VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Andrej Masár

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

MOBILNÍ APLIKACE PRO OVLÁDÁNÍ PRVKŮ CHYTRÉ DOMÁCNOSTI
MOBILE APPLICATION FOR CONTROLLING SMART HOME DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Andrej Masár**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jiří Hošek, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Andrej Masár ***ID:*** 175208
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Mobilní aplikace pro ovládání prvků chytré domácnosti

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce bude návrh a vývoj nativní mobilní aplikace Android, která bude sloužit jako rozšiřující rozhraní pro ovládání prvků chytré domácnosti. Pro komunikaci mezi senzory bude využit protokol MQTT, v rámci něhož bude definována jednotná struktura a forma zpráv. Uživatelská část (front-end) mobilní aplikace bude umožňovat kompletní správu (přidávání, odebírání, ovládání) připojených senzorů či dalších chytrých zařízení. Data získaná z chytrých senzorů budou ukládána v databázi IoT brány (vývoj není součástí tohoto zadání) a bude možné je vizualizovat v reálném čase.

DOPORUČENÁ LITERATURA:

[1] LIBERG, Olof, Mårten SUNDBERG, Y.-P. Eric WANG, Johan BERGMAN a Joachim SACHS, [2018]. Cellular Internet of things: technologies, standards, and performance. San Diego, CA, United States: Academic Press, an imprint of Elsevier. ISBN 978-012-8124-581.

[2] BOSWARTHICK, David, Omar ELLOUMI a Olivier HERSENT. 2012. M2M communications: a systems approach. Hoboken, N.J.: Wiley, xxiii, 308 p. ISBN 978-1-119-99475-6.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: doc. Ing. Jiří Hošek, Ph.D.
Konzultant: Franz Kröpfl (A1 Telekom Austria Group)

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Práca sa zaoberá problematikou internetu vecí a jeho aplikácií v domácnosti. Popisuje návrh a implementáciu natívnej mobilne aplikácie pre Android na ovládanie prvkov inteligentnej domácnosti. Aplikácia komunikuje s domácnosťou pomocou protokolu MQTT. Výsledná aplikácia demonštruje ovládanie zariadení v inteligentnej domácnosti podľa konvencie MQTT2GO.

KĽÚČOVÉ SLOVÁ

Android, inteligentná domácnosť, internet vecí, mobilná aplikácia, MQTT

ABSTRACT

The objective of this thesis is the issue of the Internet of Things and its application as a Smart Home. The work describes the design and implementation of a native mobile application for Android to control the elements of a Smart Home. The application communicates with the Smart Home using the MQTT protocol. The resulting application demonstrates the control of devices in a Smart Home according to the MQTT2GO convention.

KEYWORDS

Android, Internet of Things, mobile application, MQTT, Smart Home

MASÁR, Andrej. *Mobilní aplikace pro ovládání prvků chytré domácnosti*. Brno, 2021, 60 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: doc. Ing. Jiří Hošek, Ph.D.

VYHLÁSENIE

Vyhlasujem, že svoju diplomovú prácu na tému „Mobilní aplikace pro ovládání prvků chytré domácnosti“ som vypracoval samostatne pod vedením vedúceho diplomovej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi doc. Ing. Jiřímu Hošekovi, Ph.D. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Zoznam obrázkov

Úvod

Človek je v dnešnom svete obklopený inteligentnými zariadeniami schopnými spolu komunikovať, automatizovať a riadiť určité časti nášho života. Určitým trendom sa stalo využívanie inteligentných zariadení v domácnostiach, alebo akýchkoľvek priestoroch, kde sa ľudia často nachádzajú. Pomocou týchto zariadení je možné urýchliť, zjednodušiť a automatizovať bežné činnosti. So súčasnou dostupnosťou internetu je možné k inteligentným zariadeniam pristupovať vzdialene prakticky odkiaľkoľvek. To dáva človeku obrovskú možnosť kontroly nad priestormi, kde ich využíva. Správnym nastavením a automatizáciou týchto zariadení je navyše možné predísť rôznym nechceným situáciám, či už sa jedná o zdravie, alebo materiálne škody.

Táto práca sa zaoberá pojmom „Internet vecí“, inteligentnými domácnosťami, protokolom Message Queue Telemetry Transport (MQTT), návrhom a implementáciou natívnej mobilnej aplikácie pre systém Android. Práca je koncipovaná tak, že začína teoretickým úvodom, ktorý rozoberá pojem internet vecí, jeho architektúru, komunikačné modely a protokoly. Pozornosť je v tejto problematike upriamená na aplikačné protokoly používané pri komunikácií inteligentných zariadení. Podrobnejšie je rozobraté fungovanie protokolu MQTT, ktorý sa používa aj pri implementácií výslednej mobilnej aplikácie.

Tretia kapitola sa zaoberá inteligentnými domácnosťami, ich súčasťami a rôznymi použitiami. Je v nej popísaná architektúra ako sú inteligentné zariadenia riadené a dostupné ich užívateľovi. Kapitola popisuje aj populárne existujúce riešenia, ich prístup k problematike a výslednú funkcionalitu.

Praktická časť tejto práce začína návrhom a prípravou na tvorbu aplikácie. Sú tam stručne popísané procesy pred začatím implementácie, výber architektúry aplikácie a jej kľúčové prvky.

Poslednou kapitolou je samotná implementácia aplikácie, kde sú popísané použité technológie a prvky natívnej aplikácie pre Android. Postup pri implementácií je rozdelený na dve vývojové iterácie. Pred popisom každej vývojovej etapy je zhrnutý bodový zoznam jej cieľov. Popis vývoja je logicky členený na vlastné dôležité komponenty aplikácie a jednotlivé časti aplikácie, ktoré jej užívateľ vidí. Pri popise týchto častí sú často vypichnuté určité procesy, ktoré sa dejú na pozadí užívateľskej interakcie.

1 Internet vecí

Z anglického „Internet of Things“ (IoT) je technológia prepojenia objektov a ich schopnosť spolu komunikovať, teda posielať, získavať dáta a reagovať získané informácie v podobe ďalších (autonémnych) činností bez nutnosti zásahu človeka [1]. V súčasnosti sa technológia teší značnej popularite a je používaná v rôznych aplikáciach, medzi ktoré patria:

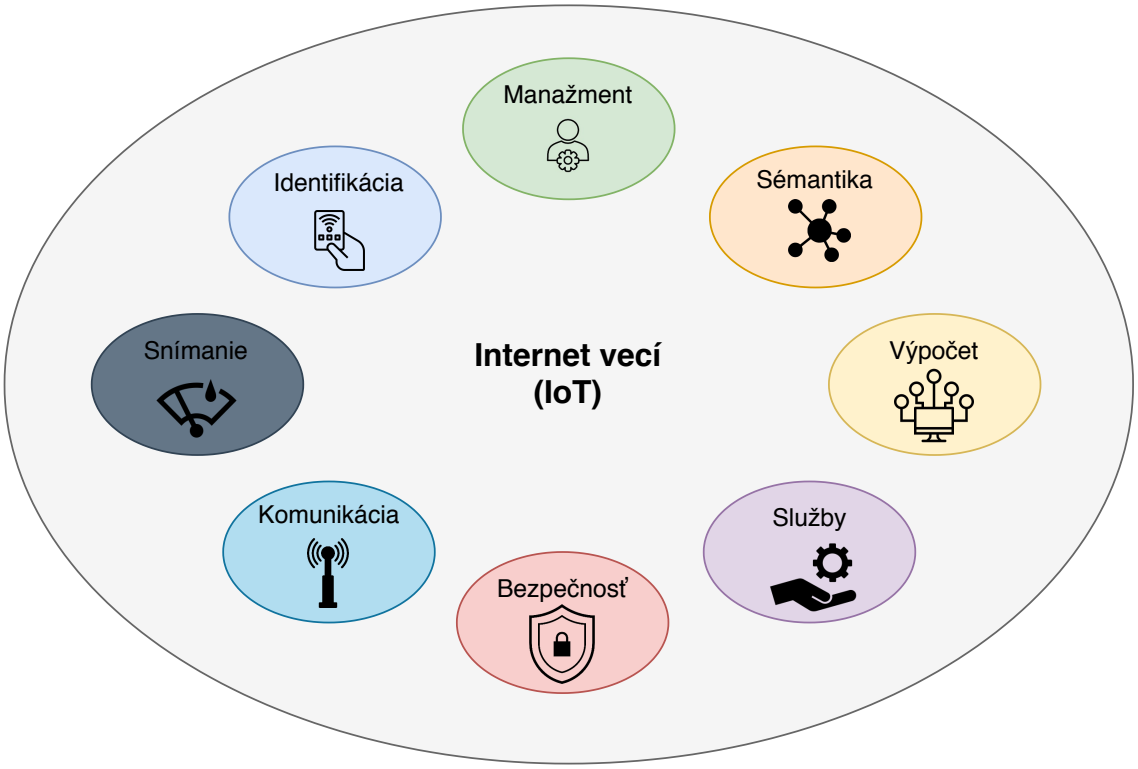
- **Nositeľná elektronika** - ide o inteligentné hodinky, okuliare, telefóny, GPS zariadenia, ktoré spolu komunikujú, pomocou senzorov merajú životné funkcie, polohu, pohyb a rôzne ďalšie informácie.
- **Inteligentná domácnosť** - zahrňuje bežné zariadenia v domácnosti rozšírené o možnosti komunikácie a ovládania na diaľku. Táto téma bude rozobraná podrobnejšie v samostatnej kapitole 3.
- **Monitorovanie dopravy** - semafory, vozidlá hromadnej dopravy, vlaky, výhybky a ďalšie. Pomocou senzorov na získavanie polohy umožňujú dispečerom riadiť plynulosť dopravy v mestách, komunikáciach. Vozidlá hromadnej dopravy môžu byť uprednostňované svetelnými znameniami na križovatkách, ako aj nevidiaci čakajúci na prechodoch. Aplikácie môžu dať šoférom informácie o zápchach, či nehodách na cestách.
- **Poľnohospodárstvo** - kde existujú senzory na získavaní informácií o pôde. Teplota, vlhkosť, kyslosť, prítomnosť minerálov. Technológia je využitá aj v strojoch, ktoré pôdu obrábajú. Stroje nie je nutné riadiť priamo a je možné ich nechať pracovať samostatne.
- **Nemocnice** - použitím elektronický kľúčov, kariet je možné zjednodušiť prístup personálu do jednotlivých častí, ku ktorým majú prístup. Diagnostické zariadenia odosielajú informácie o pacientoch a urýchľujú zásahy doktorov pri náhlom zhoršení stavu pacientov.
- **Inžinierske siete** - použitím senzorov komunikujúcich cez internet je možné včas zachytiť úniky, prerušenie na konkrétnom úseku. Správami o aktuálnej spotrebe je možné lepšie nastaviť výrobu a skladovanie konkrétnych surovín, čím sa šetrí čas aj peniaze.
- **Výroba** - zahrňuje výrobu pomocou liniek a iných strojov. Senzory zachytávajú teplotu, pohyb, poruchy, opotrebenie, čím môžu predísť škodám na zariadení ako aj úrazom obsluhujúceho personálu. Informácie, ktoré zasielajú pomáhajú manažmentu optimalizovať výrobné procesy.

Technológia teda pozostáva z mnohých rozličných komponentov, ktoré poskytujú heterogénny prístup k informáciám a ich spracovaniu. Prudký nárast používania technológie zvyšuje nároky na rýchlosť a stabilitu siete v ktorou sú zariadenia spojené. Veľký dôraz je nutné klásť na zabezpečenie prístupu k jednotlivým zariadeniam, ich

komunikácií a dátam, ktoré často obsahujú citlivé informácie o ľuďoch, či firemných tajomstvách [1].

1.1 Komponenty Internetu vecí

Na pochopenie fungovania technológie je lepšie ju rozdeliť na menšie bloky (Obrázok 1.1), ktoré budú popísané v nasledujúcich podkapitolách.



Obr. 1.1: Osem elementov Internetu vecí.

Unikátna identifikácia

Aby bola možná komunikácia s každým zariadením je nutné zaistiť ich unikátnosť. Na jej zaistenie je možné použiť viacero metód, ako elektronické produktové kódy (Electronic product code) alebo všadeprítomný kód (Ubiquitous codes). Pri adresovaní objektov Internetu vecí je dôležité rozlišovať medzi identifikátorom objektu a jeho adresou. Identifikátor objektu môže byť napríklad meno ako „kamera1“, ktoré označuje optický senzor. Adresa objektu slúži na adresáciu objektu v komunikačnej sieti. Kombináciou oboch je umožnená identifikácia objektov naprieč viacerými súkromnými sieťami [2].

Snímanie

Predstavuje zbieranie dát z objektov v sieti a ich odosielanie do dátového úložiska, ktoré reprezentuje databáza, alebo cloud. Zozbierané dáta sú potom analyzované pre potrebu vykonania špecifickej akcie definovanej službou. Na zber dát je možné použiť prakticky akékoľvek zariadenie vybavené jedným, alebo viacerými senzormi, rozhraním na pripojenie do siete, mikro-počítačom implementujúcim komunikačný protokol a modulom pre zabezpečenie [2].

Komunikácia

Zahrňuje prepojenie jednotlivých objektov Internetu vecí na zabezpečenie špecifickej služby. Typicky by zariadenia mali používať komunikačné technológie s nižším výkonom, aby sa zamedzilo vzájomnému rušeniu jednotlivých zariadení [2]. Medzi tieto technológie patria:

- **WiFi** - bezdrôtová sieť v súčasnosti založená na špecifikácií IEEE 802.11.
- **Bluetooth** - otvorený štandard pre bezdrôtovú komunikáciu. Vo verzií 4.1 poskytuje nízku energetickú náročnosť a vysokú prenosovú rýchlosť.
- **Ethernet** - súhrnná káblová technológia najčastejšie používaná v počítačových sietiach. Môže byť realizovaná krútenou dvojlinkou, optickým vláknom, alebo koaxiálnym káblom.
- **ZigBee** - bezdrôtová komunikácia postavená na štandarde IEEE 802.15.4. Je priamo určený na spojenie nízko-výkonových zariadení do 75 metrov.
- **Z-Wave** - bezdrôtová technológia špecificky vyvinutá pre inteligentné riešenia ovládania domácnosti, domov a budov.
- **Radio-Frequency IDentification (RFID)** - technológia určená na identifikáciu objektov, postavená na rádiovej frekvencií.
- **Ultra-Wideband (UWB)** - bezdrôtová technológia s nízkou úrovňou signálov a veľkou šírkou pásma. Je určená na komunikáciu senzorov.
- **Long-Term Evolution (LTE)** - vysoko-rýchlostná technológia na poskytovanie internetového pripojenia prostredníctvom mobilných sietí.
- **5G siete** - v súčasnosti najnovšia bezdrôtová technológia na báze mobilných sietí, s extrémne nízkou latenciou, vysokým dátovým prenosom a lepším zabezpečení kvality služieb v porovnaní s LTE [3].
- **SIGFOX** - bezdrôtová technológia určená pre zariadenia, ktoré neustále odosielajú malé množstvá dát.
- **WiMAX** - technológia bezdrôtovej komunikácie, na vzdialenosť až 70 kilometrov, popísaná v norme IEEE 802.16.
- **HomePlug** - umožňuje prenos dát cez elektrickú sieť. Splňuje štandard šifrovania AES-128.

- **Home Phoneline Networking Alliance (HomePNA)** - poskytuje vysokorýchlostný prenos dát po telefónnych rozvodoch.
- **Near Field Communication (NFC)** - založená na štandardoch RFID s integráciou ISO/IEC 14443. Je určená na veľmi krátku vzdialenosť od štyroch centimetrov.

Výpočet

Reprezentuje kombináciu výpočtových jednotiek (mikrokontroléry, mikroprocesory, FPGA jednotky) a softvérových aplikácií, ktoré na týchto jednotkách prevádzajú výpočty. Existuje viacero softvérových platforiem poskytujúcich potrebnú funkcionalitu pre Internet vecí. Príklady operačných systémov schopných pracovať na malých zariadenia sú: Amazon FreeRTOS, Windows 10 IoT, Nucleus RTOS, Wind River VxWorks, Contiki RTOS, TinyOS [2].

Ďalšou dôležitou výpočtovou časťou pre Internet vecí sú Cloud platformy. Nie len že zabezpečujú priestor kam sa dáta z objektov ukladajú, ale sú schopné ich spracovávať v reálnom čase a poskytovať súhrnné informácie z celej kolekcie dát.

Služby

Služby Internetu vecí je možné rozdeliť do štyroch kategórií: služby spojené s identifikáciou, služby zhromažďovania informácií, služby zabezpečujúce spoluprácu objektov, všadeprítomné služby. Služby spojené s identifikáciou sú základnou funkcionalitou poskytujúcou prenos reálnych objektov do virtuálnej podoby. Služby zhromažďovania informácií zbierajú a sumarizujú surové dáta zo senzorov, ktoré majú byť spracované a odoslané do výpočtovej časti Internetu vecí. Služby zabezpečujúce spoluprácu objektov pracujú so zhromaždenými dátami. Zabezpečujú rozhodovaciu logiku, teda správne reakcie na odoslané dáta. Všadeprítomné služby sa zameriavajú na poskytovanie služieb spolupráce kedykoľvek sú potrebné, ktokoľvek ich potrebuje a to kdekoľvek. Podľa týchto kategórií je možné zaradiť jednotlivé aplikácie Internetu vecí. Najvyšším cieľom aplikácií je dosiahnutie kategórie všadeprítomných služieb [2].

