VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HARDWAROVÝ INDIKÁTOR OZNÁMENÍ PRO TELEFONY SE SYSTÉMEM ANDROID

HARDWARE INDICATOR OF NOTIFICATIONS FOR SMARTPHONES WITH ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ROMAN BÁRTL**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Bártl Roman**
Program: Informační technologie
Název: **Hardwarový indikátor oznámení pro telefony se systémem Android**
Hardware Indicator of Notifications for Smartphones with Android
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte možnosti komunikace mobilu a malého HW zařízení přes Bluetooth.
2. Prostudujte možnosti vytváření a programování malého HW zařízení s komunikací přes Bluetooth a s možností indikace pomocí LED a/nebo malých displejů.
3. Prostudujte možnosti zachytávání událostí a informací o stavu telefonu a jejich přeposlání pomocí Bluetooth do malého zařízení.
4. Navrhněte a implementujte prototyp hardwaru pro zobrazování světelných notifikací.
5. Navrhněte a implementujte prototyp softwaru (Android) komunikujícího se zařízením.
6. Vyhodnoťte vlastnosti vytvořeného řešení a jeho možnou praktickou využitelnost.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Android Developers: https://developer.android.com/index.html
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- VODA, Zbyšek. Tým HW Kitchen. Průvodce světem Arduina, 2014, online

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 7. února 2020

Abstrakt

Tato práce se zabývá návrhem a realizací notifikačního zařízení zobrazujícího nejrůznější upozornění z chytrých telefonů, jehož jádrem je platforma Arduino. Součástí práce je i implementace aplikace pro systém Android, která bude tato upozornění zachytávat a zasílat na zařízení prostřednictvím technologie Bluetooth Low Energy. Důraz je kladen na jednoduchost a srozumitelnost ovládání celého tohoto vestavěného systému.

Abstract

This thesis deals with design and development of a notification device depicting all kinds of notifications from smartphones. The core of this device is an Arduino board. This thesis also includes an implementation of Android application which handles these notifications and sends them to the device via Bluetooth Low Energy. There is an emphasis on simplicity and intelligibility of the whole embedded system.

Klíčová slova

Android, Arduino, Bluetooth Low Energy, Vestavěný systém, Notifikace, Notifikační zařízení, NeoPixel modul, RGB LED WS2812B, Bluetooth Low Energy modul HM-10, Služba NotificationLitenerService, Zachytávání událostí

Keywords

Android, Arduino, Bluetooth Low Energy, Embeded system, Notifications, Notification device, NeoPixel modul, RGB LED WS2812B, Bluetooth Low Energy modul HM-10, NotificationLitenerService, Handling events

Citace

BÁRTL, Roman. *Hardwarový indikátor oznámení pro telefony se systémem Android*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Hardwarový indikátor oznámení pro telefony se systémem Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Adama Herouta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Roman Bártl
20. května 2020

Poděkování

Rád bych poděkoval mému vedoucímu bakalářské práce prof. Ing. Adamu Heroutovi, Ph.D. za odborné vedení, cenné rady a připomínky při konzultacích. Dále bych rád poděkoval všem, kteří se podíleli na testování a za jejich zpětnou vazbu a především pak Jiřímu Polákovi za ochotu při tisknutí pouzdra pro hardwarové zařízení.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvořit zařízení zobrazující různá upozornění z chytrých zařízení s operačním systémem Android. Mezi tato upozornění patří notifikace aplikací, příchozí hovory a stav baterie. Základním prvkem tohoto vestavěného systému je malý jednodeskový počítač Arduino. Součástí projektu je i aplikace, která má za úkol tato oznámení zachytávat a následně je zasílat na zmíněné zařízení. Uživatel má možnost specifikovat, jaké notifikace se budou zobrazovat a s jakou prioritou. Každé aplikaci může také nastavit ikonu, která posléze bude promítána na maticový displej vytvořený z RGB LED diod. Komunikace mezi těmito dvěma platformami probíhá prostřednictvím technologie Bluetooth. Aby spojení mezi zařízením a aplikací běželo nepřetržitě a to i po vypnutí aplikace, je nutné implementovat Bluetooth komunikaci jakožto službu běžící na pozadí, která ale ovšem může být u některých nadstaveb systému Android omezována kvůli funkci šetření baterie.

Motivací pro tuto práci je fakt, že se v poslední době začalo ustupovat od notifikačních LED diod u většiny mobilních zařízení kvůli tzv. „bezrámečkovým“ displejům, a uživatelé tak ztrácí možnost být bez vyrušení informování o příchozích upozorněních. A i když mobilní zařízení tuto diodu obsahuje, bývá často její nastavení dosti omezené a pro některé uživatele toto může být nedostačující. Proto podobné notifikační zařízení může přijít náročnějším uživatelům velmi vhod. Dále může toto zařízení sloužit sluchově postiženým, již jsou závislí na vizuálních podnětech.

Kapitola 2

Vývoj na platformě Android

Aktuálně je Android nejpoužívanějším operačním systémem u mobilních zařízení s tržním podílem více než 86 % [13], proto si myslím, že není potřeba jakkoliv tuto platformu blíže představovat. Stejně tak se vyhnu vysvětlování podrobností základních prvků, které každý, kdo chce vyvíjet na platformě Android, musí nastudovat. V této kapitole se spíše zaměřím detailněji na komponenty, které systém nabízí a které jsou relevantní pro tento projekt.

2.1 Základní pojmy

V této části kapitoly budou pouze vysvětleny základní pojmy používané při programování na Androidu [15], aby čtenář více rozuměl dané problematice a nebudu zde nijak tyto jednotlivé komponenty blíže specifikovat.

Android je operační systém založený na Linuxu, navržený primárně pro zařízení s dotykovými obrazovkami. Jeho historie sahá až do roku 2003 a aktuální verze je pojmenovaná jako Android 10 [3].

Software Development Kit (dále jen SDK) je sada vývojových nástrojů umožňující vytváření aplikací. Jednotlivé verze SDK se v systému Android také označují jako úroveň API [10][1]. Každá úroveň odpovídá verzi Androidu a každá nová verze přináší do systému různá vylepšení. Verze API jsou zpětně kompatibilní a tudíž aplikace navržená pro starší systémy by měla vždy fungovat na aktuálních verzích. Při vývoji je nutné dbát na to, aby byla aplikace dostupná pro co největší počet uživatelů (s rostoucí verzí API počet uživatelů klesá), proto musí vývojáři hledat kompromis mezi použitím lepších funkcí a dostupností.

Na této platformě je možné programovat v různých programovacích jazycích [2]. Nejčastěji se však využívají jazyky Java, nebo Kotlin, které jsou uvedeny jakožto oficiální jazyky pro vývoj na Androidu. Je možné ale vytvářet kód v jazycích `C/C++`, `C#` nebo i `BASIC`.

2.1.1 Soubor AndroidManifest.xml

Tento soubor musí obsahovat každý projekt v kořenovém adresáři [10][2]. Poskytuje základní informace o aplikaci samotnému systému nebo třeba i službě Google Play.

[1] https://developer.android.com/guide/topics/manifest/uses-sdk-element#ApiLevels
[2] https://developer.android.com/guide/topics/manifest/manifest-intro

`AndroidManifest.xml` vyžaduje uvést následující informace:

- **Název balíčku (package name)** – jedinečný identifikátor pro všechny aplikace na Androidu. V rámci Google Play jsou aplikace publikovány a jednoznačně identifikovány právě pod tímto údajem. Název balíčku by se měl vždy shodovat s adresářovou strukturou projektu, aby se dalo snadno přistupovat k jednotlivým komponentám. Může vypadat například takto: `com.example.myapp`.
- **Komponenty aplikace** – pro každou z těchto komponent, ze kterých se aplikace skládají, se musí definovat základní vlastnosti jako například název třídy. Více o komponentách viz kapitolu 2.1.2.
- **Oprávnění aplikace** – tyto údaje je nutné uvést v případě, že aplikace přistupuje k chráněným částem systému nebo k jiným aplikacím. Uživatel je pak vyzván aplikací, aby jí daná oprávnění udělil, v případě, že aplikaci důvěřuje.
- **Hardwarové a softwarové funkce** – které aplikace využívá, a může tím značně ovlivňovat samotné zařízení, na kterém je nainstalována. Příkladem je komunikace přes technologii Bluetooth, jež může zapříčinit rychlejší vybíjení baterie.

2.1.2 Typy komponent

V Androidu existují celkem 4 typy komponent [15]. Každá aplikace se může skládat z více komponent. Komponenty jsou spouštěny asynchronně pomocí speciálních zpráv nazývaných `Intent` (viz kapitolu 2.4.1).

Activity (Aktivita)

Aktivita je hlavní třída, která se uživateli zobrazí po spuštění aplikace. Aktivity umožňují uživatelům skrze grafické rozhraní (GUI) ovládat aplikaci a komunikovat s ní. Skrze aktivity se implementují různé úlohy, které má uživatel realizovat, například vyplnit formulář, vybrat si položku seznamu, napsat SMS zprávu, aj. Aktivita by měla být vždy navržena tak, aby se uživatel mohl soustředit na pouze jedinou věc. V souboru `AndroidManifest.xml` se pak specifikuje, která z aktivit aplikace se bude spouštět jako první.

Service (Služby)

Služba je komponenta, která je navržena speciálně pro běh na pozadí. Na rozdíl od Aktivit, nemá uživatelské rozhraní. Umožňuje provádět asynchronně déle trvající operace. Více se službám věnuji v kapitole 2.5.

BroadcastReceiver

Objekty na vysílání a přijímání, poslouchají na pozadí a reagují na události odehrávající se v systému. Jednotlivé události jsou zabaleny do třídy `Intent` a ty pak zachytávají příslušné přijímače. Broadcast receiverům je věnovaná kapitola 2.4.1.

ContentProvider (Poskytovatelé obsahu)

Tyto komponenty umožňují ukládání a sdílení dat mezi více aplikacemi a procesy. Aplikace pak mohou přistupovat k datům ostatních aplikací označené jako `ContentProvider`, pokud je to povoleno. Využívá se přitom rozhraní podobné relačním databázím.

2.2 Technologie Near Field Communication

V této kapitole se velice krátce zaměřím na technologii NFC, protože je to jedna z bezdrátových komunikačních technologií dostupná na platformě Android, a tudíž jsem zvažoval ji využít v tomto projektu. Nicméně z důvodů uvedených dále v textu nebylo možné NFC použít.

Jak jsem již zmínil, Near Field Communication [14] (dále jen NFC) je bezdrátová technologie umožňující oboustranné zasílání dat mezi zařízeními s krátkým dosahem (4 cm). V posledních letech se NFC ujalo především v oblasti bezkontaktních plateb pomocí mobilních telefonů, například Google Pay nebo Apple Pay. Další možnosti využití jsou například přenos souborů, kontaktních informací nebo jako náhrada klíčů.

U NFC jsou data, podobně jako Bluetooth či Wi-Fi, posílána/přijímána pomocí rádiových vln. Technologie vychází ze staršího standardu RFID (Radio Frequency Identification), který funguje na principu elektromagnetické indukce. Díky tomuto principu jsme totiž schopni indukovat elektromagnetický proud v pasivních součástkách (například RFID čipy), a tak s nimi komunikovat. Toto je zásadní rozdíl oproti klasickému Bluetooth nebo Wi-Fi. Další výhodou je úspora energie.

Na platformě Android se můžeme s touto technologií setkat od verze 4.0, kde je označována jako funkce Android Beam [6]. Zasílání dat pomocí Android Beam se aktivuje přiložením dvou telefonů s podporou NFC zády k sobě. Teprve po té se na zařízení, ze kterého chceme odesílat data, objeví výzva „Tap to beam“. Klepnutím na obrazovku se data začnou přenášet do cílového zařízení. Je tedy zřejmé, že není možné pomocí této technologie vytvářet automatizované systémy.

2.3 Technologie Bluetooth Low Energy

Bluetooth Low Energy (dále jen BLE) byla navržena jako doplňující technologie k již existujícímu standardu Bluetooth a zároveň také jako technologie s co nejmenší možnou úsporou energie, jaké je možno dosáhnout [11]. Ačkoliv BLE využívá názvu i některých součástí implementace svého předchůdce, jedná se o zcela odlišné technologie, které nejsou kompatibilní. Proto je nutné zkontrolovat, zda smartphone tuto funkci umožňuje. BLE je na platformě Android dostupný od verze 4.3. Kvůli snížení úspory energie došlo také k omezení rychlosti přenosu dat. Na druhou stranu má BLE mnohem větší dosah signálu oproti klasickému Bluetooth.

2.3.1 Architektura BLE

V následující kapitole budou zmíněny základní prvky této technologie používané při implementaci BLE na systému Android, na které se budu odkazovat [20].

Attribute Protocol (ATT)

Jedná se o jednoduchý bezstavový klient/server protokol, kde každé zařízení které se účastní komunikace, je buď server, klient a nebo obojí. Každý server má data organizovaná ve formě atributů a každému z nich je přiřazeno 16 nebo 128 bitové identifikační číslo `UUID` (Universally Unique Identifer). Každý identifikátor určuje typ přenášených dat. Všechny `UUID` daného zařízení jsou vysílány, aby poskytlo informace o nabízených službách.

Generic Attribute Protocol (GATT)

Tento protokol je implementovaný jako rozšíření protokolu ATT, který přidává abstrakční model přenášených dat (atributů). Hlavní logická jednotka protokolu se nazývá služba. Každá služba se váže ke konkrétní funkcionalitě zařízení a skládá se z charakteristik, které určují různé hodnoty služby. Příkladem služby[3] může být *Monitor srdečního tepu*, obsahující charakteristiku *Frekvence srdečního tepu*, která se používá k zapouzdření této informace. K jednotlivým službám a charakteristikám se pak vážou určitá `UUID`. Výchozí délka paketu přenášených zpráv (MTU – Maximum Transmission Unit) je nastavena na 20 bajtů. V některých případech lze MTU nastavit na vyšší hodnotu, ovšem za předpokladu, že to hardware umožňuje.

2.3.2 BLE oprávnění

Aby mohla aplikace využívat služeb Bluetooth, tedy i BLE, musí jí být přidělena patřičná oprávnění uvedena v souboru `AndroidManifest.xml` [10][4]. Pro komunikaci prostřednictvím technologie Bluetooth, včetně žádostí o připojení se na jiná zařízení, jsou zapotřebí následující oprávnění. Aplikace navíc bude mít možnost vypínat/zapínat funkci Bluetooth na smartphonu. Poslední deklarace říká, že aplikace bude dostupná pouze pro BLE zařízení.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

<uses-feature android:name="android.hardware.bluetooth_le"
              android:required="true"/>
```

Další oprávnění, které musí být aplikaci uděleno, je přístup k poloze smartphonu, kvůli skenování dostupných Bluetooth zařízení. Oproti předchozím oprávněním, je toto explicitně vyžadováno po uživateli prostřednictvím dialogového okna.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

2.4 Zachytávání systémových událostí

Na operačním systému Android existuje mnoho způsobů, jak lze sledovat nejrůznější události, které mohou na zařízení nastat [19]. Událostmi jsou například myšleny příchozí nebo odchozí hovory, přijetí SMS zprávy či jiné notifikace[5] ale také nainstalování nové aplikace, změna stavu nabití baterie, připojení/odpojení SD karty a další.

2.4.1 BroadcastReceiver

`BroadcastReceiver` (dále jen receiver) je komponenta, která slouží k přijímání broadcast zpráv zasílaných ostatními částmi aplikace, jinými aplikacemi nebo systémem samotným.

[3] Oficiální definice služby podle Bluetooth SIG: https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Services/org.bluetooth.service.heart_rate.xml

[4] https://developer.android.com/guide/topics/connectivity/bluetooth-le

[5] „Notifikace je synonymum pro oznámení, nejčastěji se využívá v aplikacích, v mobilních telefonech nebo na internetu.“, https://it-slovnik.cz/pojem/notifikace

Broadcasty slouží pro komunikaci mezi různými částmi systému [10][6]. Mohou být například využity pro výměnu dat mezi aplikací a její službou na pozadí. Další možností je indikace systémové události, kdy systém odešle broadcast, který pak pomocí receiveru můžeme zachytit a na tuto událost reagovat. Broadcast zpráva je zabalená do objektu `Intent`, který nese samotný obsah zprávy.

Receiver je nutné nejprve zaregistrovat, aby mohl přijímat zprávy. Kromě samotné registrace je nutné ještě specifikovat, jaká data se budou přijímat. To se určuje pomocí komponenty `IntentFilter`. Registrace receiveru může být buď statická a nebo dynamická.

Statická registrace

Statická registrace [19] se provádí v manifestačním souboru `AndroidManifest.xml` pomocí tagu `<receiver>`. Povinným argumentem tohoto tagu je `android:name`, ve kterém je uvedena cesta k receiveru, který má být zaregistrován. Dalším argumentem je například `android:exported` [10][7] určující, zda-li může receiver přijímat zprávy pouze od jiných částí aplikace (hodnota argumentu `false`), a nebo i od vnějších zdrojů (hodnota argumentu `true`). Tag `<intent-filter>` pak obsahuje seznam tagů `<action>` určující, jaké broadcasty bude receiver přijímat.

Následující příklad ukazuje zaregistrování receiveru `BatteryBroadcastReceiver`, který přijímá broadcasty při každé změně stavu baterie. Argument `android:exported` není nutné uvádět, protože bude v tomto případě automaticky systémem nastavený na hodnotu `true`.

```
<receiver android:name=".BatteryBroadcastReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BATTERY_CHANGED" />
    </intent-filter>
</receiver>
```

Dynamická registrace

Dynamické registrování [19] se provádí přímo v kódu aplikace, nejčastěji pak v aktivitách nebo službách. Pro zaregistrování receiveru slouží funkce `Context.registerReceiver()`, která přijímá dva argumenty – objekt typu `BroadcastReceiver` a druhý objekt typu `IntentFilter`. Princip je obdobný jako u statické registrace. Zde je navíc nutné receiver odregistrovat v případě ukončení aktivity nebo služby, ve které byl receiver zaregistrován, jinak může dojít k chybě *BroadcastReceiver leaked*.

Následující příklad ukazuje zaregistrování receiveru `batteryBroadcastReceiver` přijímající broadcasty při změně stavu baterie, ale tentokrát dynamicky.

