TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Metody zpracování a vizualizace záznamů z regulátorů účiníku na tenkých klientech

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Jan Moravec**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2019

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Processing and visualisation of archives from power factor controllers on thin clients

Master thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Jan Moravec**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Metody zpracování a vizualizace záznamů z regulátorů účiníku na tenkých klientech

Jméno a příjmení: **Bc. Jan Moravec**
Osobní číslo: M17000132
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2018/2019**

Zásady pro vypracování:

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

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	40–50 stran
Forma zpracování práce:	tištěná/elektronická

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]. 1.2. Dostupné z: http://www.kmb.cz/

[3] DEL LA TORRE, Adriana Escobar; CHEON, Yoonsik. Impacts of Java Language Features On the Memory Performance of Android Apps. 2017.

[4] SANTOS, Arnold N.; MACABUHAY, Mary Anne A.; DE LEON, Jeferson N. Smart Household Socket with Power Monitoring & Control Using Android Application. In: 2017 9th IEEE-GCC Conference and Exhibition (GCCCE). IEEE, 2017. p. 1-9.

[5] BARNES, Vanessa; COLLINS, Thomas K.; MILLS, Godfrey A. Design and Implementation of Home Energy and Power Management and Control System. In: 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA. 2017. p. 241-244

Vedoucí práce:	Ing. Jan Kraus, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	10. října 2018
Předpokládaný termín odevzdání:	30. dubna 2019

L. S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci 10. října 2018

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

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

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že texty tištěné verze práce a elektronické verze práce vložené do IS STAG se shodují.

30. 4. 2019 Bc. Jan Moravec

Abstrakt

Tato práce popisuje vývoj mobilní aplikace určené pro operační systém Android, která zobrazuje archivní data z regulátorů účiníku. V úvodní části práce je stručně vysvětlena problematika regulace účiníku a fungování aplikace v rámci širší perspektivy. Další částí práce je rešerše souvisejících softwarových řešení a také popis možností vývoje mobilních aplikací. Dále práce popisuje nástroje použité pro vývoj aplikace, mezi které patří vybrané třídy z Android API a další využité knihovny. Následuje praktická část popisující jednotlivé vrstvy aplikace. Aplikace je napsána v programovacím jazyce Java s využitím architektury MVVM a data bindingu. Komunikace aplikace s webovou službou je realizována pomocí HTTP a knihovny Retrofit spolu se serializační knihovnou Gson. Autentizace uživatelů je řešena použitím JSON web tokenu. V práci je také popsána problematika perzistence a obnovování tokenu na mobilních zařízeních. Mezi hlavní funkce aplikace patří porovnávání a vyhodnocování naměřených archivních dat, sledování průběhů veličin, kontrola aktivity zařízení a prezentace alarmů. Součástí aplikace je také administrativní část pro správu skupin uživatelů a zařízení, nastavení preferencí a grafické rozhraní pro zpětnou vazbu uživatelů.

Klíčová slova

vývoj mobilních aplikací, Android, vizualizace dat, regulace účiníku

Abstract

This thesis describes the development of a mobile application designed for the Android operating system, which displays archive data from power factor controllers. In the introductory part of the thesis there is the issue of power factor regulation and application functioning within a broader perspective briefly explained. The next part of the thesis is a search of related software solutions and also a description of the possibilities of mobile application development. The thesis also describes tools used for application development, which include selected classes from Android API and other used libraries. It is followed by a practical part describing the individual layers of the application. The application is written in Java programming language, using MVVM architecture and data binding. The application communication with a web service is realized by HTTP and Retrofit libraries together with Gson serialization library. User authentication is handled by using a JSON web token. The issue of persistence and token renewal on mobile devices is also described. Between the main functions of the application belong comparing and evaluating of measured archive data, monitoring of variable's development, device activity control and alarm presentation. The application also includes an administrative section for managing of the users and device groups, preference settings and a graphical user feedback interface.

Keywords

mobile application development, Android, data visualization, power factor correction

Obsah

Seznam ilustrací

Seznam zkratek

API	Application Programming Interface, rozhraní pro vývoj aplikací
CRUD	Create, Read, Update, Delete, čtyři základní operace nad daty
EMS	Energy Management Software, software pro správu energie
FTP	File Transfer Protocol, protokol pro přenos souborů
HMAC	Keyed-hash Message Authentication Code, typ autentizačního kódování
HTTP	Hypertext Transfer Protocol, komunikační protokol
IoT	Internet of Things, internet věcí (síť autonomních vestavěných zařízení)
JSON	JavaScript Object Notation, formát popisu dat
JWT	JSON Web Token, standard přístupových tokenů
LCD	Liquid crystal display, displej z tekutých krystalů
LED	Light-Emitting Diode, elektronická součástka vyřazující světlo
MVC	Model-View-Controller, softwarový architektonický vzor
MVVM	Model-View-ViewModel, softwarový architektonický vzor
OS	Operating system, operační systém
RAM	Random access memory, paměť s libovolným přístupem
REST	Representational state transfer, architektura webové služby
RS-485	Standard sériové komunikace
RTC	Real-time clock, hodiny reálného času
SDK	Software Development Kit, sada vývojových nástrojů
SHA	Secure Hash Algorithm, typ hash funkce
URI	Uniform Resource Identifier, jednotný identifikátor zdroje
URL	Uniform Resource Locator, řetězec určující umístění zdroje
VPN	Virtual Private Network, virtuální privátní síť
XAML	Extensible Application Markup Language, značkovací jazyk
XML	Extensible Markup Language, značkovací jazyk

Úvod

Cílem práce je vytvořit aplikaci určenou pro mobilní platformu, která bude zpracovávat a vizualizovat archivní záznamy z regulátorů účiníku. Přesněji se jedná o regulátory od filmy KMB systems. Tyto regulátory často bývají vybaveny displeji, na kterých jsou zobrazovány aktuální veličiny regulátoru. Aktuální hodnoty lze také sledovat pomocí desktopové aplikace Envis Daq nebo mobilní aplikace Envis M popisující práce [1].

Pro zpětné sledování a analýzu veličin je regulátor vybaven pamětí, do které ukládá své naměřené hodnoty. Tyto archivní hodnoty lze ze zařízení stáhnout a poté zkoumat v aplikaci Envis, která je určena pouze pro PC. Mobilní aplikace Envis M archivní data zobrazovat nedokáže. To je důvod, proč vznikla tato práce. Doposud neexistovalo softwarové řešení, které by fungovalo na mobilních zařízeních. S dnešním rozšířením chytrých mobilních telefonů a tabletů je logické, že vznikla poptávka po aplikaci pro tato zařízení. [2]

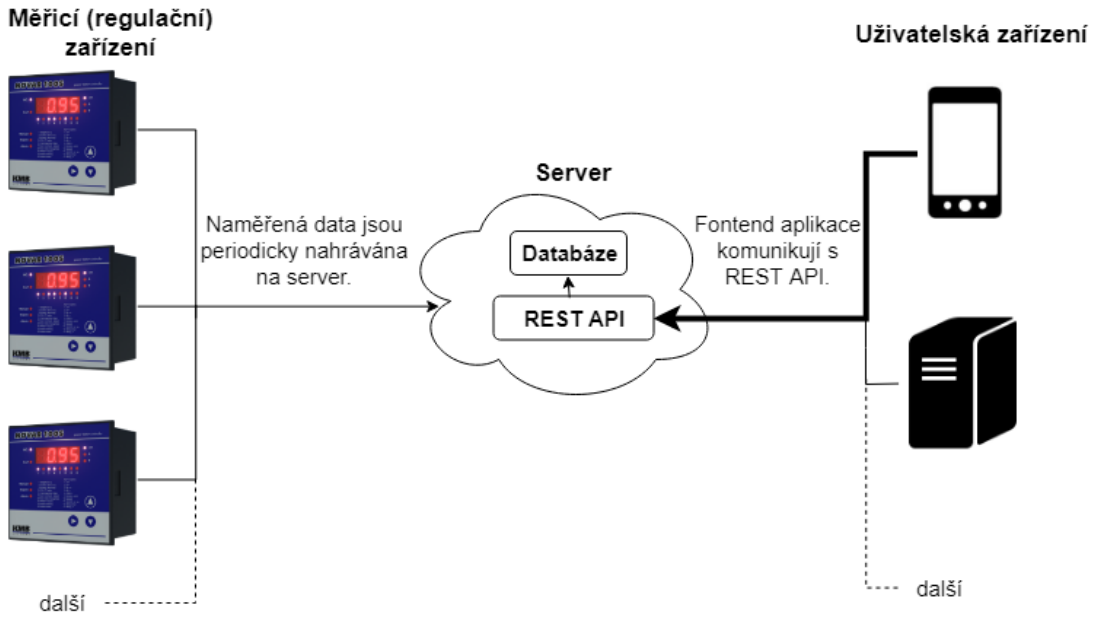
Výsledná mobilní aplikace by měla nabízet podobné funkce jako aplikace Envis, ale měla by být přizpůsobena mobilním zařízením. Rozdíl oproti předchozím řešením je, že aplikace nebude komunikovat přímo s regulátory, ale se serverem s webovou službou, kterou vyvíjel Ing. Tomáš Bedrník současně s touto prací. Jedním z možných postupů řešení bylo rozšířit zmíněnou aplikaci Envis M. Nakonec byla dána přednost vývoji nové aplikace, a to z toho důvodu, že rozšířená aplikace by byla příliš složitá a nepřehledná. Podrobnější zdůvodnění je popsané v následující kapitole.

1 Podrobnější popis problému

Tato práce tematicky navazuje na práci *Zpracování a vizualizace dat analyzátorů v OS Android* [1], praktickým výsledkem uvedené práce je aplikace Envis M sloužící především pro sledování aktuálních veličin a vzdálenou konfiguraci elektroměrů, regulátorů účiníku a analyzátorů elektrické energie. Aplikace Envis M poskytuje také stažení archivních dat do paměti mobilního zařízení, ale neumožňuje zobrazení těchto dat. Tuto chybějící funkci právě nabízí aplikace, kterou popisuje tato práce.

Aplikace Envis M komunikuje přímo s měřicími zařízeními pomocí rozhraní Ethernet a bezdrátové technologie Wi-Fi. Měřicí zařízení z bezpečnostních důvodů většinou bývají součástí privátní sítě oddělené od veřejného internetu. K tomu, aby aplikace mohla komunikovat s měřicími zařízeními je tedy nutné, aby bylo mobilní zařízení připojené ve stejné počítačové síti. Přístup aplikace z internetu k měřicím zařízením není jednoduše řešitelný, pokud není využito VPN.

To je jeden z důvodů, proč aplikace, kterou popisuje tato práce, využívá server, na který jsou nahrávána data z měřicích zařízení. Server potom tyto data poskytuje pouze ověřeným uživatelům. Globální schéma komunikace je zobrazeno na ilustraci 1. Další výhodou použití serveru je, že data získaná z měřicích zařízení lze předzpracovat, provádět na nich operace a výsledky těchto operací na serveru ukládat, což by bylo nepraktické provádět na mobilních zařízeních. Výhodou může také být uchovávání archivních dat na serveru. Měřicí zařízení mají totiž oproti úložišti na serveru malý paměťový prostor a v případě nedostatku paměti přepisují nejstarší naměřené záznamy. [3]



Ilustrace 1: Globální schéma komunikace

Měřicí zařízení periodicky posílají aktuálně naměřené hodnoty na server. Tyto hodnoty jsou na serveru agregovány na minutové záznamy a ukládány do databáze. Další možností je odečíst archivní data z měřicího zařízení a poté je nahrát na server. Na serveru také běží webová služba s REST rozhraním. Toto rozhraní zapouzdřuje databázi na serveru a jednotně definuje metody, jakými se přistupuje k datům. Mobilní aplikace potom přistupuje k datům právě pomocí tohoto rozhraní spolu s dalšími klientskými aplikacemi (například webové aplikace).

Komunikace tedy probíhá odlišným způsobem než v aplikaci Envis M. Jak už bylo uvedeno, tak aplikace bude zobrazovat pouze archivní data. Jedná se tedy o zcela odlišné funkce a odlišný způsob komunikace. Z toho důvodu by bylo komplikované zakomponovat tyto vlastnosti do původní aplikace. Proto byla vytvořena nová jednodušší aplikace.

1.1 Regulátor účiníku

Regulátor účiníku je elektrický přístroj, jehož cílem je v obvodech se střídavým elektrickým proudem jednofázové či trojfázové soustavy udržovat účiník na hodnotě blížící se k 1. Více o regulaci účiníku a souvisejících veličinách je napsáno v příloze I.

Regulátory lze rozdělit do několika skupin podle kritérií jako množství vstupů, výstupů, rychlosti reakce na změnu, proudové citlivosti nebo počtu fází, s kterými umí pracovat. Tato práce se omezuje pouze na regulátory od společnosti KMB systems, protože se záznamy těchto regulátorů pracuje výsledná aplikace popsaná v praktické části textu. [4]

Kromě hlavní poskytované funkce regulace účiníku tyto regulátory také poskytují funkci univerzálního měřicího přístroje. Mezi měřené veličiny patří například fázové, sdružené napětí, fázový proudu, jednotlivé harmonické složky napětí, proudů a jejich celkové zkreslení, zdánlivý, činný, jalový a deformační výkon, účiník a další. Pro archivaci záznamů těchto veličin je většina regulátorů vybavena vlastní pamětí, do které záznamy ukládá. Pro přiřazení jednotlivých záznamu k časovým okamžikům je také regulátor vybaven obvodem reálného času (RTC). Součástí některých typů regulátorů je také vestavěný elektroměr, který zaznamenává činnou i jalovou energii v uživatelsky definovaných tarifech. [3]

Další funkcí regulátoru jsou konfigurovatelné alarmy. Alarm je spuštěn v případě, že nastane nějaký neobvyklý jev, což může být například přerušení napětí nebo extrémní hodnoty jedné z měřených veličin. Aktivní alarm může být signalizován několika způsoby, na LCD displeji či LED diodě regulátoru, vypsáním do logu zařízení nebo například sepnutím definovaného výstupu. Hodnoty alarmů spolu s časovou informací jsou ukládány do paměti zařízení. Komunikace s regulátorem je zajištěna pomocí rozhraní Ethernet, USB nebo RS-485. Podrobnější informace jsou uvedeny v manuálu konkrétního modelu [3]. [4]

2 Existující řešení

Funkce, které má mít výsledná aplikace, bývají zahrnuty v tzv. energy management softwarech (EMS). Tento druh aplikací je primárně zaměřen na sledování spotřeby elektrické energie, vody či plynu a hledání úspor. Některé aplikace, ale poskytují i sledování kvality elektrické energie. Níže popsané aplikace byly vybrané především podle existence dokumentace, míry souvislosti s touto prací a podle množství dostupných funkcí. [5]

2.1 Wattics

Wattics je komplexní webová platforma s cloud úložištěm pro správu energie. Nahrání dat do webové aplikace je možné manuálně pomocí CSV souborů, FTP nebo v definovaném JSON formátu pomocí REST API. Pro širokou škálu zařízení (měřičů, IoT senzorů) lze také nastavit automatické nahrávání dat. Data lze přiřadit k různým zdrojům (budova, patro) a ty lze libovolně větvit ve stromové struktuře. Nahrát je možné i data z univerzálních analyzátorů elektrické energie a sledovat veličiny jako spotřebovaná elektrická energie, napětí, proud, činný výkon, jalový výkon nebo účiník.

Vizualizovaná data v podobě grafů je možné exportovat jako obrázek a uložit na používaném zařízení. Aplikace poskytuje notifikace o uplynulých událostech a umožňuje nastavení vlastních alarmů, které jsou zobrazovány v aplikaci nebo mohou být oznamovány prostřednictvím SMS a emailu. Dále aplikace umožňuje porovnání veličin jednotlivých zdrojů, správu uživatelů a přidělování práv, generování reportů, predikci vývoje veličin se zobrazením notifikací v případu anomálie a definici tarifů, pomocí kterých je vyhodnocena cena energie. Existuje pouze responzivní webová aplikace, mobilní aplikaci platforma neposkytuje. [6]

2.2 Entronix

Společnost Entronix a jejich aplikace Entronix EMP nabízí podobnou řadu funkcí jako platforma Wattics včetně sledování kvality elektrické energie a účiníku. Nasazení aplikace je možné na již existující automatizační systém.

Nahrávání dat z měřičů na jejich server je realizováno pomocí jejich vlastního hardwaru, který podporuje komunikační protokoly jako Modbus, BACnet nebo standard oBIX. Též se jedná o cloudové řešení s responzivní webovou aplikací dostupnou na všech zařízeních s prohlížečem. Sdílení dat řeší pomocí reportů, URL odkazů a QR kódů, které lze prohlížet na mobilních zařízeních pomocí aplikace Entronix-View sloužící pouze k tomu účelu. [7]

2.3 Eniscope

Společnost BEST nabízí řešení Eniscope. Oproti předchozím řešením, která byla víceméně nezávislá na použitých měřicích zařízeních, tak zde je nutné použít jejich měřicí zařízení, které zároveň slouží k odesílání dat do cloudu. Řešení nabízí podobně jako předchozí aplikace aktuální a zpětné sledování spotřeby a kvality energie včetně účiníku a také kontrolu a oznamování alarmů. Na rozdíl od předchozích řešení nabízí aplikaci určenou přímo pro mobilní zařízení, která je oproti webové aplikace zjednodušená. Mobilní aplikace zobrazuje aktuální hodnoty proudu, napětí, účiníku a spotřeby v rámci dne. Dále aplikace umožňuje prohlížení průběhů veličin prostřednictvím grafů v rámci dne, týdne a měsíce. Aplikace také poskytuje přehled a notifikace alarmů. [8]

2.4 Power Monitors

Firma Power Monitors se zabývá vývojem analyzátorů kvality elektrické energie a poskytuje k těmto zařízením vlastní software. V tomto případě už se nejedná o EMS, ale řešení přímo určené pro sledování kvality elektrické energie, čímž se jejich aplikace více shodují s tématem této práce. Firma nabízí 3 aplikace.

Aplikace ProVision je určena pro PC a umožňuje přímo komunikovat s analyzátory prostřednictvím rozhraní Ethernet, USB nebo bezdrátové technologie. Pomocí aplikace lze konfigurovat zařízení, sledovat aktuální veličiny, generovat reporty, stahovat archivní data a následně je analyzovat.

Cloudová webová aplikace Canvass je podobná již zmíněným předešlým řešením. Není tolik rozsáhlá, zaměřuje se pouze na kvalitu elektrické energie,

umožňuje prohlížení zařízení pomocí mapy, sledování aktuálních dat, srovnání veličin mezi jednotlivými zařízeními, správu uživatelů a nastavení alarmů.

Poslední nejrelevantnější aplikace iPower je určená pro Apple mobilní zařízení. Pomocí této aplikace je možné se spojit se zařízeními, která jsou vybaveny Wi-Fi modulem nebo celulárním modemem pro připojení do mobilní sítě. Aplikace umožňuje sledovat aktuální průběhy veličin formou grafu pro jedno vybrané zařízení. Možné je sledovat jednotlivé fáze napětí a proudu, jejich harmonické složky, celkové harmonické zkreslení, jednotlivé výkony a účiník. Průběh veličin lze také uložit a následně vložit do aplikace Canvass, kde je možné provádět další analýzu. [9]

2.5 Dosažené poznatky

Zkoumané aplikace se zabývají především vizualizací a optimalizací spotřeby energie. Z technologického hlediska to jsou většinou responzivní webové aplikace s cloud úložištěm. Některá řešení mají i aplikaci určenou přímo pro mobilní platformu. Desktopové aplikace jsou obecně komplexní, mobilní aplikace jsou zjednodušené poskytující základní přehled. Aplikací typu EMS je na trhu velké množství, ale většinou neposkytují detailnější informace o kvalitě elektrické energie či regulaci jalového výkonu. Z tohoto pohledu je mobilní aplikace primárně určená pro sledování archivní dat z regulátoru jalového výkonu originálním řešením. Provedená rešerše sloužila k základnímu seznámení s dostupnými aplikacemi na trhu a přispěla k návrhu výsledné aplikace.

