ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Wiesner** Jméno: **Filip** Osobní číslo: **475400**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Adaptace uživatelského rozhraní v závislosti na dostupnosti IOT zařízení

Název bakalářské práce anglicky:

User interface adaptation in regards to IOT device availability

Pokyny pro vypracování:

Cílem práce je navrhnout a vytvořit mobilní aplikaci pro platformu Android, která bude sloužit jako univerzální ovladač pro několik zařízení a jejíž obsah se bude optimalizovat/měnit ve vztahu s okolními zařízeními.
Funkčnost aplikace otestujte na několika zařízeních podle předem sepaných testovacích scénářů.

Seznam doporučené literatury:

[1] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. 2015. The Internet of Things Has a Gateway Problem. In Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile ’15). Association for Computing Machinery, New York, NY, USA, 27–32.
[2] Rüßmann, M., Lorenz, M., Gerbert, P., Waldner, M., Justus, J., Engel, P., & Harnisch, M. (2015). Industry 4.0: The future of productivity and growth in manufacturing industries. Boston Consulting Group, 9(1), 54-89.
[3] Mohammad-Mahdi Moazzami, Daisuke Mashima, Ulrich Herberg, Wei-Pen Chen, and Guoliang Xing. 2016. SPOT: a smartphone-based control app with a device-agnostic and adaptive user-interface for IoT devices. In Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct (UbiComp '16). ACM, New York, NY, USA, 670-673. DOI: https://doi.org/10.1145/2968219.2968345

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Adaptace uživatelského rozhraní v závislosti na dostupnosti IOT zařízení

Filip Wiesner

Vedoucí práce: Ing. Jiří Šebek

Studijní program: Softwarové inženýrství a technologie, Bakalářský

22. května 2020

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Jiřímu Šebkovi za téma, které mi pomohl vybrat, a za jeho čas věnovaný našim konzultacím.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 15. 5. 2020 .....

Abstract

We are at the beginning of a big boom of smart devices that are available to ordinary households. This work examines ways to optimize the form of UI and control of these devices depending on their and the user's context. It also offers an implementation of a mobile application on the Android platform, which follows the outcome of the analysis and enables control of Yeelight products. In contrast to the analysis that suggests the use of BLE, only GPS technology is implemented in the application. The results are then tested by system tests, which verify the usability of the application.

Abstrakt

Stojíme na počátku velkého boomu chytrých zařízení, která jsou dostupná v běžné domácnosti. Tato práce zkoumá způsob optimalizace zobrazení a ovládání těchto zařízení v závislosti na jejich i uživatelově kontextu. Zároveň nabízí konkrétní implementaci mobilní aplikace na platformě Android, která se řídí výsledky analýzy a umožňuje ovládání produktů Yeelight. Oproti analýze, která navrhuje využití BLE, je v aplikaci implementována pouze technologie GPS. Výsledky jsou poté otestovány systémovými testy, které ověřují použitelnost aplikace.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Čtvrtá průmyslová revoluce se pomalu stává skutečností [23, 22], a zatímco lidé sedí doma, velké korporace pomalu nahrazují levnou pracovní sílu za ještě levnější mechanickou sílu. Tato revoluce se však dostává i do běžných domácností v podobě různých virtuálních asistentů využívaných (jak už název napovídá) jako pomoc s jednoduchými věccmi jako například s nastavením budíku, zjištěním předpovědi počasí nebo odpověďmi na jednoduché otázky. Vývoj takového virtuálního asistenta však není levný, a tak není překvapivé, že přední příčky prodeje asistentů obsadili největší technologičtí giganti. Google se svým Google asistentem a Amazon s Alexou.

Mimo asistenty se však do domácností dostala i jiná chytrá zařízení. Nejběžnější jsou chytré žárovky, které se dají ovládat přes mobilní aplikaci a lze u nich měnit nejen jas, ale většinou také chromatičnost nebo dokonce barvu, dále bezpečnostní kamery, které zaznamenávají pohyb a podezřelé pohyby na pozemku pošlou obratem majiteli, domovní zvonky se zabudovanou kamerou, které majiteli hned ukáží, kdo stojí za dveřmi a podobně.

Jednu věc ale mají všechna tato zařízení společnou. Každé má svoji vlastní mobilní aplikaci. To v praxi znamená, že vlastník těchto chytrých zařízení může mít nainstalovaných třeba pět různých mobilních aplikací, a když přijde domů z práce, rozsvítí si světla v jedné, zapne topení v druhé a udělá si kávu ve třetí. Určitě se jedná o lepší zážitek než v rukou měnit fyzické ovladače, ale stále je zde viditelná velká fragmentace [32]. Tento problém se dá částečně, v mnohých případech i kompletně, vyřešit integrací s výše zmíněnými virtuálními asistenty, ale ovládat celou svou domácnost hlasem nemusí být vždy optimální řešení. Navíc je podporována jen hrstka nejpoužívanějších jazyků, mezi nimiž mimo jiné chybí i čeština. Je tedy potřeba vytvořit jednotné prostředí, které seskupí všechna chytrá zařízení a jejich možnosti ovládání.

Druhý problém vyplývající z četnosti chytrých zařízení na jednom místě je jejich přesycení [25]. Jedna místnost může v budoucnu ukrývat spousty komunikujících zařízení a hledat mezi nimi by byl problém i v případě, že by bylo ovládací prostředí ucelené a ovládací prvky na jednom místě. Prostředí by totiž muselo být uspořádáno do kategorií a podkategorií a v kompetitivním světě, kde může uživatele odradit i jedno kliknutí navíc, nemůže dlouhý seznam ovládacích prvků fungovat. Je tedy třeba upřednostňovat různá zařízení podle kontextu uživatele a následně smysluplně vystavovat jejich ovládací prvky. Tímto způsobem budou nejen všechny ovládací prvky na jednom místě, ale také chytře zobrazeny pro maximální efektivitu ovládání chytré domácnosti.

Kapitola 2

Rešerše

Jak bylo naznačeno v úvodu, tento projekt se zabývá několika problémy najednou. Z tohoto důvodu nebylo možné najít řešení, které by pokrývalo celý problém, ale pouze dvě práce, kde se každá zabývá polovinou.

Práce od Moazzami [18] řeší roztříštěnosti ovládání Internet of Things (IoT) zařízení. Rozpoznává, že tento problém vznikl velkým množstvím proprietárních aplikací a nabízí řešení s názvem SPOT, jež je platforma pro chytré spotřebiče více prodejců založená na chytrých telefonech (smartphone-based platform for multi-vendor smart-home appliances). Prototyp této platformy je implementován na systému Android a nabízí možnost ovládat zařízení od pěti prodejců najednou.

Práce od Rasch [21] se zabývá proaktivním objevováním embedded zařízení v zahlceném prostoru v závislosti na kontextu uživatele a nabízí několik algoritmů pro opakované vyhledávání relevantních zařízení v odpověď na změnu kontextu uživatele.

2.1 Android

Android je open source mobilní operační systém postavený na Linux kernelu. Je vyvíjen v rámci AOSP (Android Open-Source Project), který obsahuje samotný operační systém, middleware a aplikace jako email nebo messaging [6].
V Android ekosystému je přes milion aplikací a více než 24 tisíc různých zařízení [4]. To z něj dělá nejdiverzifikovanější mobilní platformu a ideální místo pro aplikaci, která cílí na širokou paletu uživatelů.
Hlavním programovacím jazykem byla Java, ale Google v roce 2017 veřejně podpořil Kotlin jako jazyk pro vývoj pro platformu Android a v roce 2019 oznámil, že další vývoj Androidu a jeho knihoven bude "kotlin first" a sesadil tak Javu z postu oficiálního jazyku pro Android development [13].

2.2 Internet of Things

Internet of Things (česky internet věcí) je označení pro síť propojených zařízení, která spolu mohou komunikovat za účelem optimalizace různých procesů. Často se o tomto pojmu mluví v souvislosti s Průmyslem 4.0, ve smyslu automatizace výroby či dopravy [17, 30].

V kontextu produktů pro běžné spotřebitele se zaměňuje za pojem Internet of Services [2], ve kterém je jako Service míněno hotové řešení, které řeší nějaký problém a může pod sebou mít několik zařízení, která spolu kooperují. Například u klimatizace, kde je potřeba několika teplotních senzorů a ovládací prvek.

2.3 Technologie

Mimo vybrané technologie existují i jiné varianty, které mohly být a nebyly využity. Když se řekne Android, tak kromě klasického přístupu s pomocí Kotlinu či Javy přijdou na mysl i cross-platform řešení jako Flutter, React Native nebo Xamarin. Tyto varianty nebyly zvoleny, protože jejich hlavní výhoda, vyvážená menší flexibilitou, není vůbec zapotřebí. Aplikace je v této fázi mířená pouze pro Android, a tak není cross-platform řešení potřeba.

Jako testovací IoT zařízení byly zvoleny žárovky Philips Hue kvůli velké oblíbenosti a Yeelight kvůli cenové dostupnosti. Byly zváženy i výrobky od firem Wyze a Sengled, ale jejich produkty nejsou dostupné na českém trhu.

Kapitola 3

Analýza

3.1 Propojení s chytrými zařízeními

V dnešní době je běžné, že s uvedením chytrého zařízení na trh vystaví výrobce veřejné API, které mohou vývojáři volně používat a implementovat ve svých aplikacích. Dokonce i Google umožňuje vývojářům přidávat podporu pro své produkty možností naučit Google asistenta povely, kterými mohou produkt ovládat.

Jelikož takové API výrobci stejně potřebují pro své vlastní ovládací nástroje, je toto zveřejnění prakticky levná reklama. Vývojáři nezávislých (3rd party) aplikací mohou toto API implementovat, upozornit více uživatelů na existenci produktu a současně jim nabídnout ovládání skrz prostředí, které znají. Dobrým příkladem je například Apple HomeKit, který implementuje desítky komerčních řešení a uživatelé tak mohou skrze tuto aplikaci narazit na zařízení či firmu, kterou do té doby neznali.

Mimo klasické HTTP requesty lze ještě s různými zařízeními, která to umožňují, komunikovat pomocí technologie Bluetooth Low Energy (BLE). Jedná se o Bluetooth technologii speciálně vyvinutou pro komunikaci se zařízeními, jež si nemohou dovolit spotřebu, která je potřeba ke komunikaci přes klasické Bluetooth. Přestože BLE potřebuje k provozu jen zlomek energie potřebné pro klasické Bluetooth [26], zůstává jeho funkcionalita skoro stejná.

Narozdíl od komunikace přes HTTP je však způsob komunikace přes BLE jen vzácně zveřejňován a výrobci většinou využívají proprietární protokoly, jelikož neexistuje veřejný standard, (jakým je právě HTTP pro internetovou síť) na kterém by se výrobci dohodli.

3.2 Vzdálenost a geolokace

Jak už bylo nastíněno v úvodu, jedním z cílů práce je zlepšení přehlednosti v hodně saturovaných místech pomocí řazení, filtrace a následné nabízení ovládacích prvků jednotlivých zařízení podle jejich vzdálenosti a dostupnosti. Tudíž jedním z klíčových požadavků je zjištění informací k tomu potřebných.

3.2.1 GPS a WiFi

Prvním a asi nejznámějším a nejjednodušším způsobem lokace zařízení je využití klasického souřadnicového systému pomocí kombinace GPS a jiných metod. Na platformě Android je zjišťování polohy abstrahováno do několika nástrojů (nejdůležitější je FusedLocationProviderClient), ale v základu se opírá o kombinaci GPS a WiFi technologií.

Výhod je hned několik. Jelikož se jedná o relativně staré technologie, skoro každé chytré zařízení by mělo mít k tomuto způsobu geolokace přístup. Dále není potřeba žádná interakce, tudíž nezáleží na technologiích, jež koncové zařízení ovládá.