```
IntentFilter filter = new IntentFilter();
filter.addAction(Intent.ACTION_BATTERY_CHANGED);
registerReceiver(batteryBroadcastReceiver, filter);
```

Funkce `IntentFilter.addAction()` přijímá argument typu `String`. V příkladu je uvedená konstanta `Intent.ACTION_BATTERY_CHANGED`, jejíž hodnota odpovídá řetězci, který byl použit v předchozím příkladě – `android.intent.action.BATTERY_CHANGED`.

[6] https://developer.android.com/guide/components/broadcasts.html
[7] https://developer.android.com/guide/topics/manifest/receiver-element.html#exported

Výhodou oproti statické registraci je možnost odregistrace receiveru prakticky kdykoliv. Na druhou stranu takovýto receiver nemůže fungovat bez spuštěné aplikace. A to právě statická registrace umožňuje. I v případě, že je aplikace vypnutá, systém zavolá staticky zaregistrovaný receiver zapsaný v souboru `AndroidManifest.xml` a příslušné operace se provedou na pozadí.

Implementace BroadcastReceiveru

Reciver je potomkem třídy `BroadcastReceiver`, ve kterém je nutné implementovat abstraktní metodu `onReceive()`. Jedním z jejích argumentů je objekt datového typu `Intent`. Tato metoda je pak volána automaticky pokaždé, když nastane nějaká z událostí, kterým receiver naslouchá. Podrobnosti o nastalé události jsou pak předávány právě pomocí Intentu.

Intent

Pojem Intent [15] úzce souvisí s tématem BroadcastReceiver, proto je nutné se o něm alespoň částečně zmínit. Intent je asynchronní zpráva zasílána různými částmi systému. Slouží k tomu, aby jedna aktivita spustila druhou, nebo aby spustila službu na pozadí, případně provedla nějakou činnost (například přehrát hudební soubor, otevřít obrázek atd). Jak již bylo zmíněno výše, dají se pomocí intentů posílat i broadcast zprávy. Třída `Intent` je datová struktura, která obsahuje informace o prováděné akci či události. Existují dva druhy intentů – explicitní a implicitní. Explicitní intenty přesně definují, jaká akce se bude provádět a s jakými daty, a které aktivitě případně službě jsou určeny. Implicitní intenty jsou sice nastaveny pro vykonání určité činnosti, ale nemají přesně definovanou aktivitu, která tuto činnost vykoná. V této chvíli záleží pouze na uživateli, kterou aplikaci si pro vykonání operace vybere. V případě, že je na v systému nainstalováno více aplikací, které dokáží provést stejnou akci (například otevření PDF souboru), je uživateli poskytnuta možnost výběru.

Některé receivery mohou přijímat intenty vytvořené více různými událostmi. Aby byl receiver schopen od sebe jednotlivé události odlišit, je zapotřebí implementovat metodu `getAction()` vracející řetězec označující provedenou akci. Názvy systémem vyvolaných akcí jsou předdefinovány pomocí konstant ve třídě `Intent` (viz příklad dynamické registrace v kapitole 2.4.1).

Oprávnění aplikace

Některé broadcast zprávy mohou ovšem zasílat citlivé údaje, kterých by mohlo být zneužito, a proto je povinností, aby uživatel udělil aplikacím manipulujícím z těmito informacemi patřičná oprávnění [10][8]. Mezi takovéto informace můžou například patřit SMS zprávy, GPS lokace nebo využívání hardwarových komponent, jako je kamera nebo přístup k internetu. Metoda `ActivityCompat.requestPermissions()` slouží k získání povolení od uživatele pomocí dialogových oken, díky kterým může uživatel jednoduše povolit aplikaci přístup k daným informacím. Veškerá potřebná oprávnění se uvádějí v manifestačním souboru v tagu `<uses-permission>`.

[8] https://developer.android.com/training/permissions/requesting

2.4.2 PackageManager

Třída `PackageManager` [19, 10][9] slouží k získávání informací o nainstalovaných aplikacích na zařízení (informace ze souboru `AndroidManifest.xml`) – názvy aplikací a balíčků, ikony, zda je jedná o systémovou nebo uživatelem nainstalovanou aplikaci a nebo například jaké komponenty aplikace využívá. Implementace v kombinaci s BroadcastReceiverem lze využít k získání informací ohledně nově nainstalované aplikace v zařízení.

2.4.3 Služba NotificationListenerService

`NotificationListenerService` [10][10] (dále jen NLS) je systémový nástroj, umožňující aplikacím zachytávat informace o všech příchozích notifikacích v zařízení. Při nově příchozím upozornění je službě prostřednictvím systému zaslán objekt typu `StatusBarNotification` zapouzdřující veškeré informace, které jsou prezentovány uživateli ve stavovém řádku systému. Jedná se například o název balíčku, nadpis a text notifikace, čas zobrazení, případně ikonu.

NLS poskytuje různé množství informací o notifikacích. Aplikace využívající tuto službu by mohla být hrozbou úniku soukromých dat [17]. Některé aplikace, jako například SMS či G-mail, vkládají do nadpisu název kontaktu případně telefonní číslo nebo emailovou adresu. Do podrobností pak vkládají část, nebo i dokonce celý obsah příchozí zprávy. Zde právě může nastat problém. Stále existují webové servery, které zasílají přihlašovací údaje pomocí emailu. Pokud by si útočník pomocí aplikace s implementovanou NLS přečetl email obsahující tyto údaje, měl by přístup k dalším soukromým datům napadeného uživatele. Dále pak může aplikace shromažďovat data využita například reklamními společnostmi k zobrazování relevantnějších reklam. Právě z tohoto důvodu musí uživatel opět každé aplikaci využívající NLS přidat patřičná práva. Ve srovnání s druhým nejpoužívanějším mobilním systémem iOS od společnosti Apple je tato možnost úniku soukromí vyřešena velice elegantně – žádná obdobná služba zde neexistuje.

Chování této služby dosti závisí na konkrétní verzi operačního sytému a nadstavbě na daném zařízení. Této problematice se více věnuji v kapitole 5.4.2. Podle oficiální dokumentace pro Android je NLS odstřižena od systému, pokud nemá zařízení dostatečně velké množství volné paměti RAM (výchozí nastavení Androidu je 1 GB a méně).

Implementace NotificationListenerService

Nejprve je nutné zaregistrovat novou službu v souboru `AndroidManifest.xml` pomocí tagu `<service>`. Obdobně jako u BroadcastReceiveru je zapotřebí uvést název registrované třídy, dále název služby zobrazený uživateli při udělování oprávnění (viz obrázek 2.1) a také je nutné upozornit systém, že daná služba bude vyžadovat přístup k notifikacím pomocí vlastnosti `android:persmission`. Nakonec se pomocí Intent filtru určí, že služba bude přijímat informace ohledně notifikací.

Samotná třída pak musí obsahovat tři metody, které jsou volány systémem asynchronně. První z nich `onNotificationPosted()` je spuštěna v případě, že přišlo nějaké nové upozornění. Vstupní argument této metody je typu `StatusBarNotification`, o kterém jsem se již zmiňoval. Druhá metoda `onNotificationRemoved()` se naopak zavolá, pokud uživatel odstranil notifikaci ze stavového řádku. Opět na vstupu přebírá stejný argument jako

[9] https://developer.android.com/reference/android/content/pm/PackageManager

[10] https://developer.android.com/reference/android/service/notification/NotificationListenerService

předchozí metoda. Poslední metoda `onListenerConnected()` se zavolá, pokud je služba připojena k systému, to je okamžik, kdy uživatel přidá aplikaci oprávnění.

```
<service android:name=".MyNotificationListenerService"
        android:label="Name of service"
        android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
    <intent-filter>
        <action android:name="android.service.notification.NotificationListenerService" />
    </intent-filter>
</service>
```

Oprávnění aplikace

Na rozdíl od ostatních oprávnění, jejichž příklady jsou uvedeny kapitole 2.4.1, byla pro NLS zavedena zvláštní část v nastavení systému – *Přístup k oznámením* (viz obrázek 2.1). Zde si uživatel může zvolit, které aplikace ze seznamu budou mít možnost přistupovat k notifikacím. Toto může být pro běžného uživatele velice nepohodlné, ale na druhou stranu jej to donutí zamyslet se nad tím, že opravdu přiděluje aplikaci přístup k něčemu důležitému, a proto by to neměl brát na lehkou váhu. Naštěstí je možné toto uživateli zjednodušit pomocí odkazu v aplikaci, který ho do nastavení přímo nasměruje.

Obrázek 2.1: Přidělení oprávnění pro NLS. Android vyžaduje pro některá nastavení, aby uživatel sám v systémovém dialogu zapnul citlivou funkcionalitu.

2.4.4 Další možnosti

Existuje mnoho dalších způsobů zachytávání událostí [19], kterým se ale nebudu podrobněji věnovat, jelikož nejsou potřebné pro tuto práci. Pouze zde uvedu pár zajímavých příkladů.

FileObserver

Třída `FileObserver` umožňuje sledovat události týkající se souborů bez ohledu na to, jaký proces s těmito soubory pracoval. Jedná se o vytvoření/smazání souboru, změny v souboru, přesunutí, otevření/zavření souboru. Jeho nevýhodou je, že prohledávání souborů nefunguje rekurzívně, tzn. že dovolí sledovat maximálně jednu úroveň pod aktuální adresář.

ActivityManager

Pomocí třídy `ActivityManager` lze zjišťovat informace o právě běžících aplikacích. Na základě *id procesu* (pid) poskytuje stav vybrané aplikace včetně toho, kolik operační paměti

aktuálně využívá. Dále umožňuje zjistit informace o běžících procesech a jaké aplikace v nich běží nebo podrobnosti o službách (například jméno služby, dobu od zapnutí, aj).

Senzory

Dnes již všechna zařízení s operačním systémem Android mají nějaké senzory. Ty jsou schopné měřit například teplotu, rotaci zařízení, zrychlení, tlak nebo magnetické pole. Tato měření se dají svým způsobem také považovat za události systému, které je možno zachytávat a zpracovávat poskytované informace. Například měření magnetického pole využívané pro aplikaci kompas.

2.5 Služby

Při vývoji aplikací je čas od času zapotřebí některé operace provádět na pozadí bez nutnosti interakce s uživatelem – přehrávání hudby, stahování souboru, komunikace se zařízením přes Bluetooth, synchronizace dat mezi zařízením a cloudem, aj. Služba [15, 10][11] je komponenta aplikace, která právě toto umožňuje. Rozdíl mezi službou a BroadcastReceiverem je ten, že BroadcastReceivery jsou určeny pouze pro jednoduché a specifické úkony.

Hlavními výhodami služeb jsou:

- provádění činností na pozadí i v případě, že systém ukončí aplikaci,
- jsou méně náchylné k násilnému ukončení systémem v případě nedostatku paměti,
- v případě ukončení systémem, může být opět automaticky restartována.

Ačkoliv běží služba na pozadí, sdílí stejné vlákno s aplikací. Proto, aby mohla služba běžet dlouhodobě a nezávisle na aplikaci, je nutné pro tuto službu explicitně vytvořit nové samostatné vlákno. Službu spouštíme metodou `startService()` a běží až do doby, než je ukončena sama sebou pomocí metody `stopSelf()`, jinou komponentou pomocí `stopService()` a nebo systémem.

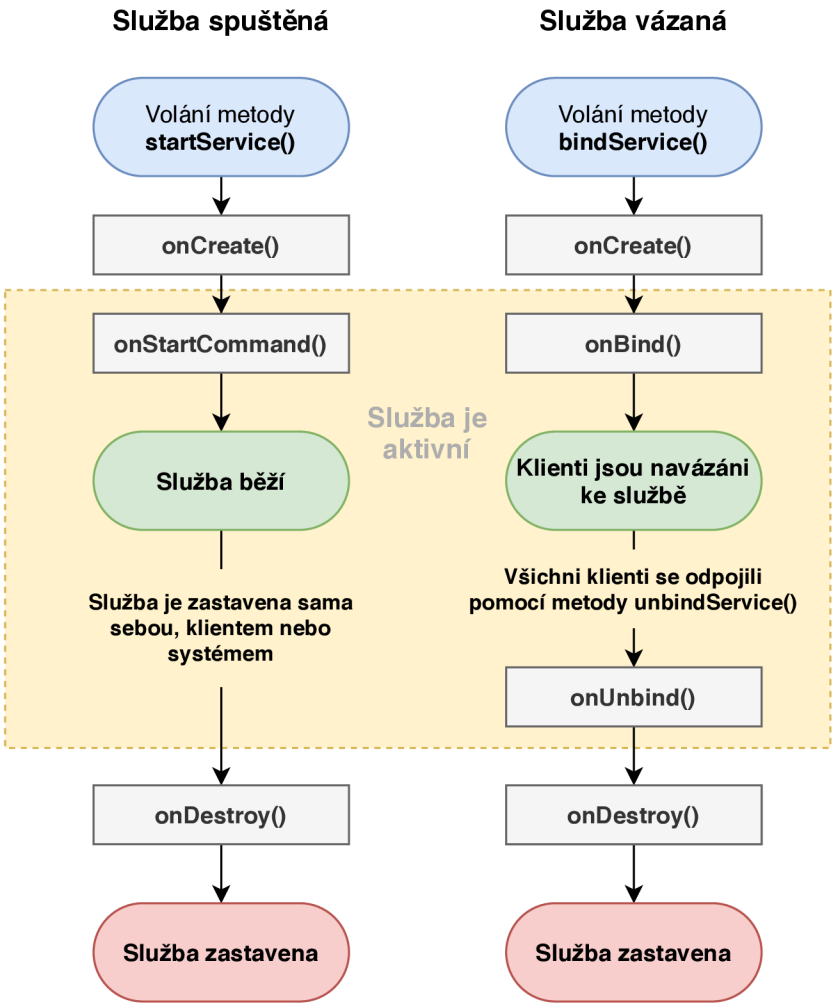
2.5.1 Životní cyklus služby

Služba má dva základní stavy, ve kterých běží, přičemž se může nacházet v obou současně:

- **Služba spuštěná** – služba běží na pozadí. Měla by se automaticky ukončit po té, co provedla všechny potřebné akce. V opačném případě ji může násilně ukončit systém, aby uvolnil prostředky. Příkladem může být hudební přehrávač – pomocí uživatelského rozhraní si uživatel zvolí jakou písničku chce přehrát, aktivita spustí službu na pozadí a předá jí informace o dané skladbě. Služba písničku začne přehrávat na pozadí. Pokud již uživatel nechce pokračovat v přehrávání, může tuto službu ukončit. Sama však bude pokračovat v přehrávání dokud nepřehraje poslední písničku v seznamu. Dalším příkladem je stahování souborů. Uživatel si zvolí soubor a aktivita spustí službu, která soubor stáhne do zařízení. Po dokončení procesu se služba sama ukončí.

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

- **Služba vázaná** – služba poskytující rozhraní klient-server. Umožňuje jednotlivým komponentám aplikace, aby spolupracovaly se službou, posílaly jí požadavky, získávaly výsledky, nebo jinak s ní komunikovaly. Tato služba je spuštěna dokud je k ní připojena alespoň jedna komponenta. Pokud se i poslední komponenta odpojí, systém službu ukončí. K navázání spojení se službou se používá metoda `bindService()`. Pokud již komponenta nepotřebuje nadále využívat službu, měla by zavolat metodu `unbindService()`.



Obrázek 2.2: Životní cyklus služby (převzato z [15])

Významy jednotlivých metod:

- **onCreate()** – zavolá se při pouze prvním vytvoření instance služby,
- **onBind()** – zavolá se v případě, že některá z komponent požaduje připojení se k již běžící službě,
- **onStartCommand()** – zavolá se po skončení metody `onCreate()` a vrací jednu z konstant, které určují, jak se bude služba chovat, když ji systém ukončí kvůli nedostatku prostředků:

  - **START_STICKY** – služba se vždy restartuje,
  - **START_NO_STICKY** – služba se restartuje pouze, pokud nějaké komponenty čekají na vyřízení jejich požadavků, jinak služba zůstane zastavená,
  - **START_REDELIVER_INTENT** – služba se chová stejně jako v předchozím případě, ale navíc je zachován původní `Intent`, který jí byl předán při prvním spuštění.

- **onUnbind()** – metoda se volá, když se odpojí všechny komponenty, které se službou komunikovaly,

- **onDestroy()** – zavolá se při zrušení služby. Prostřednictvím této metody by měla služba uvolnit veškeré prostředky.

Aby mohla služba fungovat musí se zaregistrovat pomocí tagu `<service>` v aplikačním manifestu (soubor `AndroidManifest.xml`) stejně jako v případě NLS (viz kapitola 2.4.3).

2.5.2 Foreground service

V češtině nazývaná jako služba na popředí [10][12] je speciální typ služby spuštěné nebo vázané. Tento typ je dle oficiální dokumentace doporučeno používat vždy, když aplikace provádí nějakou dlouhodobou operaci, o které by měl uživatel vědět.

Opět velice názorným příkladem je hudební přehrávač. Po dobu přehrávání skladeb se ve stavovém řádku zobrazuje nesmazatelná notifikace upozorňující uživatele na právě probíhanou akci. Uživatel může se službou komunikovat pomocí tlačítek, například zastavení přehrávání, přehrání následující skladby, apod. Služba může běžet i autonomně bez nutnosti uživatelovy interakce, například synchronizace dat s cloudovým úložištěm.

Implementace notifikace a služby na popředí

Pro fungování takovéto služby je nutné vytvořit nejprve notifikaci zobrazenou ve stavovém řádku. Vytváření notifikace se liší pro různé verze Androidu. Od verze 8.0 (API 26) a vyšší musí být všechny notifikace přiřazeny k nějakému kanálu [10][13]. Pro každý kanál jsou nastaveny různé zvukové i vizuální vlastnosti, které se aplikují pro každou notifikaci z daného kanálu. Uživatel má možnost si toto nastavení změnit. Také se může rozhodnout, které z kanálů dané aplikace jsou pro něj rušivé a zakázat tak zobrazování konkrétních notifikací. Díky tomuto systému je možné omezovat zobrazení jen některých upozornění aplikací. Zjišťování verze Androidu, na které aplikace je právě spuštěna, je ukázáno v následujícím příkladu.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O)
{
    // Code will continue here only if Android API version is 26 or above
}
```

[12] https://developer.android.com/guide/components/services
[13] https://developer.android.com/training/notify-user/channels.html

2.6 Ukládání údajů

Nejen počítačové, ale i mobilní aplikace pracují s velkým množstvím dat, které mohou být uloženy jak na vzdáleném serveru, tak i lokálně na zařízení [15]. Z tohoto důvodu Android obsahuje třídy, které umožňují práci jak s menším množstvím dat, nebo i s většími objemy dat pomocí lokální databáze SQLite. Zařazuje se sem také práce s cloudovými úložišti, které má na starosti služba Firebase.

