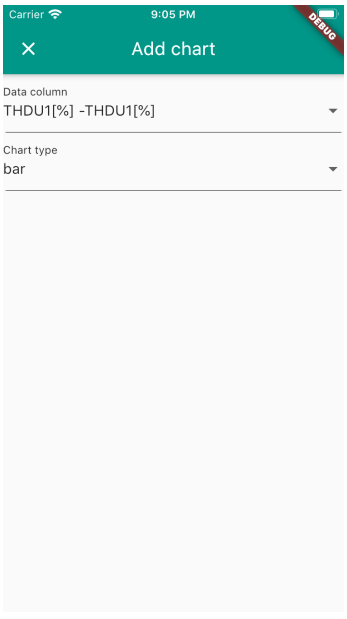
Základní záložkou této obrazovky je dashboard zvolené skupiny (viz obr. 5.12). Na dashboard skupiny lze přidávat grafy pro porovnávaní hodnot mezi zařízeními nebo skupinami v sledované skupině. Pro každý graf v dashboardu skupiny lze zvlášť zvolit jednotlivé zařízení, které na tomto grafu by uživatel chtěl sledovat. Dále se zvolí sledována veličina. Při nastavení veličiny k výběru, jsou dostupné pouze veličiny, které jsou měřené aspoň jedním zařízením této skupiny. V případě, že nějaké zařízení zvolenou veličinu neměří, data z tohoto zařízení nebudou zobrazené. V horní částí skupinového grafu je seznam zobrazených na tomto zařízení a barva, kterou jsou data z těchto zařízení vykreslená. Stisknutím na název zařízení v tomto seznamu lze schovat data z tohoto zařízení.



Obrázek 5.9: Nastavení zařízení

5.4.1 Nastavení dashboardu

V horní části obrazovky s dashbordem je panel s nastavením sledovaného období (viz obr. 5.4.2). Výchozí období je vždycky dnešní den. Pří nastavení období je nutně zvolit interval. V základní verze aplikace jsou tří intervaly na výběr: den po hodinách, týden po dnech a rok po měsících. Po nastavení intervalu uživatel si muže zvolit datum pro sledování. Dashboard se aktualizuje po každé změně nastavení. Stisknutím tlačítka přidat lze přidat do dashboardu další graf (viz obr. 5.4.1). Po rozkliknutí grafu zvolený graf se zobrazí na celou obrazovku. V dolní části obrazovky s grafem je tlačítko pro navigací na úpravu grafu.