3 Možnosti vývoje mobilních aplikací

Při vývoji aplikace určené pro mobilní zařízení je zásadní otázkou, pro jaký operační systém či systémy bude aplikace cílena. V současnosti je nejrozšířenější operační systém Android, který zastupuje 75,33 % celkového světového podílu. Na druhém místě je iOS s 22,4% zastoupením. Pouhých 2,57 % tvoří ostatní mobilní operační systémy. [10]

S výběrem operačního systému také souvisí volba programovacího jazyku. Oficiálně podporované jazyky pro iOS jsou Objective-C a Swift [11], kdežto pro Android je to jazyk Java a Kotlin [12]. Kromě rozdílného aplikačního rozhraní je tedy ještě potřeba programovat aplikaci v odlišném jazyce, čímž je vývoj pro obě platformy ještě komplikovanější. Z toho důvodu vzniklo mnoho technologií, které se snaží tento problém vyřešit. Některé dokonce umožňují současný vývoj i pro desktopové operační systémy. [13]

3.1 Nativní aplikace

Jsou aplikace určené pro jednu konkrétní platformu. Nativní zde není myšleno hardwarově, ale platformě. Příkladem je aplikace napsaná v jazyku Java/Kotlin pro operační systém Android. Nativní aplikace obecně není možné spustit na jiné platformě, než pro kterou jsou určeny, což je značným omezením. Oproti hybridním a webovým aplikacím nabízí lepší využití výkonu mobilního zařízení a využití všech funkcí zařízení bez omezení. [13]

3.2 Webové aplikace

Webové aplikace fungují na mobilních zařízeních za předpokladu nainstalovaného internetového prohlížeče. Takové aplikace ale bez dalšího přizpůsobení nejsou příliš uživatelsky přívětivé kvůli odlišným rozměrům mobilních zařízení. Tento problém řeší responzivní vývoj, což je optimalizace vzhledu aplikace pro zařízení s různými rozměry, především pomocí CSS modulu Media Queries. Responzivní aplikace ale stále potřebuje ke svému chodu prohlížeč a přístup k aplikaci je umožněn pouze pomocí URL adresy. Dokonce i tento

problém lze vyřešit a to s tzv. progresivními webovými aplikacemi. Tyto aplikace nabízí některé funkce jako klasické nativní aplikace. Progresivní aplikace lze spouštět bez prohlížeče a dokáží omezeně fungovat i bez internetového připojení. K funkčnosti těchto vlastností progresivních aplikací je ale nutná podpora operačního systému. Přesto jsou progresivní webové aplikace na mobilních zařízeních značně omezeny oproti nativním aplikacím a nejsou tak výkonné. [13], [14]

3.3 Hybridní aplikace

Hybridní aplikace se více přibližují aplikacím nativním. Princip hybridních aplikací spočívá ve využití webových technologií, jako HTML, CSS a JavaScript, pro vytvoření jedné webové aplikace, ze které je následně pomocí frameworku vytvořena aplikace pro mobilní zařízení. Hybridní aplikaci potom lze nainstalovat na mobilní zařízení stejně jako nativní aplikace. Technologicky se moc neliší od progresivních webových aplikací. Hybridní aplikace totiž využívají zobrazovací komponentu WebView, která funguje jako webový prohlížeč a renderuje vzhled aplikace. Jednotlivé frameworky také generují kód pro využití nativních funkcí jako využití fotoaparátu, Bluetooth a dalších. V podstatě mohou tedy hybridní aplikace využívat stejnou množinou funkcí jako nativní aplikace, pokud má framework pro dané funkce vytvořenou mezivrstvu. Hybridní aplikace tedy nejsou tak omezené jako webové aplikace, ale mají stále podobnou výkonnost. Mezi frameworky pro tvorbu hybridních aplikací patří například Ionic a PhoneGap. [13], [15]

3.4 Nástroje umožňující překlad do nativního kódu

Existují další frameworky, které jsou odlišné od frameworků pro tvorbu hybridních aplikací. Rozdíl je v tom, že nepoužívají WebView, ale nativní zobrazovací komponenty určité platformy. Proto lze aplikace vytvořené pomocí těchto frameworků považovat za nativní, přestože nejsou vytvořené standardní cestou jako klasické nativní aplikace. Jednotlivé frameworky se od sebe značně liší, proto není možné je úplně zobecnit z hlediska nabízených nativních funkcí.

Výhoda takto vytvořených aplikací oproti hybridním je právě v tom, že nepoužívají WebView, čímž dosahují vyššího výkonu a jsou zbaveny omezení plynoucí z jeho využití. Zástupci těchto typů frameworků jsou například Xamarin, který umožňuje psaní kódu v jazyce C# [16], React Native a NativeScript pro jazyky JavaScript/TypeScript, Flutter pro jazyk Dart a Codename One pro jazyk Java.

3.5 Zvolený typ aplikace

Výsledná mobilní aplikace je pro zjednodušení cílena pouze pro operační systém Android. Aplikace byla vyvíjena klasickým nativním způsobem, a to z toho důvodu, že velkou část aplikace tvoří prezentační vrstva. Aplikace tak může plně využít nativní zobrazovací komponenty a funkce operačního systému.

4 Použité Android nástroje

Cílovou platformou, pro kterou je výsledná aplikace určena, je Android, proto jsou v této kapitole popsané komponenty a nástroje z Android API, které aplikace využívá.

4.1 Grafické komponenty

Aktivita je základní komponentou pro tvorbu Android aplikací. Aktivita definuje vzhled a prezentační logiku jedné obrazovky aplikace. Vzhled aktivity může být definován přímo v kódu aktivity nebo pomocí XML souboru. Každá aplikace má hlavní aktivitu, která je spuštěna jako první při startu aplikace. Programátorem definované aktivity jsou pak ve zdrojovém kódu třídy dědící od třídy Activity. [17]

Fragmenty jsou podobné aktivitám. Jedná se ale o menší celek, který existuje pouze v rámci aktivity. Fragmentů může být v jedné aktivitě několik. Vlastní fragment vznikne rozšířením třídy Fragment. Výhodou a důvodem vzniku fragmentu je jejich znovupoužitelnost a dynamičnost. Fragment je možné vložit do aktivity staticky v XML souboru aktivity nebo přidat dynamicky v kódu aktivity. Pomocí fragmentů lze například zobrazit odlišný vzhled pro zařízení s většími displeji (tablety) než pro zařízení s klasickými rozměry. [18]

DialogFragment je speciální druh fragmentu, který slouží pro tvorbu dialogů. Oproti klasickému fragmentu je ale dialog zobrazený v novém okně. Navíc obsahuje více metod než klasický Fragment, které jsou určené pro správu dialogu, jako například metoda pro jeho zobrazení a skrytí. Vlastní dialog se podobně jako u aktivity definuje děděním od třídy DialogFragment a přepsáním patřičných metod. Pro jednodušší vytvoření dialogu lze také využít vnitřní třídu Builder ze třídy AlertDialog. [19]

ViewModel je třída pro uchovávání dat týkajících se grafického rozhraní. Systém Android může například při změně konfigurace restartovat aktivitu. Při restartování aktivity dojde ke ztrátě původních dat aktivity. Android nabízí

metody pro uložení dat, při kterých jsou data serializována. Alternativou je umístit data, která mají být uchována do objektu třídy ViewModel, který je zachován i při změně konfigurace aktivity. Instance třídy ViewModel je vždy spjatá s určitou aktivitou nebo fragmentem. V případě, že je fragment či aktivita zničena, je také zničen ViewModel. [20]

Navigation drawer je jednou z možností, jak vytvořit navigační menu v Android aplikaci. Jedná se o vysunovací panel s nabídkou položek, který slouží k navigaci v aplikaci. Panel je vhodný pro zařízení s malými displeji, protože je ve výchozím stavu skrytý a zobrazí se pouze při interakci uživatele. Menu lze nastavit tak, aby se zobrazovalo například po stisknutí tlačítka. Ve výchozím stavu lze menu vyvolat přejetím prstu od levého kraje displeje směrem k pravému. Pro použití tohoto menu je potřeba přidat do projektu Android knihovnu (Design support library), potom definovat menu navigace v XML souboru a následně ho v aktivitě inicializovat. [21]

RecyclerView je zobrazovací komponenta sloužící k zobrazení seznamu položek. Komponenta je optimalizovaná pro seznamy s velkým množstvím položek. Pro definici vzhledu položek slouží třída ViewHolder, každá položka v seznamu je potom reprezentována jedním objektem této třídy. Chování komponenty RecyclerView uživatel definuje pomocí vlastního adaptéru rozšířením třídy RecyclerView.Adapter. Při pohybu mezi jednotlivými položkami (scroll) RecyclerView využívá již vytvořených grafických položek a namísto toho, aby vytvářel stále nové, tak pouze mění jejich data (ViewHolder instance). Zároveň si RecyclerView připravuje položky, které bezprostředně následují za právě zobrazenými. [22]

4.2 Komunikační nástroje

Intent je třída, která slouží jako základní komunikační prostředek mezi Android komponentami. Pomocí Intentu lze předávat zprávy, ve kterých můžou být data nebo specifikovaná akce určená k vykonání. Mezi nejzákladnější využití Intentu patří například spuštění aktivity či služby nebo také zobrazení webové stránky z aktivity či poslání SMS zprávy. [23]

Intent lze rozdělit na dva druhy. Jedním druhem je Intent explicitní, který má přímo určeného příjemce zprávy. Tím druhým je implicitní Intent, který nemá přímo určeného příjemce. Implicitní Intent funguje na principu definování akce, která se má vykonat, ale není určené, kdo tuto akci vykoná. Při vyvolání implicitního Intentu operační systém Android zjistí, které aplikace mohou definovanou akci provést a ponechá výběr na uživateli. [23]

Broadcast je třída pro komunikaci mezi Android komponentami podobně jako Intent. Broadcast je ale obecně určený pro více příjemců. Broadcast bývá odeslán v případě, když nastane nějaká událost, na kterou pak ostatní komponenty mohou reagovat. Pro příjetí Broadcastu je nutné mít v aplikace definovaný BroadcastReciever. BroadcastReciever při zachycení Broadcastu vykoná akci, kterou programátor nadefinoval. Systém Android navíc nabízí vlastní Broadcast zprávy, které lze zachytit. Příkladem je Broadcast, jenž je odeslán vždy při startu zařízení. Pomocí tohoto Broadcastu lze například zajistit, aby se určitá služba vždy zapnula při startu zařízení. [24]

SyncAdapter je třída, která slouží pro synchronizaci dat mezi Android aplikací a serverem. Synchronizace může být spuštěna manuálně, například při nastání nějaké události, nebo periodicky. Pro vytvoření funkčního adaptéru je nutné ho integrovat do služby. Výhodou adaptéru je menší energetická náročnost, automatická kontrola dostupnosti internetového připojení a integrovaná podpora autentizace. Adaptér také po nastavení běží nezávisle na aplikaci i po restartování zařízení. [25]

4.3 Úlohy

Služba je prostředek určený pro práci na pozadí. Na rozdíl od aktivity nemá žádný vzhled. Své data mohou služby prezentovat pomocí notifikací nebo skrze aktivitu. Jsou vhodné pro dlouhotrvající operace jako například synchronizace dat se serverem. Standardní služba, která dědí od třídy Service běží v hlavním vlákně, což lze v definici služby změnit. Pro službu, která běží v jiném vlákně lze použít třídu IntentService. Tato třída navíc uchovává požadavky ve frontě, které postupně vykonává a po vykonání všech požadavků se automaticky ukončí. [26]

AsyncTask slouží k provedení krátkodobé úlohy mimo hlavní vlákno. Třída poskytuje k přepsání několik metod pro snadné vytvoření úlohy, která vykoná programátorem definovanou operaci v jiném vlákně. Během vykonávání operace může úloha poskytovat informaci o vývoji (například vyjádřené procentuálně) a po vykonání operace vrátit výsledek do hlavního vlákna. Třídu je vhodné použít například pro uložení souboru do úložiště zařízení nebo náročnější přípravu dat pro vykreslení v aktivitě. [27]

4.4 Ukládání dat

Android nabízí několik možností pro ukládání dat. Jedním je interní úložiště, které je určené pro ukládání privátních souborů. Tyto soubory jsou dostupné pouze z aplikace, která je vlastní. V interním úložišti jsou také ukládány instalované aplikace. Interní úložiště se fyzicky nachází v paměti mobilního zařízení, čímž je úložiště vždy dostupné na rozdíl od paměti na SD kartě. Při odinstalování aplikace se zároveň odeberou všechna její data z interní paměti. Existuje i interní cache paměť, které slouží pro ukládání dočasných privátních dat. Android dále podporuje ukládání dat do lokální SQLite databáze, která je opět dostupná pouze pro aplikace, které ji vlastní. Pro zjednodušení práce s touto databází Android SDK také nabízí vlastní knihovnu Room persistence library pro objektově relační mapování.

Dalším typem pro ukládání dat je externí úložiště, které je obecně volně přístupné, jak ostatním aplikacím, tak uživateli, a data uložená v tomto úložišti nejsou po odinstalování aplikace smazaná. Fyzicky se externí paměť může nacházet na paměti zařízení nebo na SD kartě. Proto se může cesta k externímu úložišti měnit v závislosti na přítomnost SD karty.

Posledním typem úložiště jsou tzv. Shared Preferences, které slouží pro ukládání jednoduchý dat ve formátu klíče a hodnoty. Data uložená pomocí tohoto nástroje jsou privátní a přístupné pouze pro aplikaci, která je uložila. Dříve bylo možné data sdílet mezi aplikacemi, to je ale od verze Android 7.0 zavrženo. Takto uložené údaje jsou při odinstalování aplikace odebrány. [28]

ContentProvider je nástroj sloužící ke správě dat, které aplikace poskytuje. Nástroj může být použit jak interně v aplikaci, tak při sdílení dat s jinými aplikacemi. ContentProvider může zároveň zapouzdřovat data z více zdrojů, například ze SQLite databáze a privátního úložiště. Data uložená v interním úložišti nebo v SQLite databázi jsou privátní pro konkrétní aplikaci. Pro jejich sdílení je nutné vytvořit ContentProvider. Jednotlivá data jsou identifikována pomocí URI, které se skládá z názvu autority a logické cesty k souboru. Název autority slouží pro identifikaci ContentProvideru a cesta k identifikaci dat. [29]

4.5 Sledování událostí

Observable je rozhraní, které využívá návrhový vzor Observer. Objekt, který sleduje změny Observable lze nazvat Observer. Ten pouze definuje operaci, která má po změně nastat. Úlohou Observable je spustiti operaci, kterou Observer definoval. Rozhraní je využíváno k synchronizaci mezi prezenčními daty a grafickými komponentami (data binding). Android také nabízí několik předdefinovaných implementací jako například ObservableBoolean, ObservableInt nebo ObservableArrayList. [30]

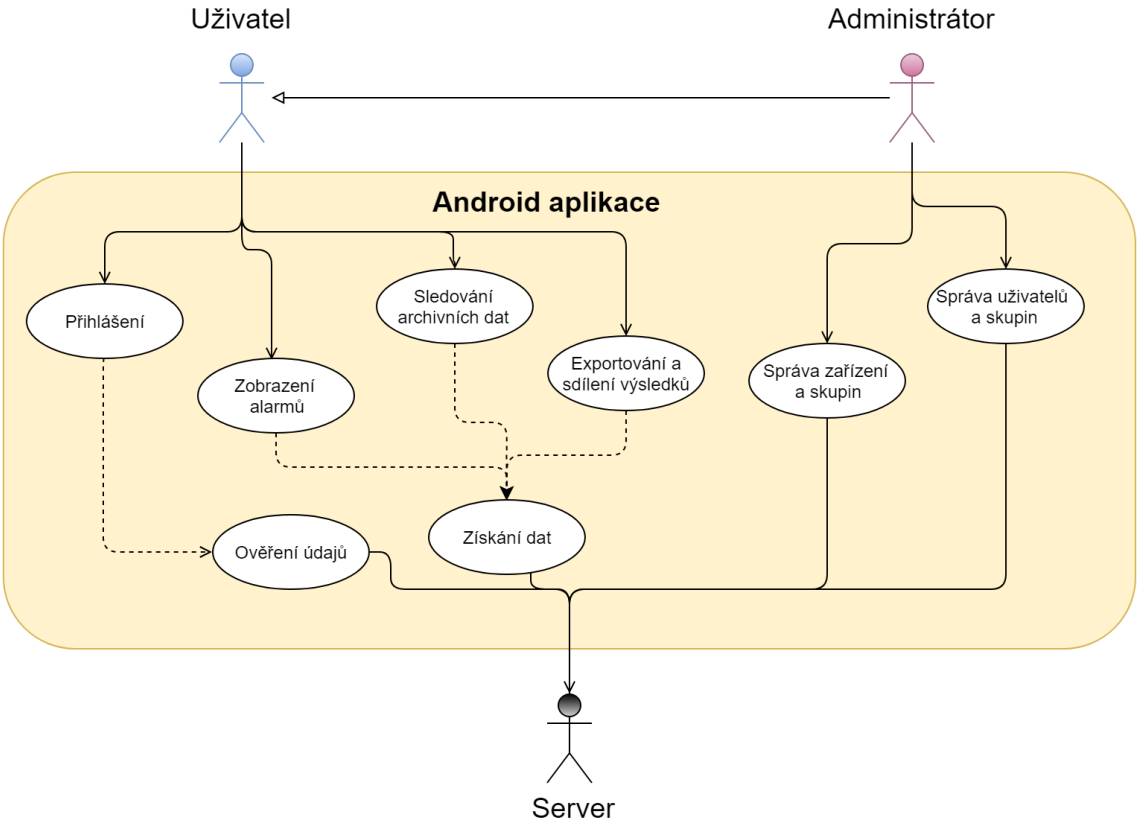
LiveData je nástroj podobný jako Observable, také se jedná o využití Observer návrhového vzoru. Tato třída byla vytvořena z toho důvodu, že Observable sebou nese určité problémy. Observable si drží reference na všechny objekty, které mají přijímat zprávu o změně. Tyto reference musí být odebrány, když už nejsou potřebné, protože jinak referencované objekty garbage collector neodstraní z paměti a dojde k úniku paměti. Navíc u aktivit a fragmentů je potřeba myslet na jejich životní cyklus. Pokud Observable nahlásí změnu a spustí se tak definovaná metoda v aktivitě, která už není v aktivním stavu, tak může dojít při volání metod aktivity k pádu aplikace, protože aktivita není uzpůsobena, aby fungovala v neaktivním stavu. Třída LiveData byla vytvořena, aby zabránila tomuto problému. Třída to řeší tak, že má informaci o životním cyklu objektu, který ji sleduje (Observer). Pokud je Observer v neaktivním stavu, tak Observer neinformuje o změnách. V případě, že je Observer zničen ve smyslu svého životního cyklu, tak třída LiveData rovnou odebere svou referenci na Observer. [31]

5 Návrh aplikace

Tato kapitola popisuje, jakým způsobem byla navrhnuta výsledná Android aplikace. V kapitole jsou také shrnuté požadavky a použité návrhové vzory.