V této části textu se krátce zmíním o všech možných způsobech ukládání dat na této platformě, více se však zaměřím na databázi SQLite a `SharedPreferences`, které jsou v této práci využity.

Možnosti ukládání údajů:

- **Shared Preferences** – je ukládání údajů pomocí základních datových typů ve formátu klíč-hodnota,
- **Interní úložiště** – uložení údajů do vnitřní paměti zařízení,
- **Externí úložiště** – uložení údajů do přídavné paměťové karty (SD karty),
- **SQLite** – uložení údajů do databáze přímo v zařízení Android,
- **Síťové úložiště** – uložení údajů na webový server,
- **Poskytovatelé obsahu** – tato komponenta je zmíněna v kapitole 2.1.2.

2.6.1 Shared Preferences

SharedPreferences dále jen SP je nejjednodušší způsob ukládání dat na Android zařízeních [10][14]. Je navržen pro ukládání a správu malého množství jednoduše strukturovaných dat. Třída je implementována jako perzistentní mapa k ukládání párových údajů typu klíč-hodnota. Podporovanými datovými typy jsou `boolean`, `int`, `long`, `float`, nebo také `string`. Třída SP vytváří XML soubor, do kterého se pak zadaná data ukládají.

Příkladem údajů, které se mohou skrze tuto třídu ukládat, je záznam nejvyššího skóre dosaženého ve hře. Existuje jeden pár klíč-hodnota (například `SCORE-235`). Dále se SP používá k uložení konfiguračních údajů aplikace. Pro tyto informace je naprosto zbytečné vytvářet SQL databázi. Data jsou perzistentní, to znamená, že zůstanou uložena na zařízení i v případě vypnutí aplikace.

PreferenceScreen a PreferenceFragment

Ukládání konfiguračních údajů je jedním z nejčastějších případů užívání třídy SP. Aby vývojáři nemuseli neustále řešit v každé aplikaci uživatelské rozhraní pro zobrazování a nastavování těchto hodnot, byla do Androidu přidána třída `PreferenceFragment`. Tato třída umožňuje vytvořit fragment (část layoutu aktivity), který pak zakomponujeme do uživatelského rozhraní aplikace. Na obrázku 4.6, obsahuje tento fragment jednotlivé položky nastavení. `PreferenceScreen` pak není nic jiného, než-li kořenová komponenta, obsahující v dané aktivitě všechny fragmenty typu `PreferenceFragment`.

[14] https://developer.android.com/reference/android/content/SharedPreferences

2.6.2 Databáze SQLite

SQLite je relativně malá, ale za to výkonná multiplatformní knihovna implementující kompletní relační databázový systém [15]. SQLite je založena na standardu SQL-92[15]. A na rozdíl od ostatních projektů jako jsou například MySQL, Oracle nebo PostgreSQL, nevyužívá klasického přístupu klient-server. Samotná data jsou uložena v lokálním souboru na dané platformě, ke kterým se přistupuje za pomoci klientské aplikace – jinými slovy aplikace využívající SQLite zastává funkci jak klienta, tak i serveru současně. V případě platformy Android není nutné instalovat databázový engine nebo jej přibalovat k aplikacím, protože SQLite je již součástí systému.

[15] Více o SQL-92 na adrese: http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

Kapitola 3

Vývoj na platformě Arduino

Arduino [22, 5] je elektronická open-source platforma založena na malé a jednoduché desce s vlastním vývojovým prostředím. Díky Arduinu je možno vytvářet interaktivní objekty, nebo-li vestavěné systémy. Vzhledem k tomu, že je tato platforma open-source, existuje celá škála různých modulů, která se stále rozšiřuje. Tyto moduly pak zastávají jiné funkce, některé se mohou chovat jako vstupy (teploměr, senzor světla, tlačítko, případně celá klávesnice), jiné jako výstupy (LED diody, LCD obrazovky), případně dokáží obojí (Wi-Fi moduly).

3.1 Základy desky Arduino

Každá deska Arduino sestává z několika základních komponent – procesor od firmy Atmel, Flash paměť, kde se ukládá program, SRAM paměť pro ukládání a manipulaci s proměnnými, napájecí konektor, konektor USB typu B (u některých verzí Arduina se můžeme setkat i s variantami mini nebo micro) pro napájení a/nebo programování desky, a dále pak vstupní/výstupní piny, pro komunikaci s dalšími externími periferiemi zmíněnými výše [22].

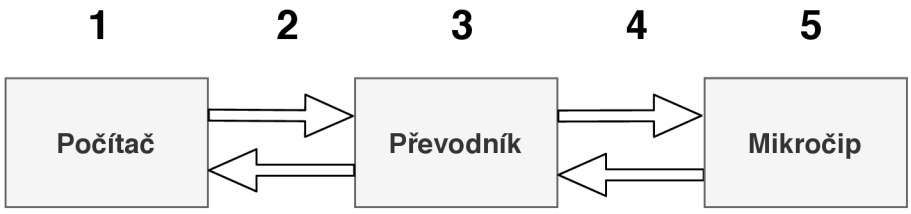
Existuje však nepřeberné množství variant těchto vývojových desek [1, 22]. Všechny se liší nejenom velikostí, ale také výkonem procesoru, velikostí pamětí a také komponentami které se na desce nachází – například verze Arduino Yún obsahuje navíc i Ethernet port. Mikrokontrolér který se na desce nachází má speciální architekturu, zvanou AVR, což označuje rodinu mikročipů typu RISC[1] s Harvardskou architekturou[2]. Pro napájení desky se běžně využívá napájecího zdroje případně externí baterie. Zdroj energie se pak používá k napájení mikropočítače a zároveň připojených komponent. Pro napájení lze využít USB konektoru a nebo napájecí konektor `DC12V`, který se může využít právě například pro připojení externí baterie.

3.1.1 Sériová komunikace

Komunikace mezi Arduinem a počítačem probíhá skrze USB port pomocí asynchronního sériového rozhraní UART (Universal Asynchronous Receiver-Transmitter) [16]. Jedná se o malé hardwarové zařízení, které pomocí dvou pinů odesílá a přijímá data. Přijímač i vysílač obsahují vlastní generátor hodinového signálu s rozdílnými rychlostmi, proto je zapotřebí přítomnosti vyrovnávací paměti (buffer), aby mohly být hodiny obou komunikujících stran synchronizovány.

[1] Jedná se o procesory s redukovanou instrukční sadou.
[2] Typ počítačové architektury, kde paměti pro data a pro program jsou odděleny.

Následující popis odpovídá většině verzí Arduina [22]. Na obrázku 3.1 je schéma sériové komunikace. Jednotlivé části procesu jsou popsány pod tímto obrázkem. Sériová komunikace nemusí ovšem nutně probíhat pouze mezi počítačem a Arduinem, ale také mezi Arduinem a nějakým připojeným modulem, například Bluetooth.



Obrázek 3.1: Schéma sériové komunikace

1. **Vývojářský počítač** – pomocí Arduino IDE probíhá oboustranná komunikace s deskou. Skrze sériový monitor je možné číst zprávy poslané z Arduina (pomocí funkce `Serial.print()`), které slouží například pro debugging, nebo naopak je možné na desku zprávy zasílat (pomocí funkce `Serial.read()`), v sériovém monitoru pak uživatel posílá zprávy pomocí uživatelského rozhraní.
2. **USB kabel**
3. **Převodník** – existují tři typy převodníků, jejich jediným rozdílem je však způsob zapojení:
   - převodníky připájené napevno k základní desce,
   - převodníky, které jsou součástí některých čipů (např. ATmega32u4),
   - externí převodníky, které se musejí při programování k desce připojit.
4. **Spojení mezi převodníkem a čipem** – při použití externího převodníku se obvykle jedná o šest vodičů, které vystupují z desky. U ostatních dvou způsobů jsou tyto komponenty spojeny pomocí plošných spojů na desce, nebo propojeními uvnitř čipu.
5. **Mikročip** – z převodníku přijímá zprávy a zpracovává je, případně zprávy převodníku předává.

Pro zahájení komunikace se používá funkce `Serial.begin()`, jejímž parametrem je rychlost přenosu udávaná v jednotách baudy[3]. Rychlost přenosu zadaná na zařízení musí odpovídat rychlosti nastavené v sériovém monitoru. Pro tuto vlastnost je vyhrazeno několik hodnot, nejčastěji se však u Arduina používá hodnota 9600 Bd.

Odesílání dat

K přenášení dat z Arduina do vývojářského počítače (případně do jiného zařízení připojené sériovou komunikací) se využívají typicky dvě funkce `Serial.write()` a `Serial.print()`, které jako povinný parametr přebírají zasílaná data. Rozdíl mezi funkcemi je ten, že funkce `Serial.print()` zobrazuje data na připojeném počítači formátované jako čitelný ASCII text, kdežto `Serial.write()` zasílá pouze binární data.

[3]Baud (Bd) je jednotka udávající počet přenosů za sekundu.

Přijímání dat

Po odeslání dat z nějakého zařízení (z počítače nebo modulu), jsou nejprve tato data uložena do vyrovnávací paměti [21]. Jeho výchozí velikost je 64 bajtů, ta se dá i navýšit, ale bude naopak snížena dostupná velikost operační paměti (SRAM). Samotné čtení pak probíhá postupně po bajtech – přečte se bajt na vrcholu zásobníku, zpracuje se procesorem, zbývající data se posunou o jednu pozici a v následujícím cyklu se čte opět z vrcholu zásobníku.

Ke čtení se využívají dvě funkce. První z nich, `Serial.available()` upozorňuje, když se v bufferu objeví nějaká data – indikuje návratovou hodnotou označující velikost dat. Funkce `Serial.read()` pak vrací jeden bajt z vrcholu zásobníku (bufferu).

3.2 Programovací jazyk a vývojové prostředí

Desku Arduino je možné programovat v jazycích C nebo C++ [22]. Je ovšem doporučeno používat knihovnu Wiring, která je určena speciálně pro účely programování mikrokontrolérů, především se používá právě pro vývoj na platformě Arduino. Tato knihovna je v poslední době velice rozšířena a často se o ní mluví jako o samostatném jazyce.

Základní strukturou jazyka Wiring jsou dvě funkce `void setup()` a `void loop()`. Funkce `setup()` se spustí pouze jednou na začátku programu a využívá se zpravidla na počáteční konfiguraci zařízení, jako například deklarace proměnných, nastavení pinů nebo inicializace sériové komunikace. Po vykonání kódu je zavolána funkce `loop()`, která je pak automaticky volána v „nekonečné“ smyčce. Ta obsahuje kód ovládající desku.

Wiring definuje různé metody usnadňující práci při vývoji. Asi nejběžnější funkci se kterou se setkáme při vývoji je funkce `delay()`, jejímž parametrem je doba v milisekundách, po kterou deska pozastaví vykonávání kódu. Dalšími funkcemi jsou například `pinMode()`, která umožňuje nastavit piny na desce jakožto vstupy nebo výstupy, nebo funkce `digitalWrite()`, která na daný pin vyšle určité napětí, pokud je tento pin nastavený jako výstupní.

V následující ukázce je program, který každou vteřinu periodicky vypíná a zapíná LED diodu zapojenou k desce.

```
int ledPin = WLED; // pojmenovani LED

void setup() {
    pinMode(ledPin, OUTPUT); // nastaveni pinu pro digitalni vystup
}

void loop() {
    digitalWrite(ledPin, HIGH); // zapnuti LED
    delay(1000); // pockame jednu sekundu (1000 milisekund)
    digitalWrite(ledPin, LOW); // vypnuti LED
    delay(1000); // pockame jednu sekundu
}
```

Arduino IDE je oficiální multiplatformní vývojové prostředí, díky kterému můžeme snadno instalovat program na vývojovou desku. Kromě textového editoru pro psaní kódu obsahuje také i okno s informacemi o stavu překladu nebo také sériový monitor, jenž může být využit pro sledování právě naměřených hodnot z nějakého modulu. Před samotným překladem je zapotřebí nastavit verzi Arduina, na kterém se bude vyvíjet, a na který port

v počítači bude Arduino připojeno. Skrze toto prostředí lze také stahovat knihovny pro práci s moduly.

3.3 Moduly

Externí komponenty, nazývané též jako moduly [22], se používají k rozšíření funkcionality vestavěného zařízení. Existuje jich obrovské množství. Jak jsem již zmínil na začátku kapitoly, může se jednat jak o vstupní, tak i o výstupní moduly. V této části textu se zaměřím pouze na BLE moduly obecně, protože LED moduly nejsou nic jiného než LED diody připájené k desce s napájecími (případně datovými) piny. Konkrétní moduly použité pro tento projekt jsou blíže popsány v kapitole 4.3.2.

3.3.1 Moduly Bluetooth Low Energy

Většina modulů ať už pro Bluetooth či BLE nejen, že vypadá, ale i dost podobně funguje. Proto zde shrnu nejdůležitější vlastnosti, které by se měly brát v úvahu při hledání správné verze.

K ovládání těchto modulů se ve užívá právě sériové komunikace – tzv. Bluetooth SPP (Serial Port Protocol). Proto se pro čtení respektive psaní využívají již známé funkce `Serial.print()` (příp. `Serial.write()`) respektive `Serial.read()`. Nastavování zařízení se provádí pomocí AT příkazů [8], které se mohou lišit v závislosti na druhu modulu, a také podle toho, co vše tento modul nabízí. Skrze tyto příkazy je možné nastavit například rychlost komunikace, název modulu viditelný při skenování, nebo také umožňují zjistit MAC adresu zařízení, případně vypsání nápovědy. Důležitým parametrem je také vstupní napětí. To obvykle bývá 3,3 V nebo 5 V, což odpovídá napětí na pinech na desce Arduino určeným pro napájení. Přesto je potřeba si dávat pozor, aby se modul nezničil příliš vysokým napětím, na které není stavěný. Různé verze se mohou lišit především pak v dosahu signálu (obvykle 60 m až 100 m), maximální možné rychlosti komunikace a především odběru proudu.

Společným rysem (aspoň pro většinu modulů) jsou také konektory. Ty jsou následující:

- **STATE** – konektor je připojen k LED diodě na desce modulu, je využíván pro kontrolu, zda-li modul pracuje správně
- **RDX** – příjem dat sériové komunikace,
- **TDX** – vysílání dat sériové komunikace,
- **GND** – zemnící konektor,
- **VCC** – napájecí konektor,
- **EN** – přerušení navázaného spojení.

Všechny podrobné informace se dají zjistit z produktových listů (datasheet) k jednotlivým modulům.

Kapitola 4

Návrh systému pro zobrazování notifikací

Tato kapitola pojednává o návrhu jednotlivých částí celého systému. Bude zde zmínka o původním konceptu tohoto projektu a proč by nebyl funkční. Následuje návrh aplikace včetně databáze a uživatelského rozhraní. V neposlední řadě zde bude popsáno zapojení zobrazovacího zařízení a také návrh komunikačního protokolu používaným mezi aplikací a zařízením.

4.1 Původní a aktuální návrh systému

V této části textu shrnu dva koncepty fungování zařízení. Původní návrh byl upraven z důvodu změny technologie použité pro komunikaci mezi tímto zařízením a aplikací.

4.1.1 Původní návrh

Prvotní myšlenka systému zobrazujícího notifikace byla taková, že by toto zařízení fungovalo jako rozšíření bezdrátové nabíječky – obdoba dokovací stanice. Pro komunikaci mezi zařízením a mobilním telefonem by se využívala technologie NFC. Po položení telefonu na dokovací stanici by se zahájilo nabíjení a samozřejmě monitorování příchozích zpráv z aplikace pomocí NFC čtečky připojené k Arduinu. Na zařízení by byly instalovány LED moduly, které by podle nastavení uživatele svítily různými barvami v závislosti na upozornění.

Výhoda tohoto návrhu je tedy především téměř nulová spotřeba baterie, díky tomu, že by se dokovací stanice chovala i jako nabíječka. Na druhou stranu technologie NFC se vyskytuje u mobilních zařízení v dosti omezeném množství.

Nicméně, jak je již uvedeno v kapitole 2.2, ke každému zaslání dat z mobilního zařízení pomocí NFC, je zapotřebí schválení uživatele. Je zřejmé, že podobné chování je pro požadovanou autonomní komunikaci nepřijatelné.

4.1.2 Aktuální návrh

Z důvodů uvedených výše jsem se nakonec jsem se rozhodl pro komunikaci využít technologii Bluetooth Low Energy (dále jen BLE). Jak již bylo řečeno, BLE je podobně jako NFC také zaměřeno na úsporu energie oproti jejím starším verzím. Nespornou výhodou oproti původnímu návrhu je fakt, že v dnešní době je BLE součástí téměř každého mobilního zařízení, a proto omezení používat tento notifikační systém bude minimální.

Díky nově zvolené komunikační technologii se i trochu upraví návrh samotného zobrazovacího zařízení. Nyní již není podmínkou, aby byly telefon i zařízení v těsné blízkosti, jak by tomu s NFC muselo být. Proto jsem se rozhodl vytvořit samostatné zařízení, které si uživatel bude moci instalovat kamkoliv.