Sémantika

Je schopnosť inteligentne extrahovať informácie rôznych zariadení na dosiahnutie potrebnej služby. Extrahovanie zahrňuje nájdenie, použitie daných zdrojov a modelovanie informácií. Sémantika tiež zahrňuje rozpoznanie a analýzu dát na vykonanie správneho rozhodnutia za účelom poskytnutia služby [2].

Manažment

Cieli na správu užívateľov, ich role, a možnosti prístupu k službám Internetu vecí.

Management užívateľov zahrňuje ich pridávanie a odoberanie zo skupín užívateľov. Pomocou skupín je možná adresácia a kontrola väčšieho množstva užívateľov. Manažment má tiež za úlohu sledovať užívateľskú aktivitu [4].

Bezpečnosť

Tento element rieši zabezpečenie jednotlivých zariadení, ich komunikácie a služieb Internetu vecí pred modifikáciou a neovereným prístupom. Pri zariadeniach je to chránenie objektov prepojených sieťou pred útočníkom. Typický životný cyklus objektu Internetu vecí sa skladá z: načítania firmvéru, inicializácie (nadviazanie spojenia a pripravenie na zber dát), prevádzka (očakávaná funkcionalita objektu), a aktualizácia (inštalácia nového firmvéru a následný reštart). Celý tento proces by mal byť zabezpečený pomocou bezpečnostných algoritmov. V prípade komunikácie je zaistená ochrana koncových bodov komunikačného kanálu medzi objektmi. Väčšina objektov v systémoch Internetu vecí používa bezdrôtové komunikačné technológie, ktoré sú viac náchylné na rôzne typy útokov. Zabezpečenie služieb aplikácie zahrňuje ochranu dát pred neoprávneným prístupom, alebo modifikáciou [4].

1.2 Architektúra Internetu vecí

Z hľadiska architektúry bolo predstavených viacero modelov na lepšie pochopenie konceptu Internetu vecí. Medzi najviac používané modely patrí troj respektíve päť vrstvová architektúra (Obrázok 1.2), ktorá len podrobnejšie rozdeľuje aplikačnú vrstvu [4].

Vrstva vnímania

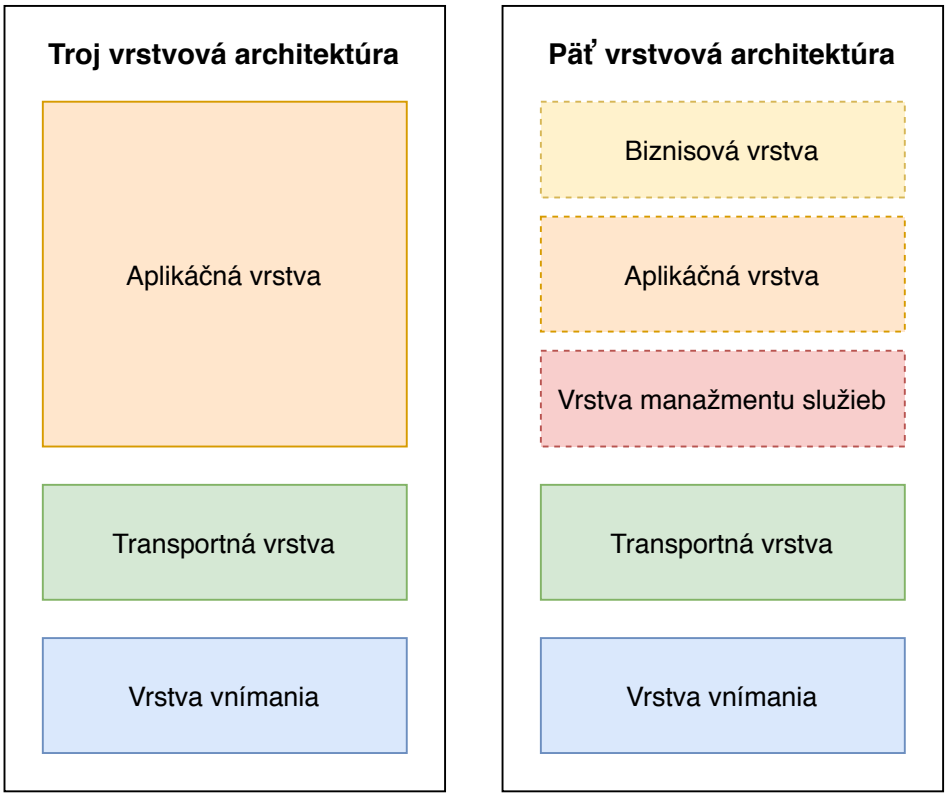
Je súbor objektov Internetu vecí. Objekty slúžia ako prostredník medzi skutočným svetom a digitálnym svetom pomocou senzorov. Hlavnou úlohou tejto vrstvy je získavanie dát z reálneho prostredia, ako teplota, vlhkosť, prítomnosť plynu, GPS (Global Positioning System), svetlo, a ďalšie, za použitia rôznych typov senzorov [4].

Transportná vrstva

Táto vrstva má na starosť prepojenie objektov a zdieľanie informácií medzi objektami bezpečným spôsobom. Redukcia spotreby energie na komunikáciu, zabezpečenie kvality služieb, schopnosť adaptovať sa na dynamické topológie, sú príklady problémov, ktoré rieši táto vrstva [4].

Vrstva manažmentu služieb

Je tiež označovaná ako middleware vrstva, ktorá uľahčuje použitie heterogénnych zariadení v aplikáciach Internetu vecí. Táto vrstva má tiež na starosť spracovanie



Obr. 1.2: Troj vrstvová architektúra vľavo a päť vrstvová architektúra na obrázku vpravo.

surových dát zozbieraných objektmi vo vrstve vnímania. Zozbierané dáta sú často v nevhodných jednotkách, z čoho je ťažké jednoducho zistiť potrebnú informáciu (čas od posledného zaznamenaného pohybu na kamere v milisekundách) [4].

Aplikačná vrstva

Táto vrstva je zodpovedná za poskytovanie aplikáciou špecifickú službu koncovému užívateľovi. Jeho súčasťou je teda užívateľské rozhranie, ktoré vhodným spôsobom žiadanú službu poskytne [4].

Biznisová vrstva

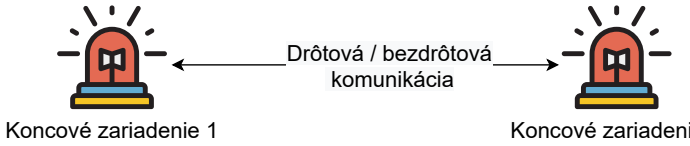
Dohliada na operácie a rôzne služby v systéme Internetu vecí. Vytvára biznis modely, grafy, tabuľky, zo surových dát ostatných vrstiev. Táto vrstva je zodpovedná za analýzu, monitorovanie, vyhodnocovanie výsledkov systému a jeho pridružených elementov. Robenie rozhodnutí je jedna z hlavných aktivít biznisovej vrstvy [4].

1.3 Komunikačné modely Internetu vecí

Existuje viacero modelov ako komunikácia systéme Internetu vecí prebieha. Od jednoduchých, v ktorých sa vyskytujú len koncové zariadenia, až po modely s viacerými cloudovými službami na rôznych miestach.

Komunikácia zariadenia so zariadením (device-to-device)

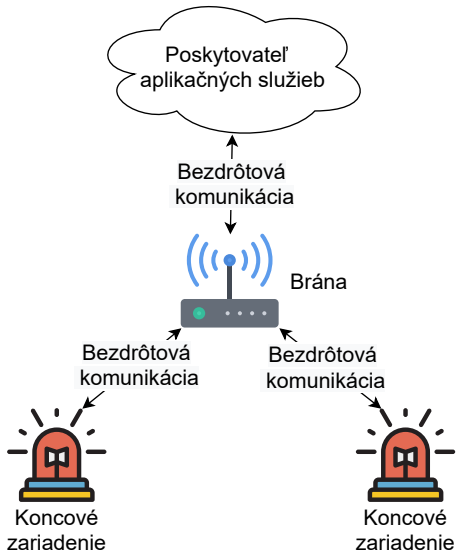
V tomto modely prebieha komunikácia zariadení priamo medzi sebou. Model používa drôtové aj bezdrôtové štandardy komunikácie. Obrázok 1.3 zobrazuje ako komunikácia funguje. Tento model hrá veľkú úlohu v industriálnej automatizácií.



Obr. 1.3: Model komunikácie koncového zariadenia s koncovým zariadením.

Komunikácia zariadenia s bránou (device-to-gateway)

Brány v tomto modely sú zariadenia so softvérom aplikačnej vrstvy a slúžia ako prepojenie medzi koncovými zariadeniami a poskytovateľmi aplikačných služieb. Model je ilustrovaný na obrázku 1.4. Tento model komunikácie sa veľmi často používa v aplikáciach inteligentnej domácnosti, športu a fitnes, nositeľnej elektroniky, a iných.



Obr. 1.4: Model komunikácie koncového zariadenia s bránou.

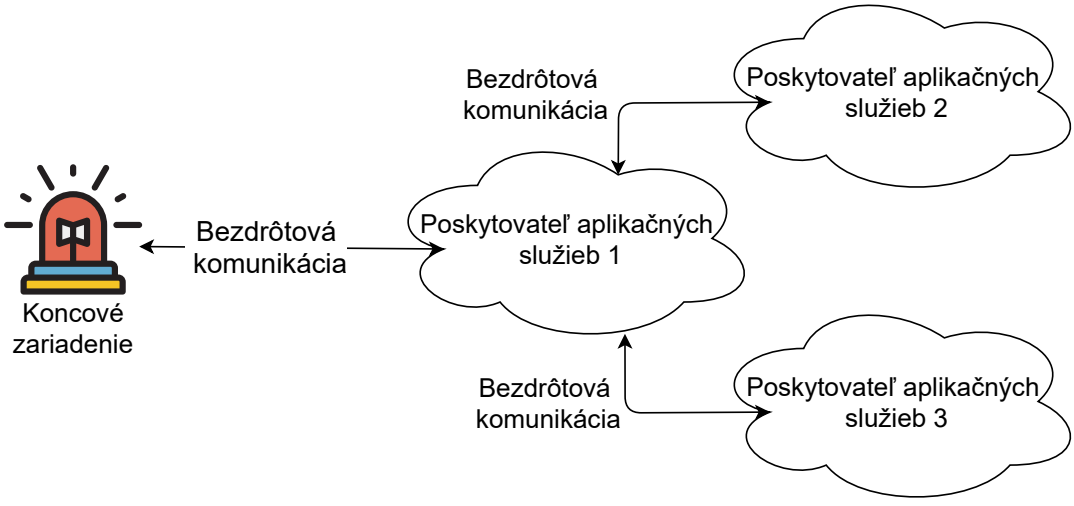
Komunikácia zariadenia s cloud-om (device-to-cloud)

V tomto komunikačnom modely sú koncové zariadenia priamo prepojené na cloud, ktorý poskytuje aplikačné služby. Komunikácia je obojstranná, kvôli možnosti ovládania zariadení (Obrázok 1.5).

Obr. 1.5: Model komunikácie koncového zariadenia s cloud-om.

Model zdieľania dát na zadnom konci (back end data sharing model)

Tento model umožňuje overeným aplikáciam tretích strán získavať dáta senzorov z cloud-u na agregáciu, alebo analýzu. Tým je možné dedikovať rôzne služby aplikácie Internetu vecí medzi viacero poskytovateľov (Obrázok 1.6).



Obr. 1.6: Model šírenia dát na zadom konci.

2 Protokoly aplikačnej vrstvy Internetu vecí

Aplikačná vrstva je najvyššia vrstva troj vrstvovej architektúry a poskytuje aplikačné služby užívateľovi pomocou iniciovania komunikácie medzi koncovými zariadeniami. Táto vrstva je zodpovedná aj za formátovanie a prezentovanie dát koncovému používateľovi. Pre zabezpečenie tejto funkcionality a pre potrebu vyššej efektivity komunikácie boli vyvinuté nové aplikačné protokoly [4]. Tu je uvedenie viacerých príkladov:

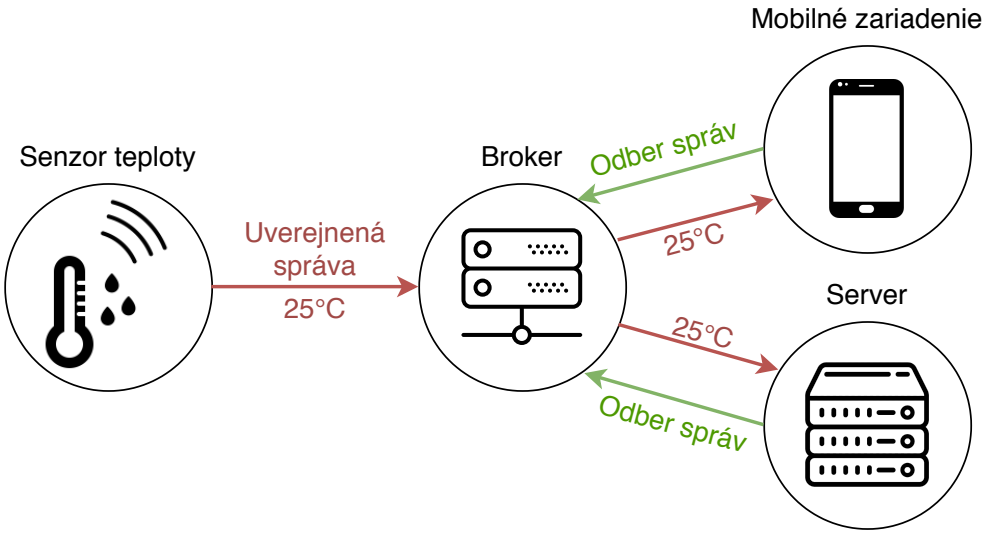
- **CoAP** (Constrained Application Protocol) - definuje webový prenosový protokol postavený na technológií REST (Representational state transfer) a HTTP (Hypertext Transfer Protocol) funkcionalite. Na rozdiel od REST využíva CoAP UDP (User Datagram Protocol) protokol, ktorý sa viac hodí na použitie v aplikáciach Internetu vecí. HTTP funkcionalita je tiež modifikovaná aby spĺňala požiadavky aplikácie Internetu vecí, ako nízka spotreba energie, fungovanie so zašumeným a vypadávajúcim pripojením. Je výhodný pre zariadenia s obmedzeným výkonom napájania a pamäťou.
- **XMPP** (Extensible Messaging and Presence Protocol) - je protokol využívajúci decentralizovanú serverovú a klientskú časť aplikácie. Podporuje asynchrónnu aj synchrónnu komunikáciu pomocou správ. Telo správ je vo formáte XML (Extensible Markup Language), čo je výhodné na prenos textových správ, ale nevýhodné z hľadiska šetrenia energie.
- **DDS** (Data Distribution Service) - je asynchrónny protokol pre komunikáciu zariadenia so zariadením (machine-to-machine) v reálnom čase. Vďaka tomuto modelu komunikácie má oproti ostatným protokol veľmi nízku latenciu. Ponúka široké nastavenia na zaistenie kvality služby, ako aj pokročilé bezpečnostné prvky.
- **MQTT** (Message Queue Telemetry Transport) - tento protokol je podrobne popísaný v sekcií 2.1.

2.1 Message Queue Telemetry Transport (MQTT)

Je protokol správ predstavený v roku 1999 a štandardizovaný v roku 2013. Bol vyvinutý ako protokol používajúci minimálne zdroje, aby bol efektívne použiteľný na malých zariadeniach (mikrokontroléroch) a s dátovo malými hlavičkami správ, pre optimalizovanie použitia šírky pásma pripojenia. Poskytuje obojstrannú komunikáciu od koncového zariadenia k serveru a naspäť [5], [6].

2.1.1 Koncept uverejňovania a odoberania správ v komunikácií protokolom MQTT

Protokol používa systém komunikácie odlišný od tradičnej komunikácie medzi klientom a serverom. Model uverejňovania/odoberania oddeľuje zariadenie, ktoré správu odosiela a zariadenia, ktoré správu dostávajú (Obrázok 2.1). Odosielateľ a prijímateľ správ spolu nikdy nekomunikujú priamo, väčšinou nevedia či na druhej strane komunikácie nejaké zariadenie existuje. Prepojenie zariadení zabezpečuje samostatná komponenta broker (jednateľ) [5].



Obr. 2.1: Komunikácia protokolom MQTT za použitia uverejňovania a odoberania správ.

Oddelenie odosielateľa a prijímateľa správ má viaceré aspekty:

- **Priestorové oddelenie** - odosielateľ a prijímateľ sa nemusia poznať, nemusia si navzájom vymeniť IP adresu a port cez ktoré komunikujú.
- **Časové oddelenie** - zariadenia nemusia komunikovať v rovnakom čase.
- **Oddelenie synchronizácie** - operácie na oboch zariadeniach nemusia byť prerušené počas komunikácie.