Toto řešení má však i spoustu nevýhod. Největší z nich je nutnost uživatele nastavit polohu manuálně (například zmáčknutím tlačítka, když se bude nacházet vedle zařízení). Také je problém s přesností této metody, která může být v ideálním případě jeden metr, ale při špatném signálu může nabývat až desítek metrů. Průměr je však chyba několika metrů [28].

3.2.2 BLE

Druhým způsobem, jak určit vzdálenost, je využití BLE. Už se tedy nejedná o geolokaci, ale pouze o určení vzdálenosti či síly signálu. Oproti GPS je také nutná aktivní interakce s cílovým zařízením. Odměnou je však větší přesnost (za podmínky správné kalibrace) a spolehlivost.

odhadu vzdálenosti se používá pouze síla signálu v decibelech, kterou mohou ovlivnit různé překážky, jako např. zdi či nábytek, a je tedy třeba aproximovat odhad vzdálenosti z několika měření pro zlepšení kvality a zmenšení chyby [14].

Druhou výhodou nad technologií GPS po přesnosti je dostupnost v uzavřených prostorách. Pro práci s GPS je nutné spojení s více satelity, pomocí kterých dojde k výpočtu polohy. Tato spojení může být však v uzavřených prostorách prakticky nemožné navázat, a v tom případě je lokace pomocí GPS nemožná. Jelikož však určení vzdálenosti pomocí BLE nevyžaduje žádná taková připojení a nutná je pouze komunikace se zařízením, které ovládáme, je tato možnost pro využití v uzavřených prostorách ideální.

3.2.3 GPS & WiFi + BLE

Sloučením dvou výše zmíněných technologií dostaneme ideální geolokační systém, který je schopen uživatele lokalizovat a poté s větší přesností určit vzdálenost od zařízení, která spravuje.

Sloučením však nezískáme jen kombinaci všech výhod, ale bohužel i všech nevýhod. Pořád musíme dbát na to, že technologie BLE není pro všechna zařízení dostupná a GPS signál nebývá všude k dispozici.

3.3 Kotlin

Veškerá logika bude psaná v programovacím jazyku Kotlin od (původně České) firmy JetBrains. Jedna z velkých výhod tohoto jazyku je jeho schopnost běžet prakticky všude a na všem. Kotlin compiler umožňuje kompilaci pro tři backendy: Javascript, JVM a Native.

Native pak nabízí různé targety jako například: Windows, iOS, Android, WebAssembly a další.

Kotlin je staticky typovaný jazyk a dal by se nejlépe přirovnat k jazykům jako Java nebo Swift, od kterých si mnoho vypůjčuje. Od Javy se nejvíce liší v následujících bodech:

- Možnost hodnoty nabývat null stavu musí být explicitně definována.
- Funkce jsou tzv. *first class citizen*, což v praxi znamená, že jakýkoliv argument, návratová hodnota nebo proměnná může být funkce, a to bez nutnosti využití předdefinovaných prototypů nebo Single Abstract Method (SAM) tříd, jež hojně využívá právě Java.
- Asynchronní (neblokující) styl podporován na jazykové úrovni klíčovým slovem suspend, které označuje funkci, která pozdrží vlákno, na kterém je volána.
- Existence *extension* funkcí, *inline* funkcí a funkcí *with receivers*.
- Existence *infix* funkcí a možnosti definovat nebo přetížit běžné operátory jako + nebo /.

3.4 Android

3.4.1 Historie

Když se v roce 2008 poprvé ukázal plnohodnotný Android světu, FOSS (Free and Open Source Software) a Linux komunita byla nadšená a očekávala velké věci. [11] Narozdíl od iOS se totiž jednalo o otevřenou platformu, jejíž zdrojový kód bude nejen veřejný, ale kdokoliv jej bude moci i upravit. Díky tomu byla adopce Androidu rychlá a rozsáhlá a dnes se kromě telefonů používá i v zařízeních jako letové monitory pro pasažéry či bankomaty.

3.4.2 User Interface a User Experience (UI & UX)

Při tvorbě mobilní aplikace je stejně jako funkcionalita a bezchybné fungování důležitý vzhled a přehlednost. Vzhled a struktura aplikace má kladný vztah k zapojení (engagementu) a návratnosti uživatelů [16] Tato vlastnost začne být obzvláště důležitá, když uvážíme, že uživatel je jen pár kliknutí od instalace další z více než milionu aplikací [4], které jsou v Google Play obchodě dostupné. To vytváří velký tlak na User Experience (UX) designéry, kteří se starají o vzhled a strukturu aplikace.

Jednou z vlastností dobrého UX designu je schopnost uživatele jen letmým pohledem na aplikaci poznat, kde se právě nachází a jaké jsou možnosti interakce. Tomu velmi napomáhá unifikovaný design celého systému, jehož poslední velkou změnu Google uvedl v Androidu 5.0 s klíčovým označením Lollipop a pojmenoval ho Material Design [12].

3.4.2.1 Material Design

Významné ale nebylo jen aktualizování systémového UI. Snad ještě významnější bylo sepsání tvz. Material Design guidelines (design pravidel), kterých se mohou vývojáři držet při vytváření designu pro jejich vlastní aplikace, což velmi pomohlo k unifikování designu v Android ekosystému.

Google jej uvedl takto: *Material metafora je sjednocující teorií racionalizovaného prostoru a systému pohybu. Materiál je založen na hmatové realitě, inspirovaný studiem papíru a inkoustu, ale technologicky pokročilý a otevřený fantazii a magii. [12] Dá se to chápat tak, že tento design je založen na designu papíru a inkoustu, ale není limitován tím, čeho lze dosáhnout v reálném světě. [12]*

Dnes již existují i komunitní projekty, které na tomto designu staví. Například knihovna Material-UI pro webový UI framework React, která má přes milion stažení týdně. [19]

3.4.2.2 Activity & Fragment

Co se týče struktury aplikace z pohledu vývojáře, tak Android historicky strukturoval obrazovky do jednotlivých Aktivit. Aktivita je samostatný prvek aplikace reprezentovaný jako jedna obrazovka s vlastním stavem a životním cyklem.

Když chce pak uživatel přejít z jedné obrazovky do druhé (z jedné Aktivity do druhé Aktivity), musí si první Aktivita uložit svůj stav, protože může být kdykoliv zabita, a případně ještě předat určité informace nové Aktivitě skrze tvz. Bundle obsahujícím primitivní hodnoty, který následně dostane nová Aktivita při startu (ve všech inicializačních metodách) [3].

Když se ale ukázalo, že chytré telefony nebudou jediné mobilní zařízení, která podporují Android, musel Google vyvinout jiný UI prvek, který bude více modulární. Na trh totiž přišly tablety, které uživatelům nabízely větší obrazovky, na nichž jedno-obrazovkové aplikace vypadaly neohrabaně. Tento nový modulární prvek uživatelského rozhraní byl představen s Android verzí 3.0 [29] a jmenoval se Fragment. Fragmenty nejsou přímou náhradou Aktivity, protože ji stále potřebují jako svého "hostitele", ale velice zjednodušily tvorbu modulárního UI. Chovají se podobně jako Aktivity v tom smyslu, že mají podobný lifecycle, ale narozdíl od Aktivity jich může být najednou zobrazeno více a mohou mezi sebou jednodušeji sdílet data (přes hostitelskou aplikaci). Jedná se tedy o takové lightweight aktivity, které slouží k zobrazení komplexních dat, se kterými lze interagovat, a která vlastní a spravují svůj stav. Současně s Fragmenty tedy přišla i podpora pro tablety a dnes je již běžný postup mít v aplikaci pouze jednu Aktivitu, na které se střídá více Fragmentů. [29]

3.4.2.3 Jetpack Compose - declarative UI

V roce 2019 ukázali na Google I/O konferenci budoucnost tvorby Android UI a představili v rámci programu Jetpack nový framework Compose, který zásadně mění způsob vytváření UI pro nativní vývoj Android aplikací [8]. Jedná se o nástroj, který umožňuje deklarativní způsob tvorby uživatelského rozhraní skrze funkce jazyku Kotlin. UI se už dnes píše částečně deklarativně pomocí XML, ale tento nový způsob cílí na eliminaci roztříštěnosti zdrojových kódů.

V této chvíli se o UI v jedné obrazovce nebo dokonce v jednom elementu (View) stará více souborů. Prvně se o vzhled stará XML soubor, který určuje strukturu a odkazuje se na jiné soubory (například styly, lokalizaci či jiné atributy). Tato XML struktura je pak "nafouknuta" při inicializaci a následně naplněna hodnotami a jinak upravena v Aktivitě, Fragmentu či View. Další problém je v nastavení a zjištění aktuálního stavu. Jednotlivé Views mají svůj vlastní stav, který vývojář nemá přímo pod kontrolou. Například Spinner (dropdown menu) oznámí změnu hodnoty až po jejím změnění a neumožní tuto změnu vetovat. Když chce tedy vývojář omezit výběr možností, musí hodnotu po výběru uživateli změnit zpět, čímž vyvolá další událost, kterou Spinner oznámí [8].

Toto a další má napravit Compose, který pomocí compiler pluginu a vlastností Kotlinu umožní deklarativní tvorbu UI uvnitř kódu bez nutnosti vytvářet XML soubory pro každý layout. Bohužel je Compose ještě v brzké fázi vývoje a feature set ještě není kompletní.

3.4.3 Shrnutí

Vývoj pro platformu Android se kromě změny preferovaného jazyku na Kotlin a vývoji first-party knihoven moc nezměnil co se obsahu týče, ale velké změny probíhají v toolingu. K dispozici je relativně nový Navigation Editor a v Beta verzi Android studia je nový Animation editor kooperující s novým Motion Layoutem. Jetpack Compose je možná velkou revolucí, ale přestože bylo oznámeno, že beta verze vyjde někdy v roce 2020, není to ještě stabilní základ, na kterém by se dal psát produkční kód. Přes snahu spokojit se se základy Compose, které jsou k dispozici už dnes a přes relativní jednoduchost tohoto projektu, nebude Compose využit a vývoj bude probíhat "klasickou cestou".

3.5 Ovládání a notifikace

Při návrhu UX (user experience) designu je nutné nejprve si uvědomit, na koho aplikace cílí a jak bude tato cílová skupina aplikaci využívat. Důležité pro nás bude:

- Jak častá by měla být interakce uživatel -> aplikace a jak častá aplikace -> uživatel. Jinými slovy, jak často by na sebe měla aplikace upozorňovat a jak často bude uživatel interagovat s aplikací z přímé potřeby. K tomuto se váže otázka, jak velká část ovladatelnosti bude možná z interaktivních notifikací a pro co bude muset uživatel otevřít plnohodnotnou aplikaci.
- Rozvržení UI v závislosti na potřebách uživatele. Jaké informace a ovládací prvky je nutné zobrazovat přednostně a co je možné skrýt pod nutnost další interakce.
- Rozvržení UI v závislosti na dostupných zařízení, se kterými je interakce možná.

3.5.1 Notifikace

Komunikace s uživatelem a schopnost nabídnout veškeré v tu chvíli potřebné ovládací prvky bez nutnosti otevřít plnohodnotnou aplikaci bude v této práci hrát velkou roli.

Co se týče návrhového hlediska, bude největší problém definovat, co všechno bude uživatel schopen udělat z notifikační lišty. Je třeba nabídnout rozumné množství ovladatelnosti, které

uživateli ušetří časté otevírání aplikace, ale na druhou stranu není cílem uživatele zaplavit přemírou informací a možností, které v naprosté většině případů nevyužije. Také je třeba brát v úvahu možnosti platformy Android. Jako například fakt, že veškerá interakce bude muset probíhat skrze jednoduchá tlačítka. Slidery či toggle buttons nejsou nijak podporovány.

Implementační stránka věci také přináší pár problémů. Mimo omezenou paletu druhů interakce bude také problém s procesy na pozadí.