4.2 Návrh aplikace

Samotná aplikace se bude chovat jako ovládací prvek celého systému. Po té, co zachytí jakoukoliv požadovanou událost, pošle zařízení zprávu, jaká ikona se bude zobrazovat dle uživatelova nastavení. Jak již bylo řečeno, čtení notifikací u mobilních zařízení je možné pouze na platformě Android a pro splnění požadavků, bude aplikace vyvíjena pouze pro tento operační systém.

Před samotným zahájením komunikace bude zapotřebí se k zařízení připojit. Nejprve bude muset aplikace najít všechna dostupná zařízení pomocí skenování. Ta zobrazí uživateli do seznamu seřazeného dle RSSI. Po stisknutí na tlačítko `Connect` se aplikace pomocí MAC adresy zvoleného zařízení připojí. Aby se urychlilo pozdější znovu-připojování (i třeba v případě výpadku), bude si aplikace pamatovat MAC adresu posledního připojeného zařízení.

Aby se zajistilo, že systém bude bez přerušení fungovat co nejdéle a to i po vypnutí samotné aplikace, bude veškerá komunikace včetně řešení výpadků implementována jakožto služba na popředí (viz 2.5.2). K zachytávání notifikací nainstalovaných aplikaci bude využívat službu NLS (viz 2.4.3). Pro události, jako jsou změna stavu baterie či příchozí hovory, se využije BroadcastReceiverů (viz 2.4.1). Nakonec v případě nainstalování nové aplikace vytvoří na pozadí nový záznam v databázi s výchozím nastavením.

4.2.1 Ukládání dat

V rámci aplikace se budou data ukládat za pomocí jak databáze SQLite, tak i SharedPreferences. V databázi se budou ukládat vzory zobrazovaných ikon. Dále se zde zaznamená seznam všech nainstalovaných aplikací, a ke každé z nich uživatelovo nastavení. SharedPreferences se využije pro informace, pro které by bylo zbytečné vytvářet speciální tabulku, protože by obsahovala jediný záznam. Řeč je o nastavení upozornění příchozích hovorů, stavu baterie, a také k uložení MAC adresy posledně připojovaného zařízení.

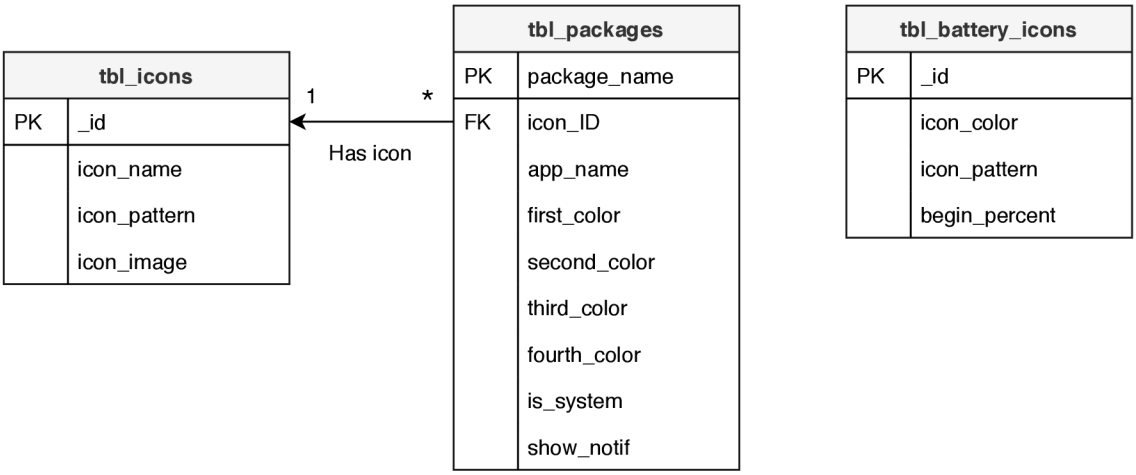
Databáze

ER diagram databáze je na obrázku 4.1. Obsahuje 3 tabulky:

- Tabulka `tbl_icons` bude obsahovat všechny ikony, které uživatel může přiřadit jednotlivým aplikacím telefonu. Konkrétně bude uchovávat číselný identifikátor `_id`. Dále pak název ikony `icon_name` a odkaz na náhledový obrázek `icon_image`, které budou zobrazeny v uživatelském rozhraní. A nakonec vzor ikony (více o vzorech ikon v kapitole 4.4.1) `icon_pattern`.

- Do tabulky `tbl_battery_icons` se opět uloží ikony, tentokrát ale pro upozornění týkající se baterie. Podobně jako v předchozí tabulce, bude i tato obsahovat číselný identifikátor `_id` a vzor ikony `icon_pattern`. Dále bude ukládat barvu dané ikony v atributu `icon_color`. Atribut `begin_percent` bude určovat procentuální stav baterie, při kterém se ikona zobrazí.

- Poslední tabulka `tbl_packages` pak již bude obsahovat samotné nastavení uživatele pro každou aplikaci. Jako jednoznačný identifikátor bohatě postačí název balíčku (vzhledem k tomu, že slouží jako globální identifikátor aplikací) `package_name`. Dále bude obsahovat informace zobrazující se v uživatelském prostředí – název aplikace `app_name` a její ikonu, respektive identifikátor ikony `icon_ID`. Atributy `first_color`, `second_color`, `third_color` a `fourth_color` uchovávají jednotlivé barvy pro ikony zobrazované na vestavěném zařízení. K rozlišení systémových aplikací od uživatelových bude sloužit atribut `is_system`. Atribut `show_notif` říká, zda uživatel notifikace dané aplikace chce zobrazovat. Poslední atribut `priority` označuje prioritu zobrazení na zařízení.



Obrázek 4.1: ER diagram databáze

4.2.2 Uživatelské rozhraní

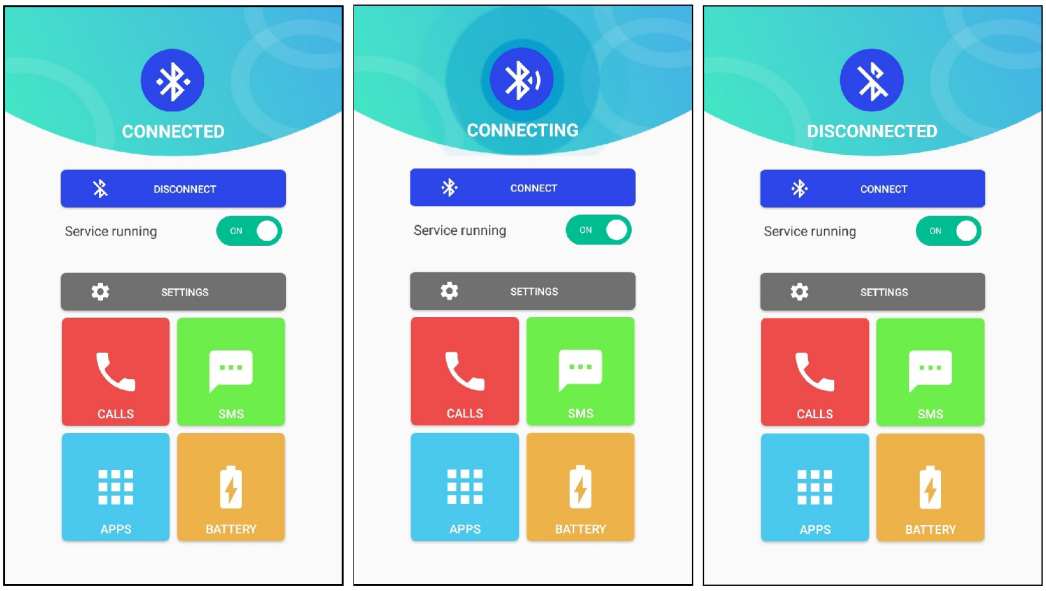
U grafické stránky aplikace je kladen důraz především na jednoduchost a srozumitelnost, a aby byly veškeré prvky co nejvíce intuitivní. Uživatel bude ve většině případů aplikaci používat pro připojování k notifikačnímu zařízení. Zbylé procento vykonaných akcí bude nastavování, jak se má tento systém chovat. V případě výpadku spojení se zařízením bude uživatel aplikací upozorněn.

Aplikace se bude skládat z osmi aktivit – hlavní obrazovka, skenování dostupných zařízení. nastavení notifikací hovorů, baterie, aplikací a zařízení, aktivita zobrazující seznam nainstalovaných aplikací a aktivitu zobrazující upozornění na nefunkční NLS nebo na nekompatibilitu s BLE.

Hlavní aktivita

Tato aktivita bude nejvíce využívanou částí celé aplikace, proto je nutné ji věnovat náležitou pozornost. Aktivita bude zobrazovat informaci o stavu připojení, která bude od zbytku prvků oddělena rozdílně zbarveným pozadím. Textový popis stavu (`Connected` / `Disconnected` / `Connecting`) bude doplněn o ikonu korespondující s tímto textem (viz obrázek 4.2). Pod stavovým indikátorem bude připojovací tlačítko, jehož text i ikonka se budou měnit v závislosti na aktuálním stavu (`Connect` / `Disconnect`). Pokud `ForegroundService`

nebude spuštěna a uživatel se bude chtít připojit k zařízení, aktivita nejprve pošle požadavek na její spuštění. Dále následuje přepínací tlačítko, které bude mít na starosti manuální spouštění, respektive vypínání `ForegroundService`. V poslední části obrazovky bude pět tlačítek sloužících jako odkazy do dalších aktivit aplikace. Tlačítka `Calls` a `Battery` spustí aktivity s nastavením pro tato upozornění. Tlačítko `Apps` spustí aktivitu se seznamem všech nainstalovaných aplikací na tomto Android zařízení. Po kliknutí na tlačítko `SMS` zjistí hlavní aktivita název balíčku výchozí aplikace pro SMS zprávy a spustí aktivitu s nastavením pro tuto aplikaci. Poslední tlačítko `Settings` spustí aktivitu pro nastavení zařízení.

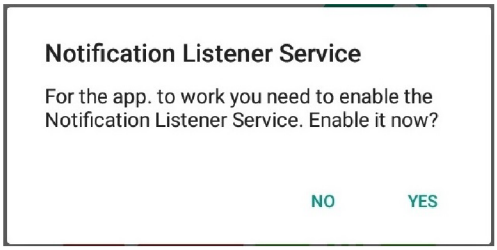


Obrázek 4.2: Hlavní aktivita – vlevo stav připojeno, uprostřed stav připojování a vpravo stav odpojeno

Při prvním spuštění aplikace zobrazí aktivita systémová dialogová okna s upozorněním na potřebná oprávnění. Tato oprávnění jsou nezbytná pro správné fungování aplikace (více o vyžadovaných oprávněních aplikace viz kapitolu 5.5). Pro udělení práv službě NLS je nutné vytvořit vlastní dialogové okno (obrázek 4.3), které po kliknutí na tlačítko `Ano` odkáže uživatele do nastavení, kde této aplikaci udělí oprávnění (více o NLS v kapitole 2.4.3), v opačném případě se spustí aktivita s upozorněním na nefunkční NLS. Posledním vyskakovacím oknem, které se může v této aktivitě zobrazit, je opět systémové dialogové okno pro zapnutí Bluetooth, které se zobrazí v případě, že se chce uživatel připojit na zařízení, ale funkce Bluetooth je vypnuta. Navíc bude aktivita kontrolovat, zda-li má dané Android zařízení podporu BLE, pokud ne, oznámí to uživateli spuštěním aktivity pro upozorňování.

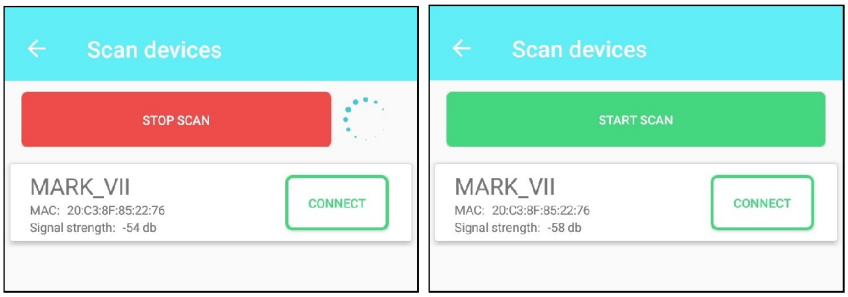
Skenování dostupných zařízení

V této aktivitě bude zobrazován seznam dostupných zařízeních, ke kterým je možné se připojit. Seznam bude průběžně řazen podle síly signálu (RSSI) sestupně. Každé zařízení bude reprezentováno kartou obsahující název zařízení, hodnotu RSSI, MAC adresu. Karta navíc bude obsahovat tlačítko pro připojení. Ve vrchní části obrazovky bude tlačítko pro zapnutí/vypnutí skenování. V případě, že bude probíhat skenování, zobrazí se i komponenta `ProgressBar` indikující uživateli probíhající akci. Skenování bude po určitém čase automaticky ukončeno z důvodu šetření baterie. Po kliknutí na tlačítko pro připojení se opět



Obrázek 4.3: Hlavní aktivita – dialogové okno jako odkaz na nastavení NLS

zobrazí `ProgressBar` tentokrát přes celou obrazovku. V případě úspěšného připojení se aktivita ukončí, jinak bude znovu zahájeno skenování. Tato aktivita bude spouštěna pouze v případě, nemá-li aplikace zaznamenanou MAC adresu posledně připojovaného zařízení nebo je toto zařízení nedostupné.



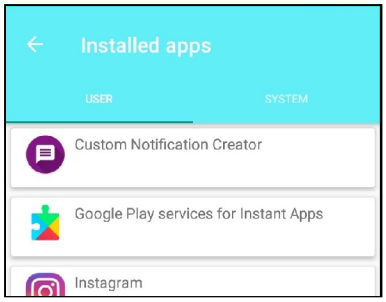
Obrázek 4.4: Aktivita pro skenování dostupných zařízení – vlevo skenování, vpravo dokončené skenování

Seznam nainstalovaných aplikací

Tato aktivita bude obsahovat dvě záložky. V první záložce bude seznam uživatelem nainstalovaných aplikací a v druhé seznam systémových aplikací. Oba seznamy budou seřazeny v abecedním pořadí. Tímto se docílí větší přehlednosti a uživatel pak nebude muset zdlouhavě hledat požadovanou aplikaci. Každá aplikace je reprezentována kartou obsahující její název a ikonu. Může se stát, že aktivitě bude trvat dlouho než se načte kvůli většímu množství nainstalovaných aplikací, proto se před načtením bude zobrazovat grafický element `ProgressBar`, aby byl uživatel informován o tom, že se pouze zpracovávají data a nenastala žádná chyba. Na jednotlivé karty bude možné klikat a budou sloužit jako odkaz do aktivity s nastavením zvolené aplikace.

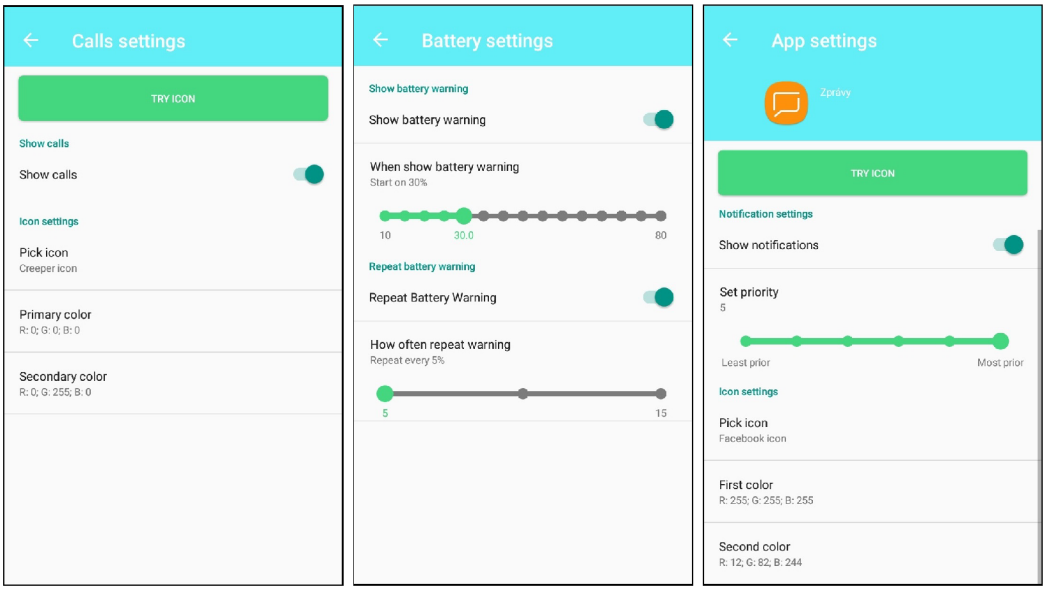
Aktivity pro nastavení

Všechny tři aktivity určené pro nastavování upozornění (tedy hovory, baterie, aplikace a zařízení) budou vypadat téměř totožně, pouze se budou lišit v některých možnostech. Uživatel si vždy bude moci nastavit, zda chce daná oznámení zobrazovat. Pokud ano, určí ikonu, která toto upozornění bude reprezentovat a jaké barvy ikona bude mít. Po kliknutí na možnost `Vybrat ikonu` se zobrazí dialogové okno (obrázek 4.7 vlevo) obsahující seznam všech dostupných ikon. Každá ikona se bude skládat ze dvou až čtyř barev (viz ukázky na obrázku 5.5). Dialogové okno pro nastavení barvy bude umožňovat uživateli vybrat si mezi třemi



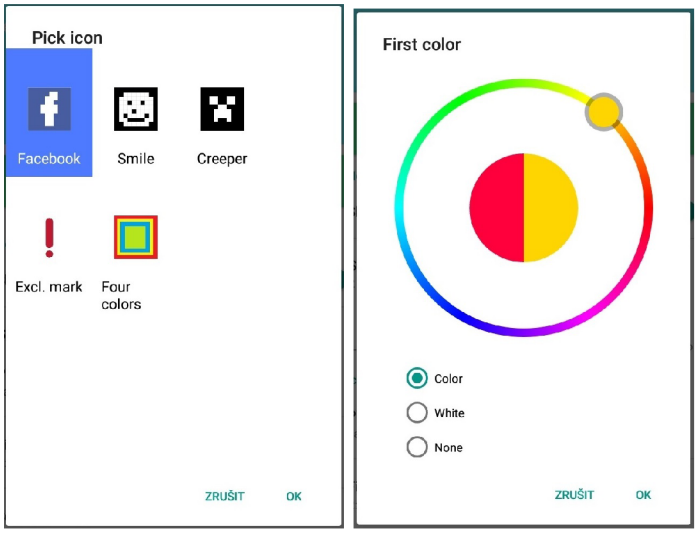
Obrázek 4.5: Aktivita seznam nainstalovaných aplikací

možnostmi — LED diody nebudou svítit, na LED diodách bude svítit bílá barva a nebo si uživatel může vybrat libovolný odstín barvy pomocí modelu HSV [12] s nastavenou nejvyšší sytostí a jasem[1] (viz obrázek 4.7 vpravo) pomocí externí knihovny `Holo ColorPicker`, kterou popisuji v kapitole 5.8. Každá aktivita (kromě nastavení baterie) bude navíc obsahovat tlačítko, pomocí kterého bude moci uživatel vyzkoušet vzhled jím nastavené ikony, která se zobrazí přímo na zařízení. V případě nastavení zařízení bude aplikace posílat přednastavenou ikonu pro demonstrování jasu. Nastavení aplikací bude navíc obsahovat možnost určení priority zobrazení jejich notifikací. V případě upozornění na stav baterie bude moci uživatel určit, při kolika procentech oznámí zařízení nízkou úroveň nabití a jak často toto upozornění bude zobrazovat. Zobrazené ikony jsou pak vybrány automaticky v závislosti na procentuálním stavu baterie (viz ukázky ikon na obrázku 5.5). Nastavení zařízení bude umožňovat pouze určení intenzity osvětlení LED diod. K dosažení přehledného layoutu pro nastavení využiji komponentu `PrereferenceScreen` (viz kapitola 2.6.1), díky které bude navíc možné snadno rozdělit jednotlivé položky nastavení do přehledných skupin.