2.1.2 Pojmy MQTT klient a broker

Pretože je komunikácia jednotlivých klientov priestorovo oddelená, o všetky prepojenia sa v MQTT komunikácií stará broker. Pod pojmom MQTT klient rozumieme aj odosielateľov aj prijímateľov komunikácie. Funkcionalita odosielania a prijímania správ môže a často je implementovaná v rovnakom klientovi. Z hľadiska hardvéru môže byť MQTT klientom akékoľvek zariadenie, od malých mikro-kontrolórov, až

po veľké servery, ktoré je schopné komunikovať MQTT protokolom a sieťovo sa pripojiť k MQTT borkeru. Implementácia klienta v MQTT komunikácií je priamočiara a škálovateľná, čo umožňuje ju nasadiť na naozaj malé zariadenia s obmedzenými zdrojmi [7].

Kľúčové zariadenie v komunikácií pomocou protokolu MQTT je broker (jednateľ). Hlavná úloha komponenty je filtrovať všetky prichádzajúce správy a distribuovať ich iba k zariadeniam, ktoré dané správy odoberajú. V závislosti od veľkosti systému môže jeden broker spravovať až milióny súbežne pripojených klientov. Broker tiež ukladá dáta všetkých klientov s perzistentným pripojením, ako informácie o aktívnych odberoch a správach, ktoré nemohli byť doručené pri výpadku spojenia. Broker má za úlohu aj autentifikáciu a autorizáciu jednotlivých klientov. Pri väčších systémoch býva táto úloha delegovaná na samostatnú časť systému, kedy je využitá rozšíriteľnosť brokeru, ako aj možnosť vlastnej implementácie autentifikácie a autorizácie. Delegovanie a integrácia brokeru do robustnejšieho systému je dôležitá, keďže broker musí riešiť odber a filtrovanie správ pre všetkých klientov a musí zostať jednoducho monitorovateľný, rozšíriteľný a odolný voči poruchám [7].

2.1.3 Prepojenie zariadení v MQTT komunikácií

MQTT protokol patrí do rodiny protokolov TCP/IP. Zariadenia, ktoré chcú protokol používať, musia podporovať sadu protokolov z tejto rodiny. Jediná možnosť prepojenia zariadení je prepojenie jedného klienta a brokeru. Dvaja MQTT klienti spolu nikdy nie sú prepojený priamo. Pre vytvorenie spojenia pošle klient brokeru správu CONNECT. Pokiaľ je správa od klienta poškodená, alebo ubehlo príliš veľa času medzi otvorením sieťového soketu a odoslaním správy, broker spojenie uzavrie. Takýto postup zabráni poškodeným, alebo nežiadúcim klientom v spomaľovaní brokeru. Ak bola správa v poriadku broker odpovie klientovi správou CONNACK.

Správa CONNECT

Vo vnútri tejto správy môže klient špecifikovať informácie, ktoré ovplyvnia výsledné spojenie a chovanie brokeru ku klientovi:

- **Identifikátor klienta (clientId)** - slúži na identifikáciu klienta, ktorý sa snaží pripojiť na broker. Broker tento identifikátor slúži na zistenie aktuálneho stavu klienta a mal by byť pre unikátny. Pokiaľ nie je nutné aby broker udržiaval stav klienta môže byť tento údaj prázdny. V tomto prípade musí byť nastavený príznak, že ide o novú reláciu, inak bude spojenie odmietnuté.
- **Nová relácia (cleanSession)** - je príznak či klient žiada o vytvorenie perzistentného pripojenia, alebo nie. Pri perzistentnom pripojení broker ukladá informácie o všetkých aktívnych odberoch klienta a všetky nedoručené správy,

ktoré majú garantované doručenie klientovi (Sekcia 2.1.5). Ak je príznak nastavený ako pravdivý, bude nové spojenie neperzistentné, broker nebude ukladať žiadne dáta a zmaže všetky uložené dáta, ktoré sa ku klientovi viažu.

- **Prihlasovacie meno a heslo (username, password)** - tieto dva údaje slúžia na autentifikáciu a autorizáciu klienta. Pokiaľ komunikácia medzi brokerom a klientom nie je šifrovaná, sú tieto údaje v správe uvedené v textovom formáte a môžu byť jednoducho odpočuté z komunikácie.
- **Interval kontroly spojenia (keepAlive)** - jedná sa o časový interval v sekundách, ktorý definuje periódu, za ktorú je spojenie považované za aktívne, bez toho aby si klient a broker vymenili akúkoľvek správu. Ak v danom intervale neprebehla žiadna komunikácia, tak sa klient zaväzuje poslať štandardnú správu PING, aby zistil či je spojenie s brokerom aktívne. Pri odpovedi od brokeru obe strany vedia, že spojenie je aktívne.
- **Posledná vôla, testament (lastWill)** - ide MQTT správu, ktorá je brokerom odoslaná klientom, ktorí sú prihlásený na jej odber, v prípade že sa spojenie so zariadením nečakane preruší.

Správa CONNACK

Je jediná odpoveď brokeru na správu CONNECT. Obsahuje dve dátové položky. Príznak, či pre pripájajúce sa zariadenie existujú uložené dáta z predchádzajúcej interakcie pomocou perzistentného pripojenia (sessionPresent). Ak sa klient pripája správou, ktorá má nastavený príznak novej relácie, bude tento príznak obsahovať zápor (false). Tento príznak bol pridaný, aby klient vedel vyhodnotiť, či sa má znova hlásiť na odber správ, alebo je tento odber stále aktívny z predchádzjúceho pripojenia. Druhou dátovou položkou je návratový kód (returnCode). Môže mať až 6 hodnôt:

Tab. 2.1: Tabuľka návratových kódov a ich významu, ktoré sa môžu vyskytovať v odpovedi brokeru CONNACK.

Návratový kód	Význam kódu
0	Spojenie prijaté.
1	Spojenie odmietnuté, neakceptovaná verzia protokolu.
2	Spojenie odmietnuté, neprijatý identifikátor klienta.
3	Spojenie odmietnuté, server je nedostupný.
4	Spojenie odmietnuté, nesprávne prihlasovacie meno alebo heslo.
5	Spojenie odmietnuté, neautorizované spojenie.

2.1.4 Filtrovanie správ v protokole MQTT

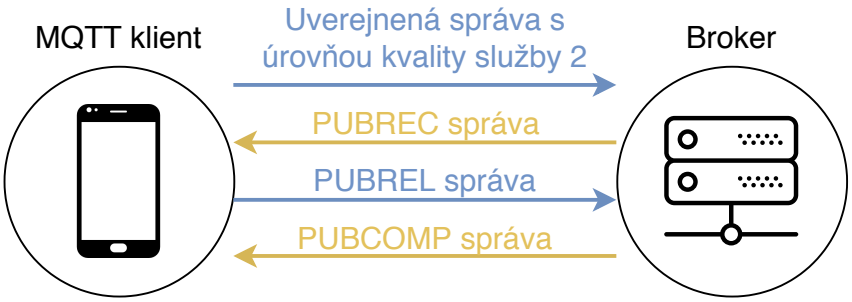
Existuje viacero prístupov ako správy filtrovať tak, aby sa zbytočne nezahlcoval komunikačný kanál:

- **Filtrovanie podľa predmetu** - je postavené na predmete, ktorý je súčasťou každej správy. Zariadenie sa prihlási k odoberaniu správ s predmetom, o ktorý má záujem. Od toho momentu broker zaručuje, že zariadenie bude dostávať všetky správy s daným predmetom. Väčšinou je predmet hierarchická štruktúra kľúčových slov, podľa ktorých sa komunikácia filtruje.
- **Filtrovanie podľa obsahu** - rozhoduje sa podľa špecifického jazykového filtra(filter-language). Zariadenia sa prihlasujú na odoberanie správ vyhovujúcich filtrovaciemu dotazu, ktorý špecifikujú. Nevýhodou je, že broker musí poznať obsah správ, takže nemôžu byť šifrované.
- **Filtrovanie podľa typu** - správy môžu obsahovať informáciu o aký typ správy ide. Týmto spôsobom je možné zachytiť všetky správy, ktoré obsahujú napríklad chybovú hlášku, alebo vyjadrujú špecifickú akciu.

2.1.5 Úroveň kvality služby v protokole MQTT

Úroveň kvality služby je dohoda medzi odosielateľom a príjemcom správy, ktorá definuje garanciu doručenia správy. V protokole MQTT existujú presne tri úrovne kvality služby:

- **Úroveň 0** - je minimálna úroveň kvality služby. Služba garantuje najvyššiu snahu doručenia správy (best-effort delivery). Reálne nie je doručenie garantované, keďže príjemca správy po jej prijatí neodosiela správu s potvrdením prijatia. Táto úroveň sa tiež nazýva „fire and forget“ (odošli a zabudni) [8].
- **Úroveň 1** - garantuje že správa bola doručená aspoň raz. Odosielateľ si správu drží v pamäti po odoslaní, až do momentu, kedy je mu od príjemcu naspäť doručený paket potvrdzujúci jej prijatie (PUBACK). Potvrdzujúci paket obsahuje identifikačné číslo paketu, ktorý potvrdzuje. Podľa toho je odosielateľ schopný nájsť uložený paket v pamäti a uvolniť ho. Pokiaľ odosielateľ nedostane naspäť potvrdzujúci paket správy, odošle ju znova. Preto je možné, že je rovnaká správa odoslaná a doručená viac krát [8].
- **Úroveň 2** - je najvyššia úroveň kvality služby v MQTT. Úroveň garantuje, že správa je prijatá iba raz. To je zaručené aspoň troma ďalšími paketmi, kedy je odoslaný paket s potvrdením o prijatí správy (PUBREC), na ktorý odosielateľ odpovedá ďalším paketom (PUBREL) a uvolní z pamäte pôvodnú správu. Príjemca po prijatí potvrdzujúceho paketu od odosielateľa zmaže uložené pakety a odošle paket o ukončení komunikácie (PUBCOMP) [8].



Obr. 2.2: Komunikácia zariadení pri použití najvyššej úrovne zabezpečenia kvality služby.

2.2 MQTT2GO

Je štandard komunikácie pomocou protokolu MQTT pri implementovaní systému inteligentnej domácnosti. Súčasná verzia štandardu umožňuje komunikovať protokolom MQTT so senzormi, ako aj zariadeniami, ktoré ich ovládajú. Podporuje tiež inteligentné zariadenia nekomunikujúce protokolom MQTT, ale použiteľné v inteligentnej domácnosti. Štandard rieši bezpečnosť komunikácie a predstavuje automatické nastavenie zabezpečenia s výmenou certifikátov. Štandard vznikol z osvedčených postupov podľa dostupnej literatúry, ale aj komerčne dostupných riešení [9].

Štandard používa komunikačný model zariadenia s bránou (sekcia 1.3), s tým rozdielom, že brána je zároveň druhý broker v architektúre tohto štandardu. To umožňuje viacero nových komunikačných scenárov, ako aj maximalizuje komunikáciu protokolom MQTT. Architektúra štandardu má viacero vrstiev, kde najvyššou vrstvou je cloud broker, ďalšou je jedna, alebo viac inteligentných domácností, kde sa nachádza lokálny broker a všetky inteligentné zariadenia. Lokálny broker je zároveň primárna brána zariadení v domácnosti, čo zvyšuje nezávislosť štandardu od stáleho pripojenia. V prípade, že užívateľ nechce vlastniť lokálny broker, štandard stále poskytuje možnosť pripojiť zariadenia priamo na cloud broker, a používať systém týmto spôsobom [9].

Štandard správ

Štruktúra predmetu MQTT správ (topic) je hierarchicky rozdelená do úrovní oddelených znakom „/“. Prvá je najvyššia úroveň, ktorá obsahuje ďalšie definované úrovne. V štandarde je hierarchia vytvorená tak, aby bola čo najefektívnejšia, s rozumnou dĺžkou a podľa osvedčených postupov. Hlavná štruktúra predmetu správy vyzerá nasledovne:

```
<domácnosť>/<brána>/<zariadenie>/<typ správy>/<smer>
```

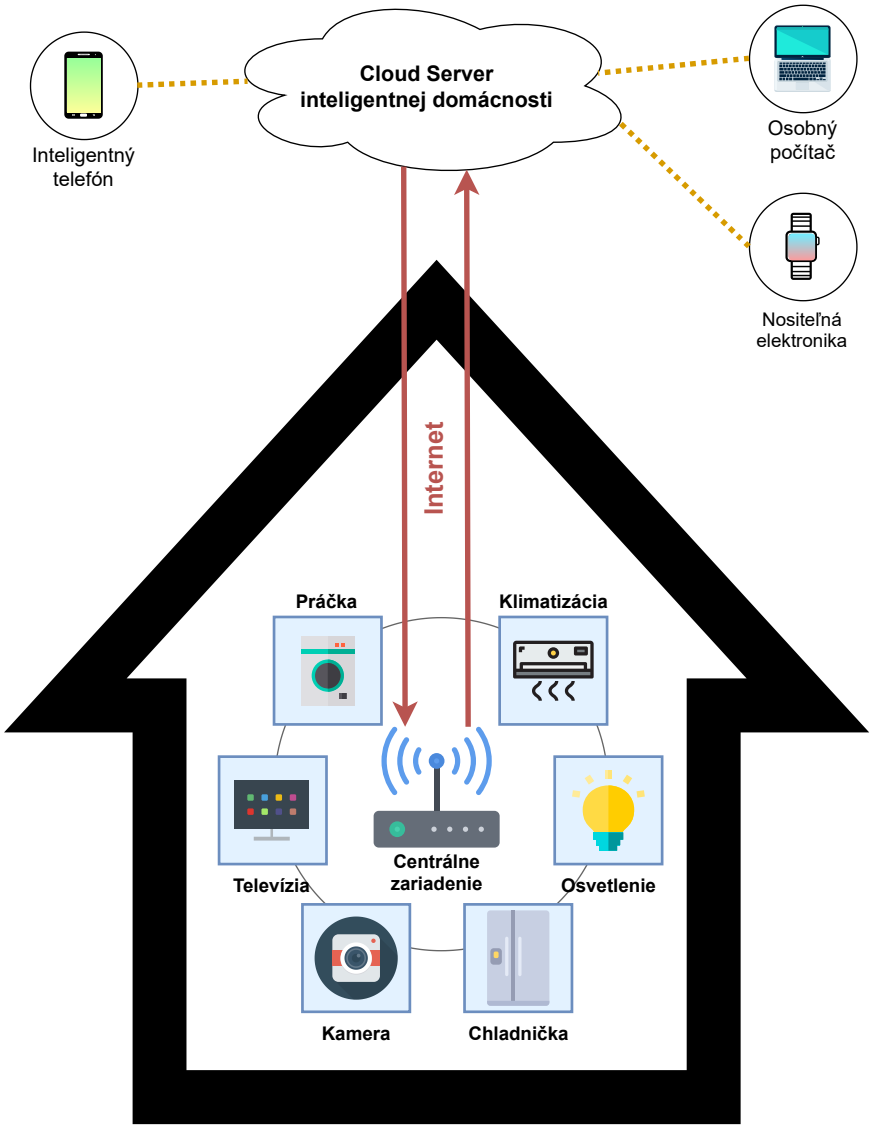
- **<domácnosť>** - je unikátny identifikátor domácnosti.
- **<brána>** - unikátny identifikátor brány.
- **<zariadenie>** - unikátny identifikátor zariadenia.
- **<typ správy>** - unikátny identifikátor typu správy.
- **<smer>** - určuje smer komunikácie. Môže nadobudnúť hodnoty „in“ (smer komunikácie je k zariadeniu), alebo „out“ (smer komunikácie je od zariadenia).

V predmetoch správ sa môžu vyskytovať aj špeciálne znaky (wildcards), ktoré nahradzujú viaceré kombinácie predmetov. Takým znakom je napríklad „+“, ktorý nahradzuje všetky možnosti v hierarchií na danej úrovni. Druhým špeciálnym znakom je „#“, ktorý nahrádza všetky kombinácie v hierarchií na danej úrovni a všetky nižšie úrovne [9].

Obsah správy je vo formáte JSON (JavaScript Object Notation) a obsahuje tri časti: typ (type), časovú značku (timestamp) a hodnotu (value). Typ označuje tému správy, časová značka určuje čas kedy bola správa vytvorená, hodnota je textový obsah správy. Hodnota môže byť jednoduchá (číslo, text), alebo komplexná štruktúra s viacerými hodnotami [9].

3 Inteligentná domácnosť

Je jedna z aplikácií Internetu vecí. Cieľom inteligentnej domácnosti je zvýšiť kvalitu života jej rezidentov. Integruje v sebe viacero rozličných domén, ako automatizáciu domácnosti (budovy), monitorovanie kvality vzduchu, zdravotnú starostlivosť, dohľad, bezpečnosť, inteligentnú záhradu. Automatizácia domácnosti umožňuje užívateľovi monitorovať, ovládať spotrebiče na diaľku, cez internet. To zahrňuje aj tvorbu vlastných súborov nastavení pri určitej spúšťacej akcií, ako že sa po príchode domov rozsvieti svetlo na chodbe, zapne televízor, a zvýši teplota na termostate [10].