5.1 Požadavky na chování aplikace

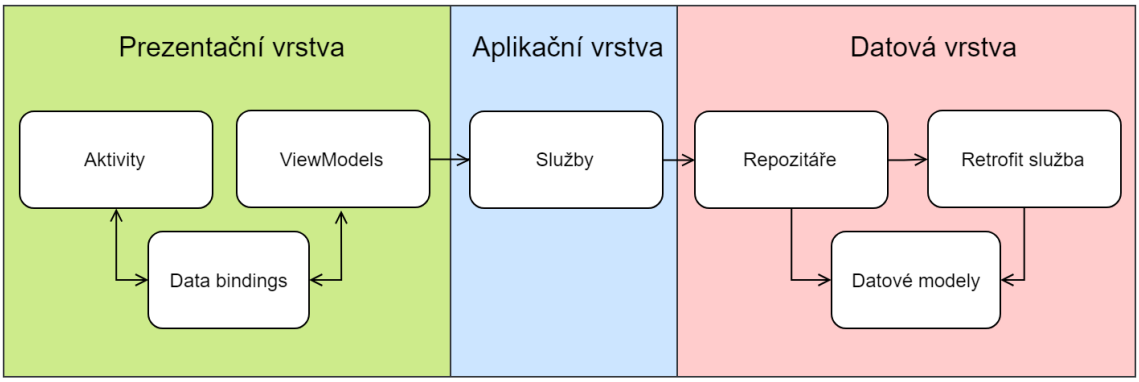
Podle zadání práce a rešerše existujících řešení byl vytvořen use case diagram popisující funkce, které by aplikace měla poskytovat. Na diagramu jsou zobrazené dvě různé role: běžný uživatel a administrátor. Každý uživatel se musí nejdříve přihlásit. Po úspěšném přihlášení může každý uživatel sledovat archivní data, stavy alarmů a exportovat zobrazené výsledky. Administrátor může provádět stejné akce jako běžný uživatel, a navíc spravovat skupiny zařízení a uživatelů. Z diagramu lze jednoduše vyčíst, že ke všem operacím potřebuje aplikace komunikovat se serverem.



Ilustrace 2: Use case diagram popisující chování aplikace

5.2 Architektura aplikace

Kód aplikace lze v tomto případě rozdělit na tři vrstvy. Datová vrstva definuje strukturu dat a stará se o získávání, zpřístupňování, cachování a perzistenci dat. V této aplikaci zajišťuje komunikaci se serverovým REST rozhraním Retrofit služba. Získaná data touto službou jsou mapována z JSON formátu na Java instance datových modelů. Tyto objekty jsou potom zapouzdřeny v repozitářích. Výhodu repozitářů je jednotný přístup k různým zdrojům dat (REST služba, databáze, souborové úložiště). Repozitáře jsou využívány službami z aplikační vrstvy, které mohou nad daty provádět další operace. Jednotlivé služby jsou potom využity v prezentační části aplikace. Příkladem může být přihlašovací služba nebo služba agregující data.



Ilustrace 3: Blokové schéma zobrazující návrh aplikace

Pro prezentační část aplikace byla využita MVVM architektura, která je popsána v dalším oddílu této kapitoly. Prezentační část aplikace je v Android prostředí vždy tvořena aktivitami. Aktivity lze považovat za View z MVVM architektury, protože definují grafické rozhraní jedné obrazovky. V Android prostředí bývá vzhled popsán v XML souborech, ze kterých je potom vygenerován kód. V těchto XML souborech jsou definované jednotlivé grafické komponenty, ze kterých se skládá grafické rozhraní. Kód aktivity je v této architektuře určen hlavně pro specifické záležitosti Androidu jako dotazování práv a spouštění jiných aktivit či služeb. Prezentační model (ViewModel) definuje data, která vykresluje aktivita a také uchovává stav prezentační vrstvy. Data mezi grafickými komponentami a prezentačním modelem jsou synchronizována pomocí tzv. data bindingu. [32]

5.2.1 Model View ViewModel

Model View ViewModel (MVVM) je architektonický vzor, který slouží pro oddělení kódu týkající se grafického rozhraní (prezentační vrstva) od zbytku kódu aplikace. Oddělením těchto vrstev je získána lepší přehlednost, znovupoužitelnost a testovatelnost kódu. Vzor je podobný Model View Contoller (MVC) vzoru. Hlavním rozdílem oproti MVC je, že MVVM navíc využívá data binding. [33]

Architektonický vzor už podle názvu definuje tři rozdílné vrstvy. Pohled (View) slouží čistě k definici vzhledu grafického rozhraní a často bývá popsán pomocí jazyků pro popis dat jako HTML, XAML, XML. Model reprezentuje data, se kterými pracuje aplikace. Pro oddělení těchto dvou vrstev slouží prezentační model (ViewModel). Tento prezentační model obsahuje stav a data grafického rozhraní. Zároveň prezentační model obsluhuje události vyvolané z uživatelského rozhraní. Mezi prezentačním modelem a pohledem jsou data synchronizována pomocí data bindingu. [33]

Android platforma pro využití tohoto architektonického vzoru poskytuje knihovny. Pohled je v Android prostředí tvořen pomocí XML souboru, ve kterém se zároveň definuje spojení (data binding) s prezentačním modelem. Pro prezentační model Android API poskytuje speciální třídu ViewModel popsanou v kapitole 4.5. [20]

5.2.2 Data Binding

Data binding je technika, která obecně slouží k propojení či synchronizaci dvou vrstev. Používá se v aplikacích s grafickým rozhraním, kde slouží k propojení prezentačního modelu s pohledem. V případě, že je viewmodel změněn, tak se změna zároveň projeví i v pohledu, tedy v grafickém rozhraní. Pokud synchronizace funguje i obráceně, tak se jedné o obousměrný data binding, jinak o jednosměrný. Obousměrný data binding je užitečný v případě, kdy může uživatel zadávat vstupní hodnoty, typickým případem je textové pole. Výhodou data bindingu je usnadnění práce programátora, protože nemusí psát kód, který může být vygenerován strojově. Android API podporuje oba typy data bindingu.

Při vývoji Android aplikace se data binding definuje uvnitř XML souboru, který definuje vzhled grafického rozhraní. [34]

Zdrojový kódu 1 ukazuje jednoduchý příklad použití data bindingu. V příkladu je definována grafická komponenta TextView. Tato komponenta má „svázané“ atributy s prezentačním modelem *viewModel*. Text komponenty je určen proměnnou *alarmName,* pozadí komponenty určuje proměnná *background* a atribut *onClick* je spojen s metodou *showDetail*. U posledního spojení je využita reference na metodu v prezentačním modelu, která implementuje rozhraní OnClickListener. Definovaná metoda je tímto nastavením spuštěna při každém „kliknutí“ na uvedenou TextView komponentu. Další ukázka data bindingu použitého ve výsledné aplikaci je uvedena v příloze ilustraci 3.

```
<TextView
      android:text="@{viewModel.alarmName}"
      android:background="@{viewModel.background}"
      android:onClick="@{viewModel::showDetail}"
/>
```

Zdrojový kód 1: Ukázka použití data bindingu v XML souboru

6 Realizace aplikace

Tato kapitola popisuje vytvořenou mobilní aplikaci. Aplikace byla programována v jazyce Java s využitím vývojového prostředí Android Studio.

6.1 Komunikace s webovou službou

Mobilní aplikace komunikuje s webovou službou, která zprostředkovává přístup k archivním datům, informacím o uživatelích, zařízeních a dalším metadatům. Webová služba implementuje REST architekturu s využitím komunikačního protokolu HTTP. Data jsou předávána ve formátu JSON. Pro komunikaci se službou byla využita knihovna Retrofit.

Knihovna Retrofit vytvoří implementaci (konkrétní třídu), která umožňuje komunikaci s webovou službou. Tuto třídu knihovna vytvoří podle rozhraní a speciálních Java anotací, které využívá programátor. Toto rozhraní se skládá z hlaviček metod, které popisují jednotlivé operace. Knihovna podporuje všechny CRUD operace. Parametry hlaviček metod popisují data, která jsou poslána službě jako součást požadavku. Tyto parametry mohou být umístěny do těla požadavku nebo do URL, například vložení parametru s názvem *deviceId* by vypadalo takto: *http://127.0.0.1/devices/{deviceId}*. Naopak návratová hodnota metody definuje strukturu dat v těle odpovědi. Knihovna zároveň umožňuje definovat konvertor, který se stará o serializaci Java objektů do JSON datového formátu a obrácenou deserializaci. K tomuto účelu byl použit konvertor Gson. [35]

Výsledná implementace rozhraní, kterou Retrofit vytvoří, nabízí dva způsoby vykonávání požadavků, a to synchronní a asynchronní [35]. V aplikaci jsou použita pouze asynchronní volání, aby nedocházelo k blokování hlavního vlákna. Zdrojový kód rozhraní RestService, které definuje komunikaci s REST službou, je uveden v příloze III ilustraci 2.

6.2 Repozitáře, sdílená data

Službu RestService z minulé kapitoly využívají takzvané repozitáře. Repozitář je jeden z návrhových vzorů využitých v aplikaci. Repozitáře vytváří abstraktní vrstvu nad daty. Výhoda je v tom, že usnadňují přístup k datům a definují stejné rozhraní pro data z různých zdrojů, například webové služby, databáze, souborového úložiště nebo operační paměti. Repozitáře dále obsahují metody, které jsou využívány z více různých míst aplikace, čímž se redukuje opakování stejného kódu napříč aplikací. [32]

Některé repozitáře byly implementovány spolu s návrhovým vzorem Singleton. Příkladem je DeviceRepository, který poskytuje data o zařízeních. Je implementován jako Singleton, protože je nežádoucí, aby šlo vytvořit více instancí. Pokud by si každá služba držela svoji vlastní instanci této třídy, byla by stejná data o zařízeních zbytečně stahována několikrát a zabírala místo v paměti. Namísto toho jsou data stažena pouze při prvním požadavku (lazy loading) a potom sdílena napříč všemi službami v aplikaci.

Odlišná situace je u MeterValuesRepository, který slouží k poskytování specifických hodnot měření pro konkrétní zařízení, veličinu a časové období. Zde je naopak potřeba více instancí, protože stažení všech dat, jako je tomu u zařízení, nepřipadá v úvahu. Dalším rozdílem je, že pokud existuje jediná služba, která má referenci na tento repozitář a je odebrána z paměti, tak je s ní odstraněn i repozitář. Toto by neplatilo v případě repozitáře implementovaného jako Singleton, který je odebrán jen při odstranění statické reference. Tento repozitář má funkci navíc. Při každém požadavku o data je uložen jeho formát, a pokud je nový požadavek stejný, tak je vrácen výsledek původního. Nevýhodou je, že musí být ošetřena situace, kdy je požadavek stejný, ale data na serveru jsou už změněna.

V repozitářích je využit nástroj LiveData popsaný v kapitole 4.5. Prezentační model či služba, která požádá o data, dostane objekt typu LiveData. Ve chvíli, kdy budou data k dispozici, je žadatel upozorněn prostřednictvím této třídy. Všechny repozitáře aplikace, které jsou závislé na internetovém připojení, dědí

od třídy NetworkRepository, která se ve svém konstruktoru zaregistruje u třídy NetworkController popsaném v následující kapitole. V případě změny dostupnosti internetového připojení je repozitář o této skutečnosti informován a může provést opakované vykonání požadavku.

6.3 Třída NetworkController

Tato vytvořená třída slouží k poskytování informací o změně dostupnosti internetového připojení. Třída funguje na principu návrhového vzoru Observer. Ke sledování využívá systémový Broadcast (viz kapitola 4.2), tedy o změně stavu připojení třídu informuje operační systém Android. Třída, která má být informována o změnách internetové dostupnosti, musí implementovat rozhraní NetworkStateListener a při běhu aplikace se instance této třídy musí registrovat pomocí metody addListener. Všechny takto registrované objekty jsou potom při nastání změny internetové dostupnosti informovány voláním metody onNetworkStateChanged prostřednictvím rozhraní NetworkStateListener. Zároveň je důležité odregistrování objektu pomocí metody removeListener. Objekt, který už dále není využíván, je nutné odregistrovat od NetworkControlleru, jinak nebude automaticky odstraněn z paměti.

6.4 Autentizace uživatelů

Pro autentizaci uživatelů slouží vytvořená služba UserService, která poskytuje metody pro přihlášení, odhlášení, získání tokenu a dalších údajích o uživateli. Prezentační část autentizace uživatelů představuje aktivita AuthenticationAcitivity, vzhled aktivity je zobrazen na ilustraci 19. Jedná se také o vstupní aktivitu aplikace, která je vytvořena minimálně při prvním spuštění aplikace. Dokud se uživatel neautentizuje, nemůže pokračovat dále do aplikace.

Autentizace uživatele probíhá pomocí emailu a hesla. Zadané údaje jsou odeslány na server, kde je ověřena jejich správnost. Pokud nejsou zadané údaje správné, tak je vrácena odpověď se stavovým kódem 401 a uživateli zobrazena zpráva informující o špatně zadaných údajích. V případě, že server vyhodnotí

údaje jako validní, tak odešle informace o přihlášeném uživateli: email, seznam skupin, do kterých uživatel patří a přístupový token.

Přístupový token implementuje otevřený standard JWT (JSON Web Token). Pro ověření pravosti tokenu server využívá algoritmus HMAC s hashovací funkcí SHA-256. Tento token slouží, jak k autentizaci, tak k autorizaci uživatele. Při přijmutí tokenu si jej UserService uloží a poté ho AuthorizationInterceptor vkládá do hlavičky každého dalšího požadavku na server. Interceptor je nástroj (rozhraní) knihovny Retrofit, který umožňuje sledovat každý požadavek na server a měnit jeho obsah [36]. Komunikace mezi klientem a serverem při úspěšné autentizaci je znázorněna pomocí sekvenčního diagramu, který je umístěn v příloze, ilustraci 18.

6.4.1 Perzistence přístupového tokenu

Jak už bylo zmíněno, přístupový token je po získání držen v operační paměti službou UserService, odkud ho lze kdykoliv vložit do požadavku. Mít token pouze v operační paměti není dostačující v případě, že je aplikace ukončena. Proto je přístupový token ukládán i do interní paměti mobilního zařízení.

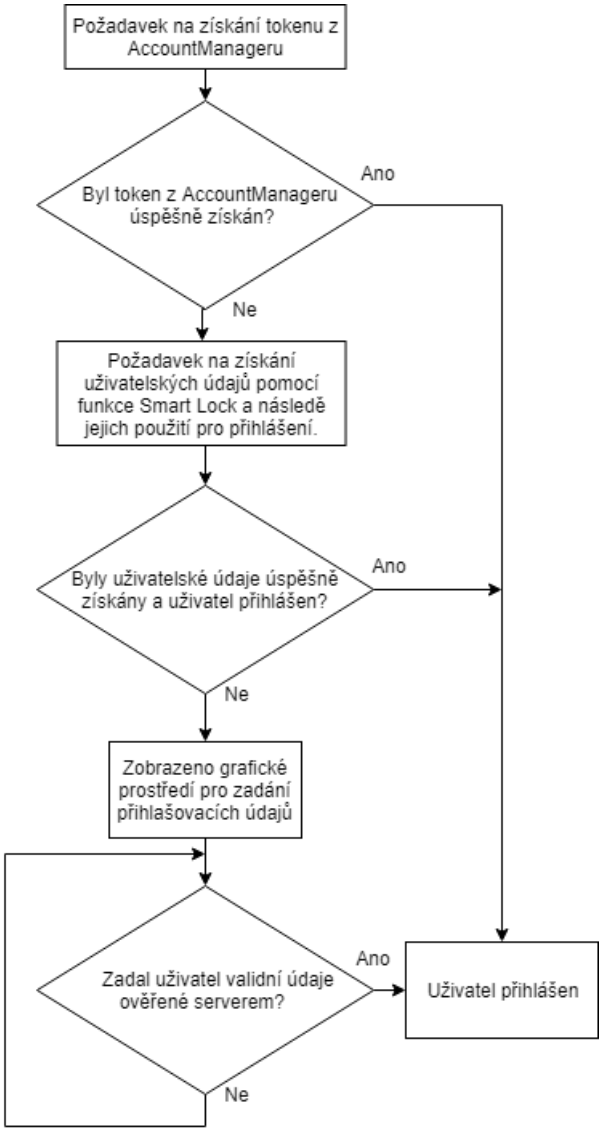
Přístupový token je ukládán pomocí nástroje AccountManager. Tento nástroj slouží pro ukládání uživatelských údajů, jako jsou jména, hesla, ale nabízí i metody přímo pro ukládání a získávání přístupových tokenů. Výhodou AccountManageru je podpora Androidu. Uživatel si můžu zobrazit a spravovat své účty přímo v nastavení operačního systému Android, jak je zobrazeno na ilustraci 22. Aplikace ukládání hesel pomocí AccountManageru nevyužívá.

Při použití vlastní služby je nutné vytvořit implementaci abstraktní třídy AbstractAccountAuthenticator a vytvořit službu využívající tuto implementaci, což v kódu aplikace splňuje třída AccountAuthenticator a služba AuthenticatorService. [37]

Aplikace se vždy při novém spuštění snaží získat uložený přístupový token z AccountManageru pro účet, který byl přihlášen jako poslední. Aplikace má informaci o posledně přihlášeném uživateli, protože si vždy po úspěšném přihlá-

šení uživatele uloží jeho email do SharedPreferences úložiště. Pokud je token úspěšně získán, je ještě zkontrolováno, jestli token již nevypršel. V případě, že je token stále validní, tak uživatel nemusí znovu zadávat přihlašovací údaje a je rovnou přesměrován do další sekce aplikace.

V případě, že token už validní není, aplikace se nejprve pokusí využít funkci Smart Lock popsanou v další kapitole. Pokud uživatel nemá tuto funkci povolenou musí se přihlásit manuálně. Celý proces přihlášení uživatele je znázorněn formou diagramu na ilustraci níže.



Ilustrace 4: Diagram popisující přihlášení uživatele

6.4.2 Funkce Smart Lock

Aplikace používá funkci Smart Lock pro automatické přihlašování uživatelů. Tato funkce není součástí standardního Android API, ale je součástí Google Play služeb. Funkce je v aplikaci použita v případě, když uživatelův přístupový token není k dispozici nebo již není platný a je potřeba se nově autentizovat.

Funkce Smart Lock si uživatelské údaje uloží a na požádání je předá aplikaci. Uživatelské údaje nejsou uloženy v mobilním zařízení, ale v Google cloudu. To je rozdíl oproti AccountManageru, který uživatelské údaje ukládá přímo na zařízení. Proto aplikace nevyužívá k ukládání hesel nástroj AccountManager, ale funkci Smart Lock. Další výhodou této funkce je, že takto uložená hesla jsou synchronizována mezi Google aplikacemi a lze je spravovat, jak na mobilním zařízení Android, tak v aplikaci Google Chrome. [38]

6.4.3 Expirace přístupového tokenu

Získaný přístupový token z webové služby je z bezpečnostních důvodů platný pouze po omezenou dobu. Zřejmě totiž platí, že čím déle je token platný, tím má potenciální útočník více času na jeho získání. Po uběhnuté expirační době je token neplatný a pro další komunikaci je potřeba získat token nový. Proto se aplikace snaží předcházet tomu, aby token expiroval. Aplikace nekontroluje expiraci tokenu periodicky z toho důvodu, aby obnovování neprobíhalo zbytečně, když ji uživatel nějakou dobu nevyužívá. Proto je token obnovován pouze v případě, kdy uživatel svou akcí vyvolá požadavek na server. Problém je řešen ve zmiňovaném AuthorizationInterceptor.