Obrázek 4.6: Aktivity pro nastavení – vlevo nastavení hovorů, uprostřed nastavení baterie, vpravo nastavení nainstalovaných aplikací

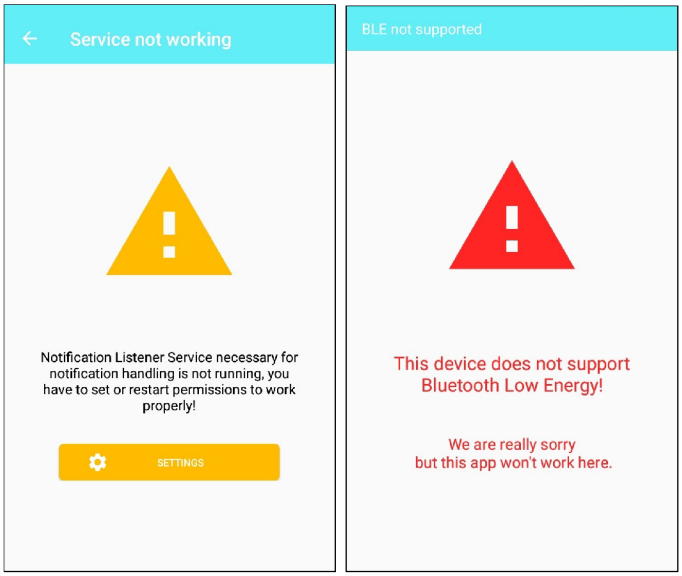
[1]Důvod, proč v aplikaci není možné nastavit sytost či jas popisuji v kapitole 4.3.1



Obrázek 4.7: Dialogová okna pro nastavení – vlevo seznam ikon zobrazovaných na zařízení, vpravo nastavení barvy ikony

Aktivita pro upozornění uživatele

Tato aktivita bude zobrazovat dvě upozornění – nefunkční NLS nebo chybějící funkci BLE na Android zařízení. Důvodem vytvoření speciální aktivity bylo, aby uživatel neměl možnost přístupu do ostatních částí aplikace, když funkce na kterých je závislý správný chod aplikace není spuštěna, respektive chybí.

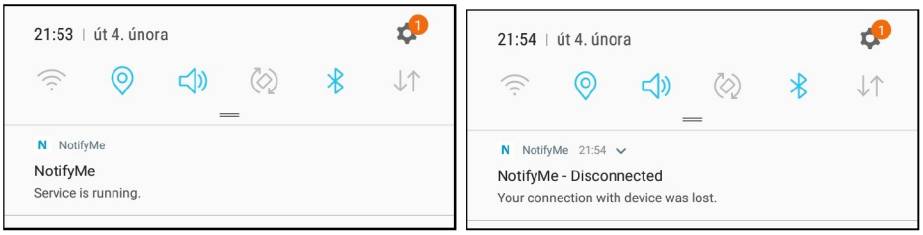


Obrázek 4.8: Aktivity s upozorněními

Notifikace a jiná upozornění

V průběhu používání aplikace se uživatel může setkat se dvěma různými notifikacemi. První z nich bude statická (nesmazatelná) notifikace oznamující uživateli, že na pozadí běží dlou-

hodobý proces – služba `ForegroundService`. Druhá notifikace se zobrazí jako upozornění v případě výpadku spojení mezi aplikací a notifikačním zařízením.



Obrázek 4.9: Ukázka notifikací

4.3 Návrh zařízení

V následující kapitole bude popsán návrh hardwarového zařízení pro indikaci notifikací.

4.3.1 Výběr komponent

Zařízení se bude skládat z následujících tří komponent:

- **Arduino UNO (klon[2])** – Vzhledem k tomu, že jsem dopředu netušil, jak moc náročný tento systém bude, rozhodl jsem se využít klon modelu Arduino UNO. Tento model je zlatou střední cestou mezi výkonem a velikostí [22]. Klony bývají zpravidla levnější než originální desky a dle recenzí na e-shopech si můžeme zjistit, zda konkrétní klon funguje, jak má. Model použitý pro tuto práci disponuje 32 kB paměti Flash a 2 kB SRAM pamětí, 14 vstupně-výstupními piny (což by mělo být dostačující) a při napájení z USB dosahuje výstupní proud hodnoty 500 mA. Dá se pořídit za 149 Kč[3].
- **Arduino Bluetooth 4.0 modul (HM-10)** – Tento modul umožňuje komunikovat s Android zařízeními od verze 4.3. Dle dokumentace má signál dosah až 60 m, spotřeba nabývá až 400 μA a jeho vstupní napětí je 3,3 V. Dá se zakoupit za 191 Kč[4].
- **RGB LED maticový displej** – Konkrétní modul který jsem vybral, je matice 10×10 LED diod. Plošný spoj se skládá z LED diod označované jako `WS2812B`. Diody ovšem nejsou mezi sebou zapojené a proto bude potřeba mezi nimi vytvořit propojky. Tento modul stál 8,39 $ tedy přibližně 193 Kč[5]. Ačkoliv existují matice s již propojenými diodami, rozhodl jsem se pořídit tuto, protože bylo mým záměrem vytvořit celé zařízení sám.

U výběru komponent bylo zapotřebí si dávat pozor na spotřebu obou modulů. V případě Bluetooth modulu to nebyl žádný problém, ovšem ne tak u RGB LED diod. Dle

[2]Společně s oficiální řadou Arduino, existuje i celá řada neoficiálních desek, tzv. klony. Protože je tato platforma open-source, může si prakticky každý vyrobit vlastní [22].

[3]Zakoupeno na: https://arduino-shop.cz/arduino/1353-klon-arduino-uno-r3-atmega328p-ch340-mini-usb.html

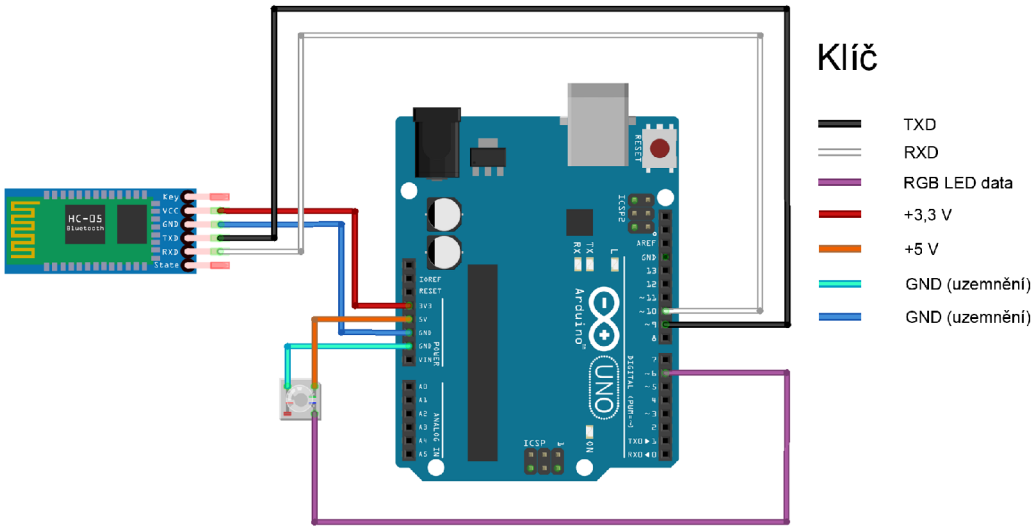
[4]Zakoupeno na: https://arduino-shop.cz/arduino/1312-arduino-android-ios-hm-10-bluetooth-4-0-ble-cc2540-cc2541-seriovy-bezdratovy-modul.html

[5]Zakoupeno na: https://www.aliexpress.com/item/32805045364.html?spm=a2g0s.12269583.0.0.283c1112NdF997

produktového listu (datasheet) diod `WS2812B` [23] má každá barevná složka odběr proudu při nejvyšším jasu až 20 mA, tedy v případě bílé barvy to činí až 60 mA. To znamená, že pro všech 100 diod může dosahovat odběr proudu až 6 A, což by při napájení z USB rozhodně nebylo dostačující. Nicméně při průběžném testování diod jsem zjistil, že maximální jas je velmi oslnivý a pro oči nepříjemný, proto tento jas snížím, což povede i k menšímu odběru proudu. Díky tomu bude mít Arduino dostatek výstupního proudu i v případě, že všechny diody budou svítit bílou barvou. Toto je i jeden z důvodů, proč aplikace neumožňuje měnit jas barvy, díky tomuto opatření je totiž možné určit maximální odběr obou modulů. Druhým důvodem je, že každá dioda by měla jiný jas, což by opět pro lidské oko nebylo příjemné.

4.3.2 Zapojení modulů

Pro sestavení celého vestavěného systému bylo nejprve nutné vymyslet schéma zapojení, to popisuje obrázek 4.10. Zapojení LED matice je znázorněno na diagramu jedinou diodou se třemi konektory – vstupní napětí 5 V, uzemnění a data ovládající barvu na diodách. Pro datový vodič je zapotřebí využít pin umožňující `PWM` (pulzně šířkovou modulaci), proto jsem vybral pin 6. Více o těchto LED diodách se zmiňuji v kapitole 5.8. U Bluetooth modulu budou zapojeny pro sériovou komunikaci dva konektory – `RXD` na pin 10 na desce Arduino a `TXD` na pin 9. Piny vyhrazeny pro sériovou komunikaci (`RX` – pin 0 a `TX` – pin 1) jsou propojeny s USB konektorem [4][6], pomocí kterého nahrávám program na tuto desku a zároveň jej využívám i pro zasílání ladících zpráv. Kdybych zapojil konektory `RXD` a `TXD` na tyto dva piny, nebyl bych schopen rozlišit, která data zaslat skrze Bluetooth modul, a která zase přes USB do počítače. A naopak kdybych chtěl nahrát novou verzi programu na desku, musel bych odpojit konektory modulu. Proto za pomocí knihovny `SoftwareSerial` (viz kapitola 5.8) budu schopen využít i jiné digitální piny pro sériovou komunikaci. Modul pak bude napájen z napájecího pinu pro 3,3 V. Pro účely testování budou všechny komponenty dočasně propojené pomocí nepájivého kontaktního pole, protože kvůli uspořádání pinů na vybraném BLE modulu jej nebylo možné přímo zapojit na desku Arduino.

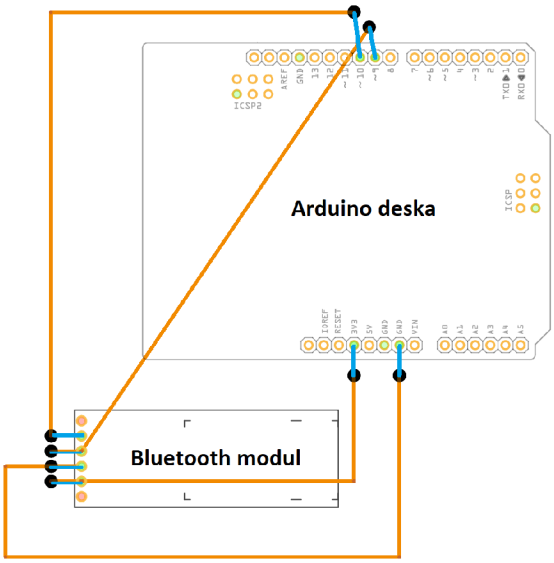


Obrázek 4.10: Diagram zapojení modulů k desce Arduino[7]

[6] https://www.arduino.cc/en/Reference/softwareSerial

4.3.3 Návrh plošného spoje

Aby bylo možné jednoduše s tímto zařízením manipulovat, rozhodl jsem se vytvořit základní desku, na které budou všechny komponenty posazeny. Diagram je na obrázku 4.11. Tato základní deska bude obsahovat jednoduché plošné spoje (reprezentované oranžovými čarami), na jejichž koncích budou vodiče připojené ke všem komponentám (reprezentované modrými čarami). LED modul bude posazen z druhé strany základní desky a připojen vodiči přímo k Arduinu. Výroba plošného spoje a výsledné zapojení jsou popsány v kapitole 5.9.



Obrázek 4.11: Diagram plošných spojů a umístění komponent na základní desce[9]. Oranžové čáry označují plošné spoje, modré čáry drátové propojky mezi plošnými spoji a piny komponent.

4.3.4 Programové vybavení

Zařízení bude v „nekonečném“ cyklu zobrazovat prolínáním na maticovém displeji až tři ikony, reprezentující nejnovější notifikace aplikací seřazené podle priority nastavené uživatelem. Po zobrazení poslední ikony následuje minutová pauza. V případě, že se začne vybíjet baterie, upozorní na tento fakt uživatele okamžitě – pokud se zobrazují ikony notifikací, bude tento cyklus přerušen a zobrazí ikonu se stavem baterie a následně bude pokračovat s ikonami aplikací. Hovory mají nejvyšší prioritu ze všech upozornění. Nehledě na to, jaké ikony se právě zobrazují, zařízení přeruší tento proces a na maticovém displeji se rozsvítí ikona nastavená pro hovory, která bude po dobu vyzvánění periodicky prolínáním mizet a znovu se objevovat, aby tak na sebe více upozornila. Po skončení vyzvánění se opět začnou zobrazovat notifikace. Samozřejmě se může stát, že i během vyzvánění nastane nějaká další událost. I tato situace je ošetřena v algoritmu. Na zařízení bude aplikace periodicky

[7]Navrženo pomocí nástroje Fritzing, viz https://fritzing.org/
[9]Navrženo pomocí nástroje Fritzing

zasílat zprávu oznamující, že stále nedošlo k výpadku. Pokud po delším čase tuto zprávu nedostane, přestane zbytečně zobrazovat notifikace.

Potencionálním problémem by mohla být nedostačující velikost vyrovnávací paměti pro sériovou komunikaci. Z návrhu komunikačního protokolu (viz kapitola 4.4) ale vyplývá, že nejdelší zpráva bude obsahovat 39 bajtů informací, proto výchozí hodnota bufferu (64 bajtů) bude dostačující.

4.4 Komunikační protokol

Základem pro fungování systému jako celku je komunikační protokol. Vzhledem k tomu, že BLE umožňuje naráz zasílat maximálně 20 bajtů, je vhodné posílat mezi sebou co nejmenší množství dat. Dalším kritériem je potřeba rozlišit jednotlivé typy zpráv (příchozí hovory, baterie, ostatní notifikace nebo například nastavení jasu), aby mohlo zařízení zobrazovat ikony dle výše zmíněných požadavků. Také je zapotřebí odlišit další druh oznamovací zprávy, jež bude sloužit ke kontrole, zda je aplikace stále připojena. Pokud po určitém čase zařízení tuto zprávu nedostane, automaticky ukončí zobrazování notifikací, protože předpokládá, že došlo k výpadku spojení.

4.4.1 Formát zpráv

Zpráva se dělí na dvě hlavní části. Informace o druhu zprávy a data zobrazované ikony. Část označující druh zprávy je zobrazena v tabulce 4.1. Tato část je dlouhá 1 bajt a dělí se na další dvě části - typ zpráv a podrobnosti. Význam jednotlivých bitů je popsán níže v tabulce 4.2.

7	6	5	4	3	2	1	0
Druh zprávy		Podrobnosti					

Tabulka 4.1: Komunikační protokol – druh zprávy

Druh zprávy	Význam
00	Hovory
01	Baterie
10	Ostatní notifikace
11	Zařízení

Tabulka 4.2: Komunikační protokol – druh zprávy význam

Formát pro zprávy obsahující informace o příchozích hovorech je popsán tabulkou 4.3. Bit *C* informuje o tom, zda-li telefon indikuje příchozí hovor, pak je bit nastaven na hodnotu jedna. Nebo již vyzvánění skončilo, pak je bit nastaven na nulu. Bit *I* říká, zda následují data pro ikonu. Vzhledem k tomu, že se ikona pro stav baterie nemusí změnit, není nutné ji znovu zasílat. Tím se ušetří čas při posílání zprávy. Díky tomu, že je tento typ zprávy nejprioritnější, rozhodl jsem jej využít také pro možnost uživatele zkusit si, jak na zařízení vypadá jím nastavená ikona. Aby ale ikona byla zobrazena staticky bez prolínání (viz kapitola 4.3.4), přidal jsem další bit *S*, který když je nastaven na hodnotu jedna, způsobí, že ikona bude staticky svítit, dokud ji uživatel sám neodstraní.