3.5.2 Procesy na pozadí

V posledních několika iteracích systému Android byly background procesy dosti omezeny či dokonce zakázány. Uživatel má nyní mnohem větší kontrolu nejen nad notifikacemi, ale také ve správě procesů, které běží na pozadí v zájmu šetření baterie a datových přenosů. Tyto změny samozřejmě přinesly spoustu výhod pro uživatele, ale také spoustu ztížily implementaci notifikací a procesů na pozadí pro vývojáře. Proto pro verzi 23+ vznikl nový nástroj JobScheduler, který zprostředkovává spouštění procesů na pozadí (mimo běh aplikace). Tím ale vznikl problém, že pro starší verze systému Android je nutné využívat starý způsob spouštění background aktivit pomocí dvojice AlarmManageru a BroadcastReceiveru a pro nové je lepší využívat rozšířené funkcionality JobScheduleru. Naštěstí tomu tvůrci androidu předcházeli a nově nabízí WorkManager utilitu, která tento proces zjednodušuje [5]. Je dostupná od API verze 14 a nabízí abstrakci nad zmíněnými nástroji. Podle verze zařízení, na kterém aplikace běží, pak vybere optimální implementaci, kterou použije k vlastnímu zprostředkování procesů na pozadí.

3.5.3 Procesy na popředí

V případě ovládání a přehledu v notifikační liště je však vhodnější využít foreground procesy, které se od background procesů liší v několika bodech. Zaprvé nejsou omezeny v přístupu ke zdrojům (jako je síť či poloha) a zadruhé je menší pravděpodobnost, že bude tento proces zabit z důvodu nedostatku zdrojů. Obě tyto vlastnosti jsou však vykoupeny nutností oznámit uživateli, že tento proces běží pomocí notifikace. To ale v případě ovládání vůbec nevadí, naopak. Tato nutná notifikace bude využita jako skupinová složka pro zbytek zařízení, která pod ní budou zobrazena v rozbalovacím menu, aby nezabírala celý notifikační prostor. Tento proces na popředí bude spuštěn pokaždé když uživatel zavře aplikaci, a opět ukončen po jejím otevření. Musí samozřejmě existovat možnost tuto funkcionalitu vypnout, protože periodické zjišťování polohy má velký dopad na baterii zařízení.

Kapitola 4

Návrh

4.1 Funkcionalita

V předchozích kapitolách je rozepsán efekt, jakého by měla aplikace dosáhnout, technologie, které při tom použije a jaké mají omezení, vlastnosti platformy, pro kterou bude aplikace vyvíjena, ale ještě zde není uvedeno, co přesně bude a co by mohla aplikace umožňovat.

4.1.1 Základní

Celá tato myšlenka je o ovládání IoT zařízení, tudíž musí aplikace umožňovat přesně toto. K testování bude využita chytrá žárovka od firmy Yeelight. Aplikace s ní bude umožňovat následující interakci:

- přidat ji do seznamu svých zařízení
- vypínat a zapínat ji ze seznamu zařízení, detailu žárovky i notifikační lišty systému Android
- měnit chromatičnost a intenzitu světla z detailu žárovky

Další klíčovou vlastností aplikace bylo zjednodušit interakci s více zařízeními. Aplikace tedy bude umět seřadit seznam zařízení podle jejich relevance v závislosti na vzdálenosti od uživatele.

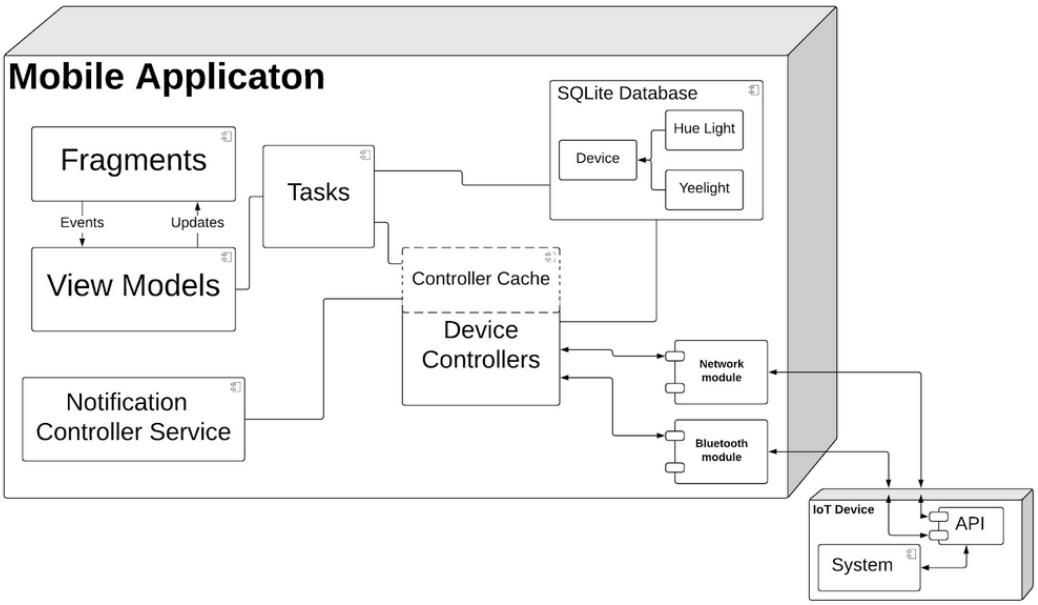
4.1.2 "Nice to have"

Na veškerou užitečnou funkcionalitu není čas, ale stejně je třeba zmínit ji a efekt, který by měla na celkovou použitelnost. V ideálním případě by tedy aplikace uměla:

- ovládat více druhů a značek IoT zařízení
- vytvářet skupiny chytrých zařízení se stejnou funkcionalitou a nabízet hromadné ovládání či hromadný výčet informací
- nastavovat časovač zapnutí/vypnutí

- vystavovat ovládání skrze notifikační lištu jen pokud je uživatel dostatečně blízko k nějakému zařízení, které je možno ovládat

4.2 Architektura



Obrázek 4.1: Architektura aplikace

Architektura aplikace (obr. 4.1) sestává z následujících modulů

- **Fragments** - Android komponenty, které se starají o obsluhu UI. Každé obrazovce náleží jeden Fragment.
- **View Models** - ke každému fragmentu náleží jeden ViewModel, který se stará o obstarávání dat, jejich transformaci a následné vystavení jako observable objekt. ViewModel neví, jaký Fragment k němu má přístup, pouze vystavuje data a přijímá eventy z UI.
- **Tasks** - plní jeden úkol s využitím jiných komponent modelu - abstrakce nad modelem ve formě jednotlivých úloh.
- **SQLite Database** - persistenční vrstva aplikace
- **Device Controllers** - ovladače pro různá zařízení a cache, která se stará o to, aby pro každé zařízení existovala jen jedna instance.
- **Notification Controller Service** - komponenta, která se stará o správu notifikací

4.2.1 MVVM architektura

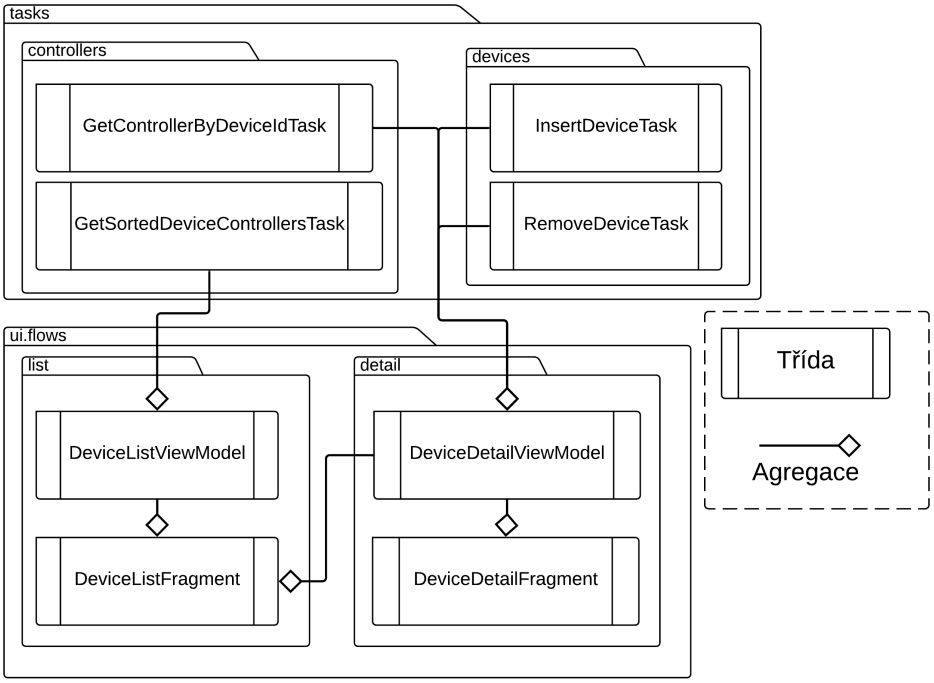
S příchodem programu Android Jetpack a konkrétně ViewModel a LiveData komponent se MVVM (Model-View-ViewModel) design pattern stal preferovaným a v oficiální dokumentaci předepisovaným řešením struktury kódu Android aplikace.

Kód je strukturován do tří vrstev. První vrstvou je Model, jehož práce je obstarávat data využívaná dalšími vrstvami. Tudíž do něj spadá persistence dat a networking. Jelikož ale do těchto dvou věcí patří spousta komponent, jsou jednotlivé úlohy abstrahovány do tvz. Tasků. Každý Task má pak na starosti jednu konkrétní věc, i když při jejím vykonávání využívá i několik komponent z persistence a networkingu (obr. 4.2).

Druhou je View, které v tomto případě zastupuje Fragment. View se stará o reprezentaci dat, která přijdou z ViewModelu, a jejich předání uživateli. Zpátky pak posílá události vyvolané uživatelskou interakcí.

Poslední vrstvou je ViewModel, a jak už název napovídá, jedná se o jakéhosi prostředníka mezi View a Modelem. Jeho práce je reagovat na události z View a předávat pak zpátky transformovaná či agregovaná data z modelu.

Vztah Fragmentu, ViewModelu a jednotlivých Tasků je vidět na obrázku 3, kde jediná závislost DeviceDetailFragmentu je na DeviceDetailViewModelu, který má ale závislost hned na tři Tasky.



Obrázek 4.2: Class diagram MVVM architektury

4.2.2 Data model

Data aplikace
Jelikož není aplikace propojená s žádnými externími službami, nemusí si pamatovat žádné uživatelské údaje či autentikační tokeny. Jediná data potřebná k běhu aplikace jsou nastavení jí samotné. V tomto případě stačí nastavení pro zapnutí procesů na popředí zmiňovaných v kapitole 3.5.3

Zařízení
Jediná data, která je třeba v nejjednodušší formě aplikace ukládat, jsou informace o připojených zařízeních, a to hlavně IP adresa a případně i metadata jako jméno zařízení či datum posledního připojení. Většina dat se však bude stahovat přímo z IoT zařízení, aby byla omezena duplicita dat. Mohlo by pak docházet k zobrazení nekorektních dat.

4.2.3 Rozhraní IoT zařízení

Na obrázku 4.1 lze kromě architektury aplikace vidět i komponent IoT zařízení. Komunikace mezi mobilní aplikací a chytrým zařízením probíhá po internetové síti a nebo pomocí technologie Bluetooth. Oba tyto přístupy však řeší každý prodejce podle svého a vystavuje jiné funkcionality v různých jazycích (JSON, XML), pod jinými jmény či dokonce nabízí věci navíc. Nebo dokonce i více způsobů ovládání. Vývojář se pak musí přizpůsobit a pro každé zařízení implementovat zvláštní ovládací prvek. Například Yeelight API [31] umožňuje načasovat vypnutí žárovky a Hue API [20] toto neumožňuje.