Interceptor zachytí každou událost, při které je poslán požadavek na server a provede kontrolu expirace tokenu, a pokud se blíží doba expirace, tak je poslán požadavek na server pro vygenerování nového token pomocí toho starého. Tento požadavek server podporuje, ale součástí požadavku musí být stále platný přístupový token. Token je sice obnovován zmíněným způsobem, ale stále je nutné, aby aplikace byla připravená na situaci, že platnost tokenu vyprší. Platnost může vypršet například v situaci, kdy uživatel delší dobu aplikaci nevyužívá.

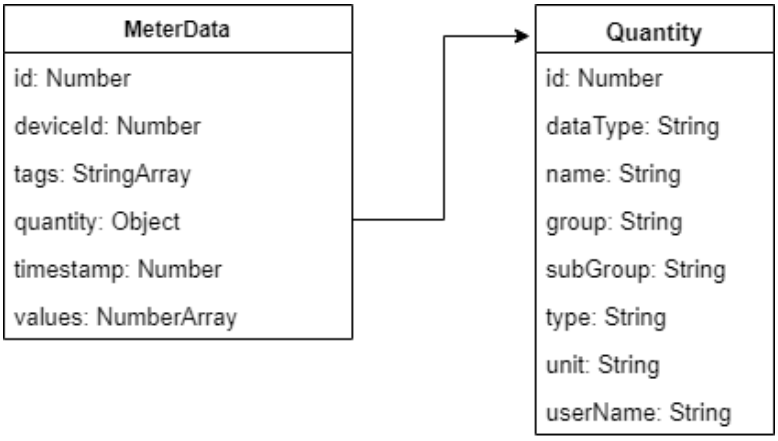
Token by mohl také být ze strany serveru zneplatněn. Aplikace identifikuje expiraci tokenu, pokud dojde k neúspěšnému požadavku se stavovým kódem 401. Tato situace může tedy nastat při každém požadavku na server. Aby mohla aplikace dále komunikovat se serverem je potřeba, aby se uživatel znovu autentizoval a aplikace získala nový token.

Událost, kdy dojde k neúspěšnému požadavku se stavovým kódem 401, je detekována na jednom místě pomocí třídy AuthorizationInterceptor. Tato třída potom zavolá službu UserService a ta se nejdříve pokusí získat přihlašovací údaje pomocí funkce Smart Lock a ty následovně poslat serveru pro získání nového tokenu. Po získání tokenu jsou obnoveny všechny nezdařené požadavky na server pomocí třídy NetworkController. Takto lze problém vyřešit bez přerušení činnost uživatele. V případě, že uživatel nepoužívá funkci Smart Lock, nezbývá nic jiného než, aby uživatel znovu vložil přihlašovací údaje manuálně.

V situaci, kdy se službě UserService nepodařilo získat údaje pomocí funkce Smart Lock, je nastavena proměnná s názvem loginRequired na hodnotu true. Tato proměnná je typu LiveData a ostatní třídy tak mohou sledovat její stav. Tuto proměnnou sledují všechny aktivity, protože proměnnou sleduje BaseActivity (viz kapitola 6.7), od které všechny ostatní aktivity dědí. Při změně proměnné je tedy současná aktivita upozorněna a zobrazí dialog informující uživatele o situaci. Dialog je zobrazen na ilustraci 24. Uživatel se může rozhodnout, jestli se přihlásí ihned, nebo až později. Při vybrání možnosti pro přihlášení je uživatel přesměrován do AuthenticationActivity pomocí metody startActivityForResult třídy Activity. Pokud v aktivitě uživatel zadá platné údaje, tak aplikace obdrží nový token a vrátí se do předešlé aktivity se stejným stavem. Při obdržení nového tokenu jsou zároveň provedeny nezdařené požadavky na server pomocí třídy NetworkController.

6.5 Formát měřených dat

Aplikace přijímá měřená data ze serveru v podobě JSON datového formátu. Přesná struktura dat je zobrazena na ilustraci níže. Naměřená data obsahují vlastní identifikátor, identifikátor zařízení, ke kterému se data vztahují, a seznam tagů, pomocí kterých lze data vyhledat. Dále objekt obsahující informace o veličině, unixovou časovou známku vyjadřující časový okamžik, ke kterému se vztahuje první hodnota, a pole naměřených hodnot.



Ilustrace 5: Formát měřených dat

Většina veličin, se kterými aplikace pracuje, je v měřicích zařízeních reprezentována jako datový typ s pohyblivou řádovou čárkou základní přesnosti (single), tedy pomocí 4 bajtů. Tomu v jazyce Java odpovídá datový typ float. Některé veličiny jsou ale reprezentované pomocí 4 bajtového celočíselného typu bez znaménka, například čítače, které čítají výskyt určité události na měřicím zařízení. Jazyk Java kóduje všechny primitivní celočíselné typy dvojkovým doplňkem a operace s datovými typy bez znaménka (unsigned) podporuje až s verzí 8 [39]. Problém je, že Android podporuje unsigned operace až od verze API 26 [40], tudíž by se použitím značně zredukovala výsledná množina zařízení, pro které by byla aplikace kompatibilní (minimální podporovaná Android verze aplikace je současně 16).

6.5.1 Problém s datovými typy

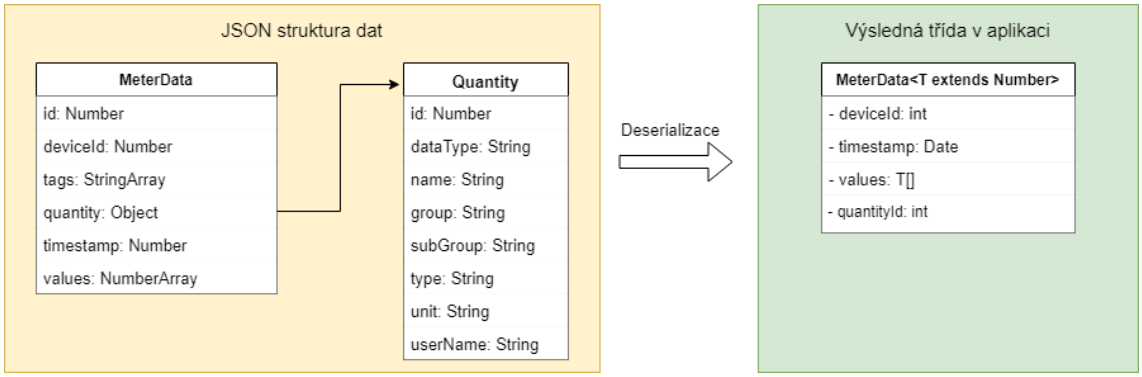
Problém s datovým typem unsigned integer lze vyřešit jednoduše pomocí využití datového typu long, který je 8 bajtový, takže je možné v něm uchovávat stejný rozsah hodnot 4 bajtového celočíselného typu bez znaménka a je i správně reprezentován. Nevýhodou tohoto řešení je větší paměťová náročnost. Problém lze také řešit využitím 4 bajtového typu int s tím, že při vyhodnocení lze hodnotu převést do datového typu long a získat správnou kladnou hodnotu. Tím je sice možné ušetřit paměť, ale je nutné myslet na to, že se datový typ před každým použitím musí konvertovat, což zvyšuje riziko chyby.

Aby se zamezilo tomuto problému, byla vytvořena třída UnsignedInteger, která obaluje primitivní datový typ int a obsahuje metodu getValue, která vrací hodnotu long. Zároveň tato třída rozšiřuje třídu Number. Použitím této třídy vzrostla paměťová náročnost pro uchování jedné hodnoty, protože se jedná o referenční datový typ. Řeší se tím ale zároveň jiný problém. V měřených datech se totiž mohou nacházet prázdné (null) záznamy. Tyto záznamy signalizují, že hodnota záznamu zkrátka není známa, například mohlo dojít k výpadku elektrického proudu a měřicí zařízení bylo nějakou dobu mimo provoz. Pokud je nutné zachovat možnost null záznamů, tak lze buď použít zmiňované řešení, nebo mít pole platných (neprázdných) hodnot a zároveň informaci o tom, které hodnoty v poli chybí. Toto řešení má nevýhodu v tom, že se s daty potom komplikovaně pracuje, proto nebylo využito.

6.5.2 Deserializace

Proces deserializace je znázorněn na ilustraci 6. Strukturu měřených dat, která je použita v aplikaci, definuje třída MeterData<T>. Na ilustraci lze vidět, že oproti původnímu formátu má výsledná třída méně atributů. Důvodem je to, že nikde v aplikaci tyto atributy nejsou použity, proto by jen zbytečně zabírali místo v paměti. Aplikace si před stahováním měřených dat stahuje seznam všech veličin a ten drží v paměti na jednom místě pomocí hash tabulky. Proto stačí u měřených dat mít pouze identifikátor veličiny a další informace o veličině lze získat skrze zmíněnou hash tabulku.

Třída MeterData má jeden generický parametr označený symbolem T. Třída má tento generický parametr, protože datový typ samotných hodnot se může lišit podle typu veličiny. Vždy se ale jedná o číselný typ, proto součástí definice parametru je omezení, že parametr musí rozšiřovat třídu Number. Konkrétní parametr nedokáže knihovna Gson sama odvodit [41], proto je v aplikaci použit pro tuto třídu speciální deserializátor. Pro získání datového typu jsou součástí odpovědi ze serveru informace o získané veličině a to včetně datového typu. Deserializace tedy probíhá tak, že je nejdříve získán datový typ veličiny a podle toho zvolen generický parametr třídy MeterData. Po specifikování parametru už knihovna Gson dokáže JSON řetězec deserializovat.



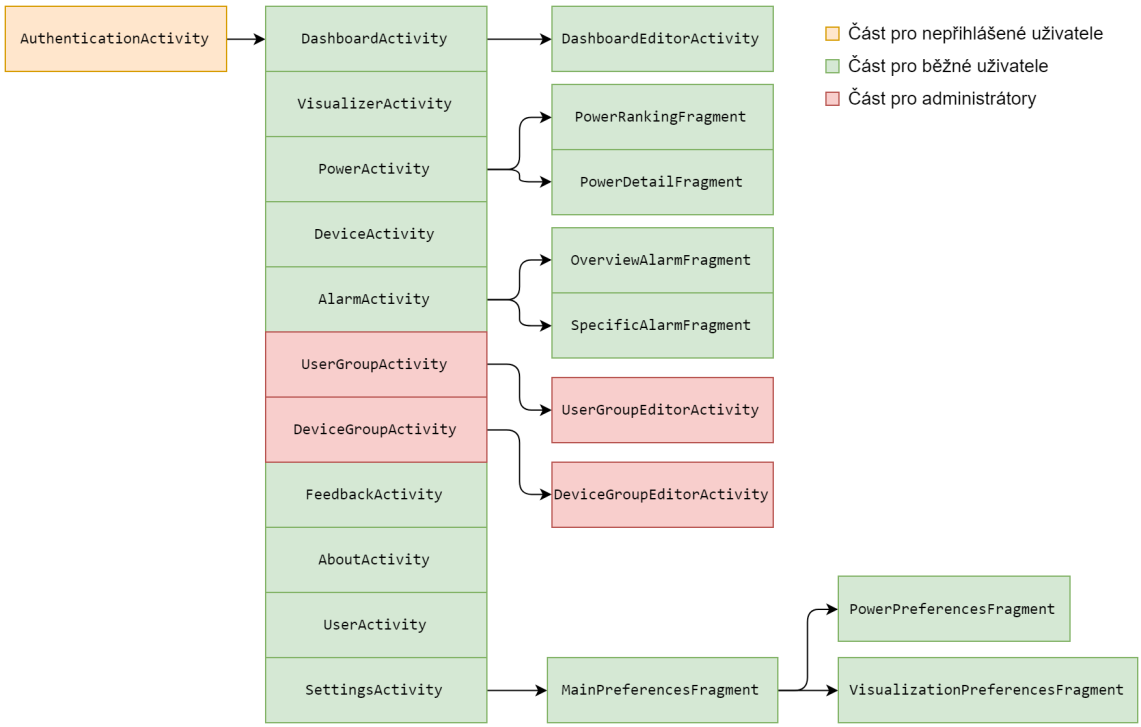
Ilustrace 6: Proces deserializace měřených dat

6.6 Navigace

Aplikace se skládá z několika různých sekcí (aktivit). K tomu, aby se v aplikace dalo snadno orientovat slouží vysunovací panel zobrazený na ilustraci 20. Ten je vytvořen pomocí komponenty DrawerLayout a NavigationView (viz kapitola 4.1). Vysunovací panel lze zobrazit pomocí gesta, přejetím z levé strany displeje k pravé nebo pomocí ikony v levém horním rohu. Některé sekce aplikace se potom ještě dále dělí na několik podsekcí. Podsekce (fragmenty) jsou děleny pomocí nástroje ViewPager, mezi kterými lze opět přepínat pomocí gest nebo menu v horní částí aplikace. ViewPager na rozdíl od DrawerLayoutu přepíná mezi fragmenty, kdežto DrawerPager mezi aktivitami. [42]

Na ilustraci níže jsou znázorněné všechny aktivity aplikace. Ilustrace také znázorňuje možnosti navigace v aplikace. Mezi aktivitami, které jsou umístěné

bezprostředně vedle sebe se lze libovolně přesouvat. Aktivity lze podle přístupu rozdělit na 3 druhy. Pro nepřihlášené uživatelé, běžné přihlášené uživatele a administrátory.



Ilustrace 7: Schéma přístupnosti všech aktivit a fragmentů aplikace

6.7 BaseActivity

Všechny aktivity v aplikaci, které jsou přístupné pomocí navigačního panelu rozšiřují třídu BaseActivity. Tato třída obsahuje metody, které jsou potřeba ve všech aktivitách, proto byl vytvořen společný předek, který tyto metody obsahuje pouze na jednom místě. Mezi tyto metody patří inicializace lišty nástrojů (toolbar), inicializace navigačního menu a také funkce pro obnovení aktivity. Každá aktivita, která tuto třídu rozšiřuje, lze aktualizovat pomocí nabídky v liště nástrojů. Další funkcí je zmiňovaná notifikace v případě vypršení platnosti tokenu. K vypršení platnosti tokenu může dojít prakticky kdykoliv a dialog nelze vytvořit bez kontextu aktivity, proto je tato funkce umístěna do této rodičovské třídy.

6.8 Vlastní komponenty

Některé grafické komponenty společně s jejich inicializačními procesy a reakcemi na události se opakují v několika aktivitách. Například komponenta pro výběr zařízení zobrazena na obrázku 24 je využita v několika aktivitách. Tato komponenta je vytvořena pomocí třídy AutoCompleteTextView a TextInputLayout. Aby tato komponenta zobrazovala nabídku zařízení, je nejdříve nutné získat data ze serveru. Po získání dat je dále potřeba vytvořit adaptér, který je předán třídě AutoCompleteTextView, která pomocí tohoto adaptéru vytvoří požadovanou grafickou nabídku zařízení. Komponenta má svou vlastní stavovou proměnnou určující její současný stav. Tato stavová proměnná poskytuje informaci, zda jsou data stahována či už stažena nebo došlo k chybě. Dále je kontrolován obsah textového pole a komponenta nabízí metodu pro zjištění, jestli se vstup textového pole shoduje s názvem nějakého zařízení.

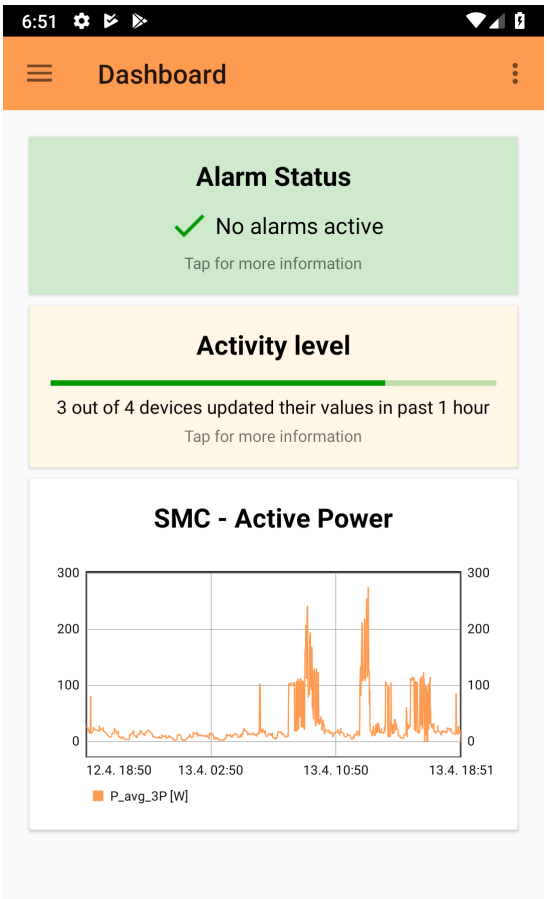
Z tohoto důvodu by bylo nevhodné definovat všechny tyto funkce v každém prezentačním modelu. Problém byl vyřešen využitím jedné ze základních technik objektově orientovaného programování, a to kompozicí. Byla tedy vytvořena samostatná komponenta, která má vlastní vzhled definovaný v XML souboru a také vlastní prezentační model. Toto řešení lze chápat jako vícestupňové využití architektonického vzoru MVVM. Výslednou komponentu lze vložit do jiného XML souboru pomocí tagu *include* a využít v kterékoliv aktivitě či prezentačním modelu. Aplikace obsahuje dalších 5 podobně vytvořených komponent.

6.9 Dashboard

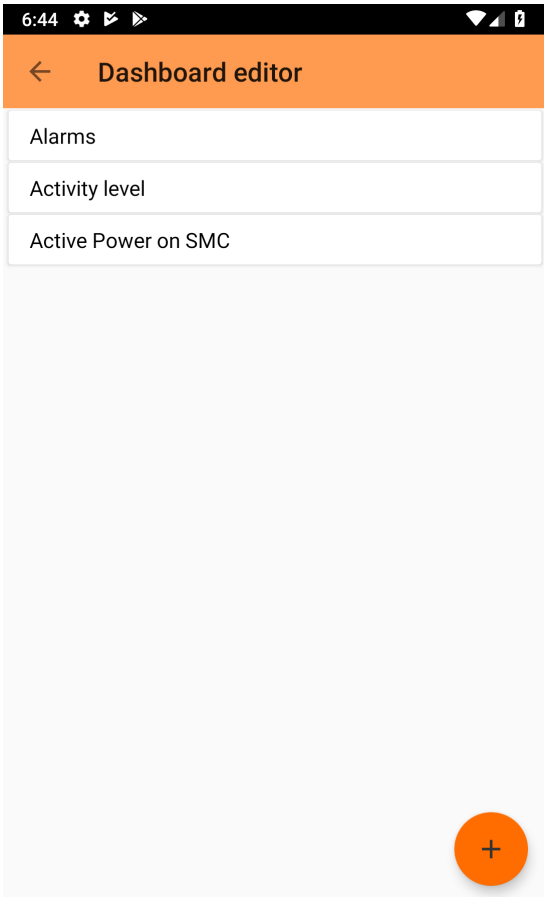
Poté, co se uživatel přihlásí, je přesměrován do úvodní části aplikace se základním přehledem o událostech a informacích aplikace. Vzhled této části aplikace zobrazuje ilustraci 9 a je definován aktivitou DashboardActivity. Na ilustraci lze vidět, že aktivita se skládá z několika komponent.