5	4	3	2	1	0
–	–	–	S	I	C

Tabulka 4.3: Komunikační protokol – podrobnosti o hovoru

V tabulce 4.4 se nachází formát podrobností pro zprávu obsahující informace o baterii. Pokud je bit C nastaven na jedničku, znamená to, že mobilní zařízení se právě nabíjí. V tomto případě není nutné tuto ikonu zobrazovat ihned. Postačí když bude ikona zobrazena až s ostatními po uplynutí minutové pauzy. Naopak pokud je bit nastaven na nulu, telefon se vybíjí, a musí se zobrazit upozornění okamžitě. Obdobně jako u hovoru bit I oznamuje, jestli následuje nová ikona. Ostatní bity jsou nevyužité.

5	4	3	2	1	0
–	–	–	–	I	C

Tabulka 4.4: Komunikační protokol – podrobnosti o stavu baterie

Tabulka 4.5 znázorňuje formát podrobností pro zprávu obsahující notifikace. Pokud je některý bit X_i v jedničce, bude se mazat ikona z dané pozice (X_1 odpovídá první zobrazované ikoně, atd.). Naopak pokud některý z bitů Y_i je nastaven na jedničku, bude se na danou pozici přidávat nová ikona.

5	4	3	2	1	0
X_0	X_1	X_2	Y_0	Y_1	Y_2

Tabulka 4.5: Komunikační protokol – podrobnosti o notifikacích

Na zařízení se bude posílat ještě jeden typ zpráv nastavující chování zařízení, který je popsán v tabulce 4.6. Zachycením zprávy `Reset` přestane zařízení zobrazovat všechny ikony. Stane se tak v případě odpojení aplikace od zařízení. Druhá zpráva slouží jako kontrola připojení (viz kapitola 4.3.4). Poslední zpráva slouží k nastavení jasu LED diod. Za touto zprávou následuje další bajt určující procentuální hodnotu jasu (hodnoty 20 % – 100 %).

Význam	**Bity 5-0**
Reset	111111
Kontrola připojení	000000
Jas	100000

Tabulka 4.6: Komunikační protokol – formát nastavení zařízení

Formát zasílané ikony je zobrazen v tabulce 4.7. Je rozdělen do tří částí – počet barev pro ikonu, barvy a `pattern` ikony. Počet barev je označen v jednom bajtu. Barvy se přenášejí ve formátu RGB, tedy 3 bajty pro barvu. Poslední část, `pattern` ikony, je navržen tak, že každé dva bity určují jednu ze čtyřech barev, která se pak namapuje na konkrétní LED diodu.

1 bajt	**6-12 bajtů**	**25 bajtů**
Počet barev	Barvy ve formátu RGB	Pattern ikony

Tabulka 4.7: Komunikační protokol – formát ikony

Kapitola 5

Implementace

V této kapitole bude popsán průběh celkového vývoje a implementace důležitých částí jak aplikace, tak i zařízení včetně použitých knihoven. Dále zde bude zmíněna problematika kompatibility mezi verzemi platformy Android, na které jsem v průběhu vývoje narazil.

5.1 Programovací jazyky a vývojová prostředí

K programování logiky uživatelské aplikace byl využit jazyk Java. Pro vývoj na operační systém Android jsem zvolil vývojové prostředí Android Studio [7], verzi 3.4.1, jelikož je pro tyto účely navrhnuto. Android Studio bylo vytvořeno kooperací firem Google a JetBrains. Je postaveno nad prostředím IntelliJ IDEA. Oproti ostatním vývojovým prostředím od JetBrains, je Android Studio dostupné zcela zdarma. Prostředí velice usnadňuje práci vývojáře nejen díky tomu, že umožňuje jednodušší psaní kódu (našeptávání, refractoring, aj.), ale také pro potřeby testování aplikace je možnost vytvořit virtuální zařízení, případně nainstalovat aplikaci přes USB kabel do fyzického mobilního zařízení. Program notifikačního zařízení je implementovaný v jazyce Wiring. Pro vývoj na desce Arduino jsem využil Arduino IDE (viz kapitola 3.2) pro jednoduchou instalaci programu na desku.

5.2 Zprovoznění Arduino klonu

Zprovozňování klonů se může lišit v závislosti na výrobci. U některých klonů (jako tomu bylo v případě desky použité pro tento projekt) stačí pouze nainstalovat na počítač ovladač pro čip `CH340`, který se právě na těchto klonech nachází. Ovladače je možné stáhnout pro operační systémy Windows, Linux i Mac. Dále pak v samotném Arduino IDE je nutné nastavit s jakou originální deskou je klon kompatibilní (Nástroje → Vývojová deska). Pro tento projekt jsem použil nastavení `Arduino/Genuino Uno`.

5.3 Databáze nainstalovaných aplikací

Jak již bylo řečeno v kapitole 4.2.1, budu používat databázi pro ukládání uživatelových nastavení včetně nabízených ikon pro notifikace a stav baterie. Ačkoliv vychází SQLite ze standardu SQL-92, je její funkcionalita dosti omezená. Kromě malého množství podporovaných datových typů neumožňuje SQLite také provádět známé příkazy, jako například `ALTER TABLE` pro úpravu tabulek [15]. Při každé změně struktury bylo tedy zapotřebí smazat a znovu vytvořit patřičnou tabulku. V případě datových typů jsem se řídil dle oficiální

dokumentace[1], která radí jak nahradit určité typy. Jediné upravené atributy jsou `is_system` a `show_notif` (významy popsány v kapitole 4.2.1), kde typ `BOOLEAN` se zobrazuje na typ `NUMERIC` následovně: `false` → `0`, `true` → `1`. Každá tabulka je implementována pomocí třídy, jejíž vlastnosti odpovídají atributům dané tabulky. Každá z tříd pak obsahuje metody pro přidávání, úpravu či získávání jednotlivých záznamů. Mazání záznamů není v tomto případě zapotřebí.

5.3.1 Udržování aktuality databáze

Při prvním spuštění této aplikace se do databáze nahrají všechny dosud nainstalované balíčky na daném chytrém zařízení. Nelze ale předpokládat, že tato aplikace bude poslední nainstalovanou, a proto se každá nová instalace bude muset detekovat. Toho je docíleno za pomocí staticky registrovaného BroadcastReceiveru v kombinaci s PackageManagerem. Díky takto registrovanému receiveru je možné ukládat záznamy do databáze i bez toho, aby byla aplikace (případně služba) spuštěna. Hodnota `package` (název balíčku aplikace) je zabalena do broadcast zprávy a následně zachycena receiverem zaregistrovaným v manifestačním souboru (viz ukázka kódu níže). Tyto zprávy jsou zasílané ale i v okamžiku, kdy dojde k aktualizaci aplikace. Tuto skutečnost je potřeba ošetřit, jinak by v databázi vznikaly duplicitní záznamy.

```
<receiver android:name=".Receivers.PackageInstalledReceiver">
    <intent-filter>
        <action android:name="android.intent.action.PACKAGE_ADDED" />
        <data android:scheme="package" />
    </intent-filter>
</receiver>
```

5.4 Služby na pozadí

Aplikace obsahuje dvě služby. První z nich je služba spuštěná a je využita pro komunikaci se zařízením pomocí Bluetooth. Přijímá broadcast zprávy od ostatních komponent aplikace a provádí určité akce, jako například připojení se k zařízení po stisknutí tlačítka v hlavní aktivitě nebo odeslání zprávy na zařízení po té, co byla zachycena některá z událostí. Kvůli tomu, že je tato služba nastavena jako `ForegroundService` (viz 2.5.2), nesmí je jednat o službu vázanou, jinak by při vypnutí aplikace nemohla nadále běžet. Proto je jediným možným způsobem využití broadcast zpráv pro komunikaci mezi částmi aplikace. Naštěstí lze použít komponentu `LocalBroadcastManager`, která dané broadcasty vysílá pouze v rámci aplikace [10][2]. Aby služba zůstala běžet i v případě ukončení aplikace uživatelem, je nutné, aby metoda `onStartCommand()` vracela hodnotu `START_STICKY` (viz kapitolu 2.5.1). Dále je zapotřebí v manifestačním souboru u deklarace služby nastavit dvě vlastnosti `andorid:enabled` (instance služby může být vytvořena systémem) a `andorid:exported` (ostatní aplikace nebo systém mohou službu spustit a přistupovat k ní) na hodnotu `true`. Podrobnější informace o komunikaci přes Bluetooth jsou v kapitole 5.6.

[1] Seznam podporovaných datových typů: https://www.sqlite.org/datatype3.html

[2] https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html

Druhá služba fungující na pozadí je potomek třídy `NotificationListenerService`. Jejím jediným úkolem je zasílat broadcast zprávy ohledně zachycení přidané/smazané notifikace výše zmíněné službě pro komunikaci. Ve zprávě se posílá název balíčku aplikace, která notifikaci vytvořila, a informaci, zda byla vytvořena nová nebo odebrána již existující.

5.4.1 Zachytávání stavu baterie, hovorů a notifikací

Zachytávání notifikací, jak již bylo zmíněno, probíhá pomocí služby NotificationListenerService. Pro získávání stavu baterie a zachytávání příchozích hovorů jsou využity staticky registrované BroadcastReceivery. Nejprve mě napadlo využít pro získávání informací o příchozích hovorech NotificationListenerService, která toto umožňuje také. Bohužel názvy balíčků aplikací zprostředkovávající telefonní hovory se liší na různých verzích Androidu. To by znamenalo, že by aplikace musela umět rozlišovat pro různé verze systému různé názvy balíčků. Jednodušším způsobem je tedy využít staticky registrovaný BroadcastReceiver, který sleduje stav hovorů. Pro tuto činnost potřebuje mít aplikace zvláštní oprávnění (viz kapitola 5.5).

5.4.2 Kompatibilita mezi různými verzemi Androidu

Jedním z telefonů, na kterém probíhalo testování je *Xiaomi Redmi 4a* s verzí API 25. Při vývoji jsem ovšem zjistil, že se služba vždy vypne společně s aplikací. Na několika navštívených fórech[3] je uvedeno, že se jedná o způsob úspory baterie na systémech vytvořených v Číně. V hlavní aktivitě stačilo přidat kontrolu na značku zařízení (v tomto případě *Xiaomi*) a odkázat uživatele do nastavení, kde je možné přidělit aplikacím právo automatického spouštění (viz ukázka kódu níže). Většině nově nainstalovaným aplikacím je tato možnost odepřena a uživatelé musí tato oprávnění přidělit manuálně. Bohužel toto řešení nebylo pro všechny telefony této značky funkční. Podobně jako u NotificationListenerService to může být pro uživatele rušivý element, ale takto je zajištěna větší podpora aplikace.

```
if (Build.BRAND.equalsIgnoreCase("xiaomi")) {
    Intent intent = new Intent();
    intent.setComponent(new ComponentName("com.miui.securitycenter",
    "com.miui.permcenter.autostart.AutoStartManagementActivity"));
    startActivity(intent);
}
```

5.5 Použitá oprávnění

Pro získávání takto citlivých informací musí mít aplikace přidělená oprávnění od uživatele. Přidělování oprávnění pro službu NotificationListenerService je blíže popsáno v kapitole 2.4.3, pro BLE v kapitole 2.3.2. Oprávnění pro zachytávání hovorů je vloženo v manifestačním souboru společně s oprávněním používat `ForegroundService`, jak je ukázáno níže. Tato oprávnění jsou vyžadována po uživateli v hlavní aktivitě (viz kapitola 2.4.1). Pro získávání stavu baterie nejsou potřeba žádná zvláštní oprávnění.

[3] Odkaz na jedno z navštívených fór: https://www.reddit.com/r/androiddev/comments/9ra0iq/workmanager_reliability_for_periodic_tasks_on/