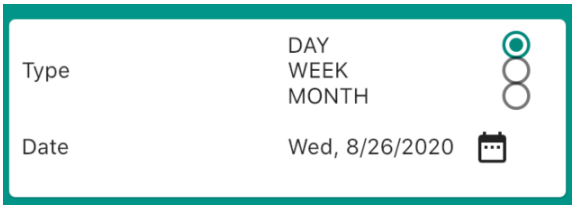


Obrázek 5.10: Přidání grafu

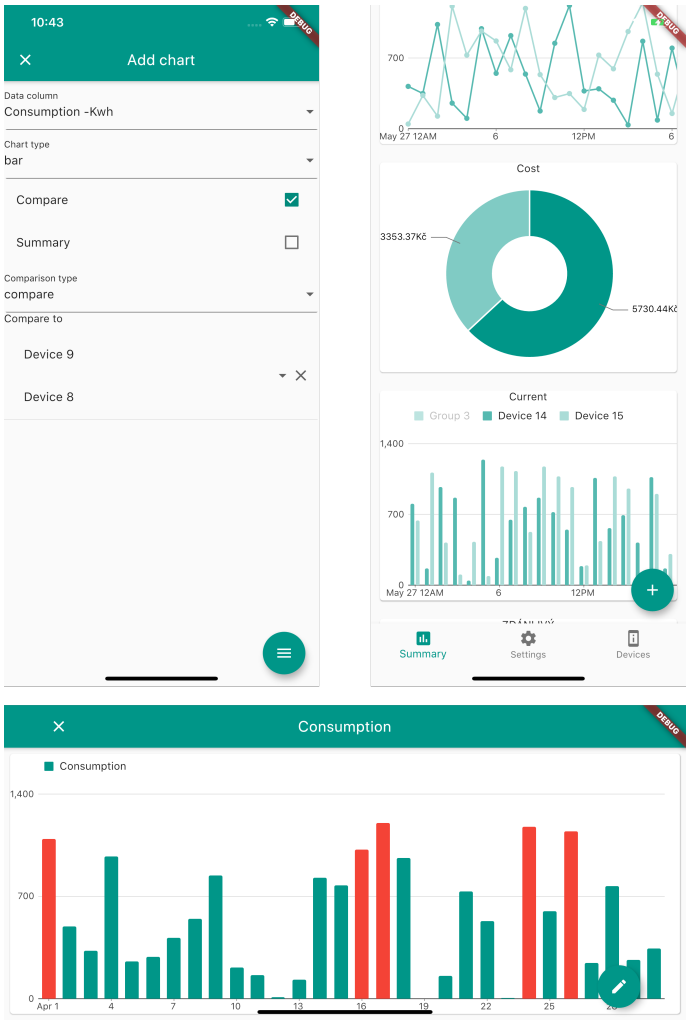
5.4.2 Vizualizace dat

Na vizualizaci dat byla použitá knihovna *charts_flutter*. Táto knihovna implementuje základní widgety pro vizualizace dat. V této aplikace byly použité sloupcové, lineární a koláčové grafy. Třídá *DeviceService* má funkci *fetchData*, která vrací data pro zobrazování na zvoleném grafu se zvoleným intervalem mezi měření. Vstupem této funkci je instance grafu, zvolený časový interval a počáteční datum vyhledávání. Z důvodu jednoduššího ovládání a vnímání grafů pro každý časový interval je zaveden určitý počet měření na jednom grafu. Pokud interval je jeden den, je potřeba vrátit vzorky za měsíc, pokud interval je měsíc, budou vracený hodnoty za rok. Třída ChartViewModel zpřístupňuje funkci pro načítání dat prezentační vrstvě.

UI vrstva vizualiaci byla navržena s ohledem na škálovatelnost. Z důvodu toho, že ve Flutteru se většinou používá kompozice byla navržen widget ChartContainer, který umožňuje abstrakci nad grafy. V něm jsou popsané zobrazovací vlastnosti a funkcionalita přislušná všem grafům. Během vykreslování tento widget , dotáže na data a vykreslí příslušný graf. Po tom až data budou načtené, tento widget překonvertuje data do tvaru který je vyžadován konkretním grafem.



Obrázek 5.11: Volba období



Obrázek 5.12: Ukázka nastavení grafu a dashboaru

6 Dosažené výsledky

6.1 Správa uživatelů

Správa uživatelů implementuje následující scénáře:

- Před použitím aplikace se musí uživatel přihlásit emailem a heslem. Po úspěšném přihlášení se přihlašovací token a refresh token uloží do paměti zařízení. Pokud jsou validní, uživatel se nemusí přihlašovat znovu.
- Administrátor registruje novou firmu a nové uživatele.
- Manažer registruje nové uživatele do své firmy.
- Manažer upravuje roli uživatelů ze své firmy.
- Uživatelé nemohou provést určitou činnost, pokud k ní nemají dostatečná práva, přístup jim je odepřen.

Pro modelování pracovníků firem byla vytvořena třída Employee. Employee obsahuje email, jméno, identifikátor firmy, ke které patří, a roli. Model uživatele, User, rozšiřuje Employee o vlastnost password a používá se jenom pro autentizace a vytvoření uživatele. Pokud uživatel pro provedení určité činnosti nemá práva, komponenty uživatelského rozhraní pro danou činnost se nezobrazí.

6.2 Správa zařízení

Správa zařízení v této aplikaci realizuje tři následujících scénáře: manažer registruje novou skupinu zařízení, manažer přidává existující zařízení do existující skupiny, manažer může zařízení ze skupiny odebrat. Z pozorování vzorového csv souboru s daty ze zařízení je vidět, že datové hlavičky z různých zařízení se mohou lišit. Proto bylo rozhodnuto, že datová struktura, která modeluje zdroj dat, musí obsahovat informace o všech druzích dat a měrných jednotkách, kterými se měří jednotlivý druh. Třída, která je navržena pro modelování druhu dat, byla pojmenovaná *DataRow*. Model zdroje dat je abstraktní třída *MeasurementSource*, díky této abstrakci se určité části aplikace nebudou muset měnit při přidání dalších druhů zdroje dat. MeasurementSource také obsahuje identifikátor firmy, která toto zařízení vlastní. V původní verzi aplikace jsou dvě

implementace zdroje dat Device, a DeviceGroup. Model DeviceGroup vznikl pro plnění potřeby systematizace jednotlivých zařízení do skupin, pro práci s několika zařízeními jako s celkem.Vlastnost 'children' třídy DeviceGroup je seznam podřízených zařízení a skupin. Vlastnost dataRows v Device Group obsahuje množinu všech druhů dat z podřízených skupin zdrojů.

6.3 Reporting

Reporting realizuje následující scénáře:

- Manažer přidává do dashboardu pro zařízení, nebo skupiny zařízení, nový graf, který vizualizuje předem zvolený datový sloupec a druh grafu. K dispozici jsou dva druhy grafů pro zařízení: sloupcový a čárový. U skupiny zařízení je také k dispozici výsečový graf. Při práci se skupinou zařízení může manažer vybrat několik zařízení pro porovnávání. Jsou implementované tři druhy porovnávání dat: vykreslování každého vzorku dat na časové ose, vykreslování sumy dat na časové ose a vizualizace poměru dat. Na grafu mají data z různých zdrojů odlišnou barvu. Součástí grafu je legenda s mapováním názvu zdroje na barvu. Po nastavení si manažer může zobrazit náhled vytvořeného grafu. Po kontrole náhledu může být graf uložen, poté je přidán do dashboardu a zpřístupněn pro všechny uživatele, kteří k tomuto dashboardu mají přístup.
- Poté, co si uživatel vybere časový úsek, za který chce vizualizovat data, zobrazí se dashboard s grafy pro zvolený časový úsek.
- Správce nastaví kritické hodnoty pro toto zařízení nebo skupinu. Tato nastavení slouží k upozornění uživatelů společnosti, ke které zařízení patří, když dojde k nahrávání na server kritických hodnot. Při zobrazení grafu se kritické hodnoty zvýrazní jinou barvou.

7 Závěr

V dnešní době roste snaha lidstva o sníženi negativního vlivu na životní prostředí. Industrializace, následný technologický rozvoj a růst populace se stáli mocnějšími důvody, vedoucími k rychlejšímu spotřebování neobnovitelných zdrojů energie a znečištění ekologie. Technologický pokrok, který ještě nedávno přispíval k distancování lidstva od přírody, může být využit pro dosažení harmonie s ní. V rozvinutých zemích aktivně zavádějí používání elektrických motorů a zelené energie a další proekologické míry. Ale spotřeba všech druhů energie stále neustále roste. Téma této práce je aktuální, protože výsledná mobilní aplikace je navržená s cílem pomoci uživatelům optimalizovat spotřebu elektronické energie.

Teoretická část této práce seznamuje čtenáře s problematikou hospodaření s energií, charakteristikami kvality elektronické energie a současnými existujícími řešeními v tomto oboru. Jsou stručně popsáné aktuální statní normy, které se týkají kontextu této práce. Výsledkem, této části práce je stanovení požadavků na funkčnosti navržené aplikace (viz 2.8). Při průzkumu existujících řešení se ukázalo, že výsledná aplikace může vyniknout mezi existujícími řešeními použitím modernějších technologií pro zrychlení vývoje. Rychlejší vývoj může mít pozitivní vliv na dostupnost výsledné aplikace.

Na začátku druhé rešeršní části práce jsou stručně popsáné mobilní platformy a porovnávané přístupy při vývoji mobilních aplikací. Pro snížení náročnosti vývoje přednost byla dána víceplatformním řešením. Po průzkumu technologií byl zvolen framework Flutter, vynikající mezi konkurenty výrazným zvyšováním efektivity a jednoduchosti vývoje. Byl prostudován standart zabezpečení komunikace mezi klientem a serverem pro použití v přihlašovacím mechanismu aplikace.

Výsledkem praktické části této práce je mobilní aplikace pro platformy Android a IOS. Aplikace je schopna načítat a zobrazovat vybraná historická data o spotřebě a kvalitativních charakteristikách elektronické energie. Aby byla aplikace pro uživatele přívětivá, bylo pomocí analýzy konkurenčních řešení rozhodnuto, nabídnout uživatelům možnost konfigurace dashboardu dle vlastních potřeb. Pro zobrazení dat se používají lineární, sloupcové a výsečové grafy. Pro použití aplikace je nutné, aby se uživatel přihlásil uživatelským jménem a heslem. Po autentizaci uživatele bude na základě jeho role nabídnuta odlišná funkcionalita. V základní verzi jsou tři uživatelské role inspirované normou ISO 50001. Alarmy jsou realizovány tak, že uživatel může definovat podmínku na alarm. Když dojde ke spoušti alarmu, kritické hodnoty budou

znázorněné na grafu . V nastavení každého zařízení lze definovat alarm nastavením kritické hodnoty jedné nebo několika veličin. Z demonstračních důvodů je základní verze aplikace v angličtině a veškerá data jsou reálné. Data jsou načítána ze lokální databáze na serveru.

Navzdory tomu, že poměrně dost podobných řešení již existuje ,tato aplikace stejně vyniká mezi svými předchůdci použitím modernějších technologií v oboru vývoje mobilních aplikací. Tato aplikace byla vytvořená za použití frameworku Flutter, umožňujícího rychlejší vývoj, což znamená, že vývoj je dostupnější jak pro organizaci, která se pokusí o komerční vývoj tohoto produktu tak i pro koncové uživatele .

V plánu dalšího rozvoje tohoto tématu jsou rozšíření funkcionality aplikace o možnost importu dat přímo ze zařízení prostřednictvím Bluetooth nebo USB. Pro některé uživatelé by mohla být přivětivá možnosti přidání dat rozpoznáváním textu z fotografie. Serverová část aplikace je vytvořena s předpokladem, že lepší přehlednost správy zařízení bude dosažená zobrazením zařízení na mapě např. Google Map. V serverové části by bylo možné použít algoritmy pro práci s velkým množstvím dat pro predikce budoucích hodnot nebo pro prevenci kritických stavů. Pro použití této aplikace pro sledování hodnot v reálném čase, je nutné, doplnit jí o zpracování datového toku ze zařízení použitím RabbitMQ pro snížení vypočtení náročnosti při zpracování dat z velkého počtu odběrových míst.

Seznam obrázků

Seznam tabulek

Literatura

[1] Software ENVIS Uživatelská příručka pro podporované měřící přístroje. Verze 1.1. 2013, doi:http://www.kmb.cz/index.php/cs/component/phocadownload/category/2-software?download=197:envis-v11-uzivatelska-prirucka.

[2] JWT. 2014.
URL https://jwt.io/introduction/

[3] ReactiveX. 2016.
URL http://reactivex.io/intro.html

[4] Apache Kafka. 2017.
URL https://kafka.apache.org/documentation/{#}introduction

[5] Developer. 2018.
URL https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html{#}//apple{_}ref/doc/uid/TP40008194-CH8-SW1

[6] Flutter. 2018.
URL https://flutter.dev/docs/development/data-and-backend/state-mgmt/options

[7] Futter. 2018.
URL https://flutter.dev/docs/resources/technical-overview

[8] Systémy energy managmentu Regulátory jalového výkonu Analyzátory kvality energie. 2018.

[9] BlueKite Apps. 2019.
URL https://www.bluekiteapps.com/blog/types-of-mobile-applications-native-hybrid-and-mobile-apps

[10] Systémy managementu hospodaření s energií - Požadavky s návodem k použití. 2019.
URL https://csnonline.agentura-cas.cz/Detailnormy.aspx?k=507051

[11] Developers. 2020.
URL https://developer.android.com/guide/platform

[12] Firebase. 2020.
URL https://firebase.google.com/docs/cloud-messaging/fcm-architecture

[13] RabbitMQ. 2020.
URL https://www.rabbitmq.com/documentation.html

[14] StatCounter. 2020.
URL https://gs.statcounter.com/os-market-share/mobile/worldwide

[15] Borza, J.: EnergyElephant. 2019.
URL https://energyelephant.com/benefits

[16] Brian Frank; John Petze: SkyFoundry. 2019.
URL https://skyfoundry.com/product

[17] Cleveroad: The Costs of Software Development: Challenges and Ready-Made Estimations. 2018.
URL https://www.cleveroad.com/blog/software-development-costs

[18] Google LLC: The Go Project. 2020.
URL https://golang.org/doc/

[19] normalizační institut, Č.: *Elektromagnetická kompatibilita (EMC) - Část 4-30: Zkušební a měřicí technika - Metody měření kvality energie*. Praha, t⊠et⊠ vyd⊠n⊠, 2017, 68 s.

[20] Laravel LLC: Laravel. 2020.
URL https://laravel.com/docs/7.x

[21] Martin, R.: *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. London: Prentice Hall, 2017, ISBN 0134494326, 432 s.

[22] OpenJS Foundation: Node.js. 2020.
URL https://nodejs.org/en/docs/

[23] Parsons, J.: Tracking Down Memory Leaks in Node.js – A Node.JS Holiday Season. 2012.
URL https://hacks.mozilla.org/2012/11/tracking-down-memory-leaks-in-node-js-a-node-js-holiday-season/

[24] PROVOZOVATELÉ DISTRIBUNÍCH SOUSTAV: PRAVIDLA PROVOZOVÁNÍ DISTRIBUČNÍCH SOUSTAV PŘÍLOHA 3 KVALITA ELEKTINY V DISTRIBUČNÍ SOUSTAV, ZPŮSOBY JEJÍHO ZJIŠŤOVÁNÍ A HODNOCENÍ, 2006.
URL https://www.cezdistribuce.cz/edee/content/file-other/distribuce/energeticka-legislativa/ppds3-priloha.pdf

[25] RIGAU, X.: Novoda. 2018.
URL https://blog.novoda.com/introduction-to-redux-in-flutter/

[26] Ruzzelli, A.: Wattics. 2019.
URL https://www.wattics.com/dashboard/

[27] Suri, S.: Medium. 2018.
URL https://medium.com/flutterpub/architecting-your-flutter-project-bd04e1

[28] Symfony SAS: Symfony. 2020.
URL https://symfony.com/doc/current/reference/index.html

[29] Testguy: testguy. 2019.
URL https://testguy.net/content/361-Power-Quality-Analysis-Basic-Theory-ar
fbclid=IwAR1NM5YbYrx2LKCobIQlh36anEvUkPAwbBRMAOIajG-KB6wBb24RWWMCuNg

[30] Winget, J.: EchoinnovateIT. 2020.
URL https://echoinnovateit.com/ios-platform-overview/