4.3 Networking

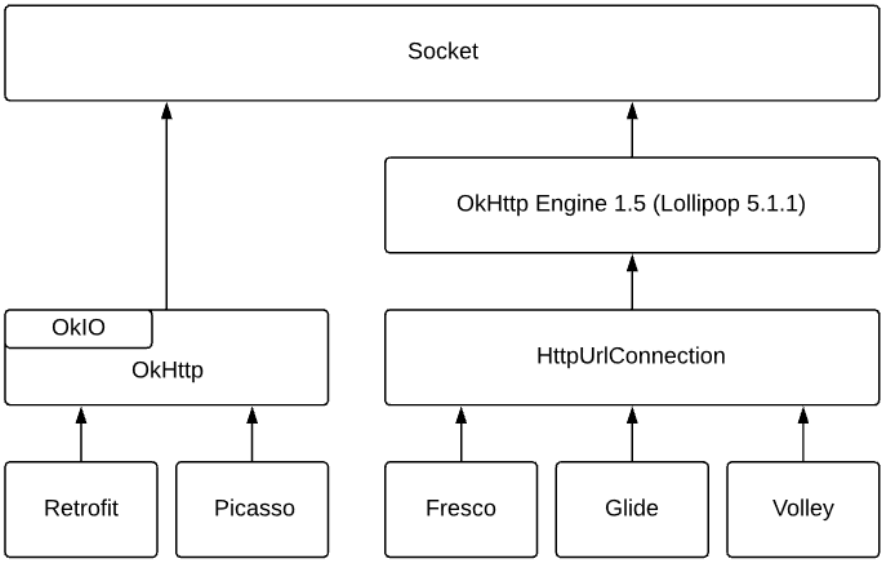
Networking je jedna z nejpoužívanějších vlastností chytrých telefonů, a tak přirozeně existuje mnoho komunitních řešení pro jeho realizaci. Dalším důvodem k jejich existenci by mohlo být neoptimální nativní řešení (HttpUrlConnection), které Android standardní knihovna nabízí. Jedná se o relativně low-level rozhraní, jež se od první verze Androidu z roku 2008 prakticky nezměnilo.

K nejpoužívanějším a nejznámějším komunitním HTTP klientům patří OkHttp a jeho nadstavba Retrofit od společnosti Square [10]. Adopce OkHttp byla tak velká, že byla její větev přidána do Androidu (API verze 21+) jako implementace výše zmíněného HttpUrlConnection a nahradila tak implementaci od Apache Harmony (která byla později vyřazena úplně) [27]. Druhý nejpoužívanější klient je knihovna Volley od Google, která mimo jiné slibuje lepší provázanost s UI či prioritizaci requestů. Byla vyvíjena přímo pro Android, a jelikož je od Google, tak je zmiňovaná i v oficiální Android developer dokumentaci [9].

4.3.1 OkHttp

Dnes skoro polovina (48%) z top pětiset aplikací na Google Play používá Retrofit, který je nadstavba nad OkHttp, a dalších 20% používá přímo OkHttp, což z ní dělá nejpoužívanější síťovací knihovnu pro Android. [10]

Mezi důležité vlastnosti OkHttp patří podle dokumentace [24]:



Obrázek 4.3: Znázornění závislostí jednotlivých knihoven

- Podpora HTTP/2, jež při volání stejného hosta více requesty umožňuje sdílení jednoho socketu.
- Pokud není HTTP/2 k dispozici, snaha o sdružování připojení pro zmenšení odezvy.
- Zmenšení souborů GZIP kompresí.
- Omezení opakovaného volání ukládáním odpovědí do mezipaměti.
- Automatické obnovení při běžných síťových problémech.

Jedním z dalších důvodů rychlé adopce OkHttp je fakt, že tato knihovna má minimum externích závislostí.
Pro file management využívá knihovnu Okio, která je ale vyvíjena stejnou společností (Square), takže se teoreticky nejedná o externí závislost. Veškerá síťovací funckionalita je pak závislá na Java Socketu, takže OkHttp má prakticky nulové závislosti na jiných knihovnách či projektech, které by její vývoj či stabilitu.

4.3.2 Volley

V roce 2013 Google, v reakci na rychlý vývoj Androidu a rapidní adopci OkHttp, oznámil na Google I/O síťovací knihovnu Volley. V dnešní době ji využívá zhruba třetina vývojářů [10], což není jen výsledkem větší viditelností díky tomu, že je Volley vnímáno jako oficiální řešení. Volley oproti OkHttp nabízí i pár benefitů [9] jako

- Automatické plánování volání vytvořením fronty.
- Možnost prioritizace jednotlivých requestů.

- Lepší debugování následkem integrace s vývojovým prostředím Android Studio.

Důležité je také zmínit, že díky tomu, že se implementace HttpUrlConnection spoléhá na OkHttp engine, je Volley svým způsobem na OkHttp závislá (Viditelné na obrázku 4.3) i když je nutno podotknout, že tato závislost není nijak omezující, protože tato implementace HttpUrlConnection je větev OkHttp verze 1.5 spravovaná Android týmem a nikoliv přímá závislost [27].

4.3.3 Ktor

Jedním z hlavních cílů JetBrains při tvorbě Kotlinu je vytvořit univerzální nástroj, který může kdokoli použít na cokoliv, a proto začali už v raných verzích pracovat Ktoru, backend frameworku, který by ke Kotlinu přilákal obrovské množství vývojářů, kteří už léta vyvíjí pro JVM platformu. Inspirovali se a později nahradili existující projekty Wasabi a Kara, které se dnes už nevyvíjí.

Verze 1.0 vyšla v listopadu 2018, ale v té době už vývojáři běžně Ktor používali v produkci. Se stabilní verzí se totiž čekalo na asynchronní knihovnu kotlinx.coroutines, která stejně jako Ktor byla do té doby sice využívaná v produkci, ale stále oficiálně nestabilní.

Jelikož chce Ktor nabídnout nástroje pro kompletní design webové aplikace, není to pouze backend framework, ale nabízí i HTTP klienta. Výhodou oproti již zmiňovaným klientům je volnost při volbě enginu, který bude klienta "pohánět". Ktor nabízí několik implementací: Apache, CIO, Jetty, OkHttp, Android (HttpUrlConnection), iOS, JS a Curl (pro Native targety). Je pak na uživateli, jakou platformu (JVM, JS, Native) a jaký engine si vybere.

4.3.4 Souhrn

Všechny tři jmenovaná řešení nabízejí určité výhody a nevýhody. Jejich vtahy lze vidět na obr. 4.3. OkHttp nabízí nejaktuálnější a robustní řešení pro klasické textové požadavky, Volley je kompletní řešení, které umožňuje i stahování obrázků a Ktor je univerzální multiplatformní řešení, které ale nenabízí tak hezké API jako třeba Retrofit (nadstavba OkHttp).

Název	**Klady**	**Zápory**	**Známka**
OkHttp	- spousta tutoriálů - vyvíjeno pro Android - přizpůsobeno pro Kotlin	- nepoužívá suspend funkce	2
Volley	- od Googlu - vyvíjeno pro Android	- obsahuje zbytečné věci (stahování obrázků) - nepoužívá suspend funkce	3
Ktor	- univerzální (volba enginu) - multiplatformní - vyvíjeno pro Kotlin	- relativně nové (možnost chyby)	1

Tabulka 4.1: Výběr networking knihovny

Stejně jako u jiných rozhodnutí se ale nakonec přikláním k univerzálnějšímu řešení a zvolím Ktor (tabulka 4.1). Nejen že jeho využití ulehčí abstrakci platformě nezávislého kódu

a možné budoucí rozšíření na jiné platformy, ale nabízí i flexibilitu co se týče výběru HTTP Client backendu a nakonec stejně bude požadavky zpracovávat OkHttp.

4.4 Persistence

U vývoje mobilních aplikací není výběr možných databázových řešení široký jako u vývoje backendu, protože zvolené řešení následně poběží na nevýkonných zařízeních s malým množstvím paměti. Je pravda, že operační paměť dnešních high-end telefonů již překračuje hranici 10GB, ale Android (s 32 bit architekturou, qHD rozlišením a podporou Google aplikací) je možné provozovat i při tak malé paměti jako je 416MB [7]. Proto není možné (a ani ve většině případů nutné) pracovat se stejnými technologiemi a množstvím dat jako v datacentrech používaných pro persistenci backend systémů.

4.4.1 Shared Preference

Nejjednodušší způsob persistence dat na platformě Android je využití tzv. Shared Preferences (dalo by se přeložit jako Sdílené Preference), pomocí kterých se dá bez příprav a s minimálními znalostmi uložit nekomplexní data. Jedná se o jednoduché souborové key-value pair úložiště, které sice není rychlé (protože se pouze jedná o ukládání a čtení dat ze souboru), ale pro hodně use-casů je dostačující.

4.4.2 SQLite

Úplně na druhé straně oproti Shared Preferences stojí SQLiteDatabase. Jak už název napovídá, jedná se o implementaci relační SQL databáze, konkrétně o implementaci SQLite, která je v Androidu nativně podporovaná. Vystavený interface je však relativně low-level a dotazy nejsou kontrolovány na správnou typovost (což lze ale napravit využitím nějaké knihovny a/nebo pluginu).

4.4.3 Dokumentová databáze

Třetí možností, která je dostupná na každé platformě, kde je umožněn přístup do filesystému, je využití dokumentové databáze. Nejedná se o nejrychlejší řešení, ale na oplátku je práce s dokumentovými databázemi většinou jednoduchá, pro všechny pochopitelná a často velmi ohebná, protože prakticky pracujeme s (většinou) JSON objektem. Dokumentové databáze by se daly označit za zlatou střední cestu.

4.4.4 Souhrn

Pro tuto práci jsem si z několika důvodů zvolil SQLight databázi s využitím knihovny (a pluginu) SQLDelight (tabulka 4.2).

1. Výsledné řešení bude rychlé, jelikož pracujeme s nativně podporovaným řešením a obecně rychlou a paměťově nenáročnou relační databází.

2. Knihovna byla vyvíjena výhradně pro Kotlin, ve kterém bude vývoj probíhat, a který je preferovaný jazyk pro vývoj na platformě Android.
3. Jedná se o multiplatformní knihovnu a jako taková usnadní možný budoucí rozvoj na jiné zařízení (jako např. od firmy Apple).

Název	**Klady**	**Zápory**	**Známka**
Shared Preferences	- přímočaré použití	- pouze key-value pair úložiště - pomalé	3
SQLight	- rychlé - strukturovaná data (relace)	- složitější nastavení	1
Dokumentová databáze	- velice ohebné	- pomalé - nutná úprava dat na jiný textový formát (JSON)	2

Tabulka 4.2: Výběr databázové knihovny

Kapitola 5

Implementace

Google v posledních letech změnil svůj neutrální postoj na doporučené architektury, postupy a 3rd party knihovny a začal být více sdílný. Velký vliv k tomuto postoji měl přechod z jazyku Java na Kotlin a pravděpodobně šance začít s čistým štítem. V závislosti na to oznámil Google roce 2018 Android Jetpack. Jedná se o sadu knihoven, které si dávají za cíl zjednodušit vývoj Android aplikací nabídnutím 1st party řešením pro běžné problémy jako např. databáze, architektura nebo práce s kamerou.