Obr. 3.1: Koncept inteligentnej domácnosti. Zariadenia v dome sú súčasťou lokálnej siete, ktorá komunikuje so serverom, cez ktorý je možné domácnosť ovládať rôznymi zariadeniami.

Monitorovanie kvality vzduchu, môže v prípade že domácnosť (budova) disponuje ovládateľnou vzduchotechnikou, po prekročení určitých hodnôt zapnúť systém na jeho odvetranie. Ak budova nemá vzduchotechniku, je možné upozorniť pomocou notifikácie rezidenta o potrebe výmeny vzduchu.

Inteligentná domácnosť sa rozšírila aj za múry budovy a pomáha aj s prácami okolo domu. Monitoruje vlhkosť pôdy trávniku a môže zahájiť zavlažovanie, alebo používateľa upozorniť. Trávnik je potom možné nechať pokosiť inteligentnou kosačku.

Dohľad a bezpečnosť má za úlohu poskytnúť bezpečie ľudí v domácnosti ako aj majetku samotného. Kamery v budove aj jej okolí sú schopné detektovať anomálie, rozpoznávať tváre ľudí a v reálnom čase upozorňovať používateľa prípadne bezpečnostné zložky. Dohľad môže byť uplatnený pri deťoch, či starších ľuďoch. Kamery môžu detektovať nehodu, ktorá sa obyvateľovi stane, hlasový asistent sa môže uistiť, či je v poriadku, prípadne privolať pomoc. Dohľad nad domácnosťou poskytuje úľavu vždy, keď ju rezident opustí, pretože sa nemusí báť či spotrebič neostal zapnutý, dvere neuzamknuté, majetok nechránený a môže sa o tom uistiť v reálnom čase [11], [4].

Inteligentná domácnosť pomáha šetriť energie. Vďaka senzorom, na rozdiel od časovača, nespustí zavlažovanie, ak je pôda vlhká, alebo ho spustí len na dobu dokedy nedosahuje správnu vlhkosť. Nespustí klimatizáciu, keď nikto nie je doma, ale môže vďaka naučenému správaniu nastaviť teplotu domu pred príchodom rezidentov z práce [11].

3.1 Systémová architektúra inteligentnej domácnosti

Architektúra sa v aplikáciach inteligentnej domácnosti a jej siete organizuje okolo centrálneho zariadenia (Obrázok 3.1). Centrálne zariadenie môže byť smerovač s priamym prístupom k internetu, alebo zariadenie, ktoré je na smerovač pripojené. Inteligentné zariadenia v domácnosti je možné nakonfigurovať tak, aby sa pripojili na domácu sieť, ktorej súčasťou je centrálne zariadenie. Všetky zariadenia sú potom pripojené na lokálnu sieť, kde spolu môžu komunikovať. Neexistuje komplexné riešenie na vybudovanie takéhoto systému, preto je nutné aby systém poskytoval možnosť dynamicky pridávať a odoberať inteligentné zariadenia pomocou užívateľom riadenej platformy [12].

Centrálne zariadenie je spojené so vzdialeným serverom, ktorý umožňuje komunikovať s domácnosťou aj keď zariadenie, ktoré systém ovláda nie je pripojené na lokálnej sieti. Vzdialený server nie len že poskytuje možnosť komunikácie na diaľku s jednou domácnosťou, ale umožňuje spravovanie viacerých systémov. Vzdialený server teda môže byť verejným a poskytovať svoje služby viacerým nezávislým

používateľom s viacerými domácnosťami, ku ktorým majú prístup. Server musí manažovať enormné množstvo dát, bezpečnostné prístupy používateľov, a zabezpečenú databázu dát z jednotlivých domácností [12].

3.2 Proprietárne riešenia inteligentnej domácnosti

Na zostavenie vlastnej inteligentnej domácnosti už v súčastnosti nie je potreba odborných znalostí. Je možné použiť jedno z existujúcich riešení od preferovanej spoločnosti. Jediné na čo si musí používateľ dať pozor je aby boli jednotlivé zariadenia, ktoré chce do svojej domácnosti pridať, kompatibilné s centrálnym zariadením. Väčšinou je uvedená kompatibilita zariadenia s rôznymi riešeniami priamo na obale nového zariadenia, alebo v jeho manuále.

Medzi najznámejšie systémy inteligentnej domácnosti môžme považovať riešenia od technologických gigantov: Google, Samsung, Apple a ďalšie.

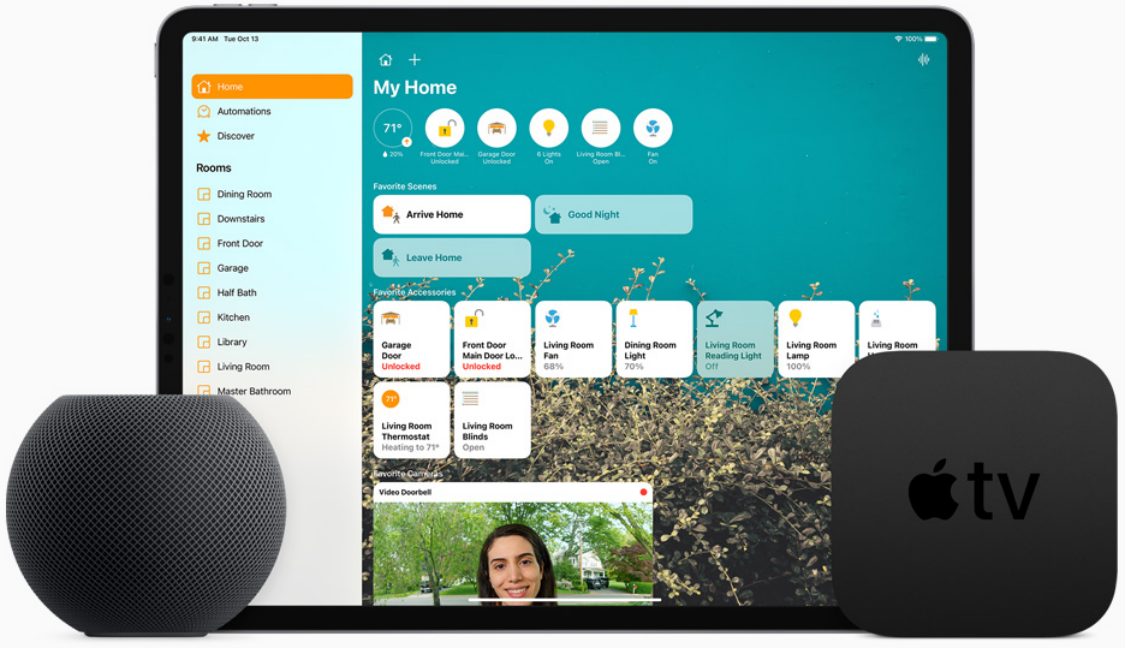
Apple HomeKit

Je mobilná aplikácia od spoločnosti Apple. Toto riešenie je zaujímavé, pretože na jeho používanie je možné zvoliť dva prístupy.

Prvou možnosťou je používanie mobilnej aplikácie na ovládanie zariadení priamo bez použitia centrálneho zariadenia. Ak je inteligentné zariadenie kompatibilné s HomeKit štandardom, tak sa po pripojení do lokálnej domácej siete samo propaguje. Mobilná aplikácia automaticky vyhľadáva kompatibilné nové zariadenia. Môže sa stať že zariadenie v sieti nebude aplikáciou nájdené a v tomto prípade je možné zariadenie pridať manuálne. Takýmto spôsobom možné je združiť ovládanie zariadení od rôznych výrobcov v jedinej aplikácií. Zoznam kompatibilných zariadení uvádza na svojich webových stránkach aj priamo spoločnosť Apple[1].

Druhou možnosťou je vytvorenie inteligentnej domácnosti s centrálnym zariadením, čo je v tomto prípade zariadenie Apple TV (Obrázok 3.2). Pridávanie zariadení funguje podobne ako pri použití aplikácie s rozdielom, že všetká komunikácia prebieha cez toto zariadenie. Keďže Apple TV je primárne set-top box, je možné inteligentnú domácnosť ovládať z obrazovky televízora. Hlavnou výhodou tohto riešenia je, že po pridaní zariadenia Apple TV do bezplatného HomeKit účtu, je možné riadiť domácnosť pomocou mobilnej aplikácie na diaľku (nie je potrebné aby sa mobilné zariadenie nachádzalo v rovnakej sieti, ako zariadenia inteligentnej domácnosti). Ďalšou výhodou je aj ovládanie domácnosti cez nositeľnú elektroniku (wearables), akou je napríklad Apple Watch.

[1] `https://www.apple.com/ios/home/accessories/`



Obr. 3.2: Aplikácie Apple HomeKit zobrazená na zariadení iPad. Po stranách sú z ľava inteligentný reproduktor HomoPod, disponujúci mikrofónom reagujúcim na hlasové povely, vpravo zariadenie Apple TV, ktoré okrem inej funkcionality zastupuje centrálne zariadenie domácnosti.[3]

Aplikácia HomeKit organizuje jednotlivé zariadenia do miestností. Miestnosti je možné v aplikácií priamo vytvoriť. Miestnosti je ďalej možné organizovať do skupín (napríklad poschodí). Pre ovládanie viacerých zariadení spoločne je možné vytvoriť takzvané scény. V scéne je možné definovať aké zariadenia sa po jej aktivácií dostanú do konkrétneho stavu. To znamená, že je možné definovať napríklad svietivosť žiarovky, nie len či je zapnutá.

Ovládanie zariadení môže prebiehať aj bez užívateľskej interakcie a to za pomoci automatických scenárov, ktoré je možné definovať priamo v aplikácií. Automatizačné scenáre môžu byť spustené interakciou s konkrétnym zariadením, podľa nastaveného času, alebo podľa lokácie zariadenia zaregistrované na ovládanie domácnosti.

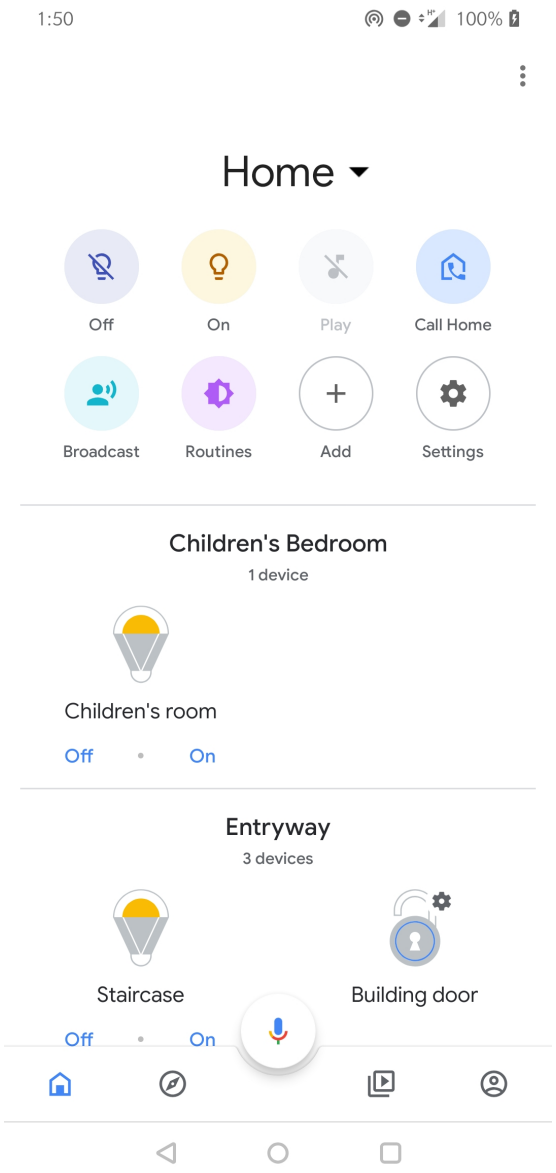
Google Home

Je zároveň názov hlasového asistenta a tiež mobilnej aplikácie. Rovnako ako v prípade Apple HomeKit je možné aplikáciu používať samostatne na združené ovládanie zariadení, ktoré podporujú Google API (Application Programming Interface). Alebo je možné použiť na kompletné ovládanie zariadenie Google Home, ktoré sa

[3] `https://www.apple.com/ios/home/`

stane centrálnym zariadením domácnosti. Výhodou tohto riešenia je väčšia otvorenosť voči zariadeniam. Za zmienku stojí už len to, že aplikácia Google Home je dostupná aj na zariadeniach s operačným systémom iOS.

Aplikácia Google Home poskytuje okamžite možnosti na riadenie viacerých zariadení (Obrázok 3.3). Napríklad sa pod jediným tlačítkom skrýva zapnutie inteligentných svetiel v celej domácnosti, určitej miestnosti, alebo je možné definovať vlastnú skupinu svetiel. Rovnako to platí pre spúšťanie hudby a videa, cez ďalšie aplikácie.



Obr. 3.3: Domovská obrazovka aplikácie Google Home.[4]

[3] https://www.androidpolice.com/

Automatizácia domácnosti je dostupná cez štandard IFTTT (If This Then That)[5]. Výhodou tohto štandardu je, že sa neobmedzuje len na ovládanie zariadení inteligentnej domácnosti, ale je možné ňou automatizovať aj udalosti z rôznych webových služieb (e-mailového klienta, sociálnej siete). Napríklad je možné zmeniť farbu svetla pri príchode dôležitej pošty na e-mail užívateľa [13].

[5]If This Then That je štandard umožňujúci vytváranie scén (applets), ktoré obsahujú udalosť, ktorá ju spúšťa, a akcie, ktoré sa vykonajú [13].

4 Návrh natívnej aplikácie pre Android

Táto kapitola zahrňuje prípravu a rozhodovací proces pred začatím samotnej implementácie aplikácie.

4.1 Dostupné technológie na implementáciu aplikácie

Prvým a to veľmi dôležitým rozhodnutí je výber programovacieho jazyka, v ktorom bude práca implementovaná. V súčastnosti existujú dva programovacie jazyky, ktoré sa bežne používajú na vývoj aplikácií pre Android. Sú to jazyky Java a Kotlin. Kotlin je pomerne mladý programovací jazyk, vytvorený tak aby dokázal spolupracovať s komponentami z jazyka Java. Napriek tomu že má jazyk plnú podporu pre vývoj na operačný systém Android, čo sa dokumentácie týka, pre tento jazyk neexistuje veľké množstvo knižníc, návodov a riešení na špecifické problémy. Naproti tomu je jazyk Java veľmi dobre zmapovaný a na riešenie problémov sa dá na internete nájsť viacero prístupov ako ich vyriešiť. Existuje preň obrovské množstvo knižníc a návodov. Najkľúčovejším aspektom pri výbere sa po prieskume dostupných komponent stala podpora pre aplikačný protokol MQTT. Pre aplikačný vývoj na operačný systém Android existuje dokumentáciou k MQTT protokolu odporúčaná jediná knižnica Eclipse Paho Android Service [14]. Knižnica je naprogramovaná v jazyku Java a je k nej dostupná aj vzorová Android aplikácia tiež v jazyku Java. Podporné informácie ku knižnici boli výlučne orientované na jazyk Java. Preto bol na implementáciu aplikácie zvolený jazyk Java.

Cieľová verzia Android zariadení

Ďalším krokom nutným pred začatím implementácie aplikácie pre systém Android je voľba cieľovej a minimálnej verzie systému Android, pre ktorú bude aplikácia spustiteľná. Ako cieľová bola zvolená najvyššia možná verzia a to Android 11. Je to z dôvodu aby bolo možné použiť najnovšie dostupné prvky systému Android. Minimálna verzia na spustenie bola zvolená verzia Android 8.0. Dôvodom je zmena v podpore textov, kedy sa zlepšili možnosti adaptívnej veľkosti, štýlov a charakteristiky písma. Veľké zmeny nastali aj v oblasti systémových notifikácií. Štatisticky bude teda aplikácia kompatibilná s 80% aktuálne používanými zariadeniami a nebude nutné používať zastaranejšie technológie so spätnou kompatibilitou.

Architektonický návrh aplikácie

Pred začatím programovania aplikácie je odporúčané, pre zaistenie robustnosti, logického rozdelenia kódu a produkčnej kvality, zvoliť typ architektúry. Medzi najznámejšie patria: Model-View-Presenter (MVP) a Model-View-ViewModel (MVVM).

Pod modelom rozumieme komponenty, ktoré sa starajú o dáta, získavanie dát z vonkajšieho prostredia, udržujú stav aplikácie, spúšťajú výpočtové procesy aplikácie. View reprezentuje jak súbory, ktoré priamo popisujú vzhľad užívateľského rozhrania, tak komponenty riadiace vykreslenie rozhrania a definujú procesy a chovanie, ktoré je možné spustiť pomocou užívateľskej interakcie. Presenter a ViewModel sú komponenty starajúce sa o prenos medzi informácií medzi súčasťami Modelu a View. Hlavným rozdielom medzi oboma prístupmi je spôsob predávania dát do View, kde pri ViewModel architektúre sa k vyžiadaným dátam komponenty viažu, čím je možné pozorovať a reagovať zmeny v načítaných dátach [15] [16].