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
```

5.6 Komunikace přes Bluetooth

V této kapitole se zaměřím na řešení komunikace prostřednictvím Bluetooth včetně problémů, na které jsem narazil v průběhu vývoje na různých zařízeních.

5.6.1 Skenování zařízení a následné připojování

Před zahájením komunikace je nutné zjistit všechna dostupná zařízení v okolí pomocí skenování. Aby však aplikace nalezla správná zařízení, bude nejefektivnější zaměřovat se na jejich názvy. Využije se tedy filtrovaného skenování. Modul bude nutné přejmenovat pomocí příkazu `AT+NAME` (viz kapitola 3.3.1). Skenování nemá žádný vestavěný mechanismus na automatické ukončení, proto bylo nutné naplánovat opožděnou úlohu, která skenování po určité době ukončí. Toho je docíleno pomocí metody `Handler.postDelayed()`, která naplánuje vytvoření objektu typu `Runnable`. Ten obsahuje abstraktní metodu, která je zavolána automaticky po vytvoření nového vlákna, ve kterém se dané operace provedou.

Pro připojení se na zařízení je potřeba mít jeho MAC adresu. Po té, co skončí skenování, vybere si uživatel jedno z dostupných zařízení zobrazených v seznamu (viz kapitola 4.2.2) a aplikace si jeho MAC adresu uloží pomocí SharedPreferences. Pokud se bude uživatel v budoucnu znovu připojovat k zařízení, nebude nutné hned provádět skenování, protože se aplikace prvně pokusí navázat spojení s naposledy připojeným zařízením. Jakmile se podaří úspěšně k zařízení připojit (indikováno pomocí callbacku), pokusí se aplikace najít službu a její charakteristiku (viz kapitola 2.3.1), která bude využívána pro přenos dat na zařízení. Využitý BLE modul poskytuje službu pro sériovou komunikaci a charakteristiku pro zasílání dat na modul. Služba i charakteristika jsou pro všechny moduly tohoto typu identifikovány pomocí stejných `UUID`, proto je možné tyto hodnoty „natvrdo“ napsat do kódu aplikace. Nejprve je potřeba nalézt službu a až díky ní její charakteristiku. V případě, že se úspěšně podaří najít i charakteristika pomocí metody `BluetoothGattService.getCharacteristic(UUID uuid)`, vrátí tato metoda objekt, který se bude využívat pro zasílání dat. Na některých zařízeních se ovšem může stát (viz kapitola 6.2), že požadovanou službu (a tedy i charakteristiku) nemusí aplikace najít hned po úspěšném připojení se k zařízení. Jinými slovy pokud v callbacku v případě úspěšného připojení, bude zavolána metoda pro nalezení služby, nemusí toto řešení vždy fungovat. Na různých fórech[4] jsem ovšem nalezl řešení, které spočívá v opožděném zavolání metody pro vyhledání dané služby. Toto řešení se ukázalo jako funkční. Další rozdíl mezi testovanými zařízeními je v čase, ve kterém je volán callback v případě neúspěšného připojení se k zařízení. U některých zařízení trvalo až půl minuty, než bylo zaznamenáno neúspěšné připojení. Proto, aby uživatel nečekal zbytečně dlouho, vyřešil jsem tento problém opět pomocí metody `postDelayed()`. Po určitém čase je v nově vytvořeném vlákně zkontrolován stav připojení. Pokud po dané době, není navázáno spojení mezi aplikací a zařízením, bude tento fakt oznámen uživateli.

[4] Jedno z navštívených fór: https://stackoverflow.com/questions/41434555/onservicesdiscovered-never-called-while-connecting-to-gatt-server

5.6.2 Zasílání dat na zařízení

Jak jsem uvedl v kapitole 2.3.1, výchozí velikost jednotlivých packetů v případě technologie BLE je 20 batjů a ne všechen hardware podporuje zvýšení MTU (včetně telefonů) [9][5]. Dle dokumentace mnou vybraný modul tuto možnost neposkytuje. Také protože nelze předpokládat, že každý telefon, na kterém aplikace bude běžet, bude umožňovat změnu MTU, rozhodl jsem vždy posílat data po packetech délky 20 bajtů. Po odeslání dat na zařízení, je zavolán callback označující stav přenosu dat (metoda `onWriteCharacteristics()`). Pokud přenos packetu proběhl v pořádku, odešle aplikace následujících 20 bajtů zprávy, v případě neúspěšného přenosu se pokusí odeslat packet znovu. Protože může nastat situace, že přijde nová notifikace v době, kdy již probíhá přenos dat, použil jsem semafor, který vždy pozastaví nově vytvořené vlákno do doby, než budou data odeslána.

5.6.3 Synchronizace dat aplikace se zařízením

Aby aplikace měla přehled o všech právě zobrazovaných notifikacích, vytvořil jsem třídu `NotificationsList`. Ta obsahuje seznam s důležitými informacemi o notifikacích načtených z databáze – prioritu, obrazec ikony včetně barev, název balíčku a informaci, zda-li se bude notifikace na zařízení posílat nebo naopak se bude ze zařízení mazat. Dále jsou v ní implementovány dvě důležité metody. První z těchto metod přidává do se seznamu novou položku s již zmíněnými informacemi a zároveň všechny notifikace v seznamu řadí automaticky dle priorit. V případě, že nově příchozí notifikace bude umístěna na jedné z prvních tří pozic seznamu, aplikace u této notifikace nastaví informaci, že se bude příslušná ikona posílat na zařízení. Druhá metoda slouží k mazání notifikací. Pokud NLS oznámí odebrání notifikace ze stavového řádku, nastaví metoda informaci o smazání této notifikace ze zařízení. Jestliže bude nějaká notifikace odebrána a zároveň jsou v seznamu více než-li tři položky, bude první následující notifikace v seznamu označena k odeslání na zařízení. Aplikace následně vygeneruje dle tohoto seznamu zprávu, ve které budou nastaveny příslušné bity dle popisu v kapitole 4.4.1.

5.6.4 Řešení výpadků spojení

V průběhu spojení může dojít k výpadkům spojení ať už vlivem oddálení se telefonu od zařízení, odpojení zařízení od zdroje napájení, nebo v případě poruchy BLE modulu. Už při zadávání požadavku o připojení k zařízení pomocí metody `BluetoothDevice.connectGatt()` je této metodě zadán parametr pro automatické znovu-připojování, tudíž je tento problém zajištěn samotným systémem. Ačkoliv nebyl na žádném testovaném zařízení s tímto řešením žádný problém, rozhodl jsem se na základě dosavadních zkušeností přidat navíc ošetření v případě, že by automatické připojování nebylo na některé verzi systému funkční. Po odpojení zařízení (indikováno opět callbackem) zavolá aplikace metodu `postDelayed()`. Po určitém čase se spustí kód pro nový pokus o připojení. Pokud se připojení nepovede ani napotřetí, oznámí aplikace uživateli ztrátu spojení.

Další situací, při které se může dojít ke ztrátě připojení, je restartování služby na popředí (viz kapitola 5.4). Pokud jsou již zaznamenány nějaké notifikace v objektu typu `NotificationsList`, budou tato data ztracena a aplikace nebude vědět, které notifikace jsou na zařízení zobrazeny. Proto jsem využil knihovnu Gson (viz kapitola 5.8), která seznam převede do formátu JSON uložený v proměnné datového typu `String`. Seznam je v této podobě průběžně ukládán pomocí SharedPreferences. Po restartování služby

[5] https://github.com/don/cordova-plugin-ble-central/issues/234

pak jen aplikace opět pomocí knihovny Gson převede tento řetězec zpět na seznam typu `NotificationsList`.

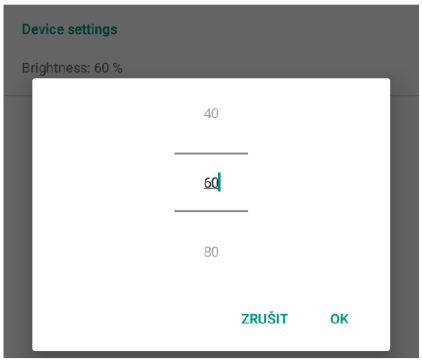
5.7 Uživatelské rozhraní

V této části textu se zaměřím na implementační detaily důležitých částí grafického uživatelského rozhraní.

5.7.1 Vlastní PreferenceFragmenty

Co je `PreferenceFragment` a `PreferenceScreen`, jsem se již zmiňoval v kapitole 2.6.1. Platforma Android sama o sobě nabízí některé předprogramované fragmenty. Jedná se například o `EditTextPreference`, který otevírá dialogové okno s komponentou pro úpravu textových dat, nebo `SwitchPreference`, který obsahuje dvoustavový přepínač. Ten je v aplikaci využit pro nastavení, zda uživatel chce zobrazovat na zařízení konkrétní upozornění. Ovšem kvůli některým požadavkům na aplikaci bylo zapotřebí vytvořit vlastní fragmenty, aby se daly jednoduše zakomponovat do layoutu `PreferenceScreen`.

Pro každý fragment byla vytvořena jedna třída. Každá z nich je potomkem buď třídy `Preference`, nebo `DialogPreference`. V případě `DialogPreference` je komponenta (případně komponenty) určená pro změnu hodnot nastavení zobrazena v dialogovém okně. V této aplikaci jsou využity tři vlastní PreferenceFragmenty. `ColorDialogPreference` vytváří prvek pro nastavování barev ikon (viz obrázek 4.7 vpravo). Dalším vlastním fragmentem je `ImageDialogPreference`. Ten slouží k přiřazování ikon jednotlivým upozorněním (viz obrázek 4.7 vlevo), přičemž využívá upravené komponenty `RadioGroup` (více viz následující kapitola 5.7.2). Poslední fragment slouží k vybírání číselných hodnot. Ten je využit v případě určování priority notifikací, procentuálního stavu baterie, při kterém se má zobrazit upozornění, nebo také nastavení jasu LED diod. První komponentou k určení dané číselné hodnoty byla `NumberPicker` (viz obrázek 5.1). Ta ale byla po testování uživatelského rozhraní nahrazena komponentou `SeekBar` (více o testování viz kapitola 6.3). Protože je fragment použit pro různé typy nastavení s různými rozsahy číselných hodnot, přidal jsem tomuto objektu tři vlastnosti – minimální hodnotu, maximální hodnotu a velikost kroku. Hodnoty těchto vlastností jsou pak vyplněny v kódu layoutu. Díky tomu stačí mít pouze jeden předpis tohoto fragmentu, který je v daném layoutu upraven dle potřeby.



Obrázek 5.1: Komponenta `NumberPicker` určená pro vybírání číselných hodnot

5.7.2 Vlastní RadioGroup a RadioButton

Při nastavování upozornění uživatel vybírá jednu z nabízených ikon, která bude toto upozornění reprezentovat na notifikačním zařízení (viz obrázek 4.7 vlevo). Pro tyto účely se hodí použít komponentu `RadioGroup` respektive `RadioButton`. Bohužel Android nabízí pouze jeden výchozí vzhled jednotlivých položek `RadioButton`. Aby měl uživatel lepší představu o vzhledu ikony, vytvořil jsem vlastní vzhled těchto položek[6]. Každé `RadioButton` obsahuje obrázek přibližné podoby ikony včetně jejího názvu. Vybraná ikona od ostatních se odlišuje rozdílnou barvou pozadí.

5.7.3 Seznamy aplikací a zařízení

Aplikace obsahuje celkem 3 seznamy – 2 pro zobrazení nainstalovaných aplikací v telefonu (uživatelské a systémové) a seznam dostupných BLE zařízení při skenování. Jelikož všem položkám obou druhů seznamů bylo vytvořeno vlastní rozložení, bylo nutné implementovat i adaptéry. Ten zajišťuje mapování dat objektů v seznamu na prvky layoutu. Všechny položky seznamu jsou řazeny kvůli větší přehlednosti. Seznam s aplikacemi je řazen podle abecedy a seznam dostupných zařízení podle síly signálu (hodnota RSSI) sestupně. Protože informace o nainstalovaných aplikacích jsou uloženy v databázi, je využito pro řazení dat jazyka SQL. V případě dostupných BLE zařízení jsem využil metodu `Collections.sort()`. Kromě samotného seznamu je jejím argumentem i komponenta `Comparator`. Ta určuje, podle které vlastnosti se mají objekty v daném seznamu řadit. Ukázka seznamu je na obrázku 4.2.2.

5.8 Použité externí knihovny

V následující kapitole budou uvedeny všechny knihovny, které jsou využívány pro tuto práci. Jedná se převážně o knihovny zkrášlující uživatelské rozhraní.

SoftwareSerial

Použití této knihovny umožňuje využít jakékoliv digitální piny na platformě Arduino pro sériovou komunikaci [4][7]. Jedná se v podstatě o programovou replikaci fukncionality UART hardwaru. Je možné vytvořit hned několik sériových kanálů, ale pouze jediný z těchto kanálů může přijímat data v daném časovém úseku. U tohoto projektu to není překážkou, protože knihovnu využivám pouze u komunikace mezi Arduino deskou a Bluetooth modulem.

NeoPixels

Pro zjednodušení práce s diodami `WS2812B` navrhla společnost Adafruit Industries knihovnu NeoPixels[8]. Knihovna využívá techniku pulzně šířkové modulace (`PWM`) k ovládání jasu a tudíž i výsledné barvy diody. Jednotlivé barevné složky (RGB) LED diody jsou velmi rychle vypínány a znovu zapínány. Poměr časů, kdy jsou tyto složky vypnuté a zapnuté, nazýván jako střída signálu, pak vyvolává v lidském oku dojem, že dioda svítí s nižším

[6] Inspirováno návodem dostupným z https://crosp.net/blog/software-development/mobile/android/creating-custom-radio-groups-radio-buttons-android/

[7] https://www.arduino.cc/en/Reference/softwareSerial

[8] https://learn.adafruit.com/sipping-power-with-neopixels/insights

jasem. Změna jasů u jednotlivých barevných složek pak vytváří výslednou barvu LED diody. Knihovna byla stažena za pomocí Arduino IDE.

BubbleSeekBar

Tato knihovna přidává esteticky navrženou komponentu `SeekBar`. Kromě změny barvy posuvné části je možné také rozdělit tuto komponentu na sekce pro zvýšení přehlednosti. Jednotlivé sekce lze také označit číselným nebo textovým popiskem. V aplikaci je využita pro nastavení priority zobrazení ikon a procentuální stav baterie, při kterém se má dané upozornění zobrazit (viz obrázek 4.6). Více o této knihovně na stránkách GitHub[9].

Adroid Holo ColorPicker

Jedná se o knihovnu poskytující prvek pro jednoduchý výběr barvy podle modelu HSV. Jednotlivé složky barvy (odstín, jas a sytost) je možné nastavit separátními komponentami, které mohou a nemusí být využity. V aplikaci je využita pouze komponenta pro změnu odstínu barvy z důvodů uvedených v předchozích kapitolách. Ukázka komponenty je na obrázku 4.7 vpravo. Více o této knihovně na GitHub[10].

Android-SpinKit

Jedná se o knihovnu, která umožňuje přidat prvek uživatelského rozhraní oznamující uživateli, že právě probíhá nějaká operace – `ProgressBar`. Knihovna nabízí spoustu různých animací a tvarů, kterým je možné měnit barvu. Více o této knihovně Android-SpinKit na GitHub[11].

Android Ripple Background

Knihovna nabízí možnost přidat pulzující animaci na pozadí, přičemž je možné přizpůsobit si barvu vln, rychlost vlnění i jejich počet. Tuto komponentu jsem využil v hlavní aktivitě, kde indikuje uživateli, že probíhá připojování k notifikačnímu zařízení. Více o této knihovně na GitHub[12].

Android Toggle

Android Toggle přidává do uživatelského rozhraní přepínač, kterému je možné určit barvu i text pro stavy vypnuto/zapnuto. Tento prvek využívám v hlavní aktivitě pro manuální vypínání/zapínání ForegroundService (viz obrázek 4.2). Knihovna je dostupná ze stránky GitHub[13].

Gson

Tato knihovna umožňuje serializaci/deserializaci Java objektů do/z formátu JSON. Výhodou oproti podobným knihovnám je ta, že není zapotřebí psát do tříd žádné anotace, což

[9] https://github.com/woxingxiao/BubbleSeekBar
[10] https://github.com/LarsWerkman/HoloColorPicker
[11] https://github.com/ybq/Android-SpinKit
[12] https://github.com/skyfishjy/android-ripple-background
[13] https://github.com/Angads25/android-toggle

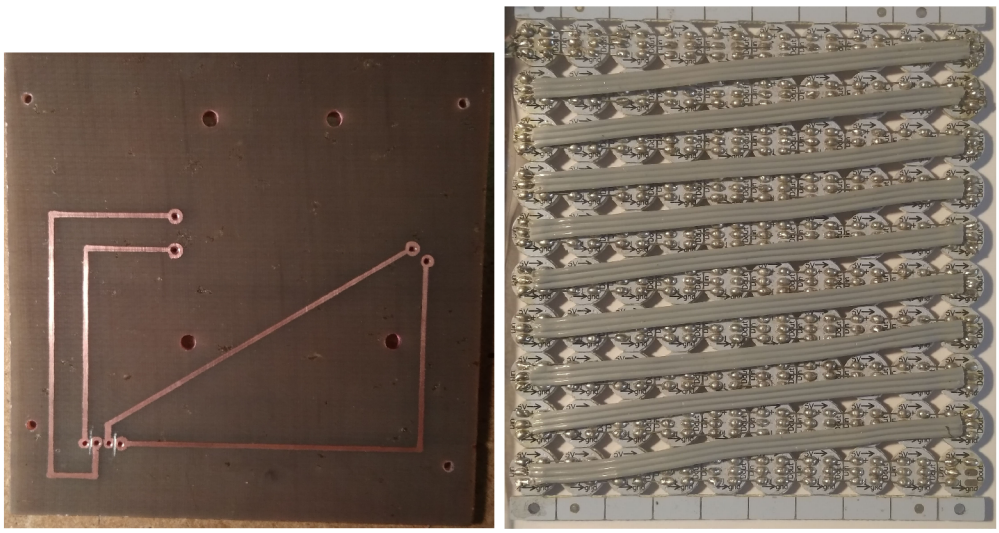
je vhodné v případě, že vývojář nemá ke zdrojovému kódu přístup. Více o knihovně na stránkách GitHub[14].

5.9 Vytvoření plošného spoje a finální zapojení

Jak jsem se již zmínil v kapitole 4.3.3 kvůli jednoduchému přenášení zařízení jsem umístil všechny moduly na jednu základní desku vytvořenou z cuprextitu, jež obsahuje plošné spoje. Tyto spoje slouží k přehlednému propojení mezi komponenty. Využil jsem metodu samostatných spojů [18] pro vytvoření této desky. Metoda spočívá v tom, že se pomocí lihové barvy nakreslí na cuprextit jednotlivé propoje mezi součástkami (viz návrh 4.11). Po zaschnutí barvy se nechá neoznačená měď odleptat v lázni chloridu železitého – $FeCl_3$ (viz obrázek 5.2). Propojení plošných spojů a komponent je tvořeno drátovými propojkami. Finální verze zařízení je na obrázku 5.3. Pro toto zařízení jsem navrhl pouzdro, které bylo vytištěné na 3D tiskárně (viz obrázek 5.4).

5.10 Zobrazování ikon na matici

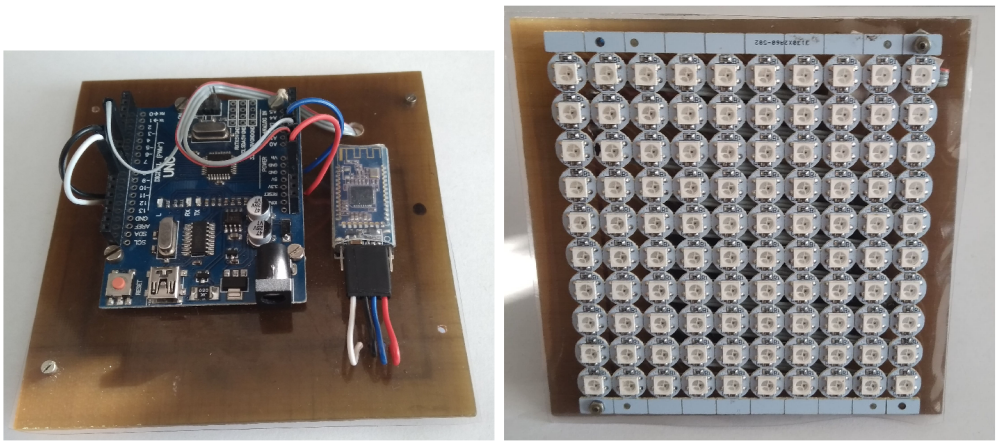
