UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2024 Bc. Josef Pavlík

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Správa vizitek pro Android

Diplomová práce

2024 Bc. Josef Pavlík

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Josef Pavlík**
Osobní číslo:	**I22170**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Správa vizitek pro Android**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem práce je vytvoření mobilní aplikace pro správu vizitek na OS Android v programovacím jazyce Kotlin. V teoretické části autor zpracuje existující technologie pro přenos dat v nadefinované vzdálenosti a techniky jejich zpracování. Aplikace bude umožňovat: uchovávání vizitek uživatelů s osobními informacemi, předávání vizitek jiným uživatelům pomocí NFC/QR kódu, úpravu vlastní vizitky / změna vzhledu vizitky, možnost vlastního pozadí (vyfocení fotografie s její následnou úpravou / oříznutí fotografie, rotace, apod.), automatická aktualizace uložených vizitek, při změně vizitky jiným uživatelem. Dále bude aplikace umožňovat vytvářet a sdílet události v kalendáři, které se přidají k vizitkám. Přístup do aplikace bude zabezpečen heslem, případně otiskem prstu. Tato aplikace využije databázi pro uchování dat, ke kterým bude umožněn přístup přes backend.

Rozsah pracovní zprávy: **50**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

JEMEROV, Dmitry a ISAKOVA, Svetlana. Kotlin in action. Shelter Island, NY: Manning Publications Co., [2017]. ISBN 978-1-61729-329-0.
ECKEL, Bruce a ISAKOVA, Svetlana. Atomic Kotlin. Online. USA: Crested Butte: Mindview, 2020. ISBN 978-0-9818725-4-4. [cit. 2023-10-07].
SABELLA, Robert P. a MUELLER, John. NFC for dummies. Hoboken, New Jersey: John Wiley, [2016]. ISBN 978-1-119-18292-4.

Vedoucí diplomové práce: **Ing. Soňa Neradová, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **8. listopadu 2023**
Termín odevzdání diplomové práce: **17. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2023

Prohlašuji:

Práci s názvem Správa vizitek pro Android jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019. Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 17. 05. 2024

Bc. Josef Pavlík

Poděkování

Tímto bych rád poděkoval své vedoucí diplomové práce Ing. Soně Neradové, Ph.D., za odbornou pomoc a cenné rady při zpracování této práce. Poděkování patří také mé rodině a přátelům, kteří mi byli oporou.

ANOTACE

Cílem diplomové práce bylo navrhnout a vytvořit nativní mobilní aplikaci pro správu vizitek pro platformu Android v programovacím jazyce Kotlin. Uživatel si v aplikaci může vytvořit vlastní vizitku a tu sdílet pomocí NFC a QR kódu. Vizitka má možnost nastavení vlastního pozadí. V aplikaci jsou vždy dostupná aktuální data. Aplikace umožňuje plánování událostí mezi uživateli. Data jsou v aplikaci zabezpečena. Teoretická část je věnována představení technologií, které byly využity pro implementaci praktické části, jako je platforma Android, programovací jazyk Kotlin, framework Koin, služby Firebase a technologie pro přenos dat. V praktické části je popsán návrh a implementace aplikace a zabezpečení aplikace.

KLÍČOVÁ SLOVA

Android, Kotlin, mobilní aplikace, NFC, Firebase

TITLE

Business card management for Android

ANNOTATION

The aim of this thesis was to design and develop a native mobile application for managing business cards for Android platform in Kotlin programming language. The user can create their own business card in the application and share it using NFC and QR code. The business card can be customized with a custom background. The app ensures that current data is always available. The app allows scheduling of events between users. Data is secure in the app. The theoretical part is dedicated to introduce the technologies that were used to implement the practical part such as Android platform, Kotlin programming language, Koin framework, Firebase services and data transfer technologies. The practical part describes the design and implementation of the application and the security of the application.

KEYWORDS

Android, Kotlin, mobile application, NFC, Firebase

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

JVM	Java Virtual Machine
API	Application Programming Interface
XML	Extensible Markup Language
DI	Dependency injection
DSL	Domain-Specific Language
SDK	Software Development Kit
NoSQL	Not Only Structured Query Language
JSON	JavaScript Object Notation
CLI	Command Line Interface
CI/CD	Continuous Integration/Continuous Deployment
ID	Identifier
BLE	Bluetooth Low Energy
IoT	Internet of Things
PIN	Personal Identification Number
NFC	Near Field Communication
ISP	Internet Service Provider
RFID	Radio-Frequency Identification
P2P	Peer-to-Peer
HCE	Host Card Emulation
SE	Secure Element
APDU	Application Protocol Data Unit
AID	Application Identifier
QR	Quick Response
ISO	International Organization for Standardization
URL	Uniform Resource Locator
WPS	Wi-Fi Protected Setup
WPA	Wi-Fi Protected Access
URI	Uniform Resource Identifier

ÚVOD

V dnešní době má téměř každý svůj mobilní telefon a digitalizace proniká do všech oblastí našeho života. Tento trend přináší s sebou neustálou poptávku po nových aplikacích, které nejen usnadňují, ale často i přetvářejí naše každodenní rutiny a zvyklosti. Mobilní aplikace se stávají nedílnou součástí našich životů, přinášejí nové možnosti a zjednodušují interakci s okolním světem.

Cílem této diplomové práce je navrhnout a vytvořit mobilní aplikaci pro správu vizitek určenou pro platformu Android v programovacím jazyce Kotlin. Výsledná mobilní aplikace umožňuje uživatelům vytvořit svou digitální vizitku a sdílet ji s ostatními uživateli aplikace. Pro sdílení vizitek jsou využity technologie jako NFC a QR kód. Aplikace dále umožňuje nastavení vlastního pozadí vizitky, přičemž uživatel může vybrat obrázek z galerie zařízení nebo vyfotit fotografii přímo v aplikaci. Uživatelé si rovněž mezi sebou mohou naplánovat události. Data v aplikaci jsou udržována aktuální díky službám Firebase. Kromě toho je aplikace zabezpečena bezpečnostními prvky jako je otisk prstu, PIN nebo heslo.

Hlavním důvodem výběru této práce bylo primárně použití technologie NFC, pro sdílení dat mezi zařízeními. Na základě NFC vznikla myšlenka této aplikace a spolu s tím i možnost použití moderních technologií pro vývoj aplikací pro platformu Android. Kromě této aplikace existují již obdobná řešení a to aplikace Swopi, nebo Popl, které jsou pro platformu Android i iOS. Tato řešení však nebyla inspirací pro tuto práci.

Teoretická část práce je věnována představení použitých technologií, které byly využity pro implementaci v praktické části. Nejprve je zde představena platforma Android spolu s možnostmi vývoje aplikací pro tuto platformu. Dále je představen samotný programovací jazyk Kotlin a jeho využití na různých platformách. Poté se text věnuje sadě vývojářských nástrojů pro tvorbu uživatelského rozhraní Jetpack Compose, včetně composable funkcí, aktualizace uživatelského rozhraní, podpory material designu a komponent pro organizaci prvků na obrazovce. V další části práce je popsán rámec Koin, který slouží pro předávání závislostí. Jsou zde také podrobně popsány jednotlivé služby Firebase, jako je Authentication, Cloud Firestore, Cloud Storage, Realtime Database a App Distribution, spolu s možností jejich využití v aplikaci. Nakonec jsou představeny technologie pro přenos dat, včetně Bluetooth, NFC, QR kódu a Wi-Fi Direct.

Praktická část obsahuje popis návrhu aplikace, včetně stanovených cílů, práce s daty, vývoje uživatelského rozhraní a navigace mezi obrazovkami. Primárně je zde popsána implementace aplikace, včetně tvorby a úpravy vizitek spolu s možností nastavení vlastního pozadí, ukládání vizitek pomocí služeb Firebase, sdílení vizitek pomocí NFC a QR kódu a kalendáře událostí. Nakonec je popsáno zabezpečení aplikace, které zahrnuje různé bezpečnostní prvky jako je otisk prstu, PIN nebo heslo, spolu s testováním aplikace.

1 POUŽITÉ TECHNOLOGIE

1.1 Android

Android je mobilním operačním systémem, který je založen na Linuxovém jádře a primárně cílí na chytré telefony a tablety. Jeho struktura zahrnuje jádro Linuxu, grafické uživatelské rozhraní, webový prohlížeč a škálu aplikací dostupných pro uživatele ke stažení. Android byl vydán pod licencí Apache v2 s otevřeným zdrojovým kódem, což umožňuje jeho přizpůsobení různým zařízením. Přestože je základní systém open-source, výrobci zařízení ho často doplňují o vlastní aplikace.

Historie Androidu sahá až do roku 2003, kdy společnost Android Inc. začala vyvíjet operační systém původně pro digitální fotoaparáty. Později byla tato společnost zakoupena společností Google, která začala prosazovat Android vůči výrobcům mobilních telefonů a operátorům. Díky své flexibilitě se Android rychle rozšířil a stal se jedním z předních operačních systémů pro mobilní zařízení.

Uživatelské rozhraní Androidu je založeno na dotykových interakcích, jako je klepnutí, posunutí nebo přiblížení. Domovská obrazovka slouží jako centrum navigace, kde uživatelé naleznou ikony aplikací a různé widgety. Statusový panel v horní části obrazovky poskytuje uživatelům důležité informace o stavu zařízení a jeho připojení.

Kromě toho Android nabízí funkce pro úsporu energie, jako je pozastavení neaktivních aplikací a optimalizace správy paměti. Dále disponuje funkcemi umělé inteligence, které zlepšují vyhledávání na platformě Google. Podporuje také širokou škálu hardwarových funkcí a komunikaci přes různé sítě a technologie, včetně Bluetooth, Wi-Fi, GPS a 3D grafiky.

Pokud jde o hardwarovou platformu, Android se původně spoléhal na architekturu ARM, ale novější verze podporují také x86 a x86-64. Minimální hardwarové požadavky se liší podle typu zařízení, ale Google stanovuje určité požadavky, které musí výrobci splnit, aby bylo zařízení oficiálně certifikováno.

Nicméně, Android čelí kritice kvůli tomu, že různá zařízení běží na různých verzích systému, což komplikuje vývoj aplikací. Další kritika směřuje k ochraně proti pirátství aplikací. Google se však snaží řešit tyto problémy prostřednictvím iniciativ jako Projekt Treble, který má zlepšit uživatelskou zkušenost a usnadnit vývoj aplikací pro Android. [1]

1.1.1 Možnosti vývoje aplikací pro Android

Při vývoji aplikací pro platformu Android je klíčové rozhodnutí mezi dvěma hlavními přístupy: nativním vývojem a vývojem multiplatformních aplikací. [2]

Nativní vývoj

Nativní aplikace jsou speciálně navrženy a optimalizovány pro konkrétní platformu, což zaručuje vysoký výkon a plnou integraci s funkcemi a rozhraním Android zařízení. Pro vývoj nativních aplikací pro Android se často používají jazyky jako Java nebo Kotlin a vývojové prostředí Android Studio. [2]

Vývoj multiplatformních aplikací

Multiplatformní aplikace jsou navrženy tak, aby fungovaly na více platformách, což může usnadnit sdílení kódu a snížit náklady na vývoj. Tyto aplikace využívají frameworky jako React Native, Flutter nebo Xamarin, které umožňují psaní kódu v jednom jazyce a jeho použití pro vytvoření aplikací pro různé platformy. Přestože vývoj multiplatformních aplikací může být efektivní z hlediska časové náročnosti a nákladů, může být náchylný na kompromisy v oblasti výkonu a přístupu k některým specifickým funkcím platformy. [2]

Shrnutí

Volba mezi nativním a multiplatformním přístupem k vývoji aplikací závisí na konkrétních potřebách projektu, jako je požadavek na výkon, dostupnost funkcí a rozsah podporovaných zařízení. Nativní aplikace často nabízejí lepší výkon a plnou integraci s konkrétní platformou, zatímco multiplatformní přístup umožňuje snížit náklady a sdílet kód mezi více platformami. Rozhodnutí je třeba učinit s ohledem na prioritu výkonu, potřeby funkčnosti a rozsah zařízení, na kterých bude aplikace používána. [2]

1.2 Programovací jazyk Kotlin

Kotlin je moderní open-source programovací jazyk se statickým typováním, navržený tak, aby byl stručný a bezpečný. Kombinuje paradigmata objektově orientovaného a funkcionálního programování a poskytuje podobnou syntaxi a koncepty z jiných známých jazyků, včetně Java, C#, JavaScript a Scala. Cílem Kotlinu není vytvářet něco radikálně nového,

ale spíše sbírat to nejlepší z jiných jazyků a kombinovat do intuitivního a efektivního celku. [3], [4]

Kotlin může být využit téměř kdekoliv. Existuje v různých variantách, které jsou například JVM (Kotlin/JVM), JavaScript (Kotlin/JS) a nebo také nativní kód (Kotlin/Native). Kotlin lze pomocí JVM zkompilovat do bajtkódu, což nám umožňuje jej použít tam, kde i Javu. Mimo to Kotlin/JS poskytuje možnost transpilace Kotlinu do JavaScriptu nebo také jej lze kompilovat do nativního kódu Kotlin/Native, který pak lze spustit i bez virtuálního stroje. Na jeho vývoj dohlíží Kotlin Foundation, což je společná iniciativa společností JetBrains a Google. [4], [5]

Pro vývoj aplikací pro Android se Kotlin stal preferovaným jazykem společnosti Google. Dokumentace a nástroje pro Android jsou navrženy s ohledem na Kotlin. I když některá rozhraní API systému Android, jako je Android KTX, jsou psána speciálně pro Kotlin, tak většina z nich je napsána v Javě a lze je bez problémů volat z obou jazyků. Tato hluboká interoperabilita s Javou je základním kamenem úspěchu Kotlinu. Existující knihovny Javy lze volně využívat v projektech psaných v Kotlinu a naopak. [4]

Závazek společnosti Google, že Kotlin bude pro vývoj aplikací pro Android na prvním místě, byl oznámen na konferenci Google I/O v roce 2019 a společnost ve své podpoře zůstala neochvějná. Jednoznačnost a stručnost Kotlinu pomáhá snižovat běžné chyby v programování a zároveň se bez problémů dá využít ve stávajících projektech. Pokud tedy chce někdo vytvořit aplikaci pro Android, je mu doporučeno začít právě s Kotlinem. [6]

1.3 Jetpack Compose

Jetpack Compose představuje moderní sadu nástrojů pro tvorbu uživatelského rozhraní na platformě Android. Jeho deklarativní přístup umožňuje vývojářům vytvářet nativní aplikace efektivněji a s větší flexibilitou. Compose využívá sílu jazyka Kotlin a kompilátor Kotlinu k optimalizaci procesu vykreslování uživatelského rozhraní. Jedná se o moderní alternativu k tradičnímu přístupu založenému na pohledech (Views) při vývoji uživatelského rozhraní. [7], [8]

Na rozdíl od tradičních XML pohledů Jetpack Compose umožňuje vytvářet uživatelská rozhraní pomocí tzv. composable funkcí, které popisují, jak má uživatelské rozhraní vypadat a chovat se. Hlavní výhodou používání Jetpack Compose je, že umožňuje psát kód uživatelského rozhraní, který je stručnější a snadněji srozumitelný. To vede k lepší udr-

žovatelnosti a snížení času vývoje. Nicméně, je důležité si uvědomit, že Jetpack Compose je stále ve vývoji a může mít některá omezení. Pokud tedy potřebujeme nějakou specifickou funkcionalitu, může být nutné ji implementovat ručně nebo hledat jiné alternativní řešení. [7], [8]

Na konferenci Android Dev Summit v roce 2022 bylo oznámeno, že již 160 z nejlepších 1 000 aplikací pro Android používá Jetpack Compose. To svědčí o rostoucí popularitě tohoto nástroje a jeho přínosu pro vývoj mobilních aplikací. [8]

1.3.1 Composable funkce

V Jetpack Compose se uživatelské rozhraní skládá z funkcí, které definují jednotlivé komponenty. Tyto funkce se nazývají composables a slouží jako stavební bloky pro tvorbu uživatelské rozhraní. Composable funkce jsou lehké a mohou se vzájemně kombinovat, aby vytvářely složité hierarchie uživatelského rozhraní. [7], [9]

Compose umožňuje vytvářet uživatelské rozhraní voláním těchto composable funkcí. Jsou to běžné funkce s anotací *@Composable*, které reprezentují jednotlivé widgety na obrazovce. Compose pracuje s pomocí pluginu kompilátoru Kotlinu ve fázi ověřování typů a generování kódu. Plugin kompilátoru Compose se ujistí, že je možné vytvářet composables. [7], [9]

Zde je příklad jednoduchých composable funkcí, které zobrazí tlačítko:

```
@Composable
fun ExampleButton(onClick: () -> Unit, text: String) {
    Button(
        onClick = onClick
    ) {
        Text(text = text)
    }
}

@Composable
fun MainScreen() {
    ExampleButton(
        onClick = { /* Action here. */ },
        text = "Click me!"
    )
}
```

Ukázkový kód obsahuje funkce *ExampleButton* a *MainScreen*, které jsou composable funkcemi s anotací *@Composable*. Bez této anotace by nebylo možné vkládat komponenty Jetpack Compose do funkcí.

Funkce *ExampleButton* obsahuje tlačítko. Tato funkce má dva parametry: *onClick*, což je funkce bez parametrů a návratové hodnoty, která se spustí po kliknutí na tlačítko a *text*, což je text, který se má zobrazit na tlačítku.

Druhou funkcí je zde *MainScreen*, která používá výše definovanou funkci *ExampleButton*. Volání *ExampleButton* předává akci v parametru *onClick*, která se spustí po kliknutí na tlačítko a text *Click me!* jako text tlačítka.

1.3.2 Aktualizace uživatelského rozhraní

Jelikož je Compose deklarativní, tak pro aktualizaci uživatelského rozhraní je nutné volat stejný composable prvek s novými parametry, které reprezentují stav uživatelského rozhraní. Při každé aktualizaci stavu následně dochází k tzv. rekompozici. [7], [10]

V Jetpack Compose hraje klíčovou roli právě koncept rekompozice. Jedná se o proces automatického opětovného spouštění composable funkcí, kdykoli se změní jejich vstupy. Když k tomu dojde, Compose znovu sestaví strom widgetů uživatelského rozhraní, v podstatě překreslí widgety na základě nových dat poskytnutých composables. [9]

Nicméně rekompozice celého uživatelského rozhraní může být výpočetně náročná. Pro optimalizaci výkonu využívá Compose chytrou techniku nazvanou inteligentní rekompozice. To znamená, že Compose znovu volá pouze ty composable funkce, které mají skutečně nové vstupní hodnoty. Funkce s nezměněnými vstupy jsou efektivně přeskočeny. [9]

Na rozdíl od imperativního přístupu založeného na XML, kde se prvky jako je například *TextField* aktualizují automaticky, je ve Compose nutné explicitně předat nový stav composable prvku, aby se uživatelského rozhraní aktualizovalo správně. [7], [10]

Jetpack Compose poskytuje systém správy stavu pro composable funkce. Tento systém umožňuje definovat dynamické prvky uživatelského rozhraní, které reagují na uživatelské interakce nebo jiné události. Klíčovou roli hraje funkce *remember*. Ta umožňuje vytvářet a spravovat stav uvnitř composable funkce. Funkce *remember* v podstatě ukládá objekty (změnitelné i neměnné) během počáteční kompozice a načítá je během rekompozice, čímž zajišťuje uchování stavu napříč aktualizacemi uživatelského rozhraní, což vede k reaktiv-

ním komponentám reagujícím na změny v datech nebo samotném stavu. Výsledkem je plynulý a responzivní uživatelský zážitek. [7], [10]

Zde je ukázka použití funkce *remember*:

```
@Composable
fun Example() {
    val count = remember { mutableIntStateOf(0) }

    Button(
        onClick = { count.value++ }
    ) {
        Text("Clicks: ${count.value}")
    }
}
```

1.3.3 Material Design

Vytváření atraktivních a soudržných uživatelských rozhraní pomocí Jetpack Compose je díky integrované podpoře pro Material Design značně zjednodušeno. Material Design představuje adaptabilní systém poskytující instrukce, komponenty a nástroje, které podporují osvědčené postupy v designu uživatelských rozhraní. Tato knihovna nabízí předem definované komponenty Material Designu, jako jsou tlačítka, textová pole, karty a další, které lze upravovat podle potřeb a vizuální identity aplikace. [7], [11]

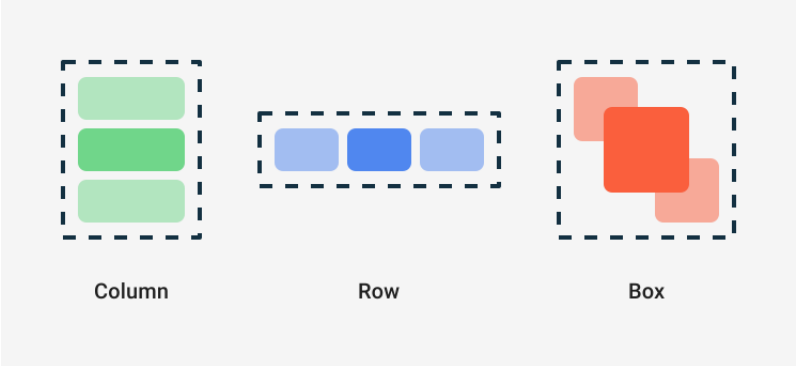
Jetpack Compose rovněž obsahuje implementaci Material Designu 3, což představuje další krok v jeho vývoji. Material 3 přináší aktualizované motivy, komponenty a funkce personalizace podle konceptu Material You, včetně dynamických barev. Byl navržen s ohledem na nový vizuální styl a systémové uživatelské rozhraní v Androidu 12 a novějších verzích. Je však důležité poznamenat, že některá API v Material Designu 3 jsou považována za experimentální. To znamená, že se v budoucnu mohou změnit. Composable funkci, která dané API používá je nutné označit jako experimentální pomocí anotace *@OptIn(ExperimentalMaterial3Api::class)*. [12]

1.3.4 Layouty

Jetpack Compose nabízí širokou škálu komponent, které usnadňují organizaci prvků na obrazovce. Komponenty jako *Column*, *Row* a *Box* slouží k tvorbě flexibilních a responzivních layoutů. Tyto komponenty lze vzájemně kombinovat a upravovat pomocí modifikátorů, čímž se dosahuje požadovaného vzhledu a chování uživatelského rozhraní. Díky

těmto vlastnostem se Jetpack Compose stává efektivním nástrojem pro tvorbu uživatelsky přívětivých aplikací s jednoduchým a přehledným kódem. [12]

- **Column** - Tento layout je ideální pro vertikální uspořádání potomků. Často se používá k vytváření vertikálních seznamů nebo sekcí v uživatelském rozhraní. [13]
- **Row** - Naopak, *Row* layout slouží k horizontálnímu uspořádání potomků. To je užitečné pro vytváření horizontálních seznamů nebo zarovnání potomků vedle sebe. [13]
- **Box** - *Box* layout je všestranný kontejner, který umožňuje dávat potomky přes sebe. Často se používá pro vytváření vlastních komponent uživatelského rozhraní nebo překrytí jiné vrstvy. [13]



Obrázek 1: Základní layouty Jetpack Compose. [14]

1.4 Koin

Koin je jedním z nejžádanějších frameworků v jazyce Kotlin, který se zaměřuje na vkládání závislostí (dependency injection) v aplikacích psaných v Kotlinu. Jeho design klade důraz na jednoduchost a snadnou použitelnost, přičemž se vyznačuje absencí generování kódu navíc a reflexe, což z něj činí efektivní a rychlý nástroj. Na rozdíl od jiných frameworků, kterým je například Hilt od společnosti Google, Koin nepoužívá anotace pro definici závislostí, ale místo toho využívá Kotlin DSL, který poskytuje jednoduchou a deklarativní syntaxi pro jejich definici a vztahy mezi nimi. [15]

1.4.1 Použití

Na začátku je nutné si prvně vytvořit Koin modul, kde budou následně definovány závislosti viz. ukázka kódu níže.

```
import org.koin.androidx.viewmodel.dsl.viewModel
import org.koin.dsl.module

private val viewModelModule = module {
    viewModel { parameters ->
        SignInViewModel(googleAuthUiClient = parameters.get())
    }
    viewModel {
        MyProfileViewModel()
    }
}

val uiModule = listOf(viewModelModule)
```

V této ukázce je vidět definice modulu, ve kterém jsou definovány závislosti pro uživatelské rozhraní. Konkrétně jde o modul nazvaný viewModelModule, který obsahuje definice ViewModelů. Pro definici ViewModel komponenty se využívá klíčové slovo DSL „viewModel“, které umožňuje vytvoření instance této třídy ViewModel podle životního cyklu aplikace na platformě Android. [15], [16]

Kromě toho existují ještě další klíčová slova, jako je „single“ a „factory“. Klíčové slovo „single“ znamená, že instance dané třídy bude vytvořena pouze jednou a poté udržována v paměti. Toho lze například efektivně využít při vytváření instance databáze, kdy je potřeba, aby byla k dispozici pouze jedna instance během běhu aplikace. [17]

Na druhou stranu, klíčové slovo „factory“ označuje generování nové instance dané třídy pokaždé, když je potřeba. Tyto instance nejsou udržovány v paměti a každý požadavek vytvoří novou instanci. Tímto způsobem se zajišťuje, že každá žádost o danou závislost získá novou a oddělenou instanci objektu. [17]

Funkce *get()* umožňuje získat instanci požadované třídy z Koin kontejneru. Funguje tak, že prvně identifikuje požadovaný datový typ a následně dosadí korespondující instanci. V případě potřeby je možné předat instanci třídy parametry konstruktoru ViewModelu. [17]

Dalším krokem je inicializace Koinu samotného. Prvně je potřeba mít třídu, která dědí od *Application()*, v tomto případě je to třída *MainApp*. V této třídě se přepisuje metoda *onCreate()*, která je volána při inicializaci aplikace. V metodě *onCreate()* se provádí inicializace Koinu pomocí funkce *startKoin()*. Nejprve se nastavuje kontext aplikace pomocí *androidContext(this@MainApp)*, což umožňuje Koinu pracovat s Android kontex-

tem. Poté následuje definice modulů pomocí funkce *modules()*, kde se spojují moduly pro různé vrstvy aplikace. Toto je vidět na ukázkovém kódu níže. [18]

```
class MainApp : Application() {

    override fun onCreate() {
        super.onCreate()

        startKoin {
            androidContext(this@MainApp)
            modules(
                    app +
                        dataModule +
                        domainModule +
                        infrastructureModule +
                        uiModule
            )
        }
    }
}
```

Posledním krokem je využití závislostí. Pro získání instance ViewModelu až když bude skutečně potřeba lze použít *by viewModel()*, nebo *getViewModel()* pro okamžité získání instance. Ostatní komponenty, jako jsou například ty s použitím klíčových slov „single“ a „factory“, lze získat s použitím *by inject()* pro získání instance až když bude skutečně potřeba, nebo *get()* pro okamžité získání instance. [16], [19]

Získání ViewModelu může tedy vypadat následovně:

```
@Composable
fun MyProfileScreen(
    viewModel: MyProfileViewModel = getViewModel { parametersOf() }
) {

}
```

1.5 Služby Firebase

1.5.1 Authentication

Klíčovou součástí každé aplikace je získání identity uživatele. Identifikace uživatele je základním krokem k zajištění bezpečnosti dat a poskytnutí personalizovaného prostředí. Authentication je služba, která umožňuje snadnou autentizaci uživatelů do aplikací. Díky

ní je možné uživatele přihlásit pomocí jeho e-mailu a hesla, telefonního čísla nebo i prostřednictvím oblíbených externích poskytovatelů jako je Google, Apple, Facebook, GitHub nebo Twitter. [20]

Služba Authentication je navržena s podporou standardů OAuth 2.0 a OpenID Connect, což jsou zavedené a široce uznávané protokoly pro autentizaci a správu přístupu. Díky této podpoře Authentication poskytuje bezpečnou a spolehlivou autentizaci uživatelů v aplikaci. [20]

Pro přihlášení uživatele do aplikace je tedy prvním krokem získání jeho autentizačních údajů. Těmito údaji může být tedy e-mailová adresa a heslo, nebo token OAuth od externích poskytovatelů. Tyto údaje jsou pak následně předány do SDK Authentication. Po předání autentizačních údajů do SDK se aplikace spojí s backend službami Firebase, které ověří platnost těchto údajů. Pokud jsou údaje platné, backend vrátí odpověď klientovi, což znamená úspěšné přihlášení uživatele. [20]

Následně, po úspěšném přihlášení, má aplikace možnost získat základní informace o uživateli, jako je například jeho e-mailová adresa či jméno. Tyto informace mohou být důležité pro personalizaci uživatelského prostředí. [20]

Authentication dále umožňuje řídit přístup uživatele k datům uloženým v dalších službách Firebase. To znamená, že je možné specifikovat, které části aplikace jsou dostupné pouze přihlášeným uživatelům a jakým způsobem mají být data chráněna. [20]

Důležité je taktéž zmínit, že ve výchozím nastavením mají autentizovaní uživatelé právo číst a zapisovat data do Realtime Database, Cloud Storage nebo Cloud Firestore. Nicméně je možné spravovat přístup těchto uživatelů tím, že se upraví bezpečnostní pravidla (Security Rules) pro tyto služby. Tímto způsobem je možné specifikovat, kdo může provádět čtení, zápis nebo jakékoliv jiné operace s daty. [20]

Takto mohou vypadat bezpečnostní pravidla Cloud Storage:

```
rules_version = '2';

service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Tato bezpečnostní pravidla definují, že pouze uživatelé, kteří se úspěšně přihlásili pomocí služby Authentication, mohou prohlížet a nahrávat soubory ve službě Firebase Storage. [21]

Celkově tedy Authentication poskytuje robustní způsob autentizace uživatelů a zabezpečení jejich dat v rámci aplikace. To umožňuje vytvoření bezpečného a personalizovaného prostředí pro uživatele, což je klíčové pro úspěšný provoz aplikace. [20]

1.5.2 Cloud Firestore

Při vývoji aplikací je nezbytné brát v úvahu nejenom funkčnost, ale také přenositelnost a bezpečnost dat. Je důležité, aby aplikace byla schopna efektivně ukládat a spravovat data, a zároveň poskytovat spolehlivý a rychlý způsob přístupu k nim. Jednou z moderních a flexibilních možností, jak dosáhnout těchto cílů, je využití Cloud Firestore, cloudové NoSQL databáze, která je součástí Firebase a Google Cloud infrastruktury. Cloud Firestore poskytuje robustní řešení pro ukládání dat, které umožňuje snadnou integraci s různými platformami a nabízí pokročilé funkce pro bezpečné spravování dat. [22]

Tato databáze umožňuje přístup k datům z různých platforem a zařízení, včetně aplikací pro Apple, Android nebo web, pomocí nativních SDK. Dále je dostupná v několika dalších jazycích, jako jsou Node.js, Java, Python, Unity, C++ nebo Go, což umožňuje široké možnosti využití v různých typech aplikací a prostředí. [22]

Cloud Firestore využívá NoSQL datový model, kde jsou data ukládána do dokumentů, které obsahují pole mapující hodnoty. Tyto dokumenty jsou organizovány do kolekcí a umožňují snadné dotazování a práci s daty. Dokumenty podporují různé datové typy, včetně polí, booleovských hodnot, bajtů pro binární data, data a času, desetinných čísel, geografických bodů, celých čísel, map pro strukturování dat ve formě klíč-hodnota, odkazů na jiné dokumenty a textových řetězců. Díky takto široké škále datových typů je možné vytvářet komplexní hierarchické struktury dat a ukládat různorodé informace v databázi. [22], [23]

Dotazování v Cloud Firestore je flexibilní a efektivní, umožňuje vytvářet různé druhy dotazů, aby bylo možné získat potřebná data. Nejprve je možné vytvářet dotazy na úrovni dokumentů, což znamená, že lze získat data z konkrétních dokumentů nebo jejich částí bez nutnosti získávat celou kolekci nebo jakékoliv vnořené podkolekce. To umožňuje rychlý přístup k potřebným informacím bez zbytečné zátěže sítě. [22]

Dále je možné přidávat různé podmínky a operátory do dotazů, včetně řazení, filtrování a omezení. Řazení umožňuje řadit výsledky podle určených kritérií, například abecedně nebo podle časového razítka. Filtrování umožňuje vybírat pouze data, která splňují určité podmínky, například získat všechny dokumenty, které mají hodnotu pole v určitém rozmezí. Omezení umožňuje limitovat počet vrácených výsledků, což je užitečné zejména při stránkování výsledků. [22]

Kromě toho je možné vytvářet dotazy s využitím kurzorů, což umožňuje efektivní stránkování výsledků. Kurzory umožňují určit, od kterého dokumentu nebo jakého bodu v kolekci se mají výsledky začít vracet, což je užitečné zejména při práci s velkým objemem dat. [22]

V neposlední řadě je možné přidat do dotazů posluchače v reálném čase (realtime listeners), které aplikaci automaticky informují o jakýchkoli změnách dat. Díky tomu je možné udržovat data v aplikaci aktuální bez potřeby ručního vyžádání aktualizovaných údajů. [22]

Takto může vypadat kód pro sledování změn dat v aplikaci pro Androidu:

```
val db = Firebase.firestore
val productRef = db.collection("products").document("productId")

productRef.addSnapshotListener { snapshot, e ->
    if (e != null) {
        return@addSnapshotListener
    }

    Log.d(TAG, "Data: ${snapshot?.data ?: "null"}")
}
```

Tento kód umožňuje kontinuálně sledovat změny v datech dokumentu „productId“ v kolekci „products“ a aktualizovat je v reálném čase. [24]

Jak již bylo zmíněno dříve v souvislosti se službou Authentication, data mohou být zabezpečena pomocí bezpečnostních pravidel, která umožňují stanovit jasné pokyny pro přístup a manipulaci s daty. [22]

1.5.3 Cloud Storage

Cloud Storage poskytuje flexibilní a škálovatelné řešení pro ukládání a synchronizaci uživatelských dat v aplikacích na platformách Android, iOS a webových aplikacích. Tato

služba je postavena na rychlé a zabezpečené infrastruktuře Google Cloud, což zajišťuje spolehlivost a bezpečnost dat. [25]

Cloud Storage ukládá soubory do tzv. „bucketů“ v úložišti Google Cloud Storage, což umožňuje snadný přístup k nim jak přes Firebase, tak přímo přes Google Cloud. [25], [26]

Buckety jsou základními kontejnery, které uchovávají data. Vše, co se ukládá do Cloud Storage, musí být obsaženo v nějakém bucketu. Buckety je možné využít k organizaci dat a k řízení přístupu. Na rozdíl od adresářů a složek však není možné vytvářet vnořené buckety. [26]

Cloud Storage umožňuje ukládat a spravovat širokou škálu souborů, včetně fotografií, videí, audio souborů a dalšího uživatelského obsahu. S využitím jednoduchých SDK pro různé platformy, jako jsou Android, iOS a webové aplikace, lze rychle integrovat tuto službu do aplikací a ukládat data přímo do cloudového úložiště. [25]

Data uložená v Cloud Storage jsou automaticky zabezpečena pomocí pokročilých bezpečnostních opatření a standardů, což zaručuje ochranu dat před neoprávněným přístupem. Díky integraci s dalšími službami Firebase, jako je Authentication, je možné snadno řídit přístup uživatelů k uloženým souborům a nastavit příslušná bezpečnostní pravidla. [25]

Cloud Storage poskytuje také širokou škálu nástrojů pro práci s uloženými daty, včetně možnosti provádět serverové operace, jako je manipulace s obrázky nebo převod formátů, pomocí Google Cloud Storage API. Tímto způsobem je možné efektivně spravovat a zpracovávat uživatelský obsah přímo v cloudu. [25]

Celkově je Cloud Storage silným nástrojem pro ukládání, správu a synchronizaci uživatelských dat z aplikací. Jeho jednoduchá integrace, spolehlivost a bezpečnost ho činí ideální volbou pro ukládání a správu uživatelského obsahu. [25]

1.5.4 Realtime Database

Realtime Database je služba poskytující možnost ukládání a synchronizace dat v reálném čase pomocí cloudové NoSQL databáze. Tato databáze byla koncipována s ohledem na potřeby moderních aplikací, které vyžadují okamžité aktualizace dat a spolehlivou synchronizaci napříč všemi připojenými zařízeními. Klíčovou vlastností této služby je její schopnost synchronizovat data mezi všemi klienty v reálném čase, což zajišťuje, že uživatelé mají vždy přístup k nejnovějším informacím. Dokonce i v případě, že aplikace přechází

do režimu offline, je stále možné pracovat s daty díky lokální perzistenci, která umožňuje uživatelům provádět změny a interagovat s aplikací i bez aktivního internetového připojení. [27]

Data v Realtime Database jsou ukládána ve formátu JSON, což usnadňuje jejich strukturování a manipulaci. Tento formát je synchronizován v reálném čase s každým připojeným klientem, což znamená, že jakmile dojde ke změně dat, je tato změna okamžitě propagována všem ostatním klientům. Při vývoji multiplatformních aplikací s dostupnými SDK pro platformy Apple, Android a JavaScript, všichni klienti sdílejí jednu instanci Realtime Database, čímž se zajistí automatická aktualizace s nejnovějšími daty. [27]

Takto může vypadat kód pro sledování změn dat v aplikaci pro Androidu:

```
val db = Firebase.database
val productRef = db.getReference("products/productId")

productRef.addValueEventListener(object : ValueEventListener {
    override fun onDataChange(dataSnapshot: DataSnapshot) {
        val product = dataSnapshot.getValue<Product>()
        Log.d(TAG, "Data: $product")
    }

    override fun onCancelled(error: DatabaseError) { }
})
```

Tento kód umožňuje sledovat změny v datech pod cestou „products/productId“ v Realtime Database a aktualizovat je v reálném čase. [28]

Kromě synchronizace dat umožňuje Realtime Database také definovat bezpečnostní pravidla pro přístup k datům. Tato pravidla jsou klíčovou součástí zabezpečení aplikace a mohou být přizpůsobena potřebám konkrétní aplikace. Tím se zvyšuje úroveň ochrany dat a zajišťuje, že pouze oprávnění uživatelé mají přístup k určitým částem datového úložiště. [27]

Realtime Database je navržena jako NoSQL řešení, což přináší odlišné optimalizace a funkcionality ve srovnání s tradičními relačními databázemi. Její API je pečlivě navrženo tak, aby povolovalo pouze operace, které lze provést rychle a efektivně. Díky tomu je možné obsluhovat miliony požadavků v reálném čase, aniž by došlo k ohrožení responzivity. Tato schopnost poskytuje uživatelům plynulý a bezproblémový zážitek při interakci s aplikací. Vzhledem k tomu je nezbytné pečlivě přemýšlet o tom, jaká data uživatel potřebuje nejvíce, a pak je podle toho vhodně strukturovat. Tato analýza datové architektury je klíčová pro optimalizaci výkonu a zajištění vysoké úrovně uživatelského zážitku. [27]

Obě Realtime Database i Cloud Firestore jsou databáze typu NoSQL v cloudu, které poskytuje Firebase. Realtime Database využívá JSON datový model, což je vhodné pro jednoduché datové struktury, zatímco Cloud Firestore nabízí bohatší datový model s možností vnořování kolekcí a dokumentů, což vyhovuje komplexnějším datovým modelům. Pokud jde o dotazování, Realtime Database poskytuje omezené dotazování založené na záznamech, zatímco Cloud Firestore nabízí bohaté dotazování umožňující složité dotazy a filtrování dat. Cloud Firestore také nabízí vyšší škálovatelnost a možnost konfigurace synchronizace dat, což je vhodné pro aplikace s větším provozem a počtem dokumentů. [29]

1.5.5 App Distribution

V dnešní době, kdy se softwarové aplikace neustále vyvíjejí a mění, je klíčové zajistit plynulou distribuci nových verzí aplikací mezi testery a další uživatele. Standardní postupy, jako je zasílání instalačních souborů e-mailem nebo jejich sdílení přes různé platformy, byly běžné, avšak často neefektivní při zajištění spolehlivého doručení a správy verzí.

Nově se objevuje trend vytváření platforem, které umožňují snadnější a transparentnější distribuci nových verzí aplikací. Tyto platformy umožňují vytváření skupin pro distribuci, sdílení odkazů a sledování historie verzí aplikací, což usnadňuje komunikaci a spolupráci v rámci týmu i s externími testery.

Tento koncept se stává realitou díky Firebase App Distribution, což je služba, která poskytuje uživatelsky přívětivé řešení pro distribuci nových verzí aplikací pro platformy Android a iOS. S pomocí této služby je snadné spravovat nasazení aplikací a získávat následně zpětnou vazbu od testerů. Mimo jiné je App Distribution propojena s Crashlytics, což umožňuje získávat důležité informace o stabilitě aplikace. App Distribution tak přináší novou úroveň automatizace, transparentnosti a spolehlivosti do procesu distribuce aplikací.

Existuje několik metod, jak lze distribuovat Android aplikace, z nichž každá má své vlastní výhody a vhodnost v různých situacích. Můžeme zvolit manuální distribuci pro specifické uživatelské skupiny nebo vytvořit odkaz na pozvánku. Další možností je využití Firebase CLI, Gradle pluginu pro Firebase, procesů kontinuální integrace a nasazení (CI/CD) nebo služby Github Actions. Každá z těchto metod nabízí své vlastní možnosti a flexibilitu, které mohou odpovídat konkrétním potřebám ve vývoji aplikací. [30]

Manuální distribuce

Pro manuální distribuci pomocí App Distribution je nejprve potřeba vytvořit ve Firebase projekt pro danou aplikaci a následně vytvořit skupinu uživatelů, kteří budou moci aplikaci otestovat. Dalším krokem je nahrání podepsané nebo ladící verze aplikace, přiřazení skupiny uživatelů a možnost popisu verze. Po dokončení pak každý člen skupiny obdrží e-mailovou pozvánku s odkazem ke stažení aplikace App Tester. Tato aplikace slouží jako správce, který umožňuje prohlížet instalační soubory různých aplikací s jejich verzemi a poznámkami. Uživatel tak má možnost si vybrat požadovanou verzi k instalaci nebo prohlížet předchozí verze. Tato metoda je vhodná pro rychlou a jednoduchou distribuci aplikace. [30]

Distribuce s odkazem na pozvánku

Při této metodě se nejprve nahraje aplikace stejným způsobem jako při manuální distribuci. Poté se vygeneruje odkaz na pozvánku, který uživatele přesměruje na stránku, kde zadá svůj e-mail. Po zadání e-mailové adresy uživatel obdrží pozvánku k testování aplikace. Tato metoda je přínosná, zejména v případech, kdy nejsou k dispozici e-mailové adresy testerů nebo je třeba pozvat více uživatelů najednou, aniž by bylo nutné manuálně přidávat jednoho po druhém. [30]

Distribuce přes Firebase CLI

Pro distribuci aplikace lze také využít Firebase CLI. Tato metoda umožňuje distribuci aplikace přímo z příkazové řádky. Stačí zadat cestu k instalačnímu souboru, Firebase App ID a další parametry, jako jsou poznámky a seznam e-mailů testerů nebo skupin. Tím lze rychle a snadno distribuovat aplikaci bez potřeby používat webové rozhraní. [30]

Příkaz může vypadat následovně:

```
firebase appdistribution:distribute app-debug.apk \
--app 1:1234567890:android:0a1b2c3d4e5f6g7h8i9j \
--release-notes "Release notes" \
--testers "tester@gmail.com" \
--groups "testers"
```

Distribuce pomocí Firebase Gradle pluginu

Pro distribuci aplikace je k dispozici Gradle plugin od Firebase. Tento plugin umožňuje specifikovat testery, skupiny nebo poznámky přímo v souboru Gradle, což umožní nastavit

distribuci pro různé typy sestavení a varianty aplikace. Prvním krokem je přidání závislosti na Firebase Distribution plugin, poté je nutné v souboru Gradle modulu specifikovat vlastnosti pro distribuci aplikace. E-maily nebo názvy skupin lze přímo definovat v Gradle souboru nebo uložit do textového souboru (.txt) pro větší konfigurovatelnost. [30]

Vlastnosti pro distribuci v Gradle souboru mohou vypadat následovně:

```
buildTypes {
    debug {
        firebaseAppDistribution {
            appId = "1:1234567890:android:0a1b2c3d4e5f6g7h8i9j"
            releaseNotes = "Release notes"
            testers = "tester1@gmail.com, tester2@gmail.com"
            groupsFile = "/path/to/tester-groups.txt"
        }
    }
}
```

Příkaz pro distribuci tohoto sestavení může vypadat následovně:

```
./gradlew assembleDebug appDistributionUploadDebug
```

Kontinuální integrace a nasazení pomocí Github Actions

Kontinuální integrace a nasazení (CI/CD) pomocí Github Actions představuje automatizovaný proces testování, sestavování a nasazování softwarových aplikací. Tato služba je poskytována platformou GitHub a umožňuje vytváření toků práce (workflows) přímo v rámci repozitářů na GitHubu.

GitHub Actions umožňuje definovat a spouštět různé workflows v reakci na události, jako je například push nového kódu do repozitáře. Workflows mohou obsahovat různé kroky, jako je sestavení aplikace, spuštění testů a nasazení aplikace.

Tato služba byla uvedena kolem listopadu 2019 a od té doby se osvědčila jako spolehlivý nástroj pro vývoj softwaru. GitHub Actions umožňuje integraci s GitHubem, což znamená, že celý proces CI/CD lze provádět přímo na jedné platformě, bez nutnosti spoléhat se na externí nástroje.

Workflow v GitHub Actions je definován pomocí souboru YAML, ve kterém jsou specifikovány jednotlivé kroky, které mají být provedeny. Tyto kroky mohou být nastaveny tak, aby provedly různé úlohy, včetně sestavení, testování a nasazení aplikace. Viz. Příloha A.

Pro distribuci aplikace lze GitHub Actions využít spolu s Firebase App Distribution. V konfiguraci workflow je možné přidat krok pro distribuci aplikace, kde lze specifikovat identifikátor aplikace ve Firebase, testery, skupiny a poznámky k vydání spolu se

sestavenou aplikací. Jakmile se workflow spustí, GitHub Actions provede potřebné kroky a distribuuje aplikaci. [31]

1.6 Technologie pro přenos dat

1.6.1 Bluetooth

Bluetooth je klíčovou technologií v oblasti bezdrátového připojení, poskytující flexibilitu a univerzálnost pro vývojáře po celém světě. Tato technologie umožňuje snadnou bezdrátovou komunikaci mezi zařízeními bez potřeby komplikované síťové infrastruktury. Díky své univerzálnosti se Bluetooth stala preferovanou volbou pro širokou škálu aplikací, ať už jde o streamování hudby mezi chytrými telefony a reproduktory, nebo přenos dat mezi lékařskými přístroji a tablety, nebo také pro komunikaci mezi tisíci uzly, které řeší automatizaci budov. Bluetooth se stala neodmyslitelnou součástí digitálního světa, poskytující spolehlivé a bezpečné propojení mezi zařízeními v různých oblastech života. [32], [33]

Bluetooth Special Interest Group (Bluetooth SIG) je standardizační organizace, která dohlíží na vývoj Bluetooth standardů a uděluje licence na Bluetooth technologie a ochranné známky výrobcům. Existují dva hlavní standardy, které se v současnosti používají: Bluetooth Classic a Bluetooth Low Energy (BLE). [32], [33]

Bluetooth Classic, také známý jako Bluetooth Basic Rate (BR) nebo Enhanced Data Rate (EDR), představuje bezdrátovou technologii přenosu dat, která funguje na frekvenci 2,4 GHz. Tento rádiový systém s nízkou spotřebou energie využívá 79 kanálů v pásmu 2,4 GHz. Je často využíván pro bezdrátové streamování zvuku a je standardem pro zařízení jako jsou reproduktory, sluchátka nebo zábavní systémy v automobilech. Kromě toho umožňuje také přenos dat a mobilní tisk. [32], [33]

SIG představila Bluetooth Low Energy ve své specifikaci Bluetooth 4.0 v roce 2010 (s pozdější specifikací Bluetooth 5 z roku 2016, věnovanou výhradně BLE). Hlavním zaměřením byl rostoucí trh s zařízeními souvisejícími se zdravím a fitness spolu s chytrými domovy a vnitřní polohou. BLE bylo navrženo s ohledem na úsporu energie. Pracuje na frekvenci 2,4 GHz v pásmu ISM, kde používá 40 kanálů, a umožňuje tak snadnou komunikaci mezi zařízeními při zachování nízké spotřeby energie, což je zvláště důležité v oblasti Internetu věcí (IoT), kde zařízení často fungují na baterie a musí být schopna dlouhodobého provozu bez možnosti častého nabíjení. Tato technologie podporuje různé

formy komunikace, včetně broadcastu a sítí typu mesh, což umožňuje vytváření spolehlivých sítí zařízení, a nabízí rozmanité možnosti využití včetně určení polohy zařízení. Jednou z klíčových vlastností BLE je jeho asymetrická architektura, která umožňuje zařízením fungovat buď jako centrální nebo periferní, což zajišťuje efektivní propojení různých zařízení s různými funkcionalitami a rolemi v síti. Celkově lze říci, že BLE poskytuje spolehlivé a energeticky efektivní bezdrátové propojení, které je nedílnou součástí moderních digitálních aplikací a IoT zařízení. [33], [34]

Bluetooth technologie je dnes nezbytná pro bezdrátové spojení periferií s mobilními telefony, počítači a notebooky. Oblíbené příslušenství jako myši, klávesnice, reproduktory a sluchátka jsou běžnými příklady využití. Tato technologie se stále více rozšiřuje do spotřební elektroniky, včetně kamer, televizí a domácích spotřebičů, jako jsou ledničky nebo trouby. V lékařském prostředí je Bluetooth nezbytný pro zařízení jako glukózové senzory nebo kardiostimulátory, které spoléhají na bezproblémové bezdrátové připojení. Moderní informační a zábavní systémy v automobilech také využívají Bluetooth ke streamování hudby a navigačních pokynů z telefonu řidiče, což zvyšuje pohodlí a bezpečnost během jízdy. [32]

Připojení přes Bluetooth je proces nazývaný párování. Během tohoto procesu si zařízení vymění informace, včetně bezpečnostního klíče, který slouží k identifikaci druhého zařízení. Tyto informace jsou uloženy na obou zařízeních, což umožňuje snadné a automatické navázání spojení v budoucnu. Aby se zařízení mohla propojit, může být potřeba porovnat řetězec čísel zobrazených na obrazovce obou zařízení nebo zadat PIN z jednoho zařízení do druhého. [32]

Fyzická poloha zařízení a okolní podmínky významně ovlivňují výkon technologie Bluetooth. Bluetooth může propojovat zařízení v rozsahu přibližně 9 až 90 metrů. Tento dosah je ovlivněn faktory jako je prostředí, rušení a výkon vysílače. Navzdory svým výhodám je Bluetooth technologie omezena, například rozsahem a náchylností k interferencím od jiných bezdrátových technologií pracujících ve spektru 2,4 GHz. Přestože Bluetooth efektivně přenáší malé datové balíčky s nízkou latencí, není ideální pro přenos velkých souborů kvůli své omezené rychlosti přenosu, která dosahuje maximálně 2 Mb/s. [32]

V oblasti bezpečnosti se Bluetooth technologie spoléhá na kombinaci šifrování a ověřování, aby zajistila bezpečnou komunikaci mezi zařízeními. Při párování zařízení se provádí procesy, jako je zadávání PIN kódu, což přispívá k ochraně proti nežádoucímu přístupu

nebo neautorizovanému spojení. Bluetooth Low Energy (BLE) přináší další vrstvu bezpečnosti díky pokročilým funkcím, jako jsou BLE Secure Connections. Tyto funkce zahrnují sofistikované metody generování bezpečnostních klíčů, které zvyšují úroveň ochrany dat přenášených mezi zařízeními. [32]

Platforma Android nabízí robustní podporu pro Bluetooth, což umožňuje zařízením vyměňovat data bezdrátově. Prostřednictvím Bluetooth API má aplikace přístup k funkcím, jako je vyhledávání a připojování k jiným zařízením, vytváření spojení a přenos dat. Bluetooth Classic je vhodný pro operace s vyšší spotřebou energie, zatímco Bluetooth Low Energy je ideální pro zařízení s nízkou spotřebou. [35]

1.6.2 NFC

Near Field Communication (NFC) představuje inovativní bezdrátovou technologii umožňující efektivní komunikaci mezi elektronickými zařízeními na krátké vzdálenosti. Tato technologie umožňuje propojení chytrých telefonů, tabletů, platebních karet a dalších zařízení, čímž zvyšuje jejich funkčnost a interoperabilitu. S NFC je možné rychle a jednoduše přenášet data mezi zařízeními prostřednictvím jednoduchého dotyku, ať už se jedná o provádění plateb, výměnu elektronických vizitek nebo sdílení dat. [36], [37]

Tato sofistikovaná technologie není pouze o efektivitě a úspoře finančních prostředků, ale také o vytváření nových možností interakce s okolím, které byly dříve nedosažitelné. NFC je široce dostupné ve většině moderních chytrých telefonů a poskytuje rozsáhlé možnosti využití jak pro vývojáře, tak i pro běžné uživatele. Tato technologie představuje revoluci v propojení elektronických zařízení a přináší značné pohodlí do každodenních životů. [36], [37]

NFC využívá kombinaci čtyř hlavních komponent pro svou funkci: mikročip NFC, fungující jako anténa a přijímač v elektronických zařízeních; čtečka/zapisovač, umožňující NFC zařízením přístup k datům; softwarová aplikace NFC, zpracovávající přijatá data od NFC čipu; a poskytovatel informačních nebo komunikačních služeb (ISP), spravující veškeré komunikace zařízení přes ISP. Z technologického hlediska lze NFC chápat jako pokrok v oblasti RFID, která využívá rádiové vlny pro sledování zásob a zboží. Na rozdíl od pasivních RFID čipů mají NFC mikročipy schopnost ukládat a šifrovat informace. Zatímco RFID zařízení jsou zcela pasivní a nemohou samy přistupovat k datům, NFC umožňuje aktivní interakci s uloženými informacemi. [36]

NFC operuje ve třech různých režimech: čtení/zápis, P2P (peer-to-peer) a emulace karty. [36]

- **Režim čtení/zápis** - V režimu čtení/zápisu funguje NFC zařízení jako aktivní čtečka a zapisovač dat. Čtečka vysílá magnetické pole, které aktivuje pasivní NFC tagy (například v chytrých kartách, štítcích nebo jiných NFC zařízeních) v jejím dosahu. Když je pasivní tag aktivován magnetickým polem čtečky, přenáší svá data zpět na čtečku, která je následně může interpretovat nebo upravit. V režimu zápisu má čtečka také schopnost zapisovat data na pasivní tagy, což může být užitečné například při označování zboží či elektronických vstupenek. [36]
- **Režim peer-to-peer** - V tomto režimu si dvě NFC zařízení vytvářejí spojení mezi sebou bez potřeby externího zařízení, jako je čtečka/zapisovač. Prostřednictvím vzájemné komunikace umožňuje P2P režim přímý přenos dat mezi zařízeními. Tento režim vyžaduje, aby obě zařízení byla schopná aktivní komunikace a podporovala tento režim. [36]
- **Režim emulace karty** - V režimu emulace karty NFC zařízení simuluje chování platební karty nebo jiné fyzické karty. Když je aktivováno v tomto režimu, NFC zařízení vytváří virtuální verzi karty, kterou mohou čtečky karet číst a zpracovávat jako fyzickou kartu. To umožňuje provádět bezkontaktní platby pomocí NFC zařízení, aniž by bylo nutné používat fyzickou platební kartu. Tento režim je často využíván pro mobilní platby a další aplikace, které vyžadují autentizaci nebo autorizaci pomocí karet. [36]

Historie

NFC je známé především jako technologie umožňující platby prostřednictvím mobilních telefonů, jako jsou Google Pay a Apple Pay. NFC má své kořeny v RFID a začalo nabírat na popularitě v roce 2004, kdy Nokia, Philips a Sony společně založily NFC Forum. V roce 2006 byla stanovena oficiální architektura pro technologii NFC, což poskytlo základy pro rozvoj nových spotřebitelských produktů. Od uvedení prvního NFC telefonu od Nokie v roce 2007 až po více než 100 pilotních projektů v telekomunikačním sektoru do roku 2010, a následně až po zavedení systému MTA v New Yorku v roce 2017, který umožňuje platby v metru pomocí NFC, tato technologie postupně nabývá na významu. Je zjevné, že NFC není pouze o platebních službách, jeho využití se rozšířilo i do dalších oblastí,

jako jsou například „chytré“ plakáty, které poskytují informace při přiložení NFC kompatibilního zařízení, či možnost sdílení dat a propojení různých zařízení v domácnosti. [36]

Výhody a nevýhody

Výhody technologie NFC spočívají v její pohodlnosti, především při platebních transakcích. Místo složitého zadávání PINu či manipulace s hotovostí stačí jednoduše přiložit telefon k platebnímu terminálu. NFC také poskytuje možnost zvýšení bezpečnosti zařízení prostřednictvím NFC klíčů, což ztěžuje neoprávněný přístup k citlivým aplikacím. Důvodem, proč se NFC stává stále populárnější, je jeho schopnost propojit fyzická zařízení s minimálním uživatelským nastavením.

Naopak nevýhody technologie NFC zahrnují její relativně vysoké náklady na implementaci pro poskytovatele. Některé společnosti mohou být odrazeny náklady spojenými s distribucí NFC klíčů svým zaměstnancům. Z pohledu uživatele pak existuje riziko zneužití technologie NFC, pokud je telefon odcizen. Bez předchozího nastavení bezpečnostních opatření může být ukradený telefon použit k neoprávněným platbám. Přestože většina zařízení má limity pro bezkontaktní platby, existuje stále riziko finančních ztrát. Nicméně, i přes tyto nevýhody, NFC zůstává relativně bezpečnou a pohodlnou platební možností ve srovnání s jinými metodami jako je čip a PIN. [38]

Zabezpečení

Zabezpečení technologie NFC je klíčovým faktorem, který ovlivňuje široké spektrum interakcí a transakcí v digitálním prostředí. Tato technologie poskytuje uživatelům možnost provádět rychlé a pohodlné platby, sdílet data a usnadňuje propojení zařízení přiložením k platebnímu terminálu či jinému NFC kompatibilnímu zařízení. Avšak, je důležité si uvědomit, že i přes své výhody není NFC bezrizikové. Existuje řada potenciálních hrozeb, které mohou ohrozit bezpečnost uživatelů. Mezi ně patří například riziko zneužití technologie v případě ztráty zařízení, kdy útočníci mohou provádět neoprávněné platby nebo získávat citlivá data. Dalším rizikem je možnost skimmingu, kdy útočníci získávají data z NFC zařízení blízkých uživatelů. Kromě toho může dojít k pokusům o neoprávněný přístup pomocí klonovaných NFC tagů, což může ohrozit bezpečnost dat a systémů. Pochopení těchto možných hrozeb je klíčové pro efektivní řízení bezpečnostních opatření a minimalizaci rizik. Navzdory těmto výzvám se však zabezpečení technologie NFC neu-

stále vyvíjí a zdokonaluje, přičemž stále zůstává relativně spolehlivé a účinné pro většinu digitálních interakcí a transakcí. [38]

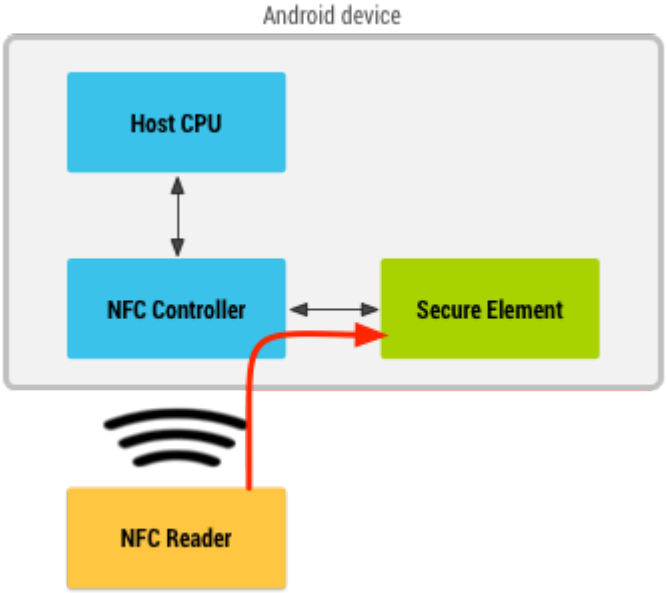
NFC na platformě Android

NFC na platformě Android přináší možnost rychlé a bezdrátové výměny dat na krátké vzdálenosti. Tato technologie, integrovaná do zařízení s operačním systémem Android, umožňuje uživatelům využívat tři hlavní režimy provozu: čtečka/zapisovač, která umožňuje komunikaci s NFC tagy pro čtení a zápis dat, peer-to-peer režim pro přímou výměnu dat mezi NFC zařízeními a režim emulace karty, který umožňuje zařízení předstírat, že je NFC kartou. Díky těmto funkcím uživatelé mohou provádět širokou škálu digitálních interakcí a transakcí, jako jsou bezkontaktní platby, sdílení dat mezi zařízeními a propojení s různými NFC zařízeními, což zlepšuje uživatelský zážitek a rozšiřuje možnosti využití mobilních zařízení s Androidem. [39]

Host-based Card Emulation (HCE) na platformě Android

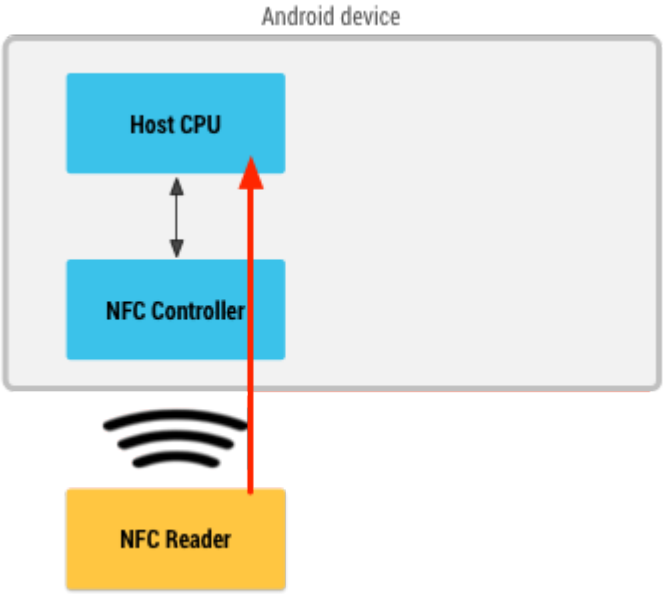
Mnoho zařízení s Androidem a NFC podporuje emulaci karet prostřednictvím speciálního čipu nazývaného secure element. Novější verze Androidu od verze 4.4 nabízejí další metodu emulace karet bez použití tohoto prvku, nazývanou host-based card emulation (HCE). HCE umožňuje aplikacím na Androidu emulovat karty a komunikovat přímo s NFC čtečkou. Tento přístup poskytuje flexibilitu v implementaci NFC aplikací a umožňuje širší využití technologie NFC na zařízeních s Androidem. [39]

Při NFC emulaci karty pomocí secure elementu je karta, která má být emulována, nainstalována do secure elementu na zařízení skrze aplikaci pro Android. Když uživatel přiblíží zařízení k NFC terminálu, řadič NFC v zařízení přenáší veškerá data z čtečky přímo do secure elementu. Samotný secure element následně zajišťuje komunikaci s NFC terminálem a v této transakci se nezapojí žádná aplikace pro Android. Po dokončení transakce může aplikace pro Android přímo dotazovat secure element na stav transakce a informovat uživatele. To lze vidět na obrázku 2. [41]



Obrázek 2: NFC emulace karty pomocí secure elementu. [41]

Při NFC emulaci karty pomocí HCE jsou data přímo směrována k hostitelskému procesoru (CPU) zařízení, namísto toho, aby byla směrována k secure elementu. Tento přístup umožňuje aplikacím na zařízení emulovat karty a komunikovat přímo s čtečkou NFC, aniž by bylo nutné použít secure element. To lze vidět na obrázku 3. [41]



Obrázek 3: NFC emulace karty bez secure elementu. [41]

I když se zdá, že secure element (SE) je bezpečnější, HCE je stále populární. HCE umožňuje aplikacím emulovat karty přímo v operačním systému, což přináší praktické

výhody. Google Pay, který využívá HCE, by neexistoval, kdyby nebylo tohoto způsobu emulace. Konflikty s mobilními operátory vedly k potřebě alternativního řešení, což vyústilo v přijetí HCE. I když HCE není tak bezpečné jako SE, poskytuje dostatečnou úroveň bezpečnosti a širokou dostupnost NFC platebních řešení. [40]

NFC standardy nabízí podporu různých protokolů a existuje několik typů karet, které lze emulovat. Android verze 4.4 a vyšší podporuje emulaci karet založených na specifikaci NFC-Forum ISO-DEP a zpracování APDU podle ISO/IEC 7816-4. Emulace ISO-DEP je povinná pouze nad technologií Nfc-A, zatímco podpora pro Nfc-B je volitelná. Tato zařízení jsou kompatibilní s mnoha běžně používanými protokoly, což zahrnuje například bezkontaktní platební karty. [41]

Architektura HCE na platformě Android je založena na službách, které mohou běžet na pozadí bez nutnosti uživatelské interakce. To přináší výhody zejména pro aplikace, které emulují karty, jako jsou například věrnostní nebo dopravní karty, protože uživatel nemusí spouštět žádnou aplikaci. Při přiložení zařízení k NFC čtečce systém Android automaticky identifikuje, s jakou HCE službou má komunikovat pomocí specifikace ISO/IEC 7816-4, která využívá identifikátory aplikací (AID). Pro existující infrastrukturu čteček jsou AID obvykle dobře známé a veřejně registrované, například AID platebních sítí jako Visa a MasterCard. Pro novou infrastrukturu je doporučeno registrovat vlastní AID podle specifikace ISO/IEC 7816-5, což minimalizuje riziko kolizí s jinými aplikacemi. Na platformě Android se pro implementaci HCE používá třída *HostApduService*. Implementaci *HostApduService* lze vidět v kapitole 3.3.1. [41]

1.6.3 QR kód

QR kód, zkratka pro „Quick Response“, představuje speciální typ čárového kódu, který ukládá data ve formě pixelů v čtvercové mřížce. Tato technologie propojuje fyzický a digitální svět tím, že umožňuje uživatelům rychlý přístup k informacím pomocí jejich mobilních zařízení. QR kódy mají široké využití, od identifikace produktů v dodavatelském řetězci až po marketingové kampaně. Jsou standardizované Mezinárodní organizací pro normalizaci (ISO) a představují pokrok oproti tradičním čárovým kódům, zejména díky své schopnosti ukládat více informací a rychlému skenování. Existují dvě hlavní kategorie QR kódů: statické a dynamické. [42], [43]

Statické QR kódy obsahují pevné informace, které nelze po vytvoření změnit. Jsou ideální pro trvalé údaje, jako jsou například identifikační čísla zaměstnanců nebo přístupové kódy. Na druhé straně dynamické QR kódy umožňují libovolné úpravy uložených informací. To je možné díky tomu, že data nejsou přímo obsažena v kódu, ale uživatele přesměrovávají na specifickou webovou adresu, kterou je možné kdykoli aktualizovat. [43]

Vznik QR kódů

QR kódy, vytvořené ve 90. letech společností Denso Wave, původně sloužily k monitorování automobilů během výrobního procesu. Na rozdíl od tradičních čárových kódů, které vyžadují optické skenování, QR kódy lze snadno dekódovat pomocí chytrých telefonů. Tyto kódy, skládající se z černých čtverců na bílém pozadí, jsou čteny specializovaným softwarem, který identifikuje vzory a extrahuje data. Sice mají vyšší kapacitu a podporují různé typy dat, ale dosud se nestaly tak populárními mezi spotřebiteli, jak se očekávalo. Místo toho jsou častěji spojovány s firemními kampaněmi a marketingem. Nicméně se stále častěji využívají v digitálních platebních systémech a pro přenos webových adres na mobilní zařízení, čímž se stávají důležitým nástrojem propojujícím fyzický a digitální svět. [42]

QR kódy vs. čárové kódy

QR kódy a čárové kódy jsou klíčovými prostředky identifikace produktů a služeb. Čárové kódy, poprvé použité v 70. letech, slouží k jednoduché identifikaci produktů a sledování cen a inventáře v maloobchodě a logistice. Naopak QR kódy, vyvinuté ve 90. letech, umožňují přenést rozsáhlejší informace a propojit fyzické objekty s digitálním obsahem.

Čárové kódy jsou běžně využívány v maloobchodě k sledování cen, inventáře a poskytování informací o produktech. Jsou jednoduché a účinné, ale mají omezenou kapacitu pro ukládání dat a jsou primárně určeny k jednoduché identifikaci produktů. S postupem času byly využívány i v dalších odvětvích, jako je sledování knih v knihovnách nebo sledování balíků v logistice.

Naopak QR kódy, s jejichž využitím začala širší populace až v posledních letech, nabízejí širší možnosti. Jsou schopny uchovat rozsáhlejší množství informací, včetně textu, URL adres, kontaktních údajů nebo dokonce obrázků. Díky tomu jsou často využívány

v marketingových kampaních, propagačních materiálech, digitálních menu v restauracích nebo v propojení offline reklamy s online obsahem.

S rozvojem digitálních technologií a zvyšujícím se využitím mobilních zařízení roste i význam QR kódů. Jsou považovány za klíčový prvek propojení fyzického světa s digitální sférou a jejich využití se stále rozšiřuje do nových oblastí, jako je zlepšení zážitku zákazníků nebo optimalizace logistických procesů. [42]

QR kódy na platformě Android

S nárůstem popularity QR kódů se stále více výrobců rozhoduje integrovat QR skenery přímo do fotoaparátů svých zařízení s Androidem. I když existuje mnoho bezplatných aplikací pro skenování QR kódů ke stažení z obchodu Google Play, jednou z nejlepších možností je využití Google Lens. Tato aplikace nenabízí pouze funkci skenování QR kódů, ale také poskytuje širokou škálu nástrojů pro rozpoznávání a překlad textu. Mnoho aplikací dostupných v obchodě Google Play také umožňuje vytváření vlastních QR kódů, což rozšiřuje jejich využití a dává uživatelům větší flexibilitu při práci s těmito kódy. [44]

Pro implementaci vlastní čtečky nebo generátoru QR kódů v aplikaci existuje několik možností. Jednou z nejčastěji používaných knihoven je ZXing (Zebra Crossing), kterou lze najít v repozitáři na adrese `https://github.com/zxing/zxing`. Tato knihovna poskytuje robustní funkce pro detekci, dekódování a generaci QR kódů. Další možností je využití knihovny ZXing Android Embedded, která staví na ZXing a nabízí hotové řešení čtečky QR kódů, čímž zjednodušuje implementaci čtečky QR kódů v aplikaci. Tuto knihovnu lze nalézt na adrese `https://github.com/journeyapps/zxing-android-embedded`. Alternativně lze pro implementaci vlastní čtečky QR kódů využít strojového učení díky mobilnímu SDK ML Kit. [45]

ML Kit je mobilní sada vývojářských nástrojů vyvinutá společností Google, která umožňuje integraci pokročilých funkcí strojového učení přímo do aplikací pro platformy Android a iOS. Tento nástroj poskytuje rozhraní API, které je možné využít pro řešení široké škály úloh, včetně rozpoznávání objektů na obrázcích nebo analýzy textu. Jednou z hlavních výhod sady ML Kit je její schopnost provádět tyto operace přímo na zařízení, což umožňuje aplikacím zpracovávat data bez nutnosti stabilního internetového připojení. [46]

ML Kit lze použít pro skenování různých formátů čárových kódů. Čárové kódy umožňují přenášet různé druhy informací do aplikací, včetně kontaktních údajů nebo informací o Wi-Fi sítích. ML Kit automaticky rozpoznává a analyzuje tyto kódy, což aplikacím umožňuje reagovat na jejich skenování. Díky podpoře široké škály formátů, včetně lineárních (Codabar, Kód 39, Kód 93, Kód 128, EAN-8, EAN-13, ITF, UPC-A, UPC-E) a 2D (Aztec, Data Matrix, PDF417, QR kód), je ML Kit vhodným nástrojem pro mnoho aplikací. [47]

ML Kit pro to poskytuje rozhraní Vision API, které umožňuje provádět analýzu obrazu v reálném čase pomocí strojového učení přímo na zařízení. Toto rozhraní poskytuje analyzátor *MLKitAnalyzer*, který implementuje rozhraní *ImageAnalysis.Analyzer*. Ten lze jednoduše integrovat do aplikace využívající knihovnu CameraX a využít například pro skenování čárových kódů, detekci obličejů, rozpoznávání textu, apod. Ukázku použití rozhraní Vision API lze vidět v kapitole 3.3.4. [48]

1.6.4 Wi-Fi Direct

Wi-Fi Direct je technologie umožňující přímé spojení mezi zařízeními bez nutnosti existující centrální sítě. Jedno zařízení funguje jako přístupový bod (Access Point), zatímco ostatní zařízení se k němu připojují prostřednictvím standardizovaných protokolů, jako jsou Wi-Fi Protected Setup (WPS) a Wi-Fi Protected Access (WPA/WPA2), což zajišťuje bezpečné a spolehlivé připojení.

V porovnání s tradičními technologiemi, jako je například Bluetooth, disponuje Wi-Fi Direct významně vyšší propustností a rychlostí přenosu dat, což umožňuje efektivnější přenos obsahu s vysokým datovým objemem. Tato vlastnost je klíčová pro situace, kdy je nutné rychle a spolehlivě sdílet rozsáhlé soubory, například multimediální obsah, v prostředích s omezeným přístupem k stabilní Wi-Fi síti.

Flexibilita a univerzálnost patří mezi hlavní výhody Wi-Fi Direct. Tato technologie umožňuje snadné propojení zařízení bez ohledu na dostupnost stávající Wi-Fi infrastruktury, což je užitečné zejména v situacích, kdy není k dispozici stabilní síťové připojení nebo je potřeba přímé komunikace mezi zařízeními.

Díky schopnosti vytvářet samostatnou síť při vyhledávání bezdrátových sítí je zařízení podporující Wi-Fi Direct snadno identifikovatelné. To značně usnadňuje uživatelům pro-

ces připojení a umožňuje rychlou a spolehlivou komunikaci mezi zařízeními bez složité konfigurace. [49]

Podpora Wi-Fi Direct

Wi-Fi Direct, díky aktualizaci Digital Living Network Alliance (DLNA) z roku 2011, nabízí spotřebitelům již přes deset let inovativní způsob přímého propojení zařízení bez nutnosti existující centrální sítě. Tato technologie umožňuje zařízením komunikovat přímo mezi sebou, což je užitečné pro širokou škálu aplikací.

Android zařízení již od verze 2.3 a Apple zařízení od verze iOS 7 podporují Wi-Fi Direct, ačkoli Apple tuto funkci označuje svými vlastními názvy, jako jsou AirDrop a AirPlay. To znamená, že majitelé mobilních zařízení obou hlavních operačních systémů mohou využívat výhod této technologie pro rychlé sdílení dat.

Wi-Fi Direct není omezené jen na mobilní zařízení. Mnoho zábavních technologií, včetně chytrých televizorů, poskytuje možnost přímého připojení pomocí této technologie. Díky Wi-Fi Direct mohou uživatelé snadno streamovat obsah přímo ze svých mobilních zařízení na své televizory či přenášet data bez zbytečného komplikovaného nastavování sítě. Kromě toho i řada periferních zařízení, jako jsou bezdrátová sluchátka s vysokým zvukovým výkonem a tiskárny, využívá Wi-Fi Direct k poskytnutí jednoduchého a spolehlivého bezdrátového připojení. Tato technologie tak nabízí široké možnosti integrace a umožňuje uživatelům pohodlnější a efektivnější používání svých zařízení.

Proces vytváření přímého připojení pomocí Wi-Fi Direct může být různý v závislosti na konkrétním zařízení. Některá vyžadují skenování QR kódu nebo zadání číselného PINu, zatímco jiná vyžadují stisknutí fyzických tlačítek. S ohledem na rostoucí důraz na bezpečnost se stále více zařízení spoléhá na kombinaci těchto metod a méně zařízení se připojuje automaticky. [49]

Zabezpečení

Wi-Fi Direct přináší minimální bezpečnostní výhody ve srovnání s jinými dostupnými metodami připojení, avšak současně s sebou nese i minimální rizika. Při použití této technologie v situaci, kdy je zařízení připojeno k jiné síti, může nastat problém. Existuje možnost, že hackeři mohou přebrat kontrolu nad spojením, zvláště v případě použití zastaralých protokolů, jako je například WPS. Zabezpečení Wi-Fi Direct je proto klíčové

a vyžaduje pečlivou pozornost. Je třeba si uvědomit, že tato technologie může být zneužita k neoprávněnému získání citlivých informací kýmkoli. Doporučuje se provádět pravidelné aktualizace a důsledně uplatňovat bezpečnostní opatření pro minimalizaci rizik spojených s používáním Wi-Fi Direct. [49]

Wi-Fi Direct na platformě Android

Android podporuje technologii Wi-Fi Direct, což je funkce umožňující zařízením vytvářet přímá spojení bez nutnosti připojení k existující síti nebo hotspotu. Tato schopnost je zásadní pro aplikace, které potřebují rychle a snadno komunikovat s nedalekými zařízeními, a to na vzdálenostech, které přesahují možnosti tradičního Bluetooth. Wi-Fi Direct nabízí řadu výhod, které ji činí atraktivní volbou pro bezdrátové spojení. Jednou z nich je podpora šifrování WPA2, která zajišťuje bezpečnost komunikace. Dále umožňuje zařízením vysílat informace o poskytovaných službách, což usnadňuje identifikaci vhodných partnerů pro připojení. Při určování vlastníka skupiny pro síť Wi-Fi Direct bere v úvahu různé faktory, jako je energetický management, uživatelské rozhraní a schopnosti poskytovaných služeb každého zařízení, čímž zajistí efektivní správu síťových zdrojů. Důležité je také zdůraznit, že Android nepodporuje režim Wi-Fi ad-hoc, což posiluje význam Wi-Fi Direct jako alternativní možnosti pro bezdrátové připojení mezi zařízeními. [50]

2 NÁVRH APLIKACE

2.1 Stanovení cílů

Cílem této diplomové práce bylo vytvoření mobilní aplikace pro platformu Android, která umožní uživatelům spravovat vizitky a snadné sdílení. Aplikace by měla umožnit uživatelům intuitivně vytvořit a upravit svou vlastní vizitku s kontaktními informacemi, jako je jméno, pracovní pozice, telefonní číslo, e-mail a adresa. Dále uživatelé by měli mít možnost přizpůsobit vzhled svých vizitek podle svých preferencí, včetně vlastního pozadí pomocí fotografie z galerie zařízení nebo vyfocením nové.

Aplikace by také měla nabízet pohodlné sdílení vizitek mezi uživateli pomocí NFC a QR kódů, což by posílilo propojení mezi uživateli a usnadnilo rychlou výměnu kontaktních informací. Tímto způsobem by se eliminovala potřeba ručního zadávání údajů a uživatelé by se nemuseli obávat ztráty důležitých kontaktů. Díky synchronizaci dat v reálném čase by měli být uživatelé vždy informováni o nejnovějších změnách. Další přidanou funkcí aplikace by měl být kalendář pro jednoduché zaznamenání plánovaných aktivit.

Bezpečnost aplikace by se měla zajistit prostřednictvím přihlašování uživatelů pomocí hesla nebo biometrických senzorů, což efektivně ochrání citlivá data a soukromí uživatele. Tato bezpečnostní opatření by měla představovat klíčový prvek pro vytvoření důvěry v aplikaci ze strany uživatelů, kteří mohou mít jistotu, že jejich osobní údaje jsou v bezpečí.

2.2 Práce s daty

Práce s daty je nedílnou součástí každé aplikace, ať už se jedná o manipulaci s lokálními nebo vzdálenými úložišti. V rámci této aplikace je nutné efektivně pracovat s online daty, která musí být okamžitě dostupná a sdílena mezi všemi uživateli v reálném čase. Proto bylo vhodné využít služby Firebase, konkrétně Cloud Firestore, Realtime Database a Cloud Storage, které poskytují spolehlivé a rychlé uchovávání dat. Tyto služby nejenže zajišťují konzistenci a aktuálnost dat, ale také umožňují uživatelům pohodlný a snadný přístup k jejich datům odkudkoliv, což je nezbytné pro efektivní fungování aplikace. Další informace o těchto službách lze nalézt v kapitole Služby Firebase.

Při vývoji aplikace je důležité myslet i na plynulost zobrazování obrázků a videí, což může výrazně ovlivnit uživatelskou zkušenost. Načítání a zobrazování obrázků může být někdy výzvou, zejména při práci s daty ze vzdáleného úložiště. Proto je vhodné využívat strategie jako je ukládání obrázků do mezipaměti, což může výrazně zlepšit rychlost jejich načítání.

Na platformě Android je jedním z efektivních nástrojů pro načítání a ukládání obrázků do mezipaměti knihovna Coil, která navíc podporuje Jetpack Compose. Tato knihovna umožňuje ukládat obrázky do mezipaměti, což výrazně zrychluje jejich načítání a zlepšuje uživatelský zážitek. Velikost mezipaměti, kterou může Coil využívat, lze specifikovat pomocí metody *newImageLoader()*, která je dostupná implementací rozhraní *ImageLoaderFactory*.

Mezipaměť se skládá ze dvou částí: *MemoryCache*, která ukládá obrázky do paměti RAM a *DiskCache*, která slouží k ukládání obrázků na disk. Obrázky uložené v paměti RAM jsou často používané a mazány, když je paměť vyčištěna, zatímco obrázky uložené na disku jsou obvykle ty s vyšším rozlišením. Velikost těchto mezipamětí lze nastavit buď procentuálně pomocí metody *maxSizePercent()* nebo pevně v bytech pomocí metody *maxSizeBytes()*. Toto lze vidět v ukázkovém kódu níže. [51]

```
class MainApp : Application(), ImageLoaderFactory {

    override fun newImageLoader(): ImageLoader {
        return ImageLoader(this)
            .newBuilder()
            .diskCachePolicy(CachePolicy.ENABLED)
            .diskCache {
                DiskCache
                    .Builder()
                    .directory(cacheDir.resolve("coil_cache"))
                    .maxSizePercent(0.025)
                    .build()
            }
            .memoryCachePolicy(CachePolicy.ENABLED)
            .memoryCache {
                MemoryCache
                    .Builder(this)
                    .maxSizePercent(0.1)
                    .strongReferencesEnabled(true)
                    .build()
            }
            .build()
    }
}
```

2.3 Uživatelské rozhraní

Pro vývoj uživatelského rozhraní jsem využil sadu vývojářských nástrojů Jetpack Compose, která přináší moderní přístup k tvorbě uživatelského rozhraní aplikací pro platformu Android.

Konkrétně jsem v aplikaci použil knihovnu Compose Material 3, která je součástí Jetpack Compose. Material 3 je navržena s ohledem na nejnovější trendy v designu a poskytuje podporu pro pokročilé designové prvky, jako jsou dynamické barvy a další vizuální efekty. Tyto prvky jsou optimalizovány pro Android 12 a novější verze, avšak mnohé z nich jsou použitelné i na starších verzích operačního systému, čímž umožňují vytvářet uživatelsky přívětivá rozhraní pro široké spektrum zařízení.

Některé funkce v Jetpack Compose jsou označeny jako experimentální, což znamená, že jejich chování a implementace se může v budoucnu změnit, nebo mohou být dokonce odebrány z frameworku. Při vývoji uživatelského rozhraní je proto důležité brát tuto skutečnost v úvahu a volit komponenty s ohledem na jejich stabilitu a budoucí podporu.

I když Jetpack Compose nabízí širokou škálu předdefinovaných komponent, může se stát, že je nezbytné si vytvořit vlastní komponentu, která není v rámci frameworku přímo dostupná. V mém případě to byla například komponenta kalendáře. V takových případech je nutné buď implementovat vlastní, nebo vyhledat nějakou externí knihovnu, která tuto komponentu poskytuje.

Navzdory těmto výzvám je Jetpack Compose silným a flexibilním nástrojem pro tvorbu moderních uživatelských rozhraní.

2.4 Navigace

Navigace je nedílnou součástí aplikace, která umožňuje uživatelům se pohybovat mezi jednotlivými obrazovkami.

Pro implementaci navigace jsem použil navigační komponentu z Jetpack Compose. Díky této navigační komponentě jsem mohl jednoduše definovat cesty k jednotlivým obrazovkám aplikace a specifikovat s nimi spojené argumenty. Všechny argumenty jsou automaticky předávány jako textové řetězce, ale v případě složitějších datových typů je nutné definovat jejich serializaci do textových řetězců, například ve formátu JSON.

V rámci bloku každé cesty je možné specifikovat *@Composable* funkci reprezentující danou obrazovku a předat jí potřebné argumenty získané navigačním controllerem. Důležité je také definovat počáteční destinaci navigace, kterou aplikace zobrazí po svém spuštění. V této aplikaci je počáteční obrazovkou přihlašovací obrazovka, kde se uživatel musí autentizovat. Ukázku kódu navigační komponenty lze vidět níže.

```
@Composable
fun AppNavGraph(
    navController: NavHostController = rememberNavController(),
    startDestination: String = AppDestinations.SIGN_IN_ROUTE,
    navActions: AppNavigationActions = remember(navController) {
        AppNavigationActions(navController)
    }
) {
    NavHost(
        navController = navController,
        startDestination = startDestination
    ) {
        composable(
            route = AppDestinations.MY_BUSINESS_CARDS_ROUTE,
            arguments = listOf(
                navArgument(CARD_ID_ARG) {
                    type = NavType.StringType
                    