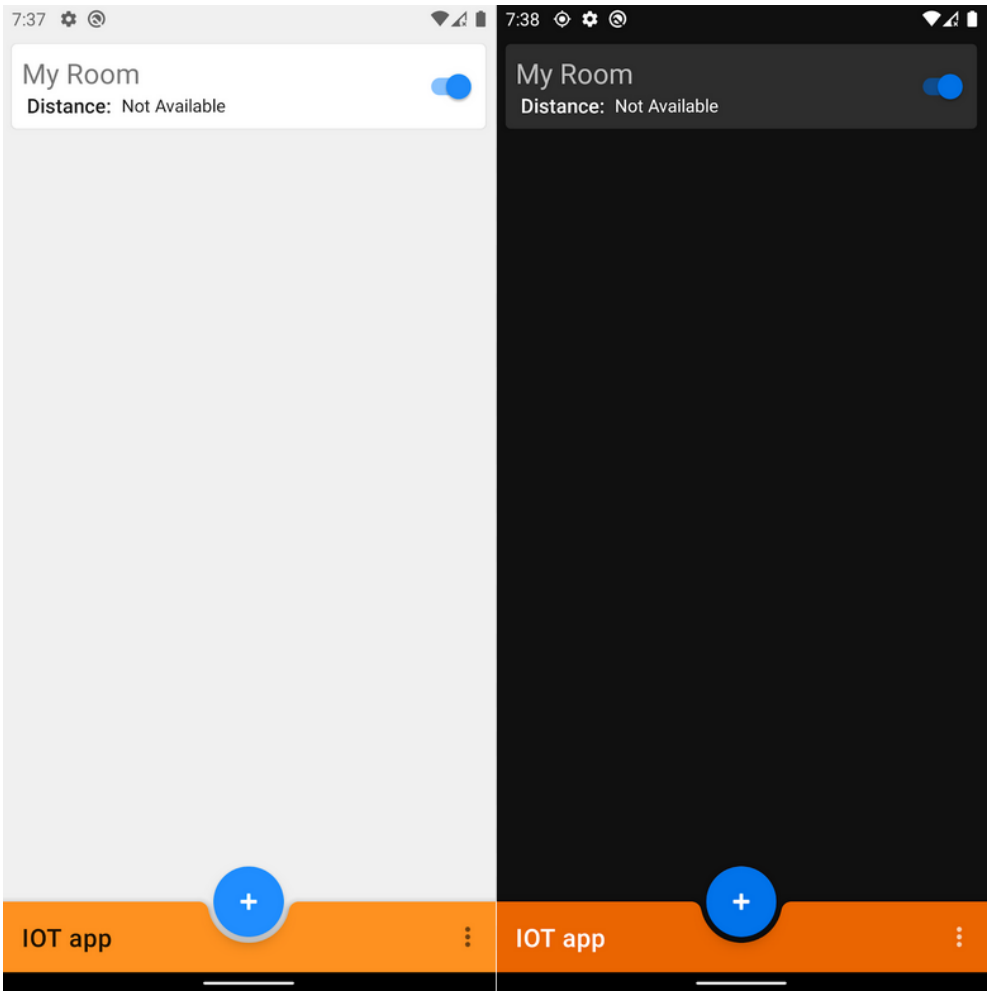
5.1 User Interface

Termín "User Interface" (občas zaměňován za UX) obnáší veškerou komunikaci s uživatelem: prezentaci dat, feedback na interakci, přechody mezi stavy či oznámení chyby. Zastává tak teoreticky roli jakéhosi posla či překladatele, který se stará mezi komunikaci s člověkem a strojem. Jako hlavní komunikátor (oproti vedlejším jako haptický feedback a audio) má tak největší podíl na schopnosti aplikace zaujmout a udržet si uživatele.

5.1.1 Material Design

Jak už bylo zmíněno v kapitole 3.4.2, Material design je aktuální Android design language a jako takový je silně podporován ze strany Googlu. Mimo základní komponenty jako např. Switch, TextView nebo ListView, nabízí Google ještě knihovnu material-components [1], která obsahuje další komponenty zmiňované v Material Design guidelines dokumentu. Tato knihovna například přidává Chips, DateTimePicker nebo CardView, které je v aplikaci použito k reprezentaci jednotlivých zařízení.

Další vlastnost aplikace, která se částečně váže na Material Design, je Dark theme. Týká se ho pouze částečně, protože to není čistě charakteristika designu, ale komponenty jej musí podporovat. Dark theme je implementován rozšířením DayNight stylu, který automaticky mění podobu podle nastavení mobilního zařízení viz obr. 5.1. Zbývá tedy už jen definovat barvy pro oba styly a o zbytek se postará DayNight styl.



Obrázek 5.1: Porovnání světlého a tmavého motivu

5.1.2 Seznam zařízení

Při prvním zapnutí aplikace je uživatel uvítán prázdným seznamem zařízení (viz obr. 5.1, ale bez položky My Room). Další (či první) zařízení lze přidat pomocí modrého tlačítka se symbolem plus neboli Floating Action Button (FAB).

Položky v seznamu jsou řazené podle aktuální vzdálenosti, která je získávána minimálně každých pět sekund. Při první implementaci byla poloha pro řazení listu získávána v jiné časy, a mohlo se tak stát, že první položka nebyla podle zobrazených údajů nejblíže.

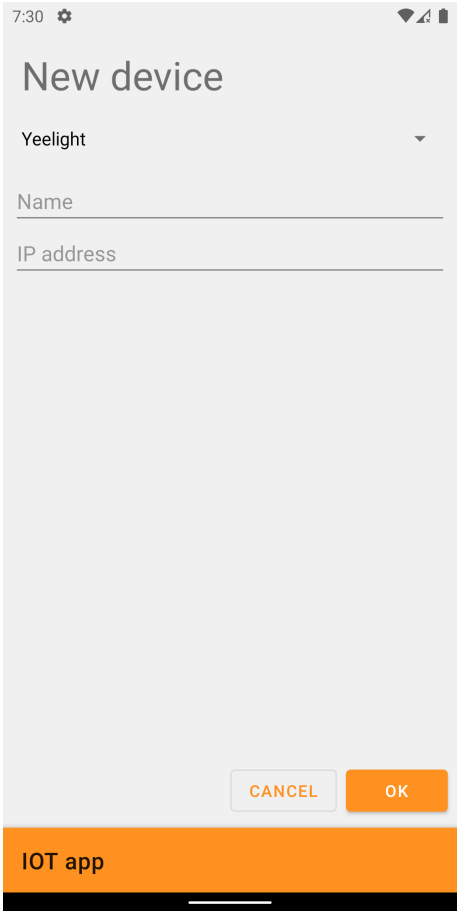
Cílem této obrazovky je nabídnout uživateli přehled o stavu jeho zařízení. Každá položka seznamu obsahuje název zařízení, jeho vzdálenost (pokud je vzdálenost nastavena) a Switch na ovládání stavu zapnuto/vypnuto. U jiného typu zařízení je možné nabídnout i jiné ovládací či informační prvky.

Dále tato obrazovka funguje jako jakýsi rozcestník do dalších částí aplikace. Kliknutí na položku nás přenese do detailu daného zařízení (kapitola 5.1.4) a kliknutí na FAB se symbolem

'+' zobrazí obrazovku, která umožní přidání zařízení (kapitola 5.1.3).

Posledním aktivním prvkem na této obrazovce je kontextní menu dostupné po kliknutí na ikonu tří teček v pravém dolním rohu. Toto menu pak nabízí možnost vypnout či zapnout ovládání skrze notifikace (kapitola 3.5.3 a 5.1.5).

5.1.3 Přidání zařízení



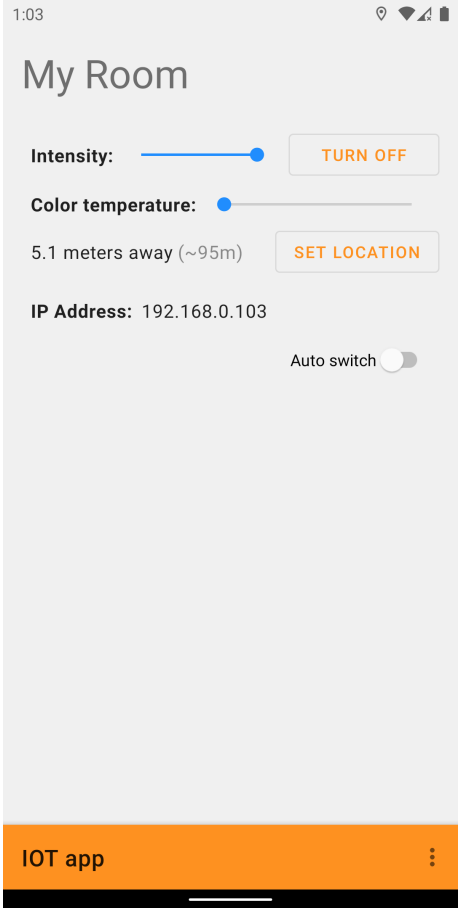
Obrázek 5.2: obrazovka přidání zařízení

Pomocí této obrazovky (obr. 5.2) bude mít uživatel možnost přidat zařízení z nabídky. První element mimo nadpis je dropdown menu, pomocí kterého je možné vybrat z dostupných zařízení. Po vybrání možnosti se změní zbytek obrazovky v závislosti na vybraném zařízení. Jak je vidět na obrázku 6, zařízení Yeelight vyžaduje jen dva parametry - jméno a IP adresu. Jiné zařízení by například mohlo vyžadovat heslo či přihlašovací jméno. V tom případě by se zde zobrazil vhodný text field.

Všechny textové pole či jiné elementy jsou dynamicky přidávány/odebírány podle vybraného zařízení. Každé zařízení musí implementovat interface DeviceCreation, jehož jediná metoda onCreate bere LinearLayout jako parametr. Do tohoto layoutu jsou pak vloženy

všechny elementy potřebné k vytvoření zařízení. Návratovou hodnotou metody je funkce, která vrací Result. Tato funkce je volána při zmáčknutí tlačítka OK a podle výsledku buď zobrazí chybovou hlášku nebo se vrátí do seznamu zařízení. Při stisknutí tlačítka Cancel není provedena žádná akce a uživatel je rovnou vrácen do seznamu zařízení.

5.1.4 Detail zařízení



Obrázek 5.3: obrazovka detailu zařízení

Detail zařízení nabízí všechny dostupné informace a ovládací prvky konkrétního zařízení. Tato obrazovka se může lišit od zařízení k zařízení, jelikož každé nabízí jiné možnosti.

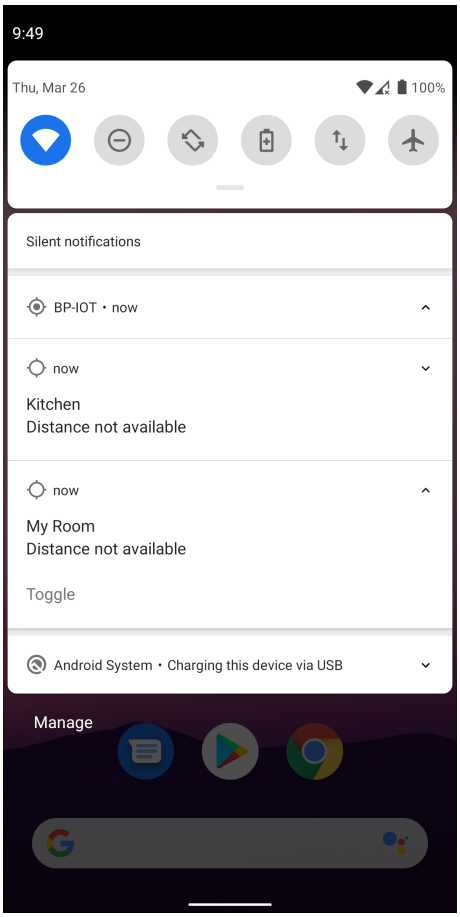
Na obrázku 5.3 je současný stav detailní obrazovky pro Yeelight žárovku. V horní části pod nadpisem je slider pro intenzitu světla, vedle kterého je tlačítko pro zapnutí a vypnutí žárovky. Pod ním je slider pro nastavení teploty barvy následovaný tlačítkem pro zaznamenání polohy.

Poloha je důležitá hodnota při určování vzdálenosti a následné pozici v seznamu zařízení. Při zmáčknutí tlačítka Set Location se k zařízení uloží aktuální GPS poloha. Jediný problém tohoto řešení je nepřesnost GPS senzoru (kapitola 3.2.1), která může být i několik metrů.

Pod IP adresou je switch pro zapnutí/vypnutí AutoSwitch funkcionality, která automaticky ovládá světlo podle vzdálenosti od zařízení.

V kontextovém menu (symbol tří teček v pravém dolním rohu) se nachází tlačítko pro odstranění aktuálně zobrazeného zařízení.