Pro zjednodušení práce s použitými diodami jsem využil knihovnu NeoPixels (viz 5.8). Ačkoliv jsou LED diody posazeny do maticového uspořádání, tak datová linka je propojena do série, a tudíž pro adresování jednotlivých diod slouží indexy, které v tomto případě mohou nabývat hodnot 0-99 (viz obrázek 5.2). Aby byl obrazec zřetelný, vytvořil jsem mřížku, která světlo z každé diody rozprostře v buňce, ve které se dioda nachází, což připomíná rastrový obrázek (viz ukázka na obrázku 5.5). Jak již bylo uvedeno v kapitole 4.3.1, nesvítí diody s maximálním možným jasem z důvodu nedostatečného výstupního proudu Arduina. U tohoto zařízení jsem nastavil maximální jas na zhruba $7,8\,\%$ maximálního možného jasu[15], což se ukázalo jako dostačující.



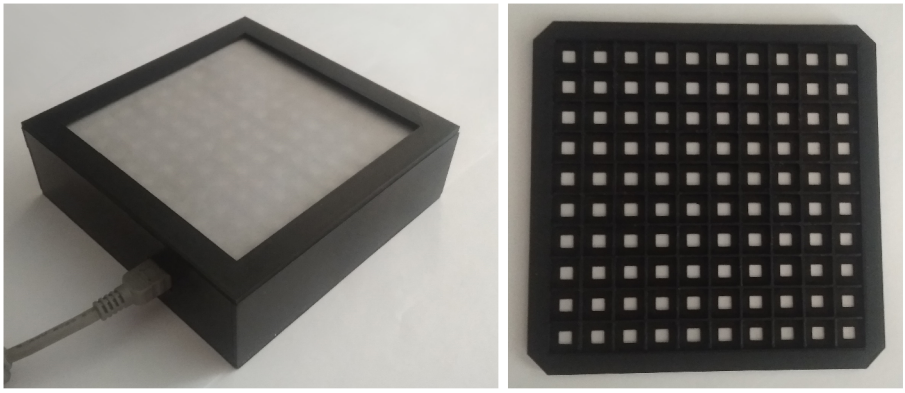
Obrázek 5.2: Vlevo hotové plošné spoje, vpravo propojky mezi LED diodami

[14] https://github.com/google/gson

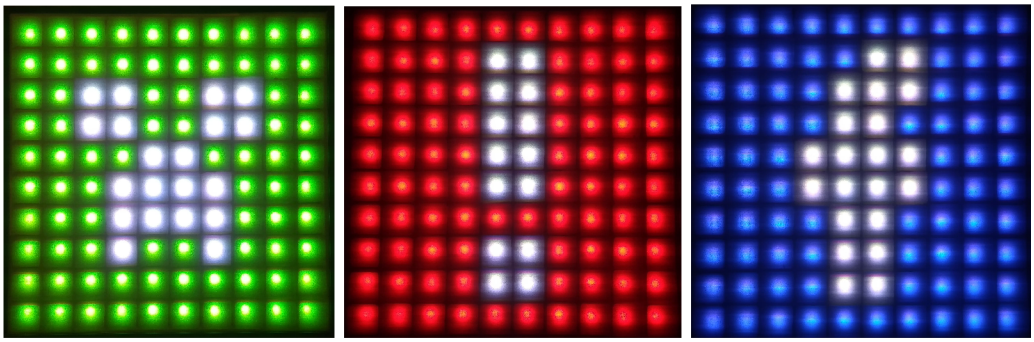
[15] Maximální hodnota jasu, která lze za pomocí knihovny Neopixel nastavit je 255, v případě tohoto zařízení je hodnota nastavena na 20.



Obrázek 5.3: Finální zapojení zařízení



Obrázek 5.4: Vlevo vytištěné pouzdro pro zařízení, vpravo mřížka, která rozprostře světlo z LED diod do čtvercových buněk



Obrázek 5.5: Ukázka zobrazených ikon na displeji

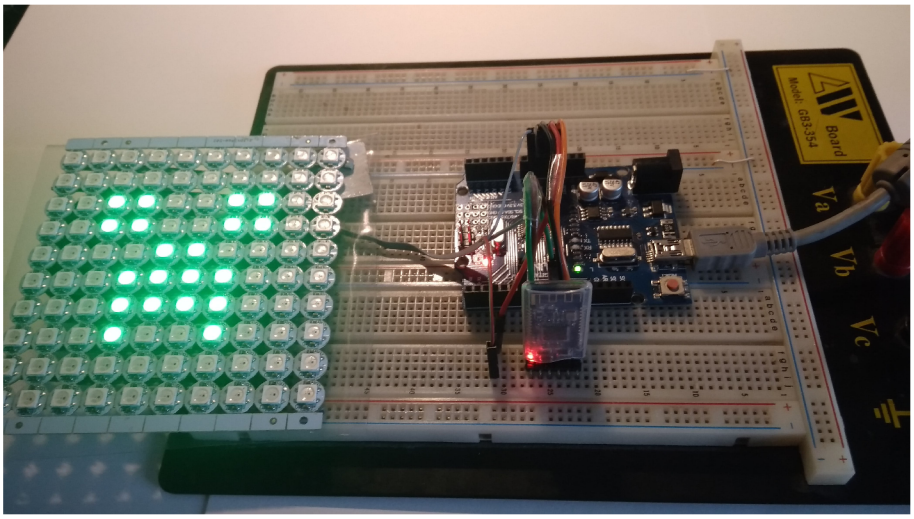
Kapitola 6

Testování

V této kapitole je shrnut postup testování při implementaci aplikace i notifikačního zařízení. Jsou zde vyhodnoceny zjištěné výsledky a zmíněny i dodatečné opravy.

6.1 Průběžné testování funkcionality

Zpočátku jsem testoval jednotlivé hlavní součásti projektu zvlášť, které jsem v pozdějších fázích vývoje spojil v jeden funkční celek. Nejprve jsem začal se zapojením zařízení na nepájivém poli (viz obrázek 6.1) včetně testování knihoven použitých pro oba moduly. K testování komunikace mezi Bluetooth modulem a Android zařízením jsem využil aplikace `Serial Bluetooth Terminal`[1], která umožňuje připojit se na Bluetooth (i BLE) zařízení a pomocí terminálu zasílat data v různém formátování (text, hexa, aj). Funkcionalitu a zachytávání notifikací pomocí NLS jsem testoval pomocí aplikace `Push Notification Tester`[2].



Obrázek 6.1: Testování zapojení zařízení na nepájivém poli

[1] Aplikace dostupná z Google Play: https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal

[2] Aplikace dostupná z Google Play: https://play.google.com/store/apps/details?id=com.firstrowria.pushnotificationtester

6.2 Testování kompatibility na různých verzích Androidu

V této kapitole srovnám všechny testované smartphony a zmíním problémy, na které jsem při jejich testování narazil, a jakým způsobem byly vyřešeny, pokud to bylo možné.

- **Xiaomi Redmi 4a (API 25)** – Jednalo se o první zařízení, na kterém jsem aplikaci testoval. Zde se mi bohužel nepodařilo, aby služba na popředí fungovala když uživatel aplikaci vypne i s využitím opatření pro telefony této značky zmíněné v kapitole 5.4.2. Stejně tak i přestane fungovat NLS. V případě aktualizace aplikace bylo nutné vždy odebrat a znovu přidělit oprávnění k NLS, jinak tato služba nemohla být využívána. U tohoto zařízení trvalo téměř půl minuty, než bylo zaznamenáno neúspěšné připojení se k notifikačnímu zařízení (řešení popsáno v kapitole 5.6.1). Jestliže ale zůstane aplikace zapnutá, vše ostatní funguje, jak by mělo.

- **Xiaomi Redmi 4a (API 23)** – Stejně tak jako u předchozího zařízení se stávalo, že služba NLS přestala fungovat v případě nainstalování nové verze. Dalším velkým problémem bylo, že pokud aplikace spadla vlivem chyby při vývoji, nepomohlo ani restartování a přeinstalace aplikace. Jediným řešením bylo změnit název balíčku aplikace. Takovéto chování je nejspíše způsobené nadstavbou MIUI[3]. U ostatních testovaných zařízení v tomto případě stačilo při spadnutí aplikace restartování. Naopak oproti stejnému modelu s verzí API 25, bylo možné udržet službu spuštěnou po té, co byla této aplikaci přidělena práva pro automatické spouštění (viz kapitola 5.4.2). Na jiné problémy jsem během testování nenarazil.

- **Xiaomi Redmi 7a; Xiaomi Redmi Note 5 (API 28)** – Kvůli rozdílnému chování spuštěných služeb u verzí API 23 a 25, jsem se rozhodl vyzkoušet tuto aplikaci i na novějších telefonech značky *Xiaomi*. Testování ukázalo, že při přidělení práv k automatickému spouštění, služba zůstane běžet i v případě, že již žádná z aktivit aplikace není spuštěna. Zřejmě tedy nemožnost ponechat službu spuštěnou byla způsobena systémovou chybou dané verze. Ostatní části aplikace fungovaly dle očekávání.

- **Doogee X5 Pro (API 22)** – U tohoto zařízení se mi opět nepodařilo, aby ForegroundService fungovala i bez zapnuté aplikace. Paradoxně však služba NLS zůstala běžet nadále (otestováno ladícími zprávami v Android Studiu). Veškeré ostatní funkce byly dle očekávání.

- **Zopo C2 (API 23)** – V případě tohoto smartphonu nebyl žádný problém, co se týče služeb aplikace. Naopak jsem zjistil, že aplikace nebyla schopna najít služby a charakteristiky BLE modulu hned po připojení se. Řešení problému je uvedeno v kapitole 5.6.1.

- **Samsung Galaxy J5 2016 (API 25)** – U tohoto modelu jsem při testování nenašel žádné nedostatky, a je to jedno ze dvou testovaných zařízení, u kterých by žádná implementovaná opatření nebyla potřeba. Proto samotný vývoj probíhal převážně na tomto zařízení. Ostatní zařízení byla využita především pro ladění chyb vzniklých rozdílnými verzemi operačního systému.

- **Lenovo A2010 (API 22)** – Ani u tohoto zařízení se během testování neobjevily žádné problémy.

[3]Jedno z navštívených fór ohledně problematiky NLS: https://c.mi.com/thread-2043797-1-0.html

V případě ForegroundService jsem se ještě pokusil využít jiný přístup restartování služby, než který je popsán v kapitole 5.4. V této službě jsem implementoval abstraktní metodu `Service.onTaskRemoved()`, která je zavolána automatiky v případě, že uživatel ukončí aplikaci ve správci úloh. V těle metody byla umístěna komponenta `AlarmManager` [10][4], pomocí které se naplánovalo spuštění této služby systémem jednu vteřinu po ukončení aplikace. Toto řešení způsobí, že po ukončení aplikace bude služba vždy vypnuta a následně spuštěna v novém procesu, na rozdíl od předchozího řešení, díky kterému služba zůstane běžet bez restartu. Bohužel u zařízení, na kterých nebylo možné zprovoznit běžící službu bez aplikace původním řešením, ani tato alternativa nepomohla. Proto jsem se rozhodl původní řešení ponechat.

6.3 Testování na uživatelích

V této části textu se zmíním o testování layoutu uživateli a jeho následné úpravě, a také o požadavcích uživatelů na přidání nových funkcí do aplikace/zařízení.

V průběhu vývoje jsem testoval zařízení na šesti uživatelích. Jelikož mám ale pouze jediné zařízení, nebylo možné provádět dlouhodobé testování na více uživatelích současně. Každé testování probíhalo tak, že jsem nejprve vysvětlil uživateli, co tento vestavěný systém umí a následně jej nechal seznámit se z uživatelským rozhraním aplikace. Po té jsem dával jednoduché úkoly jako připojit se na zařízení nebo změnit nastavení pro jednotlivá upozornění. Po upravení všech nedostatků probíhalo testování druhé.

K otestování kompatibility aplikace jsem navíc využil mobilních telefonů testovaných uživatelů (výsledky testování jsou popsány v předchozí kapitole 6.2).

Tlačítko pro připojení/odpojení

Prvním majoritním problémem původního návrhu grafického uživatelského rozhraní bylo, že uživatelé nevěděli, jak se k zařízení připojit. Ikona, která nyní indikuje stav připojení k zařízení, fungovala původně i jako tlačítko pro manuální připojení/odpojení se k/od zařízení (viz obrázek 4.2). Toto řešení se ukázalo jako nepoužitelné, protože žádný z testovaných uživatelů nebyl schopen poznat, že se jedná o tlačítko. Proto jsem nakonec do layoutu přidal pro tyto účely separátní tlačítko, jak je zobrazeno v konečném návrhu.

Výběr číselných hodnot

Druhá změna grafického uživatelského rozhraní se týkala výběru číselných hodnot v případě nastavení priority notifikací a procentuálního stavu baterie, při kterém se pošle upozornění na zařízení. Původní komponenta pro výběr čísel byla `NumberPicker`, která se zobrazovala v dialogovém okně. Dle některých testerů by bylo lepší spíše využít posuvník (`SeekBar`), který nakonec v konečném návrhu layoutu zůstal (viz obrázek 4.6).

Odkaz pro SMS aplikaci

Dalším přáním některých uživatelů bylo přidaní odkazu pro nastavení SMS aplikace do hlavní aktivity. Většinou se totiž jedná o systémovou aplikaci, a tudíž by mohlo být zdlouhavé ji mezi ostatními najít.

[4] https://developer.android.com/reference/android/app/AlarmManager

Zkoušení vzhledu nastavené ikony

Jedním z požadavků na funkcionalitu celkového systému bylo, aby si uživatelé mohli vyzkoušet vzhled ikony, která se bude zobrazovat na maticovém displeji. Více o této možnosti v kapitole 4.2.2.

Jas LED diod

Druhým žádaným rozšířením byla možnost upravit jas LED diod, protože se některým uživatelům stále připadaly diody velice jasné, i přes to, že byly sníženy na téměř 8 % své maximální svítivosti.

Druhé testování a shrnutí

Po úpravách dle výše zmíněných nedostatků a požadavků proběhlo druhé testování. Všichni uživatelé již byli schopni se bez pomoci na zařízení připojit. Dále také přidání odkazu na nastavení SMS aplikace byla velmi vítanou změnou. Největší ohlasy u uživatelů však získala možnost vyzkoušet vzhled jimi nastavených ikon.

Testování proběhlo veskrze pozitivně, protože pomohlo odhalit nedostatky a zároveň přineslo některá vylepšení pro celý tento vestavěný systém.

Kapitola 7

Závěr

Cílem této práce bylo navrhnout a implementovat zařízení pro zobrazování upozornění včetně aplikace pro telefony, která tato upozornění zachytává a zasílá na zařízení pomocí bezdrátové komunikační technologie Bluetooth. Dle výsledků testování byl tento cíl splněn. Aplikace byla testována na různých modelech smartphonů s různými verzemi operačního systémů Android. Některé chyby, které byly způsobeny ať už právě konkrétní verzí operačního systému, nadstavby nebo hardwarovou implementací telefonu, byly ošetřeny a aplikace, a tudíž i celý tento vestavěný systém, je tak dostupnější pro více uživatelů.

Nejprve bylo nutné nastudovat možnosti bezdrátové komunikace mezi telefonem a malým hardwarovým zařízením. Pro tyto účely byla vybrána technologie Bluetooth Low Energy, u které na rozdíl od NFC není zapotřebí uživatelovy součinnosti při zasílání dat, proto je vhodná pro podobně fungující autonomní systém. Navíc je zaměřena na úsporu baterie, což je při dlouhodobém používání nesporná výhoda. V případě aplikace bylo nutné nastudovat možnosti zachytávání různých událostí, čehož bylo dosaženo pomocí komponent `BroadcastReceiver` a `NotificationListenerService`. Veškerá komunikace skrze technologii Bluetooth včetně správy spojení byla implementována pomocí služby na popředí kvůli tomu, aby spojení mezi telefonem a zařízením vydrželo po co nejdelší možnou dobu, a to i v případě, že žádná z aktivit aplikace není zapnuta. Pro samotné zařízení byla vybrána platforma Arduino společně s dvěma moduly – Bluetooth modul a RGB LED matice. Všechny komponenty byly posazeny na základní desku vyrobené z cuprextitu, která obsahuje plošné spoje propojující Arduino s Bluetooth modulem. Testování bylo zaměřeno převážně na jednoduchost používání, a v případě aplikace navíc zajištění podpory na různých modelech telefonů a verzích systému Android. Bohužel jsem však narazil na případy, kdy ani s různými ošetřeními nebylo možné na delší dobu udržet službu v chodu, především kvůli striktním omezením systému šetřící baterii telefonu.

Při vývoji na platformě Android mě nejvíce zklamaly, i když ne moc překvapily, rozdíly v implementaci pro různé verze tohoto operačního systému. Celkově však pro mě byl tento projekt velmi přínosným, naučil jsem se více pracovat na nižších vrstvách Androidu, a také jak navrhnout a sestavit hardwarové zařízení. Budoucí práce na tomto projektu by mohly zahrnovat rozšiřování kompatibility na různých chytrých zařízeních, nebo i vytvoření jiných variant zobrazování notifikací, například pomocí LCD displeje.

Literatura

[1] *List of Arduino boards and compatible systems. Wikipedia, the free encyklopedia.* [online]. 2013 [cit. 2019-12-06]. Dostupné z: https://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems.

[2] *I want to develop Android Apps — What languages should I learn? Android Authority* [online]. 2019 [cit. 2019-11-09]. Dostupné z: https://www.androidauthority.com/develop-android-apps-languages-learn-391008/.

[3] *What Everybody Ought to Know About Android : Introduction, Features Applications. ELPROCUS* [online]. 2019 [cit. 2019-11-08]. Dostupné z: https://www.elprocus.com/what-is-android-introduction-features-applications/.

[4] Arduino. *Arduino – Home* [online]. 2020 [cit. 2020-02-04]. Dostupné z: https://www.arduino.cc/.

[5] Banzi, M. a Shiloh, M. *Getting started with Arduino.* 3. vyd. Maker Media, 2014. ISBN 978-1-4493-6333-8.

[6] CHANDLER, N. *What is Android Beam? How stuff works?* [online]. 2012 [cit. 2019-11-08]. Dostupné z: https://electronics.howstuffworks.com/android-beam1.htm.

[7] DiMarzio, J. F. *Beginning Android programming with Android Studio.* John Wiley Sons, 2017. ISBN 9781118705599.

[8] DSD TECH. *HM-10 DataSheet* [online]. 2017 [cit. 2019-12-07]. Dostupné z: https://people.ece.cornell.edu/land/courses/ece4760/PIC32/uart/HM10/DSD%20TECH%20HM-10%20datasheet.pdf.

[9] GitHub, Inc.. *GitHub* [online]. 2020 [cit. 2020-02-10]. Dostupné z: https://www.github.com/.

[10] Google, Inc.. *Android Developers* [online]. 2019 [cit. 2019-12-06]. Dostupné z: https://developer.android.com.

[11] Heydon, R. *Bluetooth low energy: the developer's handbook.* Upper Saddle River, NJ: Prentice Hall, 2012. ISBN 013288836X.

[12] Ibraheem, N., Hasan, M., Khan, R. Z. a Mishra, P. Understanding Color Models: A Review. *ARPN Journal of Science and Technology.* Leden 2012, sv. 2.

[13] IDC Corporate USA. *Smartphone Market Share* [online]. 2020 [cit. 2020-03-02]. Dostupné z: https://www.idc.com/promo/smartphone-market-share/os.

[14] Kilián, K. *Co je NFC a k čemu je dobré ho použít? Svět Androida* [online]. 2018 [cit. 2019-11-08]. Dostupné z: https://www.svetandroida.cz/co-je-nfc-k-cemu-je-dobre-ho-pouzit/.

[15] Lacko, L. *Mistrovství – Android*. Computer Press, 2017. ISBN 978-80-251-4875-4.

[16] Monk, S. *Programming Arduino: getting started with sketches*. McGraw-Hill/TAB Electronics, 2011.

[17] Ney, C. *Android Security — Notification Listener Service Vulnerability. Medium.com* [online]. 2016 [cit. 2019-12-06]. Dostupné z: https://medium.com/@christopherney/notification-listener-service-vulnerability-8d0c586f88d5.

[18] Novák, K. *Slabikář radioamatéra*. SNTL – Státní nakladatelství technické literatury, 1970.

[19] Piskáček, J. *Možnosti sledování událostí na systému Android a jejich využití pro odhalování podezřelého chování aplikací*. Praha, CZ, 2015. Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/61674/F3-BP-2015-Piskacek-Jan-piskaja2_2015_bp.pdf.

[20] Townsend, K., Carles, C., Akiba a Davidson, R. *Getting started with Bluetooth low energy*. OReilly Media, 2014. ISBN 9781491900550.

[21] Voda, Z. *Sériová komunikace a cykly. Arduino.cz* [online]. 2014 [cit. 2019-12-06]. Dostupné z: https://arduino.cz/seriova-komunikace-a-cykly/.

[22] Voda, Z. *Průvodce světem Arduina*. Martin Stříž, 2017. ISBN 978-80-87106-93-8.

[23] Worldsemi. *WS2812B DataSheet* [online]. 2017 [cit. 2019-12-07]. Dostupné z: https://cdn-shop.adafruit.com/datasheets/WS2812.pdf.