Pri porovnaní oboch architektúr je možné nájsť pre a proti u oboch. Architektúra MVVM má ešte podrobnejšie delenie komponent, je aktuálnejšia, odporúčaná priamo v online dokumentácií pre Android. Koncept sledovania zmeny dát podľa môjho názoru viac vyhovuje pre prácu s dátami, ktoré aplikácia dostáva z inteligentnej domácnosti a preto bola pre túto prácu zvolená [16].

Eclipse Paho Android Service

Je voľne dostupná knižnica v jazyku Java pre vývoj aplikácií na operačný systém Android, ktorá implementuje klientskú časť aplikačného protokolu MQTT. Podporuje verzie protokolu MQTT 3.1 a 3.1.1. Je v nej zahrnutá podpora šifrovania komunikácie pomocou protokolu SSL, automatického pripojenia k serveru v prípade výpadku, asynchrónne komunikačné rozhranie, správy v prípade nežiadúceho odpojenia a ďalšie. Hlavnou časťou knižnice je podpora na pripojenie k serveru, uverejňovania správ a prihlásenia na odber správ.

4.2 Užívateľské rozhranie mobilnej aplikácie

Cieľom pri vytváraní užívateľského rozhrania je, aby sa jeho používateľ dokázal bez akéhokoľvek návodu v rozhraní orientovať, mal hlavné funkcie aplikácie dostupné bez nutnosti ich hľadania a mal by mať jasnú informáciu, kde v aplikácií sa nachádza. Pri uskutočnení akcie používateľom, ktorá vyžaduje komunikáciu so serverom, alebo dlhšie spracovávanie informácií, by mal byť používateľ informovaný čo sa deje. Po vykonaní akcie by sa efekt mal prejaviť v rozhraní vizuálne, ak je to možné. Ak efekt akcie nie je viditeľný, je nutné používateľa informovať o jej výsledku.

Pre jednoduchosť používania rozhrania ju nutné brať do úvahy aj dosiahnuteľnosť komponentov rozhrania. Keďže zariadenia majú navigačné prvky umiestnené na spodnej hrane, je výhodné umiestňovať navigáciu v aplikácií tiež na spodnú hranu. Naopak na vrchnej hrane zariadenia by nemali byť často používané komponenty, pretože pri súčasnom trende zväčšovania displejov zariadení sa často stáva, že používateľ na vrchnú hranu nedočiahne a musí si zariadenie prechytiť [17].

Texty a preklady v mobilnej aplikácií

Pri textoch je najdôležitejšie aby boli čitateľné, zmysluplné a v jazyku, ktorému používateľ rozumie. Čitateľnosť závisí hlavne od veľkosti písma a kontrastnosti jeho farby voči farbe pozadia. Pri aplikáciach pre platformu Android je nutné pamätať, že veľkosť zariadení, na ktorých bude aplikácia spustiteľná, je rôzna. Preto je nutné používať relatívne veľkosti písma, ktorých veľkosť je pomerná k veľkosti obrazovky zariadenia. Ako hlavný jazyk aplikácie je použitá angličtina. Pri vytváraní používateľského rozhranie je lepšie používať zástupné texty, ku ktorým je možné vytvoriť zoznam prekladov a umožniť tak jednoduché implementovanie ďalšieho jazyka.

5 Implementácia mobilnej aplikácie na ovládanie prvkov inteligentnej domácnosti

Táto kapitola popisuje požiadavky na funkcionalitu v prvej iterácií vývoja, použité komponenty a vytváranie mobilnej aplikácie.

5.1 Použité komponenty Android aplikácie

Vývojové prostredie Android aplikácií oddeľuje súbory, ktoré definujú chovanie aplikácie a súbory definujúce zobrazenie. Súbory definujúce chovanie obsahujú triedy, ktoré riešia komunikáciu s užívateľským rozhraním, implementujú chovanie použitých komponent, ukladajú dátovú časť aplikácie, alebo pomáhajú a zaobaľujú chovanie použiteľné naprieč ostatnými triedami.

Aktivity a fragmenty

Android aplikácie sú rozdelené do jednotlivých aktivít. Aktivita sa skladá z vlastnej triedy rozširujúcej chovanie preddefinovanej triedy *Activity* a súboru vo formáte XML definujúcom zobrazenie danej aktivity. Aktivita je jedno celé okno aplikácie. Má svoj životný cyklus, ktorý definuje potupnosť udalostí, ktoré nastanú pri jej vytvorení, vykreslení, behu a nakoniec aj ukončení. V Android aplikáciach nikdy nie sú dostupné pre používateľa dve aktivity naraz. Výnimkou existencie dvoch aktivít je prechod z jednej na ďalšiu, kedy prebieha zároveň ukončenie prvej aktivity a vytvorenie nasledujúcej, čo ale používateľ nevníma.

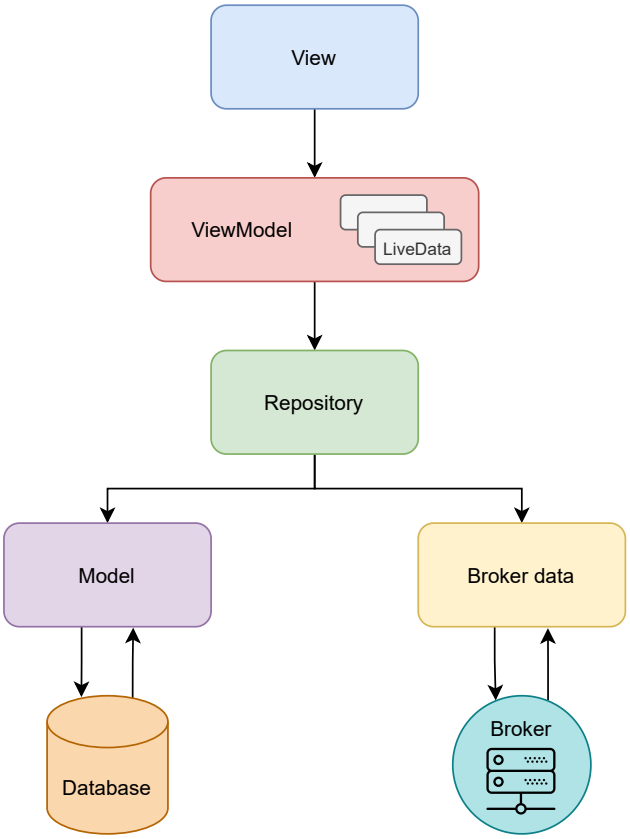
Aby bolo možné rozdeliť aktivitu na viac samostatných častí existuje v prostredí Android dostupná trieda *Fragment*. Na vytvorenie fragmentu aktivity je nutná vlastná trieda rozširujúca triedu *Fragment* a XML súbor definujúci jej zobrazenie. V zobrazení aktivity je potom možné definovať časť zobrazenia (kontajner), ktorá bude obsahovať zobrazenie fragmentu. Trieda fragmentu rieši chovanie svojho zobrazenia, pri čom má dostupné dáta z aktivity, ktorá ju spustila.

Za behu aplikácie môže nastať zmena konfigurácie, napríklad pri zmene orientácie obrazovky, kedy Android reštartuje aktuálnu aktivitu. Keďže tá obsahuje načítané (neuložené) dáta je nutné ukladať ich v samostatnom objekte *ViewModel* [18].

Architektúra MVVM

Tento architektonický vzor pomáha kompletne oddeliť logiku, prácu s dátami a procesy na pozadí od užívateľského rozhrania. Ústredným prvkom architektúry je *ViewModel* (Obrázok 5.1). Tento názov je rovnaký ako názov komponenty systému Android, ktorá implementuje jej chovanie. Podľa architektúry sa *ViewModel* neviaže

so žiadnou aktivitou, fragmentom, ani ničím čo má referenciu na ich kontext. Tým je dosiahnuté, že pri zmene zobrazenia táto komponenta prežije a dáta v nej nie sú stratené [19].



Obr. 5.1: Vizualizácia závislostí komponent pri použití architektonického vzoru MVVM. Každá komponenta je závislá iba na komponente o jeden stupeň nižšie.

Ďalším špecifikom komponenty *ViewModel* je, že umožňuje dáta obaliť triedou *LiveData*, alebo triedami, ktoré rozširujú jej chovanie. Takto zaobalené dáta je možné za behu aplikácie sledovať z aktivity, alebo fragmentu a na základe zmeny dát asynchrónne vyžiadať zmenu zobrazenia, alebo jeho prvkov.

Architektúra tiež vyčleňuje triedy, ktoré sa starajú o načítanie dát či už uložených lokálne, alebo v tomto prípade od zariadenia Broker, pod pojmom *Repository*. Triedy spadajúce pod tento spoločný pojem ponúkajú centralizovaný prístup k určitým dátam rovnakého typu. V takejto triede je napríklad možné inicializovať informácie o stave inteligentnej domácnosti, ku ktorej je aplikácia pripojená.

Model a *Broker data* z obrázku 5.1 stelesňujú triedy, ktoré popisujú dáta a mapujú sa na tabuľky v databáze, alebo na dáta zo zariadenia broker. Je to vlastne objektové vyjadrenie dát, ktoré sú za behu načítané do operačnej pamäte zariadenia

za účelom ich použitia. Prvky *Model* nesmú byť závislé od ovládacích prvkov. Zároveň je možné aby boli jednotlivé *Modely* závislé na sebe. Týmto spôsobom je možné organizovať dáta do väčších logických celkov a zároveň je ich štruktúra rozložená do viacerých tried.

Služba

Je komponenta Android aplikácií určená na vykonávanie dlhodobých operácií na pozadí[1]. Na pozadí v tomto prípade znamená, že služba nemá užívateľské rozhranie, nie na akom vlákne procesoru je spustená. Služba pracuje na vlákne procesoru, kde bola vytvorená jej inštancia. Spustená služba môže pracovať aj v dobe, kedy je na zariadení spustená iná aplikácia. Systém Android pozná tri typy služieb: na popredí (Foreground), na pozadí (Background) a naviazané (Bound) služby. V tomto projekte implementuje službu napríklad knižnica *Eclipse Paho Android Service*. Je to preto, aby spojenie so serverom nezaniklo pri prepnutí na inú aplikáciu a opätovnom návrate [18].

SQLite

Na ukladanie dát v Android zariadení je možné použiť viacero technológií. V tejto aplikácie je použitá databáza SQLite. Je to, z hľadiska platforiem, najrozšírenejší relačný databázový systém na svete. Celá databáza je uložená v jednom súbore. Je objemovo nenáročná, rýchla. Hlavnou výhodou jej použitia je možnosť ju stále rozširovať pre ukladanie nových dát [20].

V súčasnom stave má iba dve tabuľky obsahujúce informácie o prihlásení na broker a tabuľku jednotlivých miestností v inteligentnej domácnosti pre možnosť prispôsobenia ich zobrazenia v aplikácií.

RecycleView

Je jedna z grafických komponent dostupná pre Android. Dokáže dynamicky vytvárať samostatné elementy na základe dodaných dát a zobraziť ich ako zoznam s dynamickým načítavaním položiek, ktoré zatiaľ nie sú zobrazené. Pre správne fungovanie potrebuje komponenta dve triedy. Prvá rozširuje triedu *RecyclerView.Adapter*, ktorá sa stará o predanie dát komponente. Druhá rozširuje triedu *RecyclerView.ViewHolder*, ktorá implementuje mapovanie dát na jednotlivé elementy a ich vykreslenie.

Ako bude výsledný zoznam zobrazený je možné ovplyvniť pomocou nastavenia manažéra rozloženia. Ten je implementovaný triedou *RecyclerView.LayoutManager*. Trieda zodpovedá za zmeranie a pozíciu jednotlivých elementov, spôsob rolovania zoznamu (vertikálne alebo horizontálne) a nastavenie čo sa stane s elementami, ktoré sa na obrazovku nezmestia [20].

[1] `https://developer.android.com/guide/components/services`

Gradle

Je nástroj na automatické zostavenie aplikácie a jeho konverziu na inštalačný balíček systému Android. Používa acyklický orientovaný graf na vyriešenie závislostí jednotlivých knižníc a ich pridanie v správnom poradí. Nástroj umožňuje definovať rôzne verzie zostavenia balíčku určené pre fázu ladenia, testovania a vydania aplikácie.

Glide

Je voľne šíriteľná knižnica pre systém Android na efektívne načítanie a zobrazenie obrázkov[2]. Umožňuje dekódovanie rôznych formátov, ukladanie obrázkov do vlastnej rýchlej pamäte, načítanie obrázkov cez protokol HTTP, dynamickú zmenu veľkosti. Výbornou vlastnosťou je možnosť načítania miniatúry, ktorá sa dočasne zobrazí, kým je dostupný obrázok načítavaní cez protokol HTTP. Hlavnou výhodou knižnice je jej rýchlosť zobrazenia obrázku, ktorá je rýchlejšia ako pri použití štandardných komponent systému Android, ako aj jednoduchosť a rozšíriteľnosť jej programového rozhrania.

5.2 Požiadavky na funkcionalitu v prvej iterácií vývoja

Vytvorenie funkčnej natívnej mobilnej aplikácie pre systém Android. Vývoj zahrňuje užívateľské rozhranie aj logickú časť aplikácie. Aplikácia musí zvládnuť:

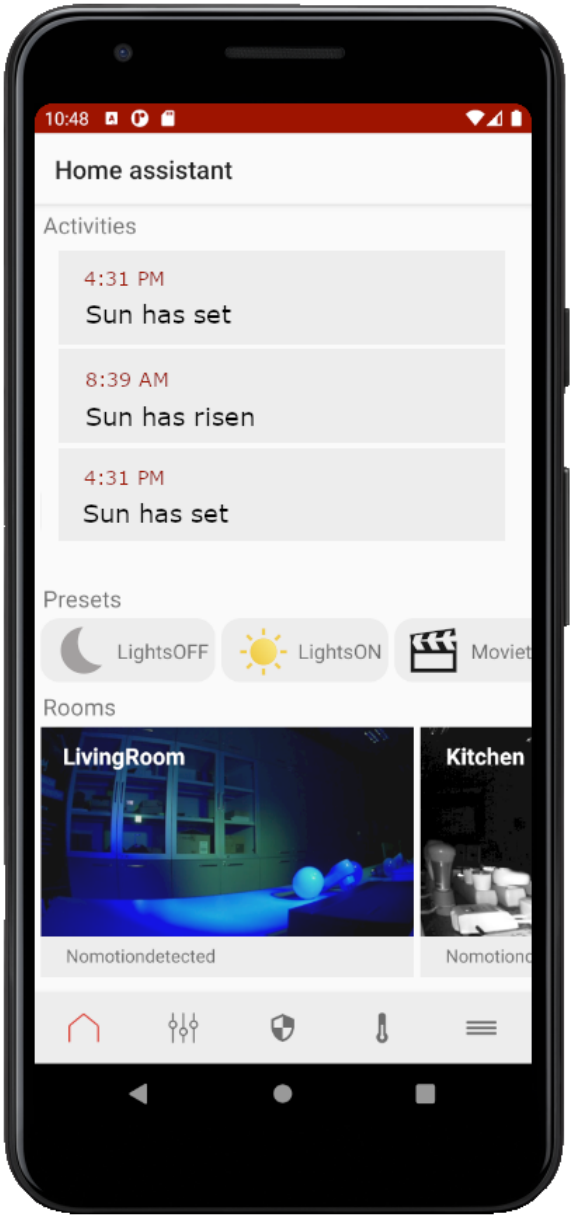
- Pripojenie na lokálny broker domácnosti.
- Komunikáciu pomocou uverejňovania a odoberania správ.
- Získanie a spracovanie informácií o inteligentnej domácnosti.
- Zobrazenie a použitie aspoň jedného inteligentného zariadenia.

5.3 Prvá iterácia vývoja aplikácie

Základom celej aplikácie je schopnosť komunikovať s inteligentnou domácnosťou, teda pripojiť sa na lokálny broker. Keďže broker sa nenachádza priamo na lokálnej sieti počítača, kde je aplikácia tvorená, bolo nutné sa na sieť pripojiť pomocou virtuálnej privátnej siete (VPN), čo skomplikovalo komunikáciu. Po spustení aplikácie je automaticky spustená hlavná aktivita. Tá najskôr zistí či existuje databáza dát a ak nie vytvorí ju. Aby sa pri zmene konfigurácie aplikácie nestratili dáta, bola vytvorená trieda *MainViewModel*. Tá iniciuje pripojenie. Na manažment pripojenia bola vytvorená pomocná trieda *MqttHelper*, ktorá rieši pripojenie na broker. Po úspešnom pripojení je spustená anonymná funkcia, ktorá sa prihlási na odber správ,

[2] `https://bumptech.github.io/glide/`

kam broker pošle informácie o zariadeniach v inteligentnej domácnosti. Odoslanie informácií o inteligentnej domácnosti je vyžiadané odoslaním MQTT správy typu *query_gui_dev*. Následne aplikácia odoberá správy a filtruje ich kým nepríde správa s požadovanými informáciami. Keďže dáta správ sú podľa štandardu MQTT2GO vo formáte JSON, bola vytvorená pomocná trieda *JsonHelper*, ktorá sa stará o rozbor správy. Dáta zo správy sú uložené v objekte *MainViewModel*. Následne je na displej vykreslená hlavná aktivita a jej prvý fragment. *HomeFragment*.