5.1.5 Notifikace



Obrázek 5.4: ovládání zařízení skrze notifikaci

Ovládání přes notifikace je klíčovou součástí aplikace. Každé zařízení je (stejně jako v seznamu zařízení) reprezentováno jedním řádkem. Ty jsou pak seřazeny podle vzdálenosti od uživatele. Kliknutím na notifikaci je pak uživatel přenesen do detailního nastavení konkrétního zařízení. Tato akce je implementována pomocí deep links a Android Jetpack Navigation komponentů.

Každá notifikace (zařízení) může mít navíc jednu nebo více akcí. V tomto případě (obr. 5.4) nabízí obě zařízení akci Toggle, která podle aktuálního stavu zařízení vypne či zapne. Implementace těchto akcí není však tak přímočará, jak by se na první pohled mohlo zdát. Pro každou akci se musí definovat třída, která dědí BroadcastReceiver. Tato třída pak přetíží

metodu onReceive, která je pak volána systémem při stisknutí akce v notifikaci. Pomocí dependency injection se pak dohledá instance controlleru, na který se pak akce deleguje.

Toto ovládání se objeví při zavření aplikace a znovu zmizí po jejím otevření.

5.2 Networking

Nejpoužívanější způsob komunikace domácích IoT zařízení je s využitím TCP (potažmo HTTP) protokolu. [15] Hlavní výhodou je jednoduchost, protože vývojáři nemusí implementovat vlastní proprietární protokol a mohou využít zažité a odzkoušené standardy přenosu a šifrování. Další výhodou je i dostupnost síťových komponent a fakt, že každý chytrý telefon je schopen se připojit k síti a následně v ní komunikovat.

5.2.1 Požadavky

Komunikace s různými zařízeními se může lišit. Například Philips Hue žárovka se ovládá pomocí klasických HTTP requestů [20] (často se tento styl označuje jako REST - Representational state transfer), ale Yeelight žárovka pomocí JSON objektů posílaných přes TCP Socket. Navíc ještě nějaká zařízení pracující na lokální síti nabízejí způsob jejich "objevení" tím, že periodicky či po spuštění vysílají discovery multicast UDP datagramy, které může mobilní zařízení odposlouchávat a zajistit tak plug&play funkcionalitu [31]. Z tohoto důvodu musí být aplikace schopna komunikovat na transportní vrstvě OSI modelu.

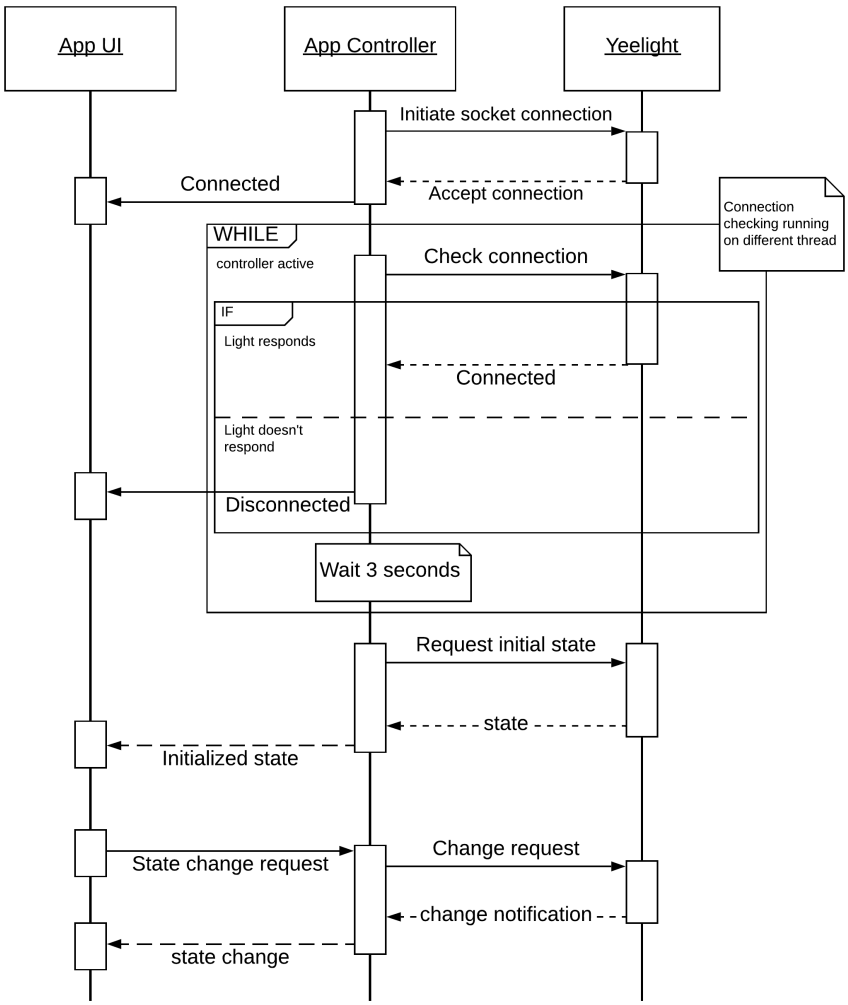
5.2.2 Implementace ovládání Yeelight

K implementaci controlleru pro Yeelight API byla pro networking použita knihovna OkHttp, Okio, jež je součástí OkHttp a k parsování JSONu byla použita Kotlin Serialization knihovna, jež je zatím v experimentální fázi. Jak vyplývá z kapitoly 4.3.4, komunikace měla být implementována pomocí knihovny Ktor, ale během násilného přerušení spojení ze strany žárovky docházelo k pádům aplikace z důvodu vyhození neodchytitelné výjimky uvnitř knihovny. Proto už během návrhu Yeelight API byl Ktor, jakožto druhý nejlepší kandidát, vyměněn za OkHttp.

Následný vývoj postupoval podle Yeelight dokumentace [31], která je veřejně k dispozici, a během implementace základních funkcionalit nedošlo k většímu problému. První zádrhel vznikl až při snaze zjistit stav připojení. U socketu totiž nelze poznat, když druhá strana násilně uzavře spojení (např. fyzické vypnutí žárovky), protože je postavena na TCP. Na vedlejším vlákně tedy musel být spuštěn loop, který se periodicky dotazuje na IP adresu žárovky a zjišťuje, jestli je aktivní (viz obr. 5.5).

Další nevýhoda socket varianty (oproti REST) je nutnost udržovat spojení a instanci Readeru a Writeru. To má za následek složité dohledávání konkrétní instance při anonymním volání, jako například volání akce z notifikace (viz 5.1.5).

Poslední problém, který vznikl, byla nemožnost identifikovat zdroj změny stavu žárovky. Při změně stavu odešle žárovka všem připojeným zařízením notifikaci (viz obr. 5.5), která ale neobsahuje žádný identifikátor. To má za následek opakování událostí. UI element změní svůj stav a aplikace odešle požadavek o změnu žárovce. Zpátky se však vrátí anonymní notifikace,



Obrázek 5.5: komunikace mobilní aplikace se světlem Yeelight

která musí změnit UI element. Touto změnou se pak akce zacyklí. Naštěstí však Android komponenty ignorují změnu stavu na identickou hodnotu.

5.3 Databáze

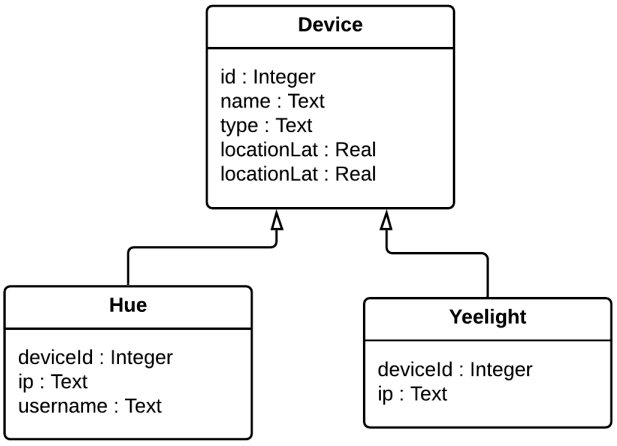
Implementace databáze byla asi nejvíce bezproblémová část při tvorbě aplikace. Knihovna SQLDelight funguje bez problému a android SQLight implementace je dostatečně rychlá, aby byla prodleva při načítání UI prakticky nepoznatelná.

5.3.1 SQLDelight

SQLDelight se od ostatních databázových přístupů (Flatfile - operace přímo s filesystémem, Object - objektové databáze, Object-Relational Mapping (ORM)) liší v tom, že nepřenáší většinu dotazovacích funkcionalit do kódu (Flatfile, Object DB) a nesnaží se mapovat jazykové konstrukty na relační tabulky (ORM), ale přistupuje k problému z opačné strany. Vývojář píše normální SQLite kód a pro pojmenované dotazy je pak vytvořena typesafe funkce, kterou je možno volat v kódu aplikace.Nejen že je pak výsledný kód rychlejší, protože většina výpočtu je provedena v optimalizovaném prostředí databáze, ale navíc je pro samotné dotazování použit jazyk, který je pro to vytvořený.

K dispozici je i plugin do Intellij Idea IDE, který přidává podporu pro .sq soubory a zajišťuje generaci Kotlin kódu při každé změně SQL kódu. Není pak třeba po každé malé změně opakovaný build projektu.

5.3.2 Implementace



Obrázek 5.6: struktura tabulek v databázi

Pro persistenci zařízení tedy stačilo vytvořit jeden soubor Devices.sq pro generická data vztahující se ke každému zařízení, a poté další soubor pro každý druh zařízení (viz obr. 5.6). Každý takový soubor obsahuje definici tabulky a dotazy pro vložení, mazání a hledání řádků. V dependency injection modulu se pak jen vytvoří databáze a z ní se pak "vytáhnou" jednotlivé interfaces pro každý soubor.

Pro usnadnění vytváření a mazání zařízení byly poté vytvořeny pomocné DAO (Data Access Object) utility, které se starají o správnou inheritanci objektů. Například každá Yeelight žárovka vlastní jeden řádek v Devices tabulce a jeden v Yeelight tabulce, takže při jejím vytváření musí být nejprve vytvořen záznam v Devices tabulce a nové id je pak využito v novém záznamu Yeelight tabulky. Opačný proces je pak třeba u mazání zařízení.

5.4 Location Tracking

Lokace je jedna z nejdůležitějších a nejvíce chráněných informací dostupných mobilní aplikaci, a jako takovou není zrovna nejlehčí ji získat.

V tomto projektu je využívána u určení relevance konkrétního zařízení. To znamená, že čím dále se od zařízení uživatel nachází, tím menší je jeho priorita při zobrazování.

5.4.1 Android location API

Prvním problémem této funkcionality byla nutnost zažádat a následně získat povolení pro získání lokace. Je tedy třeba prvně vyzvat uživatele k udělení povolení a poté očekávat jeho odpověď v přetížené metodě *Activity::onRequestPermissionsResult()*. Uživatel pak může odpovědět třemi způsoby. Buď zjišťování lokace povolí (když je aplikace aktivní), zakáže, anebo zakáže a omezí budoucí dotazování na toto povolení. To znamená, že aplikace musí být schopná fungovat i bez tohoto povolení, protože ho nemusí nikdy dostat.

```
private fun requestUpdates(callback: LocationCallback) {
    if (ActivityCompat.checkSelfPermission(
            context,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) == PackageManager.PERMISSION_GRANTED
    ) locationProvider.requestLocationUpdates(
            LocationRequest().apply { this: LocationRequest
                interval = UPDATE_INTERVAL
                priority = LocationRequest.PRIORITY_HIGH_ACCURACY
            },
            callback,
            Looper.getMainLooper()
        )
    }
}
```

Obrázek 5.7: Žádost o opakované získávání polohy

Po ujištění, že uživatel povolil přístup k poloze zařízení lze teprve přistupovat k rozhraní *FusedLocationProviderClient*. Pokud se k jistým metodám této třídy snažíme přistoupit dříve, Android Studio nám dotyčný kód červeně podtrhne a řekne, že se musíme nejdříve dotázat. Tuto kontrolu považuji za dobrý krok, ale občas se stává, že statická analýza kódu vyhodnotí, že ke kontrole nedošlo, i když k ní došlo a je třeba error ignorovat pomocí anotace.