První komponenta poskytuje informaci o aktuálním stavu alarmů. Uživatel se dotykem na komponentu může přesunout do aktivity s alarmy. Další komponenta zobrazuje, kolik zařízení aktualizovalo svoje hodnoty alespoň

před hodinou. Dotykem na tuto komponentu se zobrazí aktivita se zařízeními, kde lze zjistit přesný čas, kdy naposledy nahrálo zařízení své hodnoty. Třetí komponentou je spojnicový graf zobrazující průběh uživatelsky definované veličiny. Tato třetí komponenta není ve výchozím nastavení zobrazena, pouze předchozí dvě. Komponenty jsou aktualizovaný každých 5 minut pomocí Android třídy Handler.



Ilustrace 9: Dashboard aktivita



Ilustrace 8: Editor vzhledu dashboard aktivity

6.9.1 Komponenty

Komponenty byly naprogramovány takovým způsobem, aby uživatel mohl měnit jejich nastavení, vytvořit si jejich vlastní rozvržení a také, aby bylo pro programátora jednoduše řešitelné přidání nové komponenty.

Pro definici komponenty slouží abstraktní třída DashboardComponent. Třída má za úkol získat potřebná data k vykreslení a ty následně prezentovat. Dědící třída musí implementovat dvě metody, jedna vrací identifikátor vzhledu a druhá

identifikátor prezentační modelu. Díky tomu je možné vytvořit vlastní vzhled. Třída má také metodu pro aktualizaci, po volání této metody by se měla komponenta aktualizovat v případě, že je závislá na datech, které se mohly za uplynulý čas změnit.

K tomu, aby třída DashboardActivity mohla vytvořit komponentu bez znalosti vstupních parametrů konstruktoru a aby komponenty mohly mít různé konfigurace, byla vytvořena další abstraktní třída DashboardComponentBuilder. Tato třída obsahuje všechna potřebná data pro vytvoření komponenty a poskytuje abstraktní metodu buildComponent pro sestavení objektu podle konfigurace. Díky tomu je možné přidat další komponenty, které mohou mít libovolnou konfiguraci. Builder je také připraven na situaci změny konfigurace. Vhodným způsobem, jak to udělat, je zobrazit dialog, nechat uživatele vložit parametry konfigurace a potom parametry uložit do konfigurace. Builder implementuje rozhraní Serializable, aby výsledná třída byla serializovatelná. To je z toho důvodu, aby seznam builderů a jejich konfigurace mohly být uloženy do úložiště zařízení a načíst se i po ukončení aplikace.

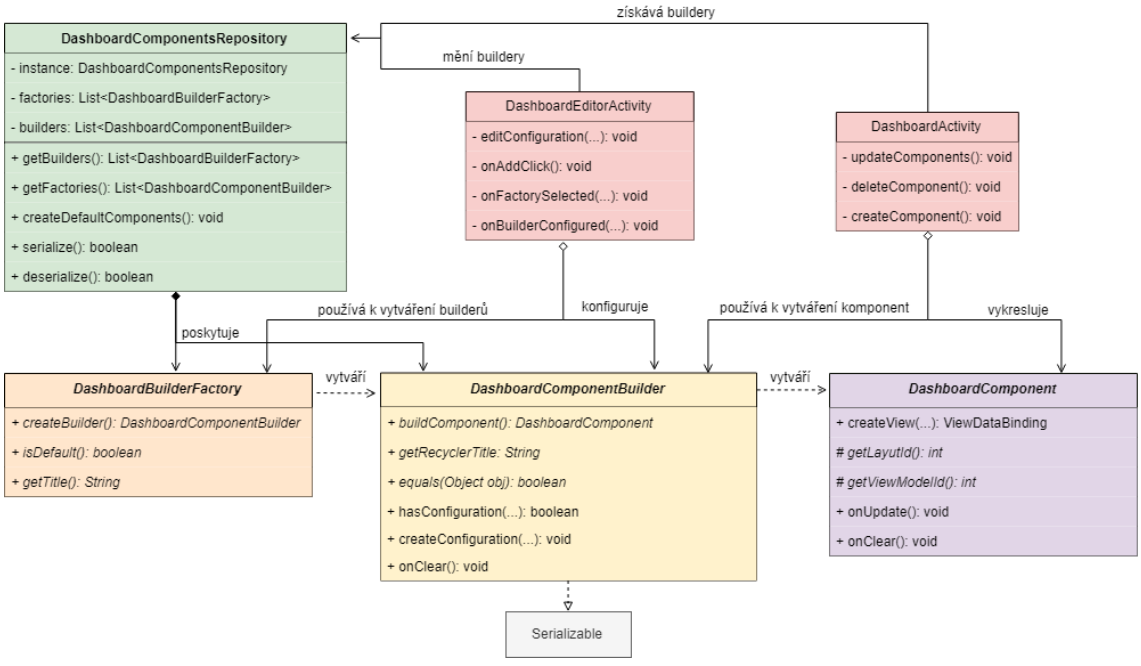
K ukládání builderů slouží třída DashboardComponentsRepository. Tato třída ukládá seznam builderů do interního úložiště mobilního zařízení pomocí serializace. Zároveň při novém startu aplikace data z úložiště načítá.

Aby mohl být abstrahován proces vytváření instancí třídy DashboardComponentBuilder, byla vytvořena abstraktní třída DashboardBuilderFactory. Třída obsahuje metodu createBuilder pro vytvoření builderu, metodu isDefault, která definuje, zdali má být builder vytvořen při prvním spuštění aplikace, poslední metoda getTitle má vracet název builder. Pro vytvoření nové komponenty je tak potřeba vytvořit tři třídy, které implementují výše zmíněné abstraktní třídy a poté zaregistrovat třídu DashboardBuilderFactory u DashboardComponentsRepository, která si drží seznam všech továren.

6.9.2 Editor

Uživatel si může definovat vlastní vzhled úvodní obrazovky pomocí dashboard komponent. K nastavení vlastního vzhledu dashboardu slouží třída DashboardEditorActivity. Do aktivity je možné se přesunout z DashboardActivity přes menu v horní části obrazovky. Vzhled této aktivity je zobrazen na ilustraci 8. Každá položka v seznamu představuje jednu komponentu.

Uživatel může přidávat komponenty pomocí tlačítka v levém dolním rohu. Po stisknutí tlačítka je zobrazen dialog s výběrem komponenty. Tento výběr je vytvořen podle seznamu továren, který poskytuje třída s názvem DashboardComponentsRepository. Názvy v nabídce jsou vytvořeny podle zavolání metody getTitle na danou továrnu. Při vybrání továrny je vytvořen builder pomocí metody createBuilder a v případě, že má builder konfiguraci, je zobrazen konfigurační dialog. Po vyplnění dialogu je přidána položka do seznamu komponent. Dotykem na komponentu lze znovu zobrazit konfigurační dialog a měnit konfiguraci komponenty. Jednotlivé položky v seznamu lze mazat a měnit jejich pořadí pomocí dotykových gest. To je docíleno použitím grafické komponenty RecyclerView s nástrojem ItemTouchHelper. Zjednodušený přehled všech tříd týkajících se dashboardu a editoru je zobrazen na ilustraci níže.



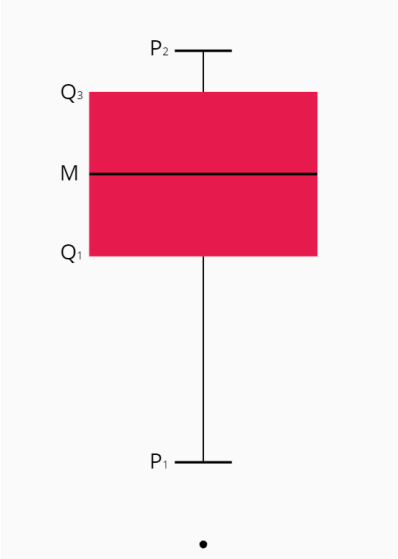
Ilustrace 10: UML diagram tříd týkajících se dashboardu

6.10 Seznam zařízení

Další částí aplikace je aktivita zobrazující seznam zařízení. Vzhled této aktivity je k vidění na ilustraci 23. Na ilustraci lze vidět, že každé zařízení je popsáno názvem a sériovým číslem. Seznam je vytvořen pomocí nástroje RecyclerView popsaný v kapitole 4.1. Po vybrání jednoho ze zařízení se zobrazí dialog s více informacemi o zařízení. Mezi informace patří celý typ zařízení, sériové číslo, čas registrace zařízení a čas, kdy naposledy zařízení nahrálo data na server.

6.11 Grafy

K vizualizaci veličin a jejich statistik aplikace využívá grafy. Protože implementování grafů pomocí standardních Android prostředků by byla časově náročná práce, tak byla v aplikaci pro tento účel použita knihovna. Jedná se o knihovnu MPAndroidChart, která poskytuje 7 různých druhů grafů [35], což je z větší části vystačující pro účely této aplikace. S využitím knihovny byly vytvořeny další dva typy grafů. Nové grafy byl vytvořeny děděním od už existujících tříd grafů, tříd popisující vstupní formát dat grafu, tvorbou matematických funkcí pro získání vstupních dat a tříd sloužící pro vykreslování grafu ze vstupních dat.



Ilustrace 11: Box plot

Prvním přidaným grafem je krabicový graf (box plot). Tělem grafu je obdelníková část, která je shora ohraničená horním a zdola dolním kvartil. V těle grafu je úsečkou vyznačen medián zkoumané sady hodnot. Za odlehlé hodnoty jsou považovány hodnoty menší než L_1 nebo vetší než L_2 z následujících vztahů. Kde Q_1 je dolní kvartil, Q_3 horní kvartil a IQR mezikvartilové rozpětí.

$$L_1 = Q_1 - \frac{3}{2} IQR \quad \textit{(rovnice 1)} \qquad L_2 = Q_3 + \frac{3}{2} IQR \quad \textit{(rovnice 2)}$$

Odlehlé hodnoty jsou v grafu zobrazeny pomocí bodů. Hodnoty L_1 a L_2 nejsou v grafu zobrazeny, namísto nich jsou vykresleny hodnoty P_1 a P_2. P_1 je minimum z množiny všech zkoumaných neodlehlých hodnot, P_2 potom maximum ze stejné množiny. Graf byl vytvořen podle vzoru [43].

Druhý přidaný graf je v angličtině označován jako violin plot (houslový graf). Jedná se o graf, který zobrazuje hustotu rozdělení zkoumané sady hodnot. Tělo grafu tvoří odhad hustoty rozdělení, který je zrcadlený podle vertikální osy. K odhadu hustoty byl použit jádrový odhad hustoty s Epanechnikovou jádrovou funkcí [44]. Oproti krabicovému grafu dokáže houslový graf lépe vizualizovat multimodální rozdělení [45].

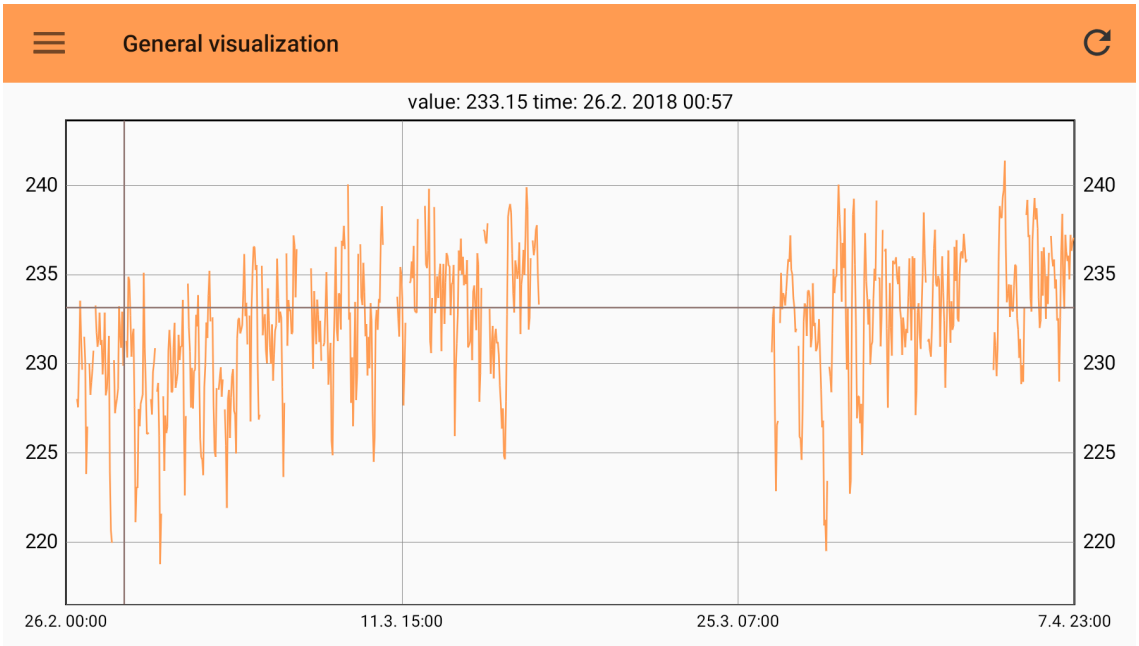
6.12 Vizualizace průběhu veličiny

Tato část aplikace slouží ke zkoumání průběhů veličin. Grafické rozhraní obsahuje 3 vstupní parametry, podle kterých se vybírají cílová data. Prvním parametrem je zařízení. Zařízení lze vyplnit ručně nebo vybrat z nabídky. Seznam všech zařízení je stažen z webové služby a podle toho je vytvořena nabídka. Druhým parametrem je veličina, kterou lze vyplnit stejným způsobem jako zařízení (seznam veličin je také stažený z webové služby). Třetím parametrem je interval, v nabídce je několik často používaných intervalů od poslední hodiny až po minulý rok, na výběr je také volba vlastního intervalu. Při výběru vlastního intervalu se zobrazí další dvě pole pro vybrání začátku a konce požadovaného intervalu. K vybrání data a času slouží speciální dialog knihovny SwitchDateTimePicker.

Data pro vykreslení grafu jsou stažena pouze tehdy, pokud jsou vyplněny všechny potřebné parametry a zároveň jsou validní. Pokud je zadáno neexistující zařízení nebo veličina, je zobrazena ikona signalizující chybu. Veškeré stahování dat i jejich příprava probíhá asynchronně. Pokud není dostupné internetové připojení, je místo grafu zobrazen text s informací o nedostupnosti. V případě, že dojde k opětovnému připojení k internetu, tak jsou data automaticky stažena.

Graf popisuje vývoj veličiny v čase pomocí spojnic. Pod grafem se nachází legenda s názvem a jednotkou veličiny. Graf reaguje na gesta přiblížení, oddálení a posunu ve směru obou os. V grafu lze také vybírat jednotlivé body, hodnoty vybraného bodu jsou zobrazeny nad grafem. Aktivita je také přizpůsobena zobrazení na šířku, při němž je graf vykreslen přes celou obrazovku. Dále je graf přizpůsoben situaci, kdy ve sledovaném období chybí některé hodnoty. Zmíněných funkcí bylo dosaženo pomocí knihovny MPAndroidChart.

Pod legendou se nachází tři ikony. Ikona vlevo slouží pro zobrazení bližších informací o sadě hodnot formou dialogu. V dialogu jsou zobrazeny informace o veličině, dále je zobrazeno, kolik chybí hodnot ve sledovaném časovém úseku, a je zobrazena granularita hodnot, tj. po jakých časových úsecích jsou data agregována. Prostřední ikona umožňuje sdílení a ikona napravo uložení výstupu jako obrázku. Vzhled této část aplikace popisuje aktivita VisualizerActivity s VisualizerViewModel. Pro přístup k datům slouží třída MeterDataRepository. K vložení vstupních hodnot a nastavení grafu slouží třída ChartBuilder.



Ilustrace 12: Aktivita s vizualizací zobrazena na šířku s chybějícími hodnotami

6.13 Regulace účiníku

Tato sekce aplikace slouží k analýze naměřených dat, které se týkají regulace účiníku. Sekce je vytvořena pomocí třídy PowerFactorActivity a dále ji tvoří dva fragmenty. Mezi fragmenty se lze přesouvat pomocí horního menu a dotykových gest, což umožňuje Android nástroj ViewPager. První fragment obsahuje porovnání a celkové zhodnocení jednotlivých účiníků na všech zařízeních. Druhý fragment obsahuje podrobnější údaje o účiníku a výkonových veličinách pro jedno zvolené zařízení.

6.13.1 Porovnání účiníků

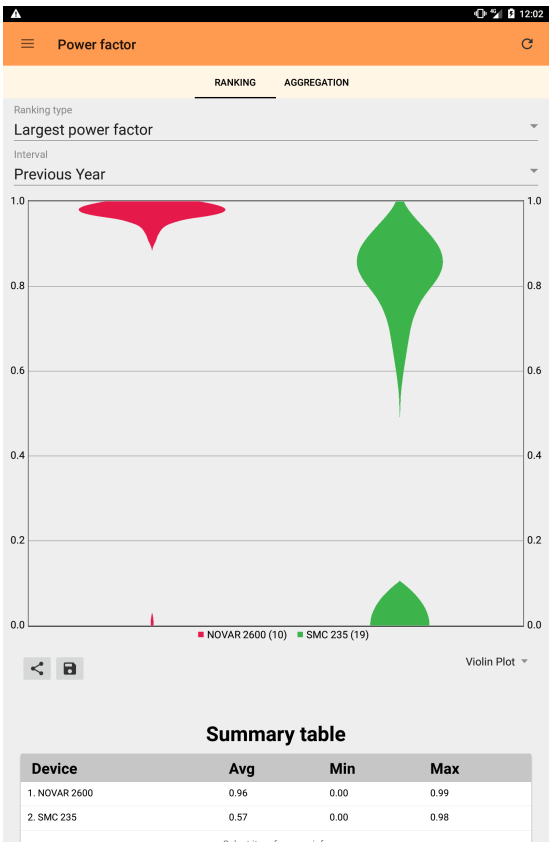
Porovnání a vyhodnocení účiníku zobrazuje fragment PowerRankingFragment. Fragment nabízí 2 druhy vyhodnocení, buď podle skutečného účiníku, nebo účiníku základní harmonické složky. Dále může uživatel určit způsob řazení a období, pro které se má provést vyhodnocení. Po vybrání těchto parametrů je zobrazen graf porovnávající hodnoty na jednotlivých zařízeních. Zařízení jsou hodnocena podle průměrné hodnoty veličiny za vybrané období vypočítané pomocí aritmetického průměru.

Zařízení jsou na grafu srovnána vždy zleva doprava, tedy buď je na prvním místě zleva zařízení s nejmenší, nebo největší hodnotou účiníku podle vybraného způsobu řazení. Fragment umožňuje vizualizovat data pomocí sloupcového, krabicového a houslového grafu (violin plot). Sloupcový graf poskytuje pouze informaci o průměrné hodnotě. Další dva grafy poskytují více informací o rozdělení veličiny. Pod grafem se nachází grafická komponenta pro výběr typu grafu. V grafu je zobrazeno maximálně 5 zařízení. Toto omezení je kvůli náročnosti výpočtu a menším mobilním zařízením, na kterých by pro více jak 5 zařízení byly grafické útvary příliš úzké.