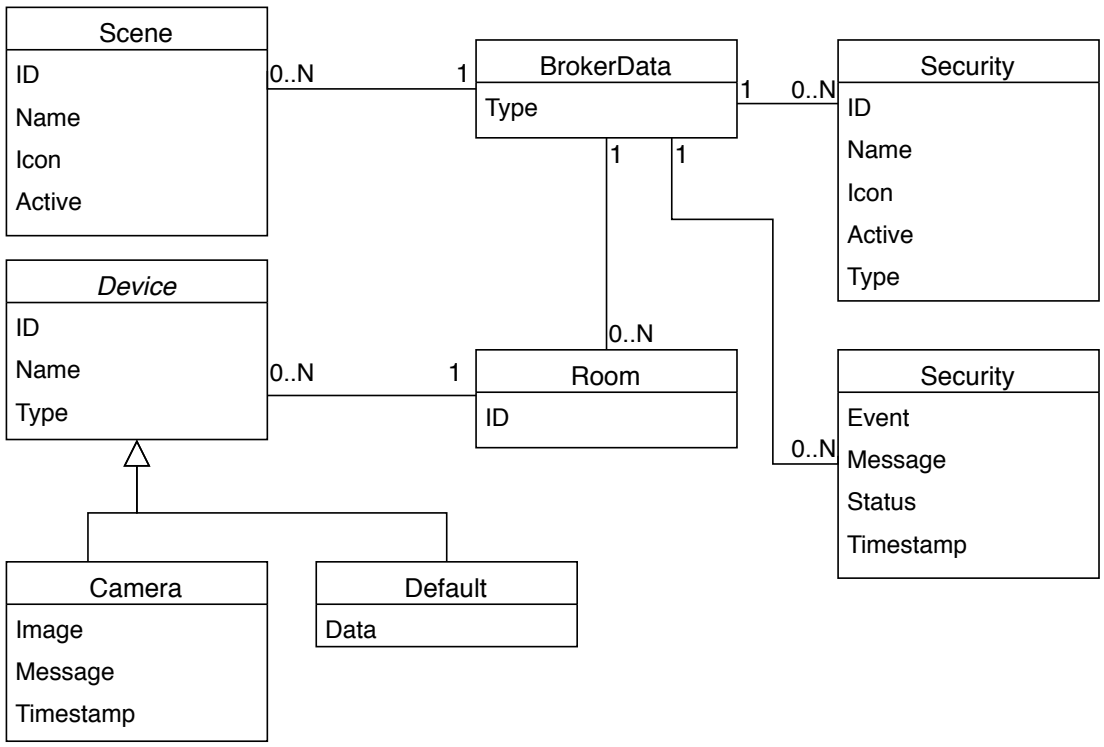
Obr. 5.2: Ukážka hlavnej aktivity a prvého fragmentu aplikácie v prvej iterácií. Ďalšie scény (presets) alebo kamery z miestností je možné zobraziť pomocou horizontálneho rolovania.

Hlavné zobrazenie

Podľa návrhu bola vytvorená navigácia v spodnej časti aplikácie. Vo vrchnej časti displeja sa nachádza panel, ktorý zatiaľ len informuje používateľa o tom kde v aplikácií sa nachádza. Každá z ikon v navigácií spúšťa pridružený fragment. Hlavná aktivita teda pozostáva z piatich fragmentov: domovského, miestnosti a ich zariadenia, zabezpečenie domácnosti, termostat a nastavenia (Obrázok 5.2).

Objektová reprezentácia dát inteligentnej domácnosti

Štandard MQTT2GO, hovorí že klientské zariadenie určené na ovládanie inteligentnej domácnosti si má, ako prvú vec po pripojení na broker, vyžiadať správu s informáciami o všetkých zariadenia v domácnosti. V tejto hlavnej správe prídu informácie o prednastavených scénach, zabezpečení, upozorneniach, bezpečnostných systémoch, miestnostiach a zariadeniach, ktoré obsahujú. Je to veľmi veľa informácií a všetky musí aplikácia mať k dispozícií. Aby bolo možné s dátami jednoducho pracovať bola vyvinutá objektová štruktúra týchto dát.



Obr. 5.3: Objektová reprezentácia dát inteligentnej domácnosti.

Na vrchole štruktúry je trieda *BrokerData*, ktorá obsahuje všetky miestnosti, nastavené upozornenia, prednastavené scény, zabezpečenia v samostatných stromových štruktúrach pre rýchlejšie vyhľadávanie. Všetky tieto dáta sú objektami vytvorených tried špecificky pre každú položku. Objekt miestnosti *Room* obsahuje stro-

movú štruktúru zariadení v objektoch triedy *Device*. Každý typ zariadenia obsahuje iné informácie. Preto od triedy *Device* dedí vlastnosti celá rada objektov, vytvorená podľa typov jednotlivých zariadení. Štruktúra je zobrazená na obrázku 5.3.

5.4 Požiadavky na funkcionalitu v druhej iterácií vývoja

Po dohode s vedúcim práce bol spravený ústupok voči zadaniu práce. Vynecháva sa implementácia pridávania a odoberania zariadení inteligentnej domácnosti, kvôli nedostupnosti laboratória s testovacou domácnosťou, na základe proti-pandemických opatrení, ktoré boli platné v priebehu implementácie.

V druhej iterácií vývoja bol dôraz kladení na rozšírenie základnej funkcionality, ovládanie všetkých zariadení, ktoré sa v testovacej domácnosti vyskytujú. Takže celkovo ide o:

- Ovládanie zabezpečenia domácnosti.
- Odosielanie systémových a aplikačných notifikácií o nastavených anomáliách v domácnosti.
- Dynamické zbieranie a zobrazenie zoznamu aktivít, ktoré sa v domácnosti stali, ako aj načítanie a zobrazenie zoznamu aktivít za celý deň.
- Sumárne zobrazenie všetkých zariadení v jednotlivých miestnostiach a ich aktuálneho stavu.
- Detailné zobrazenie každého zariadenia v domácnosti, možnosť zmeniť jeho hodnoty a zobraziť informatívne hodnoty, ktoré zariadenie poskytuje.
- Vytvorenie prihlasovacieho formulára, ukladanie jeho hodnôt a automatické prihlasovanie podľa uložených informácií.
- Šifrovanie komunikácie protokolom MQTT, získanie a použitie certifikátu na šifrovanie.

5.5 Druhá iterácia vývoja

Na začiatku tejto iterácie bolo nutné zmeniť spôsob, ako sa aplikácia k inteligentnej domácnosti pripojí, keďže v prvej iterácií boli prihlasovacie údaje nastavené staticky. V rámci analýzy tejto zmeny vyšla najavo aj potreba vytvorenia vlastnej služby, ktorá sa bude starať o spojenie a zobrazenie systémových notifikácií. Preto bola najskôr implementovaná vlastná služba s názvom *MqttService*.

MqttService

Ako bolo spomenuté v sekcií 5.1, systém Android pozná tri typy služieb. Na im-

plementáciu tejto služby bol zvolený typ naviazanej služby (Bound service) a to z viacerých dôvodov. Služba nevyhovuje definícií služby na popredí, keďže operácie, ktoré robí, nie sú pre užívateľa rušivé (pri práci s inou aplikáciou). Služba je naviazaná pretože nevyhnutne súvisí s aplikáciou. S týmto typom služby sa dá jednoducho komunikovať cez komponentu *ServiceConnection* a so zaniknutím všetkých komponent, ktoré sú na ňu naviazané, sa sama ukončí.

Ak služba už nie je vytvorená, tak sa vytvorí a naviaže hneď po spustení aplikácie a vytvorení triedy *MainViewModel*, ktorá uchováva informáciu o jej naviazaní na hlavnú aktivitu. Po jej vytvorení služba vytvorí kanál pre uverejňovanie systémových notifikácií *NotificationChannel*, vyžiada si z SQLite databáze posledné prihlasovacie údaje na pripojenie k zariadeniu broker inteligentnej domácnosti. Ak žiadne prihlasovacie údaje neexistujú, služba ďalej nerobí nič. V tomto momente už je zobrazený fragment s prihlasovacím formulárom, kde je možné údaje zadať a vyzvať službu na pripojenie. O výsledku pokusu o pripojenie služba informuje pomocou vyskakovacieho okna.

```
1  private void createNotificationChannel() {
2      CharSequence name = getString(R.string.channel_name);
3      String description = getString(R.string.channel_description);
4      int importance = NotificationManager.IMPORTANCE_DEFAULT;
5      NotificationChannel channel =
6              new NotificationChannel(CHANNEL_ID, name, importance);
7      channel.setDescription(description);
8      NotificationManager notificationManager =
9              getSystemService(NotificationManager.class);
10     notificationManager.createNotificationChannel(channel);
11 }
```

Výpis 5.1: Metóda triedy MqttService, ktorá vytvára notifikačný kanál na posielanie systémových notifikácií. Vytvorenie komunikačného kanála je od Android verzie 8.0 povinné, na odosielanie notifikácií.

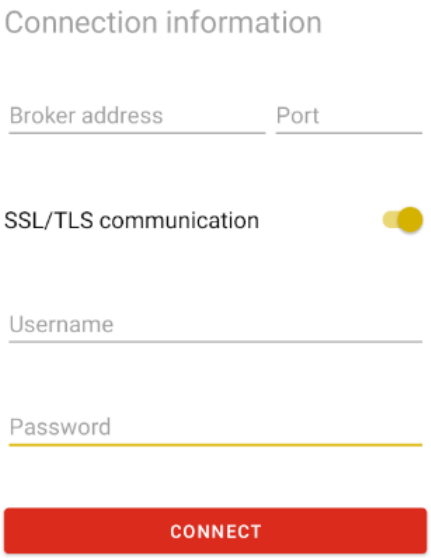
Za behu aplikácie je ďalej služba dostupná cez *MainViewModel*. Na manažment komunikácie používa, rovnako ako v prvej iterácií, pomocnú triedu *MqttHelper*. Všetká komunikácia s inteligentnou domácnosťou prebieha cez túto pomocnú triedu.

Prihlasovací formulár

Je úplne prvým zobrazeným fragmentom. Obsahuje štyri textové vstupy, prepínač a tlačítko na pripojenie (Obrázok 5.4). Informácie, ktoré je nutné vyplniť na pripojenie k zariadeniu broker sú:

- Url adresa zariadenia broker, s ktorým sa snažíme nadviazať spojenie. Adresa môže byť verejná alebo privátna, ak sa zariadenie nachádza v rovnakej sieti.
- Číslo portu cez, ktorý bude komunikácia prebiehať.
- Prihlasovacie meno, ktoré bude poslané v správe CONNECT spomínanej v sekcií 2.1.3.
- Heslo k prihlasovaciemu menu.

Prepínač vo formulári rozhoduje, či sa bude komunikácia šifrovať, alebo nie. Na šifrovanie komunikácie je zatiaľ použitý certifikát testovacieho zariadenia broker, ktorý bol pridaný pri implementácií. Aplikácia zatiaľ neposkytuje možnosť ako by mohol jej užívateľ pridať iný certifikát.



Obr. 5.4: Výsek z obrazovky aplikácie zachytávajúci prihlasovací formulár.

Polia formulára majú nastavený atribút *autofillHints*, ktorý umožňuje uložiť prihlasovacie meno a heslo ku svojmu Google účtu. Takže ak sa používateľ prihlási ku svojmu účtu na inom zariadení, bude mu ponúknuté vyplnenie údajov.

Implementácia zabezpečenej komunikácie protokolom SSL

Na vytvorenie spojenia protokolom SSL je najskôr nutné získať od zariadenia broker certifikát autenticity a v ňom obsiahnutý verejný kľúč. Spočiatku bolo zamýšľané aby si aplikácia certifikát získala sama, z adresy zariadenia broker vyplnenej v prihlasovacom formulári. Problémom však bolo, že testovacia inteligentná domácnosť používa certifikát autenticity vydaný univerzitou. Keďže Vysoké Učení Technické nepatrí medzi certifikačné autority uznané systémom Android, bol certifikát označený ako nedôveryhodný (self-signed). Vytvorenie vlastného konfiguračného súboru pre

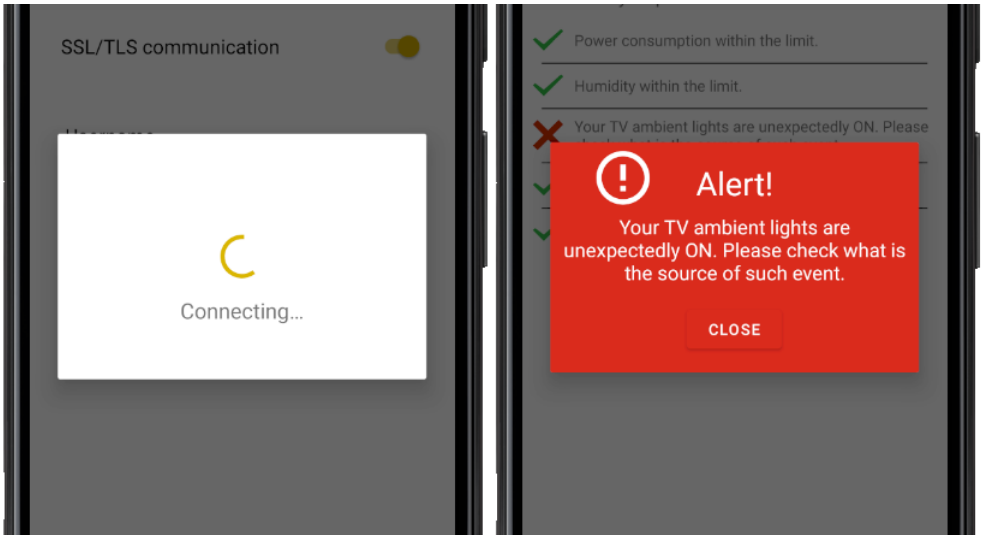
zabezpečenie pripojenia aplikácie (network_security_config.xml), tak ako je popísané v manuálových stránkach pre systém Android[3] sa ukázalo ako nefunkčné.

Ako funkčné riešenie sa ukázalo implementovať vlastnú triedu *CustomKeyStores*, ktorá implementuje systémovú abstraktnú triedu *SSLSocketFactory*. Tá je schopná načítať informácie z certifikátu ak je vo formáte „BKS“[4]. Keďže nie je možné od serveru (v tomto prípade zariadenia broker) očakávať, že svoj certifikát pošle v tomto formáte a prevod certifikátu je mimo zameranie tejto aplikácie, bol tento certifikát prevedený pri vývoji. Takéto riešenie je funkčné, ale pre širšie nasadenie je nutné, aby broker poskytoval certifikát od certifikačnej autority uznávanej systémom Android.

Pre testovacie účely bola implementovaná aj možnosť kedy zariadenie Android certifikát neoveruje. Toto riešenie je z bezpečnostného hľadiska absolútne nepoužiteľné, pretože je možné komunikáciu so serverom kompletne ovládnuť pomocou útoku „človek uprostred“ („Man-in-the-middle attack“).

Dialógové okná na zobrazenie interných notifikácií

Pri vytváraní spojenia ako aj získavaní informácií o inteligentnej domácnosti, nastávajú situácie, kedy je treba počkať na odpoveď od zariadenia broker. O čakaní treba používateľa informovať.



Obr. 5.5: Ukážka dialógových okien. Vľavo okno signalizujúce čakanie na naviazanie spojenia, vpravo upozornenie od zariadenia broker na podozrivé chovanie v domácnosti.

[3] `https://developer.android.com/training/articles/security-config`

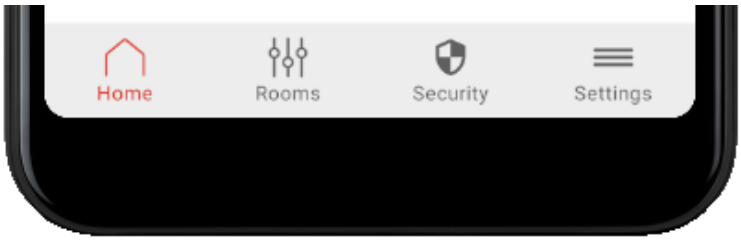
[4] Je formát definovaný v kryptografickej knižnici Bouncy Castle pre jazyk Java. `https://bouncycastle.org/specifications.html`

Rovnako je treba používateľa informovať o upozorneniach, ktoré môžu prísť z inteligentnej domácnosti. Na tento účel boli vytvorené triedy *WaitingDialog* a *BrokerAlertDialog* využívajúce Android komponentu *AlertDialog.Builder*. Komponenta umožňuje vytvoriť dialógové okno aj s polo-transparentným pozadím, znemožňujúcim ďalšej interakcie s užívateľským rozhraním (Obrázok 5.5). Okno ponúka viaceré módy ako bude pracovať. V prípade signalizácie čakania je okno nastavené tak, aby ho nebolo možné užívateľsky zrušiť. Je jednoducho nutné počkať, kým operácia neskončí. V prípade upozornení z inteligentnej domácnosti, má okno čisto informatívny charakter a preto je ho možné zrušiť systémovým tlačítkom späť, stlačením do polo-transparentnej plochy okolo dialógu, alebo stlačením tlačítka vo vnútri dialógového okna na jeho zatvorenie.