FusedLocationProviderClient je relativně nové rozhraní k přistupování k poloze, není přímo součástí Android standardní knihovny a je třeba využít balíček *com.google.android.gms:play-services-location*. Nejdůležitější funkce, kterou toto rozhraní vystavuje, je requestLocationUpdate(), pomocí které vyžádáme periodické aktualizace pozice. Na obrázku 5.7 je vidět, jak může vypadat požadavek o získávání polohy přímo v IDE. Dle mého názoru je však jméno metody zavádějící a v rozporu se zbytkem Android metod pro nastavení listeneru. V naprosté většině případů jsou tyto metody prefixované slovem add nebo set, které jasně určují, jestli může existovat více než jeden listener. V tomto případě to není jasné a předpokládal bych, že

požádat o aktualizace můžeme kolikrát chceme. V dokumentaci je však explicitně napsáno, že zažádat lze pouze jednou a každá další žádost přepíše tu předchozí.

5.4.2 Přesnost

Při průběžném testování aplikace vyšlo hned najevo, že zjišťování GPS pozice nebude vůbec přesné. Nejen že se pozice průběžně mění o jednotky metrů, ale občas dojde ke chvilkové změně až padesáti metrů.

Na druhou stranu je vidět, že *FusedLocationProviderClient* snaží polohy nějakým způsobem interpolovat, protože až na velké výkyvy jsou změny polohy pozvolné. To ale vede ke zpoždění oznámení změny polohy, což může v různých případech zhoršit *user experience* (např. při automatickém zapnutí světel při vstoupení do místnosti).

5.4.3 Implementace

Skutečnost, že je možné s jedním location clientem sledovat polohu jen jednou, vedla k vytvoření jednoho *LocationProvider* singletonu, který vystavuje *lastLocation* pro jednorázové zjištění polohy a Flow (suspending cold stream) pro stálé sledování polohy.

Tento stream pak sleduje *NotificationControllerService* pro aktualizování polohy v notifikacích, *DeviceController* k zobrazení vzdálenosti v seznamu a detailu zařízení a *GetSortedDeviceControllersTask*, který tento Flow kombinuje s Flow device controllerů z databáze, a na základě polohy je třídí od nejbližšího po nejvzdálenější.

Kapitola 6

Testování

S ohledem na současnou pandemii COVID-19 nebude UI testováno s více testery. Toto testování je nahrazeno systémovými testy. Podle jednotlivých scénářů se opakovaně otestují klíčové vlastnosti aplikace a podle výsledků se odvodí jejich funkčnost a použitelnost.

6.1 Testovací scénáře

6.1.1 Změna stavu z detailu zařízení

Popis testu
Tento test má prověřit funkčnost a spolehlivost detailu zařízení, jenž je jediné místo, kde jsou všechny ovládací prvky konkrétního zařízení. Proto musí být tato obrazovka spolehlivá.

Náplní tohoto testu bude otevření detailu zařízení, změna jeho stavu a následná kontrola, že zařízení skutečně změnilo stav a že se stav správně zpropagoval do zbytku aplikace.

Očekávaný výsledek
Při změně stavu v detailu zařízení dojde nejen k fyzické změně zařízení, ale tato změna bude také viditelná v notifikaci (pokud je zobrazená) a v seznamu zařízení (pokud je tam změna viditelná).

Postup
Předpoklady: v aplikaci je uloženo minimálně jedno zařízení.

- Otevřít aplikaci
- Kliknout na jakoukoliv položku v seznamu
- Vypnout/Zapnout zařízení kliknutím na tlačítko Power Off / Power On
- Zkontrolovat změnu fyzického zařízení
- Vrátit se zpět do seznamu zařízení
- Zkontrolovat, jestli odpovídá stav změněného zařízení

- Zavřít aplikaci
- Zkontrolovat, jestli odpovídá stav zařízení v notifikaci

Výsledky

číslo testu	výsledek	(popis)
1	OK	
2	OK	
3	OK	
4	OK	
5	OK	

Tabulka 6.1: Výsledky prvního testu

Stoprocentní úspěšnost tohoto testu ukázala, že tato kritická část aplikace by měla ve většině případů bezchybně fungovat (viz tabulka 6.1).

6.1.2 Změna stavu nejbližšího zařízení

Popis testu
Klíčovou součástí aplikace je sledování vzdálenosti jednotlivých zařízení a následné jejich řazení, aby měl uživatel vždy po ruce jen to, co potřebuje. Cílem tohoto testu je tedy zjistit, jestli je GPS poloha dostačující a spolehlivý údaj.

Test bude probíhat tak, že se telefon umístí k blízkosti jednoho zařízení a po určité době se zkontroluje, že je první v seznamu. Poté se telefon přesune do blízkosti druhého zařízení a tak dále. Důležité je také, jestli zařízení zůstane na prvním místě v seznamu po celou dobu, a jaká je minimální vzdálenost zařízení, aby nedocházelo k tak velké chybě geolokace, že by nejbližší zařízení nebylo první.

Očekávaný výsledek
Nejbližší zařízení bude vždy první v seznamu zařízení aplikace i mezi notifikacemi ostatních zařízení. Pokud bude pozice telefonu statická, bude statické i pořadí zařízení v seznamu.

Postup
Předpoklady: v aplikaci jsou uložena dvě zařízení, která mají nastavenou polohu.

- Otevřít aplikaci a nechat 10 sekund kalibrovat
- Umístit telefon do blízkosti prvního zařízení
- Počkat 20 sekund
- Zkontrolovat první zobrazené zařízení v seznamu
- Umístit telefon do blízkosti druhého zařízení
- Počkat 20 sekund

- Zkontrolovat první zobrazené zařízení v seznamu

Tento test se bude opakovat pro zařízení, která jsou od sebe 2, 5 a 10 metrů.

Výsledky

10 metrů		
číslo testu	**výsledek**	**(popis)**
1	OK	
2	FAIL	První zařízení správně asi po 25 sekundách, druhé OK.
3	OK	
4	OK	
5	OK	

Tabulka 6.2: Výsledky druhého testu - **Vzdálenost 10 metrů**

5 metrů		
číslo testu	**výsledek**	**(popis)**
1	FAIL	GPS pozice se prakticky vůbec nezměnila.
2	FAIL	
3	FAIL	U prvního zařízení OK asi až po 25 sekundách. U druhého asi po 20.
4	FAIL	GPS pozice se prakticky vůbec nezměnila.
5	OK	Přesně ve dvacátou sekundu se ukázalo správně první zařízení na prvním místě, druhé dříve.

Tabulka 6.3: Výsledky druhého testu - **Vzdálenost 5 metrů**

2 metry		
číslo testu	**výsledek**	**(popis)**
1	FAIL	Na tuto vzdálenost není pozorovatelný žádný vliv fyzické pozice na hodnotu GPS pozice. GPS souřadnice skáčou náhodně.
2	FAIL	
3	FAIL	
4	FAIL	
5	FAIL	

Tabulka 6.4: Výsledky druhého testu - **Vzdálenost 2 metry**

Téměř stoprocentní selhání testů při vzdálenosti pět a méně metrů (viz tabulku 6.3 a 6.4) potvrdila velkou nepřesnost GPS. Relativně spolehlivě a v relativně přijatelném čase dokáže GPS systém nabídnout dostatečnou změnu až při deseti a více metrech. Úsek pěti až deseti metrů je proměnlivě spolehlivý úsek.

Poznámka: tento a další testy závislé na GPS mohou být ovlivněny proměnlivostí aktuálního stavu GPS signálu.

6.1.3 Funkčnost notifikací

Popis testu
Musí být možnost zařízení ovládat i skrze notifikační lištu telefonu, protože ne vždy chce uživatel otevírat aplikaci a stačí mu jednoduché ovládání.

Je tedy třeba otestovat, že se notifikace korektně zobrazují po zavření aplikace a následné mizí po jejím spuštění, že ovládací prvky fungují, reakční doba není příliš vysoká, a hlavně jsou zobrazeny správně. Například když je světlo vypnuté, tak tlačítko musí být označeno jako "Power On" a když zapnuté, tak "Power Off". Notifikace také musí být správně seřazeny podle vzdálenosti jednotlivých zařízení.

Očekávaný výsledek
Pokud je notifikační ovladač v aplikaci povolený, musí se po zavření v notifikační liště objevit minimálně notifikace pro foreground aktivitu označená "Device Controller". Dále se zobrazí notifikace pro každé zařízení, které je připojeno a dostupné. Tyto notifikace musí po otevření aplikace opět zmizet. Kliknutí na notifikaci konkrétního zařízení otevře obrazovku detail v aplikaci. Kliknutí na tlačítko provede dotyčnou akci se zpožděním maximálně dvě sekundy.

Postup
Předpoklady: v aplikaci je uloženo minimálně jedno zařízení.

- Otevřít aplikaci
- Ujistit se, že ovládání skrze notifikace je zapnuté
- Zavřít aplikaci
- Rozbalit notifikační lištu
- Zkontrolovat pořadí, stav a počet notifikací zařízení
- Změnit stav jednoho zařízení a zkontrolovat výsledek
- Kliknout na notifikaci změněného zařízení
- Zkontrolovat, jestli je změna viditelná i v detailu zařízení

Výsledky

číslo testu	výsledek	(popis)
1	OK	
2	OK	
3	OK	
4	OK	
5	OK	

Tabulka 6.5: Výsledky třetího testu

Po 6.1 druhý test zaměřený čistě na ovládání světel se stoprocentní úspěšností ukazuje stabilitu jádra aplikace a propojení se světly Yeelight (tabulka 6.5).

6.1.4 Obnovení spojení po odpojení zařízení

Popis testu
Občas se připojené zařízení náhle odpojí z důvodu nekvalitního internetového připojení nebo v případě lokálně ovládaných zařízení z důvodu odpojení od místní sítě. Tento test má ověřit schopnost aplikace reprezentovat tento stav a schopnost se z tohoto "šoku" vzpamatovat a znovu se připojit.

Očekávaný výsledek
Po odpojení zařízení se v seznamu i v detailu zařízení zobrazí červená ikona signalizující ztrátu spojení. Pokud není aplikace otevřená a je zapnuto ovládání skrze notifikace, tak notifikace odpojeného zařízení zmizí. Pokud je zařízení znovu aktivní, aplikace se do deseti sekund opět sama připojí.

Postup
Předpoklad: v aplikaci je uloženo minimálně jedno zařízení, jež je možné při testování fyzicky vypnout.

- Otevřít aplikaci
- Zkontrolovat, že je zařízení připojené
- Fyzicky vypnout zařízení
- Zkontrolovat, že je zařízení v aplikaci označené jako odpojené
- Opět zařízení fyzicky zapnout
- Počkat 10 sekund
- Zkontrolovat, že se zařízení znovu připojilo a funguje
- Opakovat s ovládáním skrze notifikaci

Výsledky

číslo testu	výsledek	(popis)
1	FAIL	Zařízení se připojilo asi až po 15 sekundách
2	OK	
3	OK	
4	OK	
5	OK	

Tabulka 6.6: Výsledky čtvrtého testu

Vzhledem k tomu, že selhal jen jediný test, a to jen z časových důvodů (tabulka 6.6), je možno tento test označit jako úspěšný. Čas nehraje v tomto případě velkou roli. Důležité je hlavně opětovné připojení.

6.1.5 Automatické ovládání světel

Popis testu
Ne vždy uživatel chce nebo může ovládat světla manuálně. Proto existuje funkcionalita AutoSwitch, která se stará o automatické vypínání a zapínání světel. Tento test ukáže, jestli je tato funkcionalita použitelná v praxi.