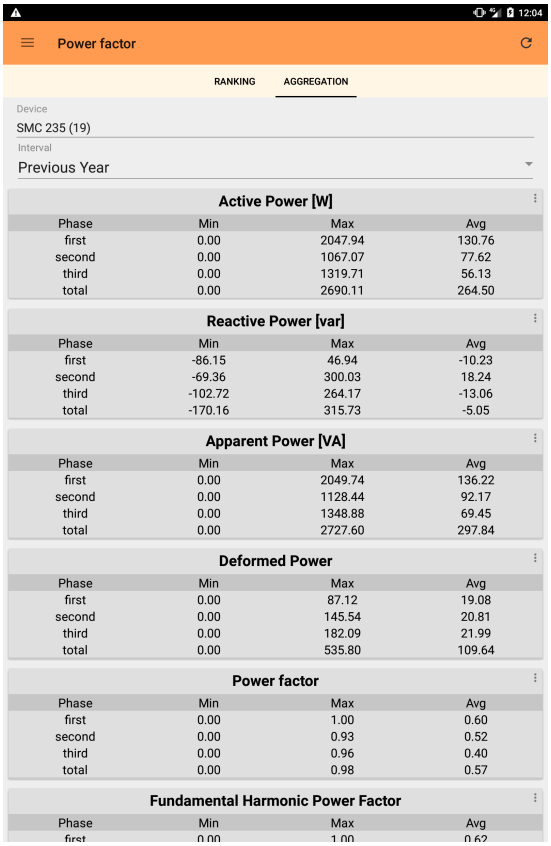
Graf lze jako obrázek vyexportovat a uložit do úložiště zařízení nebo sdílet s ostatními aplikacemi. Pod grafem se nachází tabulka, ve které jsou zobrazena všechna zařízení. U každého zařízení je minimální, maximální a průměrná hodnota účiníku za vybrané období. Obsah tabulky lze, podobně jako graf,

exportovat v podobě obrázku. Při výběru jedné z položek v tabulce, je uživatel přesměrován do druhého fragmentu, který obsahuje podrobnější informace o zvoleném zařízení. Vzhled prvního fragmentu je zobrazen na ilustraci 14. Na ilustraci jsou porovnány hodnoty ze dvou zařízení pomocí houslového grafu.

Aby bylo možné za běhu měnit typy grafů, tak byla vytvořena abstraktní třída ChartFactory. Pro každý typ grafu existuje jiná implementace této třídy. Proces tvorby grafu je totiž poměrně složitý. Nejdřív je nutné vytvořit pro graf vstupní data, která jsou u každého grafu odlišná, potom se musí správně nakonfigurovat a vytvořit grafická komponenta. Při změně typu grafu to potom funguje tak, že se změní používaná tovární třída a ze stažených dat proběhne v jiném vlákně příprava dat. Po dokončení přípravy dat je vytvořena a nakonfigurována nová instance třídy Chart, která už slouží k vykreslení grafu. Po vytvoření tohoto kroku je teprve odebraná původní instance grafu a namísto ní vložena nová, která je následovně vykreslena.



Ilustrace 14: Regulace účiníku – fragment s porovnáním



Ilustrace 13: Regulace účiníku – fragment s agregačními daty

6.13.2 Agregace

Druhým fragmentem je PowerDetailFragment, který na rozdíl od předešlého fragmentu zobrazuje informace pouze jednoho konkrétního zařízení a kromě účiníku obsahuje také agregace ostatních veličin. Po vybrání zařízení a intervalu jsou staženy jednotlivé hodnoty veličin. Fragment zobrazuje agregace činného, zdánlivého, jalového a deformačního výkonu, účiníku a účiníku základní harmonické složky. Tyto agregace jsou zobrazené, jak pro všechny 3 fáze dohromady, tak pro každou fázi zvlášť. Každé okno s veličinou má také ikonu pro zobrazení vysunovací nabídky, pomocí které se lze přesunout do aktivity se spojnicovým grafem, kde je možné detailně zkoumat průběh veličiny. Fragment s agregacemi je zobrazen na ilustraci 13.

V době, kdy byla aplikace vyvíjena, byla data na serveru agregována pouze po minutových a hodinových intervalech a nebylo možné je agregovat pro konkrétní vybraný interval. Proto jsou data stahována po hodinách a potom se dále agregují na mobilním zařízení. Toto řešení by bylo vhodné v budoucnu předělat a agregaci provádět už na serveru, protože takhle dochází k zbytečném přenosu velkého množství dat.

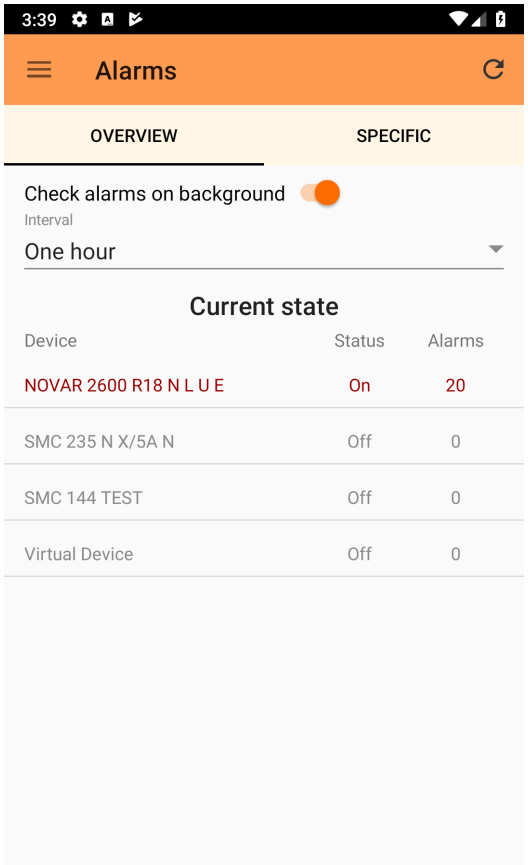
6.14 Alarmy

Další funkce, kterou aplikace nabízí, je prezentace a kontrola alarmů. Nejzákladnější přehled o alarmech poskytuje již popsaná komponenta na úvodní obrazovce aplikace. Pro podrobnější informace slouží aktivita AlarmActivity, která je dále dělena na dva fragmenty. První fragment definuje třída OverviewAlarmFragment, která zobrazuje přehled o všech zařízeních. U každého zařízení je informace o tom, jestli má alespoň jeden alarm aktivní, a počtu alarmů, které zařízení poskytuje. Po vybrání jednoho ze zařízení je uživatel přesměrován do druhého fragmentu.

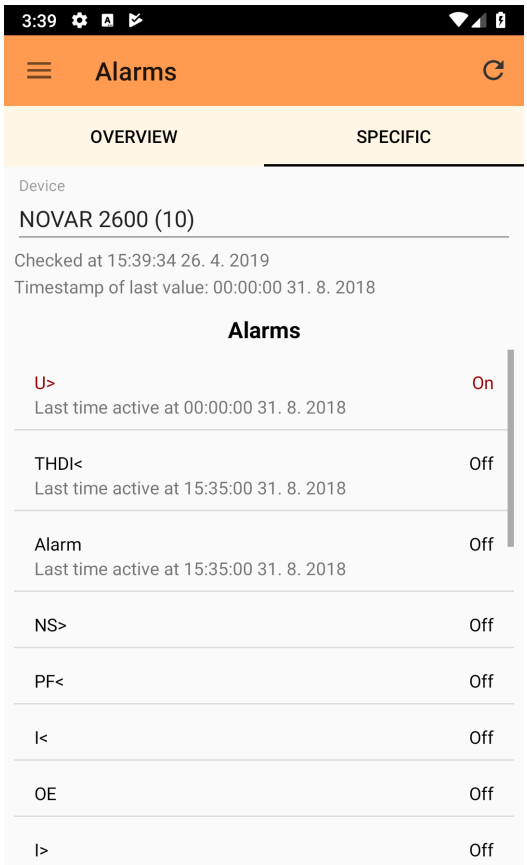
Součástí fragmentu je také nastavení pro kontrolu alarmu na pozadí. Tuto funkci může uživatel aktivovat a poté si nastavit periodu, s níž se budou alarmy kontrolovat. Na změnu parametrů je přímo navázáno zapnutí či vypnutí služby

pro kontrolu alarmů. Vybrané parametry jsou také po změně ukládány do SharedPreferences úložiště, aby se při restartování fragmentu nemusely parametry komplikovaně načítat ze služby. Kontrole alarmu se věnuje další kapitola.

Druhý fragment popisuje třída SpecificAlarmFragment. Oproti předchozímu fragmentu zobrazuje podrobnější informace o alarmech jednoho konkrétního zařízení. Po vybrání zařízení je zobrazen údaj, kdy byly alarmy naposledy kontrolovány podle časového nastavení mobilního zařízení. Dále je zobrazen čas, ke kterému se vztahují zobrazené hodnoty alarmů. Tento časový údaj je důležitý, protože se může jednat už o zastaralé údaje. Pod těmito časovými údaji se nachází seznam všech alarmů zařízení. U každého alarmu je uveden jeho název, stav a případně, kdy byl v minulosti aktivní. Aktivní alarmy jsou zvýrazněny červeným písmem. Alarmy jsou v seznamu srovnány podle stavu. Nejdříve se nachází aktivní alarmy, potom alarmy, které byly aktivní v minulosti a až potom neaktivní alarmy.



Ilustrace 15: Přehled s alarmy všech zařízení



Ilustrace 16: Seznam alarmů pro konkrétní zařízení

6.14.1 Kontrola alarmů na pozadí

V případě, že uživatel aktivuje sledování alarmu na pozadí, tak je spuštěna služba SyncService. Jedná se o službu, která využívá synchronizační adaptér (třída SyncAdapter). Synchronizační adaptér je nástroj Android API, který slouží k synchronizaci dat mezi mobilní aplikací a serverem. Synchronizace proběhne vždy, pokud je dostupné internetové připojení i v případě, že neběží aplikace, která by proces spustila, a to i po restartování zařízení. Nejmenší možná perioda je od novějších Android verzí nastavená na 15 minut. [46] Synchronizace probíhá periodicky po intervalu, který uživatel zvolí v grafickém rozhraní. Třída SyncAdapter má metodu onPerformSync, která je spouštěna v případě, že mají být synchronizována data mezi aplikací a serverem. [25]

Metoda onPerformSync byla implementována, využívá třídu CheckAlarmsTask, která provádí samotnou kontrolu alarmů. Třída CheckAlarmsTask stáhne ze serveru informace o všech alarmech ze všech zařízení. Potom postupně zkontroluje, jestli je nějaký aktivní a vrátí výsledek operace. A ten se vrátí zpět do metody onPerformSync. V případě, že je alespoň jeden alarm aktivní, tak metoda probudí displej zařízení a zobrazí uživateli notifikaci. Vzhled notifikace je na ilustraci 25 v příloze. Kliknutím na notifikaci je uživatel přesměrován do aktivity s alarmy.

Vhodnější řešení by bylo, kdyby kontrola alarmů byla prováděna na serveru při každém nahrání nových hodnot ze zařízení. V případě, že by byl nalezen aktivní alarm, tak by server o této skutečnosti informoval mobilní aplikaci. Periodickým dotazováním je server zbytečně zatěžován a na mobilním zařízení je zbytečně spotřebovávána energie, případně i mobilní data.

6.15 Sdílení výsledků

Části aplikace, které obsahují výsledky v podobě tabulky hodnot nebo grafu nabízí možnost tyto výsledky sdílet s ostatními aplikacemi nainstalovanými na zařízení uživatele. Z grafické komponenty (View) je vygenerována bitmapa, která je pak sdílena. Před sdílením je ještě nutné bitmapu dočasně uložit [47].

Obrázek je v jiném vlákně uložen do interní cache paměti zařízení pomocí metody saveImageToInternalCache, která je definována ve třídě SaveImageTaskCreator. Poté, co je obrázek uložen, je pomocí ContentProvideru získáno jeho URI. Potom je vytvořen implicitní Intent s akcí Intent.ACTION_SEND a URI určující cestu k vytvořenému obrázku. Vyvoláním tohoto Intentu pomocí metody startActivityForResult je zobrazeno dialogové okno s výběrem aplikace (ilustrace 26), se kterou má být obrázek sdílen. V dialogu jsou zobrazeny pouze aplikace, které dokáží s obrázkem pracovat, to je například aplikace Gmail, která umožňuje odeslat obrázek v příloze emailu. Po vybrání aplikace je uživatel do této aplikace přesunut a při dokončení operace se vrátí zpět do původní aktivity. Při vrácení z cizí aplikace je ještě obrázek z cache úložiště vymazán, protože už není potřebný.

6.16 Administrace

Administrace nebyla součástí zadání práce, ale byla vytvořena pro případ pozdější implementace. Pokud má uživatel administrátorské práva, tak se mu v nabídce na ilustraci 20 zobrazí dvě položky: device groups, user groups. Seznam práv aktuálního uživatele poskytuje služba UserService. Skupiny uživatelů slouží k definování uživatelských práv. Vzhled aktivity UserGroupActivity, která umožňuje správu uživatelských skupin, je zobrazen na ilustraci 29. Vzhled aktivity tvoří seznam uživatelských skupin, které lze editovat a mazat. Tlačítko v levém dolním rohu slouží k přidání nové skupiny. Editace skupiny probíhá v aktivitě UserGroupEditorActivity, ve které lze přidávat a odebírat uživatele. Grafické rozhraní pro správu skupin zařízení funguje obdobně.

6.17 Nastavení

Součástí aplikace je také nastavení preferencí. Pro tento účel slouží aktivita s názvem SettingsActivity a 3 další fragmenty. MainPreferences fragment se načte při spuštění aktivity a dále člení nastavení na 2 kategorie, kde pro každou kategorii existuje další fragment. Uživatel si může nastavit výchozí zařízení, které bude vždy vybrané při novém startu aktivity. Ve výchozím stavu se vždy

nahraje posledně vybrané zařízení. Obdobně je to s volbou intervalu, veličiny v obecné vizualizaci, a také typem zhodnocení a grafu v aktivitě se zhodnocením účiníku. Preference jsou ukládány pomocí SharedPreferences úložiště. Grafické rozhraní je vytvořené pomocí Android Preference grafických komponent v XML souborech. Dodatečná logika pro načtení dynamických dat, jako například seznam veličin a zařízení, je definována v předem zmiňovaných fragmentech.

6.18 Zpětná vazba uživatelů

Jedním z požadavků na aplikaci je také funkce pro zpětnou vazbu uživatelů. Tento problém většina aplikací neřeší, protože zpětnou vazbu nabízí Google Play obchod, kde může každý uživatel aplikaci hodnotit a napsat komentář [48]. Zároveň Google nabízí posílání notifikačních emailů v případě nových recenzí [48]. V aplikaci byla pro tyto účely vytvořena aktivita. Vzhled aktivity je k prohlédnutí na ilustraci 27. Aktivita slouží k jednoduchému odeslání emailu. Pro záznam nestandardních událostí a pádů aplikace byl využita nástroj Firebase Crashlytics. Firebase je platforma pro vývoj aplikací. Nabízí služby jako cloud úložiště, real-time databázi, autentizační systém, analytické a podpůrné nástroje. [49]

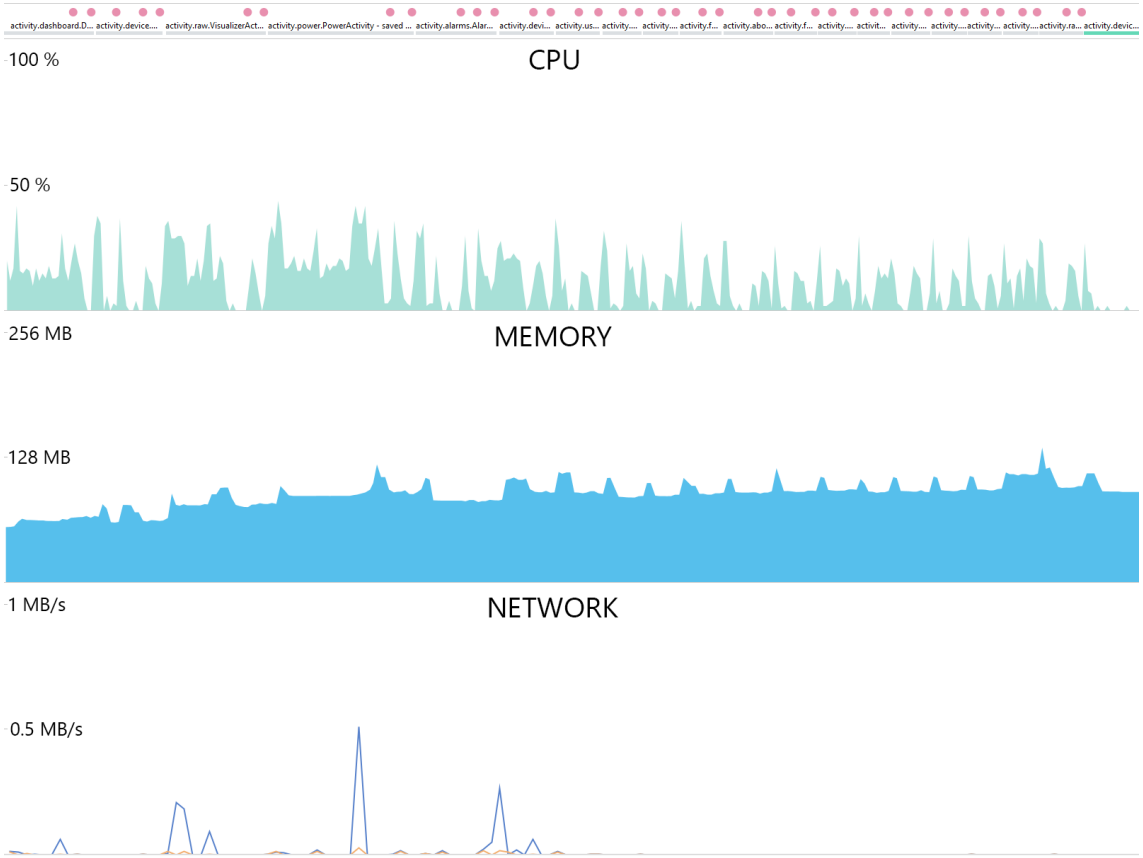
V rámci této aplikace je platforma využita ke sledování chování aplikace. Jedná se o nástroj Crashlytics, který slouží k záznamu chyb aplikace. Nástroj je vložen do projektu jako knihovna a ta sbírá data o pádech aplikace a posílá je do cloudu. Získané údaje o pádu aplikace, jako čas pádu, typ zařízení, Android verze, spuštěná aktivita, typ a stacktrace výjimky, je potom možné sledovat ve webovém prostředí. Knihovna také umožňuje zaznamenávání vlastních událostí. V aplikace je knihovna také využita při zachycení výjimek pro přehled, jak často a k jakým ošetřeným výjimkám v aplikacím dochází. [50]

Součástí aplikace je také sekce „O aplikaci“. Vzhled této sekce je zobrazen na ilustraci 28. Na obrazovce je zobrazen název aplikace spolu s její verzí. V této sekci jsou také zobrazeny všechny použité knihovny, které přispěly ke zjednodušení vývoje aplikace. Pro každou knihovnu je zobrazen odkaz na webové stránky knihovny a její licence.

7 Profilování a testování aplikace

Pro profilování aplikace byl použit vestavěný Android Profiler vývojového prostředí Android Studio. Profilování probíhalo na mobilním telefonu Doogee BL5000 s osmijádrovým 1.5GHz procesorem ARM-A53, 4GB operační pamětí, 5,5palcovým full HD displejem a operačním systémem Android 7.0. Na obrázku níže jsou zobrazeny 3 grafy zobrazující vytížení procesoru, operační paměti a síťového rozhraní. Graf zachycuje běh finální aplikace po dobu 71 vteřin, při kterých byly procházeny všechny aktivity aplikace. Jako první byla spuštěna AuthenticationActivity, poté DashboardActivity, až po AboutActivity, následně byly aktivity procházeny obráceně zpátky do DashboardActivity.

Procesor byl nejvytíženější (40 %) hned při startu aplikace a poté také v aktivitě s regulací výkonu PowerActivity (42 %). Lze vidět, že procesor je vytíženější vždy při přechodu mezi aktivitami. Dále lze vidět, že je procesor vytíženější v první



Ilustrace 17: Profilování výsledné aplikace

polovině pozorování, kdy byly aktivity otevřeny poprvé. Procházení aktivit podruhé už tolik náročné nebylo. To je důsledek toho, že potřebná data aktivit už se nemusela znovu stahovat a připravovat pro vykreslení.

Na druhém grafu obrázku je zobrazena náročnost aplikace na operační paměť. V grafu je zahrnuta veškerá paměť aplikace, to je Java alokovaná paměť, nativní (C, C++) paměť, paměť pro vykreslení grafických komponent, zásobníková paměť a paměť obsahující kód aplikace [51]. Celkově aplikace ke svému chodu potřebovala nejvýše 131 MB paměti. Při spuštění aplikace zabrala necelých 80 MB paměti, po spuštění všech další aktivity se náročnost zvýšila na přibližně 100 MB, potom už se paměť zvyšovala pouze při přechodu mezi aktivitami. Po podrobnější analýze v profileru byl zjištěno, že nejvíce paměti je potřeba pro vykreslování grafických komponent, přibližně 30 až 50 MB. Právě tato paměť nejvíce způsobuje výkyvy v druhém grafu při přechodu mezi aktivitami. Kód aplikace v paměti zabírá 7 MB. Při spuštění všech aktivity Java paměť zabírala v průměru 23 MB, nativní paměť 25 MB. Ve zdrojovém kódu aplikace se sice nenachází žádný C/C++ kód, ale nativní paměť používá Android API [51].

Třetí graf zobrazuje síťový provoz aplikace. Z grafu lze vyčíst, že aplikace komunikovala pouze v první polovině pozorování. Pomocí profileru, který umožňuje i zkoumání jednotlivých požadavků bylo zjištěno, že největší stažený soubor měl 151 KB a obsahoval seznam všech veličin. Nejdelší odeslání požadavku a přijmutí odpovědi trvalo 3,1 vteřin.

Během dřívějšího profilováni aplikace bylo zjištěno, že při znovuvytvoření aktivity VisualizerActivity se neustále nepatrně zvyšuje paměť aplikace. Příčinou byla proměnná DataSetObservable třídy BaseAdapter [52] v prezentačním modelu aktivity, která si ve svém listeneru držela instanci na aktivitu v podobě třídy Context. Při znovuvytvoření aktivity totiž nedochází k odstranění prezentačního modelu [20], takže nemohly být staré instance aktivity odebrány garbage collectorem z paměti a vznikal únik paměti. Přesná příčina úniku paměti byla objevena s pomocí knihovny LeakCanary. Potom, co byla odebrána proměnná z prezentačního modelu už k úniku nedochází.

Aplikace byla manuálně testována na již zmiňovaném mobilním telefonu Doogee BL5000 s Android API 24, tabletu Huawei MediaPad T3 s Android API 23 a také na dvou emulovaných zařízeních. První emulované zařízení mělo nejstarší podporované Android API aplikace 16 a druhé zařízení naopak nejnovější API 28 (duben 2019). Aplikace byla testována na archivních datech z regulátoru účiníku NOVAR 2600 a analyzátorech SMC 235 a SMC 144. Během testování na zařízení s API 16 bylo objeveno několik drobnějších problémů, které se převážně týkaly grafického rozhraní. Tyto problémy byly odstraněny. Na zařízení s API 28 nebyly nalezeny žádné problémy.

Kromě manuálního testování bylo také vytvořeno několik jednotkových testů s využitím knihovny JUnit. K tvorbě zástupných objektů (mocků) byl využit framework Mockito. Testované byly zejména třídy z aplikační vrstvy kódu, například služba pro vyhodnocení stavu alarmů, kde byl využit mock webové služby, který vracel různé testovací datové sady, což by bylo obtížnější nasimulovat v reálném prostředí. Před nasazení aplikace do produkčního prostředí by bylo vhodné sadu testů značně rozšířit.

Pro sestavení výsledného instalačního balíčku bylo použité Android Studio s nástrojem Gradle. Při sestavení byla také provedena obfuskace a minifikace pomocí nástroje Proguard, tím byla velikost instalačního balíčku redukována z 5,2 MB na 3,9 MB.

8 Závěr

Teoretická část práce v úvodní kapitole popisuje důvod vzniku práce, předchozí řešení, funkce regulátoru účiníku a globální komunikaci mezi regulátory a cílovou aplikací. V druhé kapitole je uvedena rešerše aplikací zabývajících se podobnou tematikou. Třetí kapitola se zabývá možnostmi vývoje mobilních aplikací a čtvrtá kapitola nástroji operačního systému Android, které byly využity při vývoji výsledné aplikace.

Výsledkem této práce je aplikace cílená pro operační systém Android, která je kompatibilní se zařízeními od verze Android 4.1 z roku 2012. Finální zdrojový kód aplikace bez použitých knihoven má dohromady 13659 neprázdných řádků rozdělených do 215 tříd. Aplikace byla naprogramována ve vývojovém prostředí Android Studio v jazyce Java. Kromě Android API bylo využito několik dalších knihoven uvedených v příloze II.

Aplikace využívá webovou službu k získání archivních dat regulátorů. Komunikace s webovou službou probíhá přes HTTP, data jsou přenášena v JSON formátu a přístup k datům je definován REST rozhraním. Základní agregace dat probíhá na serveru, další agregace, příprava a vyhodnocení dat pro výstupní grafy a tabulky probíhá v aplikační vrstvě aplikace. Autentizace a autorizace mezi aplikací a serverem probíhá pomocí přístupového JSON web tokenu. Token je na mobilním zařízení ukládán v interní paměti pro zapamatování uživatele. Token má omezenou dobu platnosti, proto jej aplikace obnovuje v případě, že se jeho platnost blíží ke konci.

V prezentační vrstvě aplikace byl využit MVVM vzor s technikou data binding. Aplikace je z prezentačního hlediska rozdělena na 13 aktivit a 9 fragmentů. Vstupní částí aplikace je přihlašovací aktivita, po přihlášení je seznam přístupných aktivit závislý na právech uživatele. Po přihlášení je uživatel přesměrován do Dashboard aktivity, kde jsou ve výchozím stavu zobrazeny základní informace o stavu zařízení a jejich alarmech. Vzhled této aktivity si může uživatel sám přizpůsobit.

Power Factor aktivita poskytuje porovnání účiníku z jednotlivých zařízení pro vybrané období formou 3 druhů grafů. Pro podrobnější analýzu lze využít aktivitu s obecnou vizualizací průběhů veličin. Další důležitá aktivita zobrazuje stav alarmů prostřednictvím celkového přehledu nad všemi zařízeními a také detailnějšího přehledu se všemi alarmy konkrétního zařízení. Aplikace umožňuje exportovat výsledky ve formě obrázku a dále je ukládat nebo sdílet s ostatními aplikacemi. Součástí aplikace je formulář pro uživatele k odeslání zpětné vazby. Zaznamenávání nestandardních situací a pádů aplikace je zajištěno pomocí integrované knihovny Firebase Crashlytics.

Výsledná aplikace byla podrobena profilování, aby byla zjištěna její náročnost na hardwarové prostředky. Také byla manuálně testována na dvou fyzických a dvou emulovaných zařízeních k ověření funkčnosti a nalezení nekompatibility mezi Android verzemi. Testování také probíhalo formou jednotkových testů s využitím knihovny JUnit a Mockito. Pro nasazení do produkčního prostředí by ale bylo vhodné sadu testů značné rozšířit.

Práce nabízí mnoho prostoru pro rozšíření či vylepšení. Tím by mohl být přehled a porovnání naměřené spotřeby elektrické energie jednotlivých zařízení. Dále by do aktivity s vizualizací průběhu veličin mohla být přidána možnost sledovat více veličin. Aplikace by také mohla být rozšířena o funkčnost na operačním systém iOS.

Seznam použité literatury

[1] Bc. Jan Moravec. *Zpracování a vizualizace dat analyzátorů v OS Android*. Liberec. 2017. Bakalářská práce. Technická univerzita v Liberci, Fakulta mechatroniky, informatiky a mezioborových studií

[2] KMB systems s. r. o. *Software ENVIS, Uživatelská příručka pro podporované měřicí přístroje, verze 1.1* [online]. [vid 23. 4. 2019]. Dostupné z: http://www.kmb.cz/index.php/cs/ke-stazeni/category/2-software?download=197:envis-v11-uzivatelska-prirucka

[3] KMB systems s. r. o. *Třífázové regulátory jalového výkonu a síťové analyzátory NOVAR 2600* [online]. [vid 23. 4. 2019]. Dostupné z: http://www.kmb.cz/index.php/cs/component/phocadownload/category/4-docspfc?download=211:novar-2600-manual-k-pristroji

[4] KMB systems s. r. o. *Regulátory účiníku – KMB systems* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/about/dashboards/

[5] Capterra Inc. *Energy Management Software | 2019 Reviews of the Most Popular Systems* [online]. [vid 23. 4. 2019]. Dostupné z: https://www.capterra.com/energy-management-software/

[6] Wattics Ltd. *Wattics Docs | All you need to know about Wattics solutions* [online]. [vid 23. 4. 2019]. Dostupné z: https://docs.wattics.com/

[7] Entronix Energy Management inc. *Entronix Energy Management* [online]. [vid 23. 4. 2019]. Dostupné z: https://entronix.io/index.php/support-center/

[8] Best Energy Saving Technology Ltd (BEST). *Energy, Tenant Management & Maintenance Solutions* [online]. [vid 23. 4. 2019]. Dostupné z: https://bestenergysaving.com/solutions/

[9] Power Monitors Inc. *PMI – Power Quality Analysis Software* [online]. [vid 23. 4. 2019]. Dostupné z: https://powermonitors.com/product-category/software/

[10] StatCounter. *Mobile Operating System Market Share Worldwide* [online]. [vid 23. 4. 2019]. Dostupné z: http://gs.statcounter.com/os-market-share/mobile/worldwide

[11] Apple Inc. *Swift - Apple Developer* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.apple.com/swift/

[12] Google Developers. *Kotlin and Android* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/kotlin

[13] Spallino, Alessio. *Native versus hybrid mobile application development for professional membership services.* Espoo. 23. 4. 2019. Diplomová práce. Aalto University

[14] Google Developers. *Progressive Web Apps | Web* [online]. [vid 23. 4. 2019]. Dostupné z: https://developers.google.com/web/progressive-web-apps/

[15] Ionic. *Ionic Article: What is Hybrid App Development?* [online]. [vid 23. 4. 2019]. Dostupné z: https://ionicframework.com/enterprise/resources/articles/what-is-hybrid-app-development

[16] Microsoft. *Funkce platformy Xamarin.Forms - Xamarin* [online] [vid 23. 4. 2019]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/platform/

[17] Android Developers. *Activity* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/reference/android/app/Activity

[18] Android Developers. *Fragments* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/guide/components/fragments

[19] Android Developers. *Dialogs* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/guide/topics/ui/dialogs

[20] Android Developers. *ViewModel* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/topic/libraries/architecture/viewmodel

[21] Android Developers. *LayoutInflater* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/reference/android/view/LayoutInflater

[22] Android Developers. *RecyclerView* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/guide/topics/ui/layout/recyclerview

[23] Android Developers. *Intent* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/reference/android/content/Intent

[24] Android Developers. *Broadcasts* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/guide/components/broadcasts

[25] Android Developers. *SyncAdapter* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/training/sync-adapters

[26] Android Developers. *Service* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/reference/android/app/Service

[27] Android Developers. *AsyncTask* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/reference/android/os/AsyncTask

[28] Android Developers. *Data and file storage overview* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/guide/topics/data/data-storage

[29] Android Developers. *Content providers* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/guide/topics/providers/content-providers

[30] Android Developers. *Work with observable data objects* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/topic/libraries/data-binding/observability

[31] Android Developers. *LiveData Overview* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/topic/libraries/architecture/livedata

[32] Android Developers. *Recommended app architecture* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/jetpack/docs/guide#overview

[33] Tian Lou. *A Comparison of Android Native App Architecture – MVC, MVP and MVVM*. Espoo. 2016. Diplomová práce. Aalto University

[34] Android Developers. *Data Binding Library* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/topic/libraries/data-binding

[35] Square Inc. *Retrofit* [online]. [vid 23. 4. 2019]. Dostupné z: https://square.github.io/retrofit/

[36] Square Inc. *Interceptors* [online]. [vid 23. 4. 2019]. Dostupné z: https://github.com/square/okhttp/wiki/Interceptors

[37] Android Developers. *AccountManager* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/reference/android/accounts/AccountManager.html

[38] Google Developers. *Start integrating Smart Lock for Passwords into your Android app* [online]. [vid 23. 4. 2019]. Dostupné z: https://developers.google.com/identity/smartlock-passwords/android/get-started

[39] Oracle. *Primitive Data Types* [online]. [vid 23. 4. 2019]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

[40] Android Developers. *Integer* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/reference/java/lang/Integer

[41] Google. *Gson User Guide* [online]. [vid 23. 4. 2019]. Dostupné z: https://github.com/google/gson/blob/master/UserGuide.md#serializing-and-deserializing-generic-types

[42] Android Developers. *Slide between fragments using ViewPager* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/training/animation/screen-slide

[43] Wolfram Research, *Inc. Box-and-Whisker Plot* [online]. [vid 23. 4. 2019]. Dostupné z: http://mathworld.wolfram.com/Box-and-WhiskerPlot.html

[44] Mgr. Tomáš Svoboda. *Implementace statistické metody KDE+*. Brno. 2016. Diplomová práce. Vysoké učení technické v Brně

[45] The Data Visualisation Catalogue. *Violin Plot* [online]. [vid 23. 4. 2019]. Dostupné z: https://datavizcatalogue.com/methods/violin_plot.html

[46] Android Developers. *Content resolver – addPeriodicSync* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/reference/ android/content/ ContentResolver.html#addPeriodicSync(android.accounts.Account, %2520java.lang.String,%2520android.os.Bundle,%2520long)

[47] Android Developers. *Sending simple data to other apps* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/training/sharing/send#send-binary-content

[48] Google. *View & analyze your app's ratings & reviews* [online]. [vid 23. 4. 2019]. Dostupné z: https://support.google.com/googleplay/android-developer/answer/138230?hl=en

[49] Google Developers. *Firebase* [online]. [vid 23. 4. 2019]. Dostupné z: https://firebase.google.com/

[50] Google Developers. *Firebase Crashlytics* [online]. [vid 23. 4. 2019]. Dostupné z: https://firebase.google.com/docs/crashlytics/

[51] Google Developers. *View the Java heap and memory allocations with Memory Profiler* [online]. [vid 23. 4. 2019]. Dostupné z: https://developer.android.com/studio/profile/memory-profiler

[52] The Android Open Source Project. *BaseAdapter.java* [online]. [vid 23. 4. 2019]. Dostupné z: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/widget/BaseAdapter.java

[53] Energetický regulační úřad. *Energetický regulační věstník* [online]. [vid 23. 4. 2019]. Dostupné z: http://www.eru.cz/documents/10540/2887244/ERV_8_2017.pdf/96b4e385-52f3-48ac-a446-fc588182b5cb

[54] Ladislav Kotěšovec. *Technické a ekonomické posouzení variant skupinové a centrální kompenzace*. Brno. 2012. Diplomová práce. Západočeská univerzita v Plzni, Fakulta elektrotechnická

[55] Bc. Martin Synek. *Analýza účiníku v distribučních sítích*. Praha. 2014. Diplomová práce. České vysoké učení technické v Praze, Fakulta elektrotechnická

[56] doc. Ing. Jiří Drápela, Ph.D. *Výkony v třífázových sítích s obecně nesymetrickou a deformovanou proudovou a napěťovou soustavou* [online]. [vid 23. 4. 2019]. Dostupné z: http://www.ueen.feec.vutbr.cz/cz/images/stories/OPVK_ePower/MPQ1/Prezentace_MPQ1.pdf

Přílohy

I. Jalový výkon a kompenzace účiníku

Pro vytvoření aplikace, která má vizualizovat veličiny regulátoru jalového výkonu, je potřeba rozumět významu veličin, proto jsou v této kapitole jednoduše popsané. Nejedná o soupis všech veličin měřených regulátorem, ale pouze o základní přehled veličin, které jsou zobrazovány ve výsledné aplikaci.

Zdánlivý výkon (S) je definován jako součin efektivní hodnoty napětí a proudu. Podle rovnice 3 je patrné, že se zdánlivý výkon skládá ze 3 složek: činného výkonu, jalového výkonu a deformačního výkonu. Název veličiny napovídá, že vyjadřuje maximální dosažitelnou hodnotu činného výkonu při nulovém fázovém posunu φ mezi proudem a napětí.

$$S = U \cdot I \quad \textit{(Rovnice 3)} \qquad\qquad S = \sqrt{\left(P^2 + Q^2 + D^2\right)} \quad \textit{(Rovnice 4)}$$

Činný výkon (P) je složka zdánlivého výkonu, která se přenáší pouze od zdroje ke spotřebiči, kde následně koná užitečnou práci.

Jalový výkon (Q) se přesouvá cyklicky mezi zdrojem a spotřebičem a nekoná užitečnou práci. Jalový výkon vzniká vlivem indukčních a kapacitních prvků v obvodu, ve kterých vytváří magnetické či elektrické pole. Jalový výkon kapacitního charakteru vyjadřuje zpoždění napětí vůči proudu. U indukčního jalového výkonu je to obráceně.

Deformační výkon (D) hraje roli v obvodech s nelineárními spotřebiči, které deformují tvar průběhu proudu na neharmonický. Tento neharmonický průběh lze rozložit pomocí Fourierovy transformace na jednotlivé harmonické složky. Podíl vyšších harmonických složek v zdánlivém výkonu se označuje jako deformační a stejně jako jalový výkon nekoná užitečnou práci.

Účiník cos(φ) je bezrozměrná veličina, která je rovna kosinu fázového posunu φ mezi proudem a napětím základní harmonické složky. Nabývá pouze hodnoty od 0 do 1. U harmonických průběhu proudu a napětí je účiník cos(φ) roven

poměru mezi činným a zdánlivým výkonem. Pro neharmonické průběhy proudu a napětí, kde se vyskytuje deformační výkon, byl zaveden ještě tzv. skutečný účiník.

Skutečný účiník (**PF)** udává poměr mezi činným a zdánlivým výkonem i v obvodech s neharmonickými průběhy proudu a napětí. V případě, že je PF rovno 1 (ideální stav), tak je veškerý výkon činný, naopak pokud je PF=0, tak je veškerý výkon jalový či deformační. Jedná se tak o důležitý ukazatel efektivnosti elektrické soustavy.

$$PF=\frac{P}{S} \quad \textit{(rovnice 5)}$$

Kompenzace účiníku je snaha omezit přenos jalového a deformačního výkonu neboli dosáhnout hodnot účiníku blížící se hodnotě 1. Kompenzací účiníku se snižují ztráty na vedení a dochází k efektivnějšímu využití elektrické energie. Spotřebitelé vysokého (nad 1 kV) a velmi vysokého napětí jsou státem finančně penalizovány v případě nedodržení hodnot účiníku cos(φ) v rozmezí 0,95-1 induktivního charakteru [53].