Zmeny užívateľského rozhrania z prvej iterácie

V druhej iterácií vývoja bolo zmenené aj užívateľské rozhranie aplikácie. Bol odstránený horný panel nástrojov, keďže sa nepoužíval v každom zobrazení a väčšinou obsahoval len nadpis zobrazenia. Pôsobil tiež problémy pri výpočte veľkosti dynamických systémových komponent ako *RecyclerView*, kedy sa veľkosť komponent počítala vzhľadom na veľkosť obrazovky zariadenia bez ohľadu na zmenšenie obsahu fragmentu o veľkosť horného panelu nástrojov.

Ďalšie zmeny prebehli v rámci spodného navigačného menu. Oproti prvej iterácií vývoja bola z menu odstránená položka termostatu. Globálne nastavenie teploty má význam iba pri domácnostiach s jednotným termostatom, čo ale nedáva zmysel pokiaľ používateľ využíva aplikáciu na riadenie viacerých ohrevných zariadení pre domácnosť, alebo veľkú plochu akou môže byť celé poschodie s kanceláriami. V takom prípade je z hľadiska aplikácie nemožné rozhodnúť, čo by sa v danom zobrazení malo zobrazovať. Nehovoriac o prípade kedy v súčasťou inteligentnej domácnosti žiadne zariadenie na ovládanie teploty nie je, ako to bolo aj v prípade testovacej domácnosti.



Obr. 5.6: Ukážka spodnej navigácie aplikácie v druhej iterácií vývoja s aktívnym domovským zobrazením (Home).

Pri testovaní aplikácie bolo odhalené, že ikony v spodnej navigácií samostatne nevysvetľujú aké zobrazenie predstavujú. Bol preto pod jednotlivé ikony doplnený popis fragmentu aplikácie, ktorý spúšťajú (Obrázok 5.6).

Domáca obrazovka aplikácie

Koncept rozloženia bol zachovaný z prvej iterácie, až na zjavné zmeny dizajnu niektorých prvkov (Obrázok 5.7). Funkcionalitu bolo treba doplniť o interaktívne chovanie aktivít (activities), keďže v prvej iterácií išlo iba o statické informácie na vyplnenie priestoru. V tejto iterácií sa vyžiadajú posledné aktivity od zariadenia broker pomocou MQTT správy do predmetu:

```
<home_id>/<gateway_id>/in
```

Časti *<home_id>* a *<gateway_id>* sú samozrejme zamenené za identifikátory domácnosti, ku ktorej je aplikácia aktuálne pripojená. Telo správy vyzerá nasledovne:

```
{
   "type": "query_activities",
   "timestamp": 1621194204040,
   "priority_level": 1
}
```

Výpis 5.2: Príklad obsahu MQTT správy, ktorá vyzve broker na poskytnutie zoznamu posledných aktivít v domácnosti.

Broker na správu odpovie do predmetu:

```
<home_id>/<gateway_id>/activities/out
```

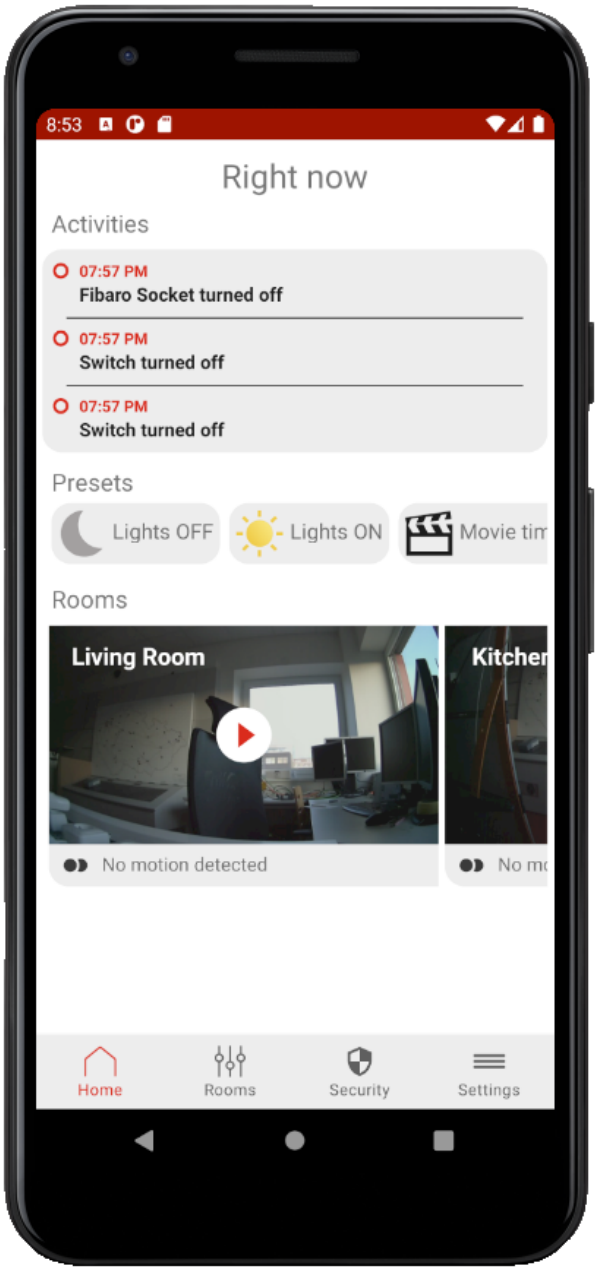
Aby sa informácie do aplikácie dostali je nutné sa najskôr prihlásiť k odoberaniu správ z daného predmetu. Keďže túto správu aplikácia potrebuje len raz, po jej doručení sa aplikácia z odberu predmetu odhlási. Namiesto tohto predmetu sa aplikácia prihlási na odber predmetu:

```
<home_id>/<gateway_id>/events/in
```

Do tohto predmetu broker uverejňuje aktuálne aktivity a upozornenia. Doručené aktivity aj upozornenia sú uložené medzi živé dáta aplikácie a na ich základe je vyvolaná odpoveď užívateľského rozhrania. V prípade domovskej obrazovky je to aktualizovanie zoznamu aktivít, ktorý sa takto obnovuje v reálnom čase.

Ďalšou zmenou na zlepšenie užívateľského rozhrania je vytvorenie novej stratégie na ukladanie obrázkov z kamier, ktoré sú zobrazované v prehľade. V prvej iterácií boli obrázky stiahnuté zo serveru pri prvom zobrazení domovského fragmentu a ďalej neboli aktualizované. Problémom bolo, že adresa, na ktorej sú obrázky dostupné,

sa nemení. Takže externá knižnica *Glide* použila obrázky z pamäte zariadenia, ktoré si pre danú adresu uložila po prvom stiahnutí zo serveru. Aby sa zachovalo použitie pamäte a zároveň boli obrázky aktuálne bola vytvorená trieda *GlideSignature*. Táto trieda obsahuje časové razítko, kedy bol obrázok uložený v pamäti stiahnutý. Pri opätovnom načítaní domovskej obrazovky, je táto časová stopa porovnaná s aktuálnym časom a ak je obrázok starší ako 10 sekúnd, je z rovnakej adresy stiahnutý nový. Aby pri načítavaní nového obrázku nezostalo okno prázdne, bol ako dočasný obrázok nastavený práve starý obrázok uložený v pamäti.



Obr. 5.7: Ukážka domovského zobrazenia aplikácie v druhej iterácií.

Zobrazenie miestností a ich zariadení

Aj štandard MQTT2GO organizuje jednotlivé zariadenia podľa miestností. Keďže ovládanie zariadení je hlavná funkcionalita aplikácie, boli miestnosti so zariadeniami umiestnené ako druhá položka navigačného menu. Zobrazenie pozostáva z komponenty *RecyclerView*, ktorej položky sú jednotlivé miestnosti. *LayoutManager* tejto komponenty je nastavený ako horizontálny, takže medzi jednotlivými miestnosťami sa užívateľ pohybuje rolovaním doprava a doľava (Obrázok 5.8).

Obr. 5.8: Ukážka zobrazenia miestností a ich zariadení. Obrázok na pravo ukazuje situáciu, kedy sa pri rolovaní doprava mení aktuálna miestnosť.

Aby bola na obrazovke len jediná miestnosť, mimo prechodu medzi miestnosťami,

bola implementovaná abstraktná trieda *SnapHelper*. Trieda rieši výpočet pozície položiek *RecyclerView* pri jeho rolovaní. Pri skúšaní rôznych implementácií tejto triedy, bola zvolená možnosť, kedy sa používateľ môže pohnúť iba na predošlú, alebo nasledujúcu miestnosť. Po ukončení gesta dotyku a ťahu sa na celú veľkosť fragmentu zobrazí tá miestnosť, ktorej väčšia časť bola viditeľná na obrazovke. Na obrázku 5.8, vpravo by to bola miestnosť kuchyňa (Kitchen).

```
1  SnapHelper helper = new LinearSnapHelper(){
2      @Override
3      public int findTargetSnapPosition(RecyclerView.LayoutManager
           layoutManager, int velocityX, int velocityY){
4          if (!(layoutManager instanceof
               RecyclerView.SmoothScroller.ScrollVectorProvider)) {
5              return RecyclerView.NO_POSITION;
6          }
7
8          final View currentItemView = findSnapView(layoutManager);
9          if(currentItemView == null) {
10             return RecyclerView.NO_POSITION;
11         }
12
13         return layoutManager.getPosition(currentItemView);
14     }
15 };
16 helper.attachToRecyclerView(rvRoom);
```

Výpis 5.3: Implementácia abstraktnej triedy SnapHelper. Kód zaistí, že sa nepočíta pozícia položiek RecyclerView, ako pri plynulom rolovaní.

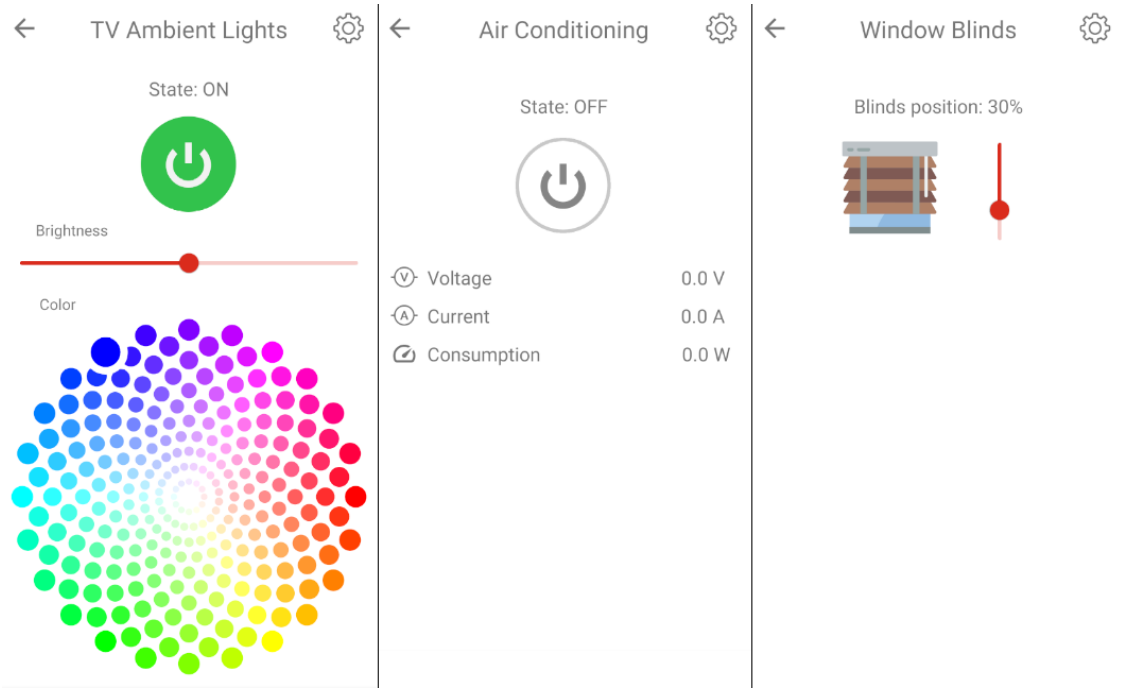
Jedna položka miestnosti sa skladá z jej názvu, ikony nastavení a dynamického zoznamu zariadení, ktoré sa v nej nachádzajú. Zoznam zariadení je zobrazený pomocou vlastnej komponenty *FoldingRecyclerView*, ktorá mení chovanie systémovej komponenty. Chovanie je zmenené tak, aby sa položky zariadení zobrazovali vedľa seba a zároveň pod sebou. Bolo nutné rozšíriť chovanie metódy *onMeasure* na výpočet počtu stĺpcov podľa šírky obrazovky zariadenia, na ktorom je aplikácia spustená.

So zariadeniami je možné interagovať dvomi spôsobmi. Prvým je krátky dotyk, ktorý zmení stav zariadenia. Stav zariadení je definovaný podľa typu zariadenia. Väčšinou ide o stav vypnutý, alebo zapnutý, no pri zariadeniach, ktoré takýto stav nemajú je definovaný inak. Napríklad zariadenia typu rolety (window blinds) majú jediný nastaviteľný parameter svoju pozíciu. V tomto prípade bola zmena stavu definovaná, ako vysunutie úplne nahor a spustenie úplne dolu. Druhým spôsobom

interakcie je dlhé stlačenie položky zariadenia, ktoré zobrazí fragment s detailom zariadenia.

Detailné zobrazenie jednotlivých zariadení inteligentnej domácnosti

Keďže jednotlivé zariadenia majú rôzne parametre, či už nastaviteľné, alebo len informatívne, bolo nutné pre jednotlivé typy zariadení vytvoriť ich detailné zobrazenia (Obrázok 5.9). Spoločnou časťou zobrazení je horný panel, v ktorom sa nachádza tlačítko späť, názov zariadenia a tlačítko nastavení. Hodnoty zo zariadení sú dynamicky aktualizované, takže ak zariadenie niekto ovláda pomocou iného zariadenia zmeny sa prejavia okamžite, ako o nich bude informovať broker. Z tohto zobrazenia je samozrejme možné jednotlivé parametre nastavovať. Pri zmene parametru je od objektu daného zariadenia, pomocou metódy *getSetMessage*, získaný predmet, kam má byť správa uverejnená a jej obsah. Následne je cez službu *MqttService* a jej pomocnú triedu *MqttHelper*, odoslaná zariadeniu broker. Históriu zmien v domácnosti je možné sledovať pomocou aktivít na domovskej obrazovke aplikácie.



Obr. 5.9: Ukážka troch rôznych detailov zariadení. V ľavo detail svetla s nastaviteľnou svietivosťou a farbou. V strede detail inteligentnej zásuvky s meraním napätia, prúdu a spotreby. Na obrázku je vypnutá, takže sú merané hodnoty nulové. V pravo detail elektronických roliet. Obrázok vedľa posuvníka je tiež dynamický.

Zobrazenie živých záberov z kamery

Úplne iným typom zariadenia je samozrejme kamera. Detailným zobrazením kamery je jej živý prenos (Obrázok 5.10). Najskôr je nutné si vyžiadať od zariadenia broker URL adresu, kde bude prenos dostupný. Z objektu reprezentujúceho kameru získame pomocou metódy *getStream* predmet, kam má byť správa uverejnená, jej obsah a predmet, do ktorého bude adresa uverejnená.

```
{
   "type":"stream",
   "timestamp":1621274584321,
   "value":"GET_STREAM"
}
```

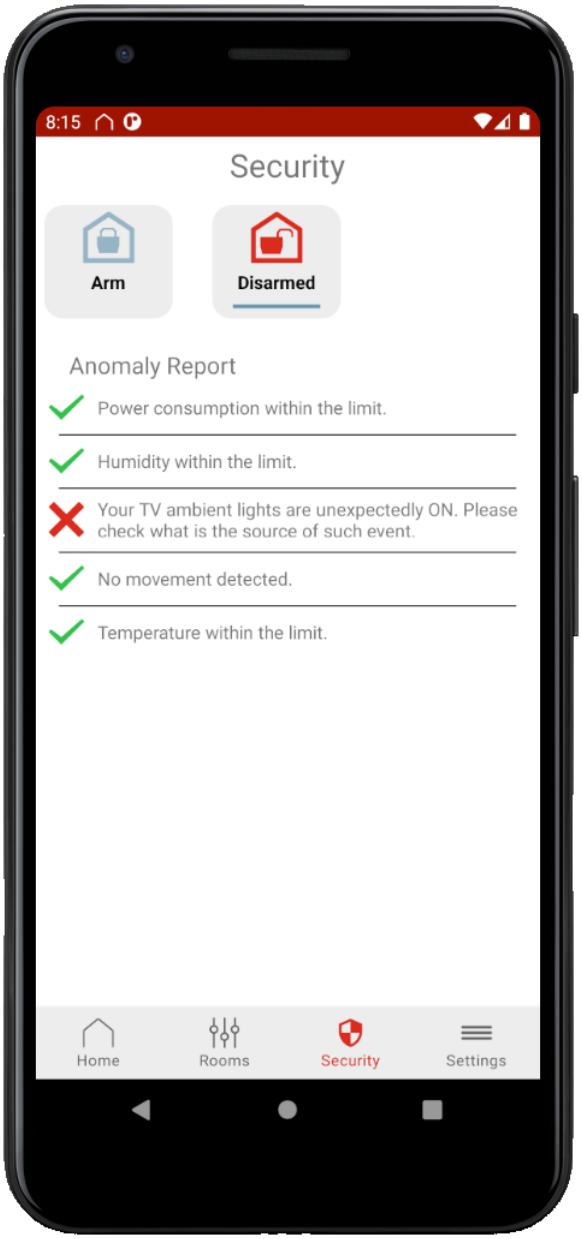
Výpis 5.4: Príklad obsahu MQTT správy na získanie URL adresy, kde bude vysielaný živý prenos.