Očekávaný výsledek
Světla by se měla rozsvítit, když se uživatel dostane do jejich blízkosti a opět zhasnout, pokud se uživatel vzdálí. Je očekávaná několika sekundová prodleva reakce z důvodu nepřesnosti GPS.

Postup
Předpoklady: v aplikaci je uloženo minimálně jedno zařízení.

- Otevřít aplikaci
- Zapnout na jakémkoliv zařízení funkcionalitu AutoSwitch v jeho blízkosti a zapnout zařízení
- Vzdálit se 10 metrů od zařízení
- Počkat 10 sekund
- Zkontrolovat, že se zařízení vypnulo
- Zkontrolovat, že se zařízení v následujících deseti sekundách opět nezapne

Výsledky

číslo testu	výsledek	(popis)
1	FAIL	Zařízení se vypnulo pozdě
2	FAIL	
3	FAIL	
4	OK	
5	FAIL	Zařízení se jednou vypnulo před vypršením 10 sekund a poté opět zapnulo na dalších 10s.

Tabulka 6.7: Výsledky pátého testu

Třetí test závislý na GPS, který ukazuje nedostatky v přesnosti této technologie (viz tabulku 6.7). Výsledky tohoto testu by však mohly být vylepšeny "chytřejším" algoritmem, který by pozoroval změny polohy v čase a porovnával vzdálenosti více zařízení. V tomto stavu je však AutoSwitch funkcionalita prakticky nepoužitelná.

6.2 Shrnutí testování

Testování jen potvrdilo předem očekávané problémy s nepřesností GPS. Mimo funkcionalitu závislou na GPS je aplikace stabilní a chová se podle očekávání.

Kapitola 7

Závěr

V úvodu této práce byly zmíněny dva hlavní problémy s chytrými zařízeními v domácnosti. Tím první byl roztříštěnost ovládacích prvků, kdy k ovládání dvou zařízení od dvou rozdílných firem je potřeba nainstalovat dvě mobilní aplikace. Tento problém byl částečně, ale ne úplně vyřešen. Aplikace umí bezproblémově komunikovat s více druhy žárovek od firmy Yeelight, ale to je jediná značka, se kterou lze komunikaci navázat. Byly pokusy o implementování komunikace s žárovkami Philips Hue [20], ale kvůli karanténě nebylo možné je testovat a implementovat protokol bez reálného zařízení není optimálně proveditelné. Architektura aplikace by však umožňovala repertoár zařízení rozšířit, takže první problém roztříštěnosti ovládacích prvků je vyřešen pouze částečně.

Druhým problémem, jež se tento projekt snažil vyřešit, bylo možné přehlcení prostoru, a následně UI aplikace, velkým množstvím zařízení. Reakcí na toto byla implementace chytrého řazení zařízení. U každého zařízení je možné uložit GPS pozici, která je použita k vypočítání vzdálenosti od uživatele a následném seřazením zařízení v seznamu. To znamená, že zařízení, které chce uživatel používat, jsou vždycky po ruce.

7.1 Aplikace

Jak bylo nastíněno v analýze, je vhodné u mobilní aplikace sledovat design operačního systému, jelikož je pak UI pro uživatele přehlednější. UI této aplikace splňuje Material design guidelines a je tedy v souladu s tímto postupem. Jako bonus navíc respektuje Light a Dark theme nastavení Androidu.

Z pohledu architektury je aplikace také v souladu s doporučeními Android týmu. Řídí se klasickým MVVM modelem a v celé aplikaci je jen jedna *Activity*, která funguje jako host pro *Fragmenty*. Jediným problémem je nemožnost dynamicky načíst jiné zařízení, protože části aplikace spoléhají na předem známé hodnoty, i přestože to nebylo na začátku plánované.

7.2 Geolokace a GPS

Při testování se ihned potvrdily obavy ohledně nepřesnosti GPS projevené v analýze. Na vzdálenost menší než deset metrů je nepřesnost a reakční doba tak velká, že ovládat podle

této informace zařízení je prakticky nemožné, a tato funkcionalita by zcela jistě způsobila změnu stavu zařízení, jež nemělo být změněno a naopak.

7.3 Využití

Vzhledem k tomu, že je aplikace v aktuálním stavu využitelná pouze pro zařízení Yeelight a to ne v plném rozsahu (nepodporuje funkci *flow*), tak je ve většině případech výhodnější využívat oficiální Yeelight aplikaci. Jediný use-case, který se přiklání k využití této aplikace je, kdyby byla pro uživatele klíčová funkce ovládání žárovek skrze notifikaci. I kdyby však byla aplikace v kompletnějším stavu, nebylo by její doporučení tak přímočaré. Posuzování vzdálenosti podle GPS se ukázalo pro určité případy jako nedostatečně přesné a pro vylepšení přesnosti by bylo třeba využít jinou geolokační technologii nebo GPS sledování polohy optimalizovat jinou cestou (např. využít akcelerometr, bluetooth majáky či *machine learning*).

7.4 Budoucnost projektu

Na přesné určování polohy se GPS technologie ukázala jako nedostatečná. Na co by ale mohla dostačovat, je reakce na časově nenáchylné aktivity. Pokud by se například uživatel dlouhou dobu nepohnul a z předchozí činnosti by vycházelo, že v tuto hodinu bývají světla zhasnutá, mohla by aplikace navrhnout zhasnutí všech světel. Pokud by pak uživatel na tuto výzvu do určitého intervalu nezareagoval, byla by světla zhasnuta. Využití tohoto projektu však nebylo mířeno pouze na světla. Aplikace může být integrována s více druhy zařízení. Poté by na základě statistik a machine learning mohla rozpoznávat i jiné vzorce chování než jenom pohyb. Například pokud by uživatel pral své prádlo pouze o víkendu, nabízela by v příslušné dny a hodiny pračku na dominantní pozici v seznamu zařízení.

7.5 Shrnutí

V současném stavu nenabízí aplikace mnoho praktických využití, ale směrem, kterým se práce ubírá, má cenu pokračovat. Počet chytrých zařízení v domácnostech bude růst a s nimi bude růst i potřeba je chytře a jednoduše ovládat. Poloha uživatele je informace, se kterou dnes žádná jiná aplikace nepracuje do té míry, že by podle ní automaticky měnila stav zařízení či optimalizovala výčet možností.

Literatura

[1] Material Components GitHub page. <https://github.com/material-components/material-components-android>. cit. 2020-05-11.

[2] AL-FAGIH, A. E. et al. A priced public sensing framework for heterogeneous IoT architectures. *IEEE Transactions on Emerging Topics in Computing*. 2013, 1, 1, s. 133–147.

[3] ANDROID. Understand the Activity Lifecycle: Activity-lifecycle concepts. <https://developer.android.com/guide/components/activities/activity-lifecycle>, . cit. 2020-03-20.

[4] ANDROID. Android is for everyone: Android's open platform helps people around the globe enjoy greater access to more information and opportunity than ever before. <https://www.android.com/everyone/>, . cit. 2020-01-09.

[5] ANDROID. Guide to background processing. <https://developer.android.com/guide/background/>, . cit. 2020-02-19.

[6] ANDROID. Understanding Android #18: From facts to terminology, learn how Android helps support the mobile market. <https://www.android.com/everyone/facts/>, . cit. 2020-01-09.

[7] ANDROID. Android 10 Compatibility Definition: Handheld Requirements. <https://source.android.com/compatibility/android-cdd>, 2020. cit. 2020-03-24.

[8] ANDROID. Declarative UI Patterns: Google I/O'19. <https://youtu.be/VsStyq4Lzxo?t=252>, 2019. cit. 2020-03-20.

[9] ANDROID. Volley overview. <https://developer.android.com/training/volley>, . cit. 2020-02-24.

[10] BHADE, J. – YADAV, H. Evaluation and Identification of Android Networking Libraries. 2019.

[11] CHOPRA, R. Editorial. *Linux For You Magazine*. 2008, 69, s. 6.

[12] CLIFTON, I. G. *Android User Interface design: implementing material design for developers*. Addison-Wesley Professional, 2015.

[13] HAASE, C. Google I/O 2019: Empowering developers to build the best experiences on Android + Play. <https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html>. cit. 2020-01-09.

[14] KACZMAREK, M. – RUMINSKI, J. – BUJNOWSKI, A. Accuracy analysis of the RSSI BLE SensorTag signal for indoor localization purposes. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, s. 1413–1416. IEEE, 2016.

[15] KHAKIMOV, A. et al. Investigation of methods for remote control IoT-devices based on cloud platforms and different interaction protocols. In *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, s. 160–163. IEEE, 2017.

[16] KIM, J. The effect of design characteristics of mobile applications on user retention: an environmental psychology perspective. 2012.

[17] LEE, J. – BAGHERI, B. – KAO, H.-A. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing letters*. 2015, 3, s. 18–23.

[18] MOAZZAMI, M.-M. et al. SPOT: A smartphone-based platform to tackle heterogeneity in smart-home IoT systems. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, s. 514–519. IEEE, 2016.

[19] NPM. Material-UI: NPM package registry. <https://www.npmjs.com/package/@material-ui/core>. cit. 2020-05-11.

[20] PHILIPS. Philips Hue lights 'getting Started' guide. <https://developers.meethue.com/develop/get-started-2/>. cit. 2020-04-04.

[21] RASCH, K. et al. Context-driven personalized service discovery in pervasive environments. *World Wide Web*. 2011, 14, 4, s. 295–319.

[22] ROBLEK, V. – MEŠKO, M. – KRAPEŽ, A. A complex view of industry 4.0. *Sage Open*. 2016, 6, 2, s. 2158244016653987.

[23] RÜSSMANN, M. et al. Industry 4.0: The future of productivity and growth in manufacturing industries. *Boston Consulting Group*. 2015, 9, 1, s. 54–89.

[24] SQUARE. OkHttp documentation. <https://square.github.io/okhttp/>. cit. 2020-02-24.

[25] THEODORIDIS, E. – MYLONAS, G. – CHATZIGIANNAKIS, I. Developing an iot smart city framework. In *IISA 2013*, s. 1–6. IEEE, 2013.

[26] TREURNIET, J. J. et al. Energy consumption and latency in BLE devices under mutual interference: An experimental study. In *2015 3rd International Conference on Future Internet of Things and Cloud*, s. 333–340. IEEE, 2015.

[27] VOVK, L. How to choose an Android HTTP Library. <https://appdevelopermagazine.com/how-to-choose-an-android-http-library/>, 2017. cit. 2020-02-19.

[28] WANG, C. et al. Multi-sensor fusion method using kalman filter to improve localization accuracy based on android smart phone. In *2014 IEEE International Conference on Vehicular Electronics and Safety*, s. 180–184. IEEE, 2014.

[29] WILSON, J. *Creating Dynamic UIs with Android Fragments*. Packt Publishing Ltd, 2016.

[30] WOLLSCHLAEGER, M. – SAUTER, T. – JASPERNEITE, J. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE industrial electronics magazine*. 2017, 11, 1, s. 17–27.

[31] YEELIGHT. Yeelight WiFi Light Inter-Operation Specification. <https://www.yeelight.com/download/Yeelight_Inter-Operation_Spec.pdf>, 2015. cit. 2020-04-04.

[32] ZACHARIAH, T. et al. The internet of things has a gateway problem. In *Proceedings of the 16th international workshop on mobile computing systems and applications*, s. 27–32, 2015.

Příloha A

Seznam použitých zkratek

API	Aplication Programming Interface
BLE	Bluetooth Low Energy
DAO	Data Access Object
FAB	Floating Action Button
FOSS	Free and Open Source Software
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
JS	JavaScrip
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVVM	Model View ViewModel
ORM	Object-Relational Mapping
qHD	quarter of a Full HD (960x540)
REST	Representational State Transfer
SAM	Single Abstract Method
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface

UX User Experience

XML Extensible Markup Language