[54], [55], [56]

II. Seznam použitých programovacích knihoven

1. Google Play Services Auth (https://developers.google.com/android/guides/setup)
2. Firebase Crashlytics (https://firebase.google.com/docs/crashlytics)
3. Firebase Performance (https://firebase.google.com/docs/perf-mon)
4. Gson (https://github.com/google/gson)
5. Retrofit 2 (https://square.github.io/retrofit)
6. Retrofit Gson Convertor (https://github.com/square/retrofit/tree/master/retrofit-converters/gson)
7. Okhttp3 Logging Interceptor (https://github.com/square/okhttp/tree/master/okhttp-logging-interceptor)
8. MPAndroidChart (https://github.com/PhilJay/MPAndroidChart)
9. Joda-Time (https://github.com/dlew/joda-time-android)
10. Simple Statful Layout (https://github.com/jakubkinst/Android-StatefulLayout)
11. Material Spinner (https://github.com/Moravec-Jan/MaterialSpinner)
12. SwitchDateTimePicker (https://github.com/Kunzisoft/Android-SwitchDateTimePicker)
13. LeakCanary (https://github.com/square/leakcanary)
14. JUnit (https://junit.org/junit5)
15. Mockito (https://site.mockito.org/)

III. Zdrojové kódy

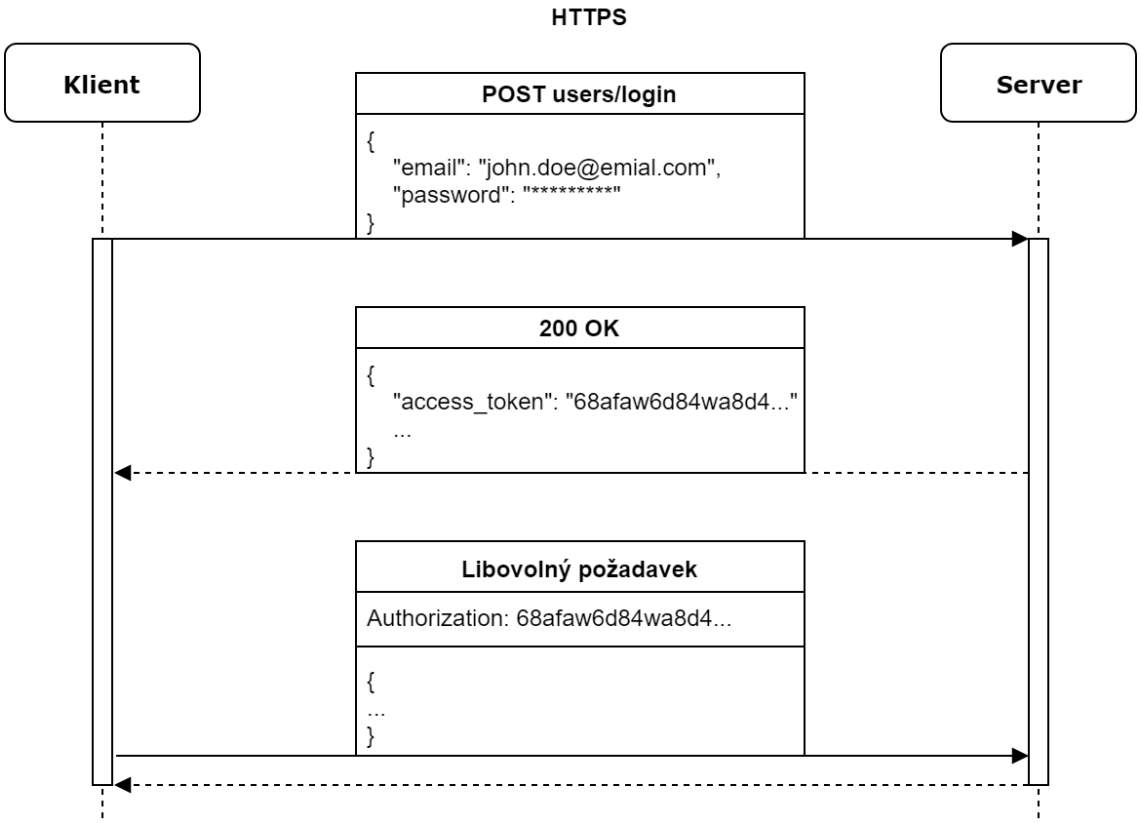
```
public interface RestService {

    @GET("data_minutes")
    Call<List<MeterData>> getMinutesData(@Query("time") UnixRange from,
    @Query(value = "tags", encoded = true) String tag);

    @GET("data_hours")
    Call<List<MeterData>> getHoursData(@Query("time") UnixRange from,
    @Query(value = "tags", encoded = true) String tag);

    @GET("variables")
    Call<List<Quantity>> getQuantities();

    @GET("devices/variables/{id}")
    Call<List<Quantity>> getQuantitiesForDevice(@Path("id") int id);

    @GET("devices")
    Call<List<Device>> getDevices();

    @GET("tags")
    Call<List<Tag>> getTags();

    @DELETE("tags/{id}")
    Call<Void> removeTag(@Path("id") int id);

    @GET("tag_relations")
    Call<List<TagRelation>> getTagRelations();

    @GET("users")
    Call<List<User>> getUsers();

    @POST("users/login")
    Call<LoginInfo> loginUser(@Body UserCredentials userCredentials);

    @GET("users/refresh")
    Call<LoginInfo> refreshToken();

    @GET("device_groups")
    Call<List<DeviceGroup>> getDeviceGroups();

    @POST("device_groups")
    Call<DeviceGroup> addDeviceGroup(@Body DeviceGroupAddData deviceGroup);

    @DELETE("device_groups/{id}")
    Call<Message> removeDeviceGroup(@Path("id") int id);

    @GET("alarms")
    Call<List<Alarm>> getAllAlarms();

    …
}
```

Zdrojový kód 2: Část zdrojového kódu rozhraní RestService

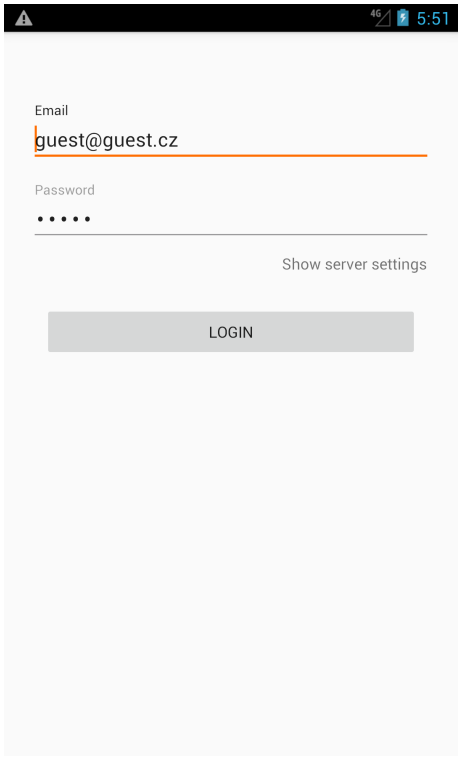
```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>
        <variable
            name="itemViewModel"
            type="cz.tul.ev.activity.device.DeviceItemViewModel" />
    </data>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

            <TextView
                android:id="@+id/device_name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@{itemViewModel.title}"
                android:textSize="18sp" />

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@{itemViewModel.serialNumber}"
                android:textSize="14sp" />
        </RelativeLayout>
    </LinearLayout>
</layout>
```

Zdrojový kód 3: XML soubor propisující vzhled položky zařízení s využitím data bindingu

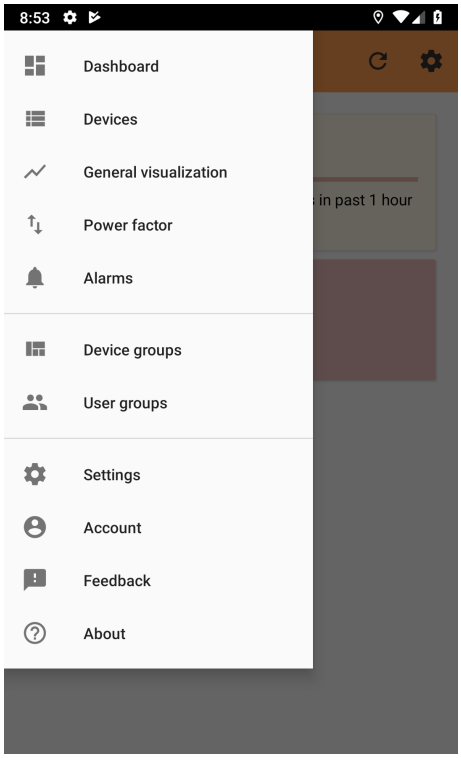
IV. Obrázky a snímky obrazovky



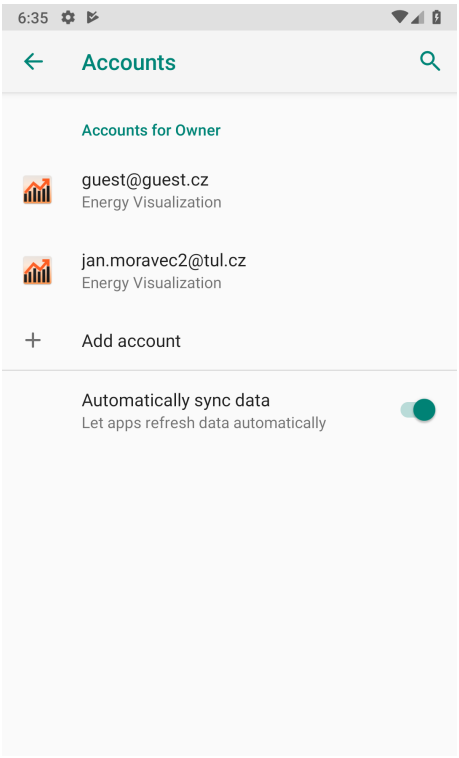
Ilustrace 18: Sekvenční diagram úspěšné autentizace



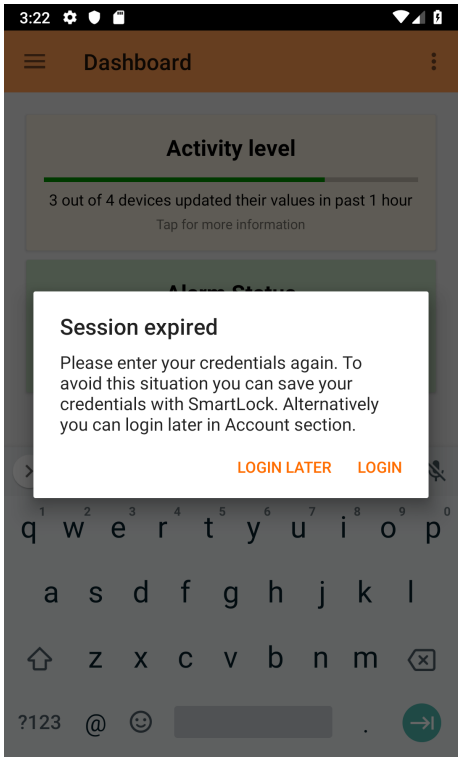
Ilustrace 19: Přihlašovací obrazovka



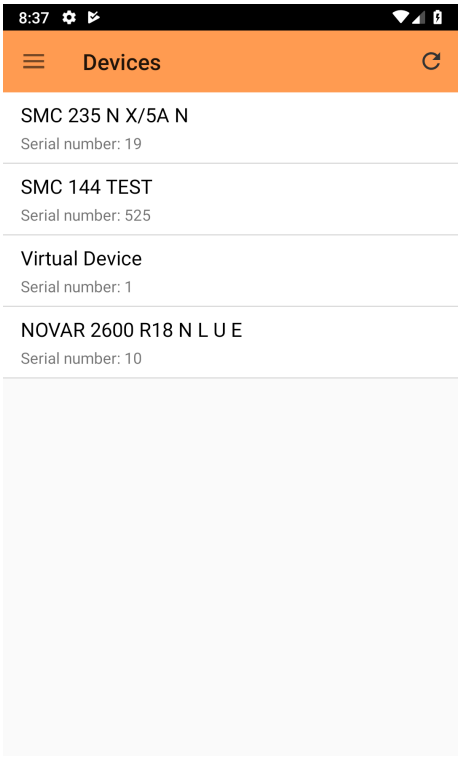
Ilustrace 20: Navigační panel aplikace



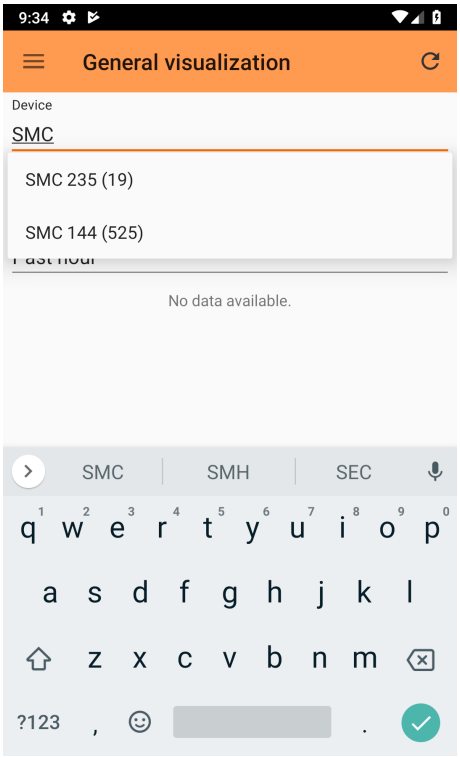
Ilustrace 22: Správa účtů aplikace v nastavení OS Android



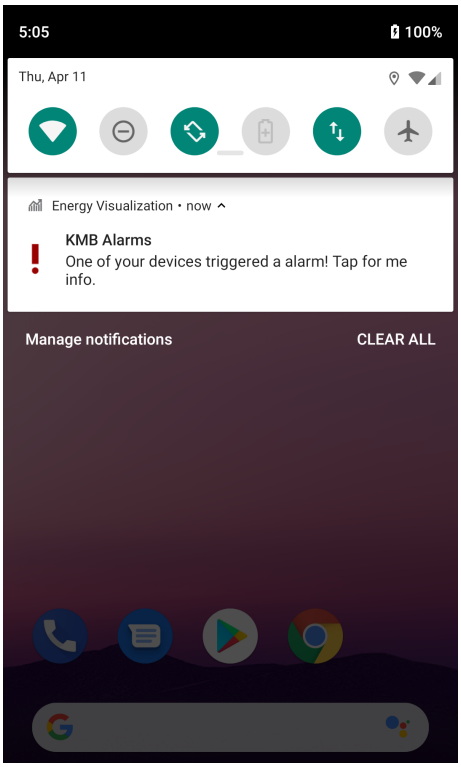
Ilustrace 21: Dialog oznamující expiraci tokenu



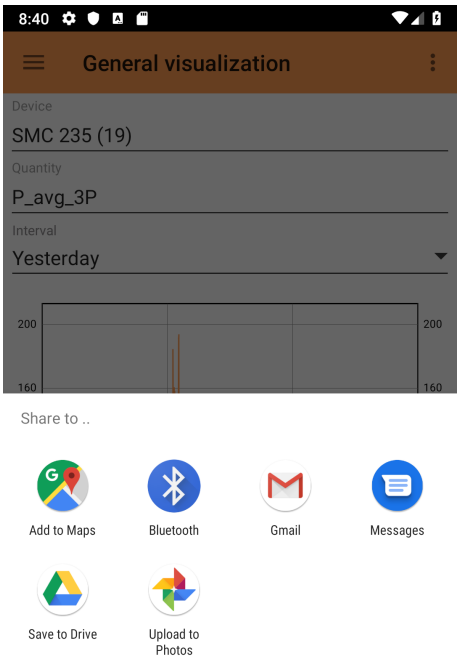
Ilustrace 23: Aktivita se seznamem zařízení



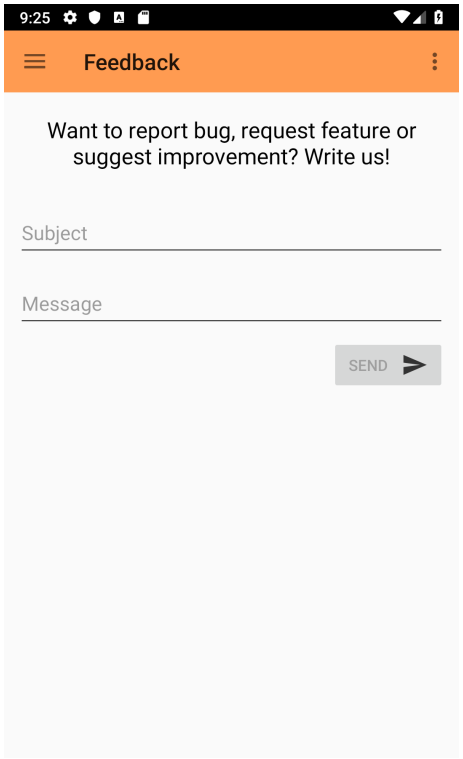
Ilustrace 24: Grafická komponenty pro výběr zařízení



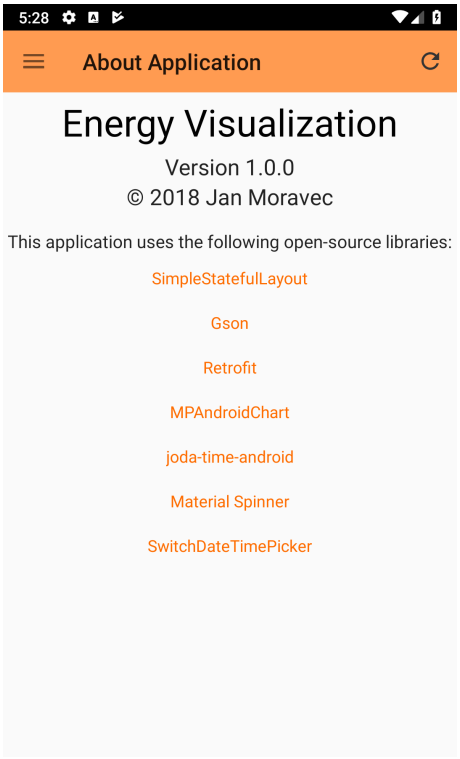
Ilustrace 25: Notifikace informující o aktivním alarmu



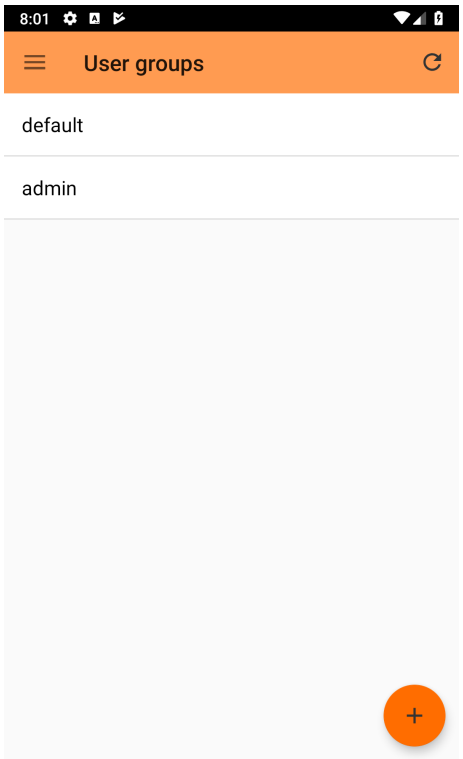
Ilustrace 26: Sdílení dat s ostatními aplikacemi



Ilustrace 27: Aktivita pro zpětnou vazbu uživatelů



Ilustrace 28: Aktivita s informacemi o aplikaci



Ilustrace 29: Správa uživatelských skupin