Po získaní URL adresy v anonymnej funkcií, ktorá je spustená pri príchode odpovede zo zariadenia broker, ju predáme ako parameter akcie zmeny fragmentu. Adresa je predaná komponente *MjpegView*, ktorá prenos zobrazuje. Keďže prenos z kamery v testovacej domácnosti je vo formáte *MJPEG*, bola na jeho zobrazenie použitá externá, voľne dostupná knižnica *android-mjpeg-view*. Pri ladení zobrazenia bolo zistené, že prenos sa vykreslí určitým časovým odstupom. Aby nebolo počas tohto oneskorenia zobrazenie prázdne, je na obrazovke zobrazený obrázok získaný v predošlom fragmente.

Obr. 5.10: Ukážka zobrazenia živého prenosu z kamery v testovacej domácnosti.

Zobrazenie zabezpečenia domácnosti

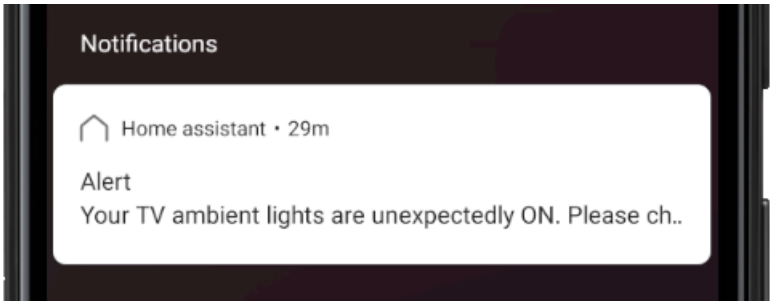
Ďalšou dôležitou súčasťou inteligentnej domácnosti je zabezpečenie (Security). Do detailného zobrazenia jednotlivých režimov zabezpečenia sa používateľ dostane cez spodnú navigáciu, konkrétne jej tretiu položku.



Obr. 5.11: Ukážka zobrazenia zabezpečenia domácnosti. Obsahuje režimy zabezpečenia a dynamický zoznam anomálií. Aktívny režim je neaktívny (Disarmed) a je zistená jedna anomália znázornená ikonou červeného krížiku.

Systém môže mať iba jeden aktívny režim zabezpečenia, preto aktiváciou iného sa automaticky deaktivujú ostatné. Aktivovaný režim je znázornený podčiarknutím

jeho názvu (Obrázok 5.11). V testovacej domácnosti sú len dva režimy zabezpečenia a to aktívne a neaktívne. Aplikácia je ale pripravená zobrazovať akýkoľvek počet režimov, ktoré sa budú ďalej radiť za sebou a po naplnení šírky obrazovky do ďalšieho radu. Pod režimami sa nachádza zoznam definovaných anomálií v domácnosti. Aktuálne anomálie sú znázornené výstražnou ikonou červeného krížiku. Pokiaľ od zariadenia broker príde správa informujúca o anomálií je cez službu *MqttService* vytvorená systémová notifikácia (Obrázok 5.12). Notifikácia informuje o zistenej anomálií a po jej otvorení zobrazí aplikáciu, ktorá informuje o anomálií pomocou dialógového okna (Obrázok 5.5). Služba spracováva správy od zariadenia broker, aj keď sa s aplikáciou aktuálne nepracuje a môže preto vytvárať systémové notifikácie.



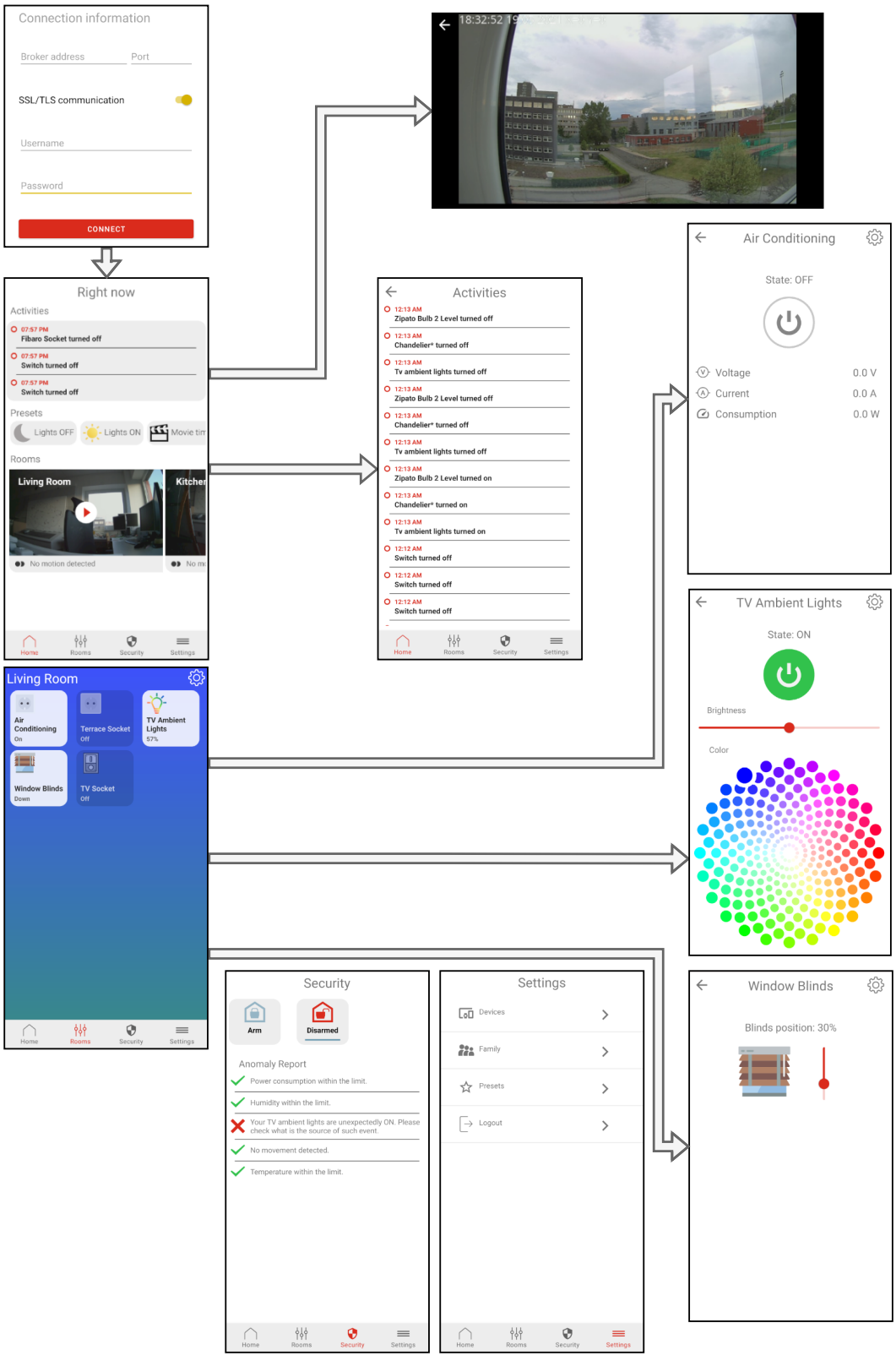
Obr. 5.12: Ukážka systémovej notifikácie vytvorenej službou *MqttService*, ktorá pracuje aj keď je aplikácia neaktívna.

Navigácia v aplikácií pomocou grafu

Presun medzi jednotlivými zobrazeniami v aplikácií je zabezpečený pomocou navigačnej komponenty. Tá sa skladá z troch častí:

- **Navigačný graf** – je zápis všetkých destinácií a prechodov medzi nimi vo formáte XML (Obrázok 5.13). Graf obsahuje aj informácie o argumentoch, parametroch, ktoré je možné zobrazeniu poslať.
- **Navigačný kontajner** (NavHost, NavHostFragment) – je kontajner, ktorý obsahuje možnosti destinácií, kam sa z aktuálneho zobrazenia môžme dostať.
- **Navigačný kontroler** (NavController) – je objekt, ktorý riadi zmeny obsahu kontajneru po zmene aktuálneho zobrazenia. Pohyb medzi zobrazeniami môže byť vykonaný pomocou jedného prechodu, alebo definovania konečnej destinácie.

Takýto spôsob navigácie automaticky rieši transakcie medzi zobrazeniami, pohyb späť, poskytuje typovú kontrolu posielaných argumentov, podporuje zdieľanie komponenty ViewModel medzi destináciami navigačného grafu. Graf je možné zostaviť, vo vývojovom prostredí Android Studio, aj pomocou grafického editoru.



Obr. 5.13: Ukážka navigačného grafu aplikácie. Jednotlivé zobrazenia sú destinácie a prechody medzi nimi sú navigačné akcie.

Záver

Cieľom diplomovej práce bolo vytvoriť natívnu mobilnú aplikáciu pre operačný systém Android, ktorá bude umožňovať riadenie inteligentnej domácnosti komunikujúcej protokolom MQTT. V rámci diplomovej práce autor definoval pojem Internet vecí, jeho súčasti, architektúru a jeho komunikačné modely. Rozobral protokoly aplikačnej vrstvy, ktoré sa v problematike vyskytujú a podrobnejšie popísal fungovanie a súčasti protokolu MQTT a jeho konvencie MQTT2GO. Preskúmal existujúce aplikácie na riadenie inteligentnej domácnosti a definoval ich špecifiká.

V praktickej časti zdôvodnil rozhodnutia pri návrhu aplikácie a popísal správne chovanie aplikácie. Podľa návrhu bola aplikácia úspešne implementovaná. Umožňuje pripojenie a prihlásenie k inteligentnej domácnosti a ďalšiu komunikáciu protokolom MQTT, ktorá je navyše zabezpečená. Aplikácia zobrazuje a ovláda všetky dostupné zariadenia v referenčnej domácnosti. Keďže komunikácia v domácnosti sa riadi konvenciou MQTT2GO, ktorá sa stále vyvíja a nie je verejne používaná, nebola aplikácia vydaná v obchode Google Play. V prílohe sa ale nachádza inštalačný súbor aplikácie, pomocou ktorého je možné aplikáciu spustiť a vyskúšať.

V priebehu implementácie bolo narazené na niekoľko problémov s testovacou domácnosťou, ktoré komplikovali vývoj. Nedostupnosť laboratória znemožnila implementáciu pridávania a editácie zariadení v domácnosti, z čoho bolo po dohode s vedúcim práce ustúpené.

Aplikácia sa v žiadnom prípade nemôže z hľadiska funkcionality, alebo dizajnu porovnávať s existujúcimi komerčnými riešeniami, no na demonštrovania ovládania inteligentnej domácnosti komunikujúcej podľa konvencie MQTT2GO postačuje.

Literatúra

[1] Alaa Abbood, Qahtan Shallal, and Mohammed Fadhel. Internet of things (iot): a technology review, security issues, threats, and open challenges. *Indonesian Journal of Electrical Engineering and Computer Science*, 21, 08 2020. doi:10.11591/ijeecs.v20.i3.pp1685-1692.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, 2015. doi:10.1109/COMST.2015.2444095.

[3] M. A. Bouras, F. Farha, and H. Ning. Convergence of computing, communication, and caching in internet of things. *Intelligent and Converged Networks*, 1(1):18–36, 2020. doi:10.23919/ICN.2020.0001.

[4] S. N. Swamy and S. R. Kota. An empirical study on system level aspects of internet of things (iot). *IEEE Access*, 8:188082–188134, 2020. doi:10.1109/ACCESS.2020.3029847.

[5] The HiveMQ Team. Publish subscribe - mqtt essentials: Part 2. [online], 1 2015. URL: <https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/>.

[6] B. Mishra and A. Kertesz. The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, 8:201071–201086, 2020. doi:10.1109/ACCESS.2020.3035849.

[7] The HiveMQ Team. Mqtt client and broker and mqtt server and connection establishment explained - mqtt essentials: Part 3. [online], 7 2019. URL: <https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/>.

[8] The HiveMQ Team. Quality of service 0,1 2 - mqtt essentials: Part 6. [online], 2 2015. URL: <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>.

[9] MQTT2GO. Mqtt2go convention. [online], 9 2020. URL: <https://mqtt2go.github.io/>.

[10] Richard Harper. *Inside the Smart Home*. Springer London, London, 2003.

[11] Margaret Rouse. smart home or building (home automation or domotics). [online], 7 2020. URL: <https://internetofthingsagenda.techtarget.com/definition/smart-home-or-building>.

[12] Y. Wenbo, W. Quanyu, and G. Zhenwei. Smart home implementation based on internet and wifi technology. In *2015 34th Chinese Control Conference (CCC)*, pages 9072–9077, 2015. doi:10.1109/ChiCC.2015.7261075.

[13] Rixin Xu, Qiang Zeng, Liehuang Zhu, Haotian Chi, and Xiaojiang Du. Privacy leakage in smart homes and its mitigation: Ifttt as a case study. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2018. doi:10.1109/PCCC.2018.8711216.

[14] Mqtt software. [online], 2 2021. URL: <https://mqtt.org/software/>.

[15] Ginanjar Prabowo, Hatma Suryotrisongko, and Aris Tjahyanto. A tale of two development approach: Empirical study on the maintainability and modularity of android mobile application with anti-pattern and model-view-presenter design pattern. In *2018 International Conference on Electrical Engineering and Informatics (ICELTICs)*, pages 149–154, 2018. doi:10.1109/ICELTICS.2018.8548784.

[16] Hugo Källström. Increasing maintainability for android applications. Master's thesis, Umea University, Department of Computing Science: Implementation and Evaluation of Three Software Architectures on the Android Framework, 2016.

[17] Luke Wroblewski. *Mobile First.* A Book Apart, 1 2011.

[18] Ryan Cohen. *GUI Design for Android Apps.* Apress, L.P., 2014.

[19] John Kouraklis. *MVVM as Design Pattern*, pages 1–11. Apress, 10 2016. doi:10.1007/978-1-4842-2214-0_1.

[20] Dawn Griffiths. *Head first Android development.* O'Reilly, Beijing, 2nd edition edition, 2017.

Zoznam symbolov, veličín a skratiek

API Rozhranie pre programovanie aplikácií – Application Programming Interface

CoAP Aplikačný protokol s obmedzenými prostriedkami – Constrained Application Protocol

DDS Služba distribúcie dát – Data Distribution Service

GPS Globálny lokalizačný systém – Global Positioning System

HomePNA Aliancia domácich telefónnych sietí – Home Phoneline Networking Alliance

HTTP Hypertextový prenosový protokol – Hypertext Transfer Protocol

IFTTT Ak toto potom tak – If This Then That

IoT Internet vecí – Internet of Things

IP Protokol sieťovej vrstvy – Internet Protocol

JSON Objektový zápis jazyka JavaScript – JavaScript Object Notation

LTE Dlhodobý vývoj – Long-Term Evolution

MQTT Telemetria prenosu správ vo frontách – Message Queue Telemetry Transpor

MVP Model-Zobrazenie-Prezentér – Model–View–Presenter

MVVM Model-Zobrazenie-Model pre zobrazenie – Model–View–ViewModel

NFC Bezdrôtová komunikácia krátkeho dosahu – Near Field Communication

REST Prenos reprezentačného stavu – Representational state transfer

RFID Identifikácia na rádiovej frekvencií – Radio Frequency Identification

TCP Protokol s kontrolovaným prenosom – Transmission Control Protocol

UDP Používateľský datagramový protokol – User Datagram Protocol

UWB Ultra-širokopásmová bezdrôtová technológia – Ultra-Wideband

XML Rozšíriteľný značkovací jazyk – Extensible Markup Language

XMPP Rozšíriteľný protokol správ a prítomnosti – Extensible Messaging and Presence Protocol

A Obsah priloženého média

- `/` ..... koreňový adresár priloženého média
  - `pdf` ..... pdf
    - `diplomova-praca.pdf`
  - `src` ..... zdrojové kódy aplikácie
  - `text` ..... zdrojové textové soubory
  - `app.apk` ..... inštalačný súbor aplikácie
  - `readme.txt` ..... súbor s popisom obsahu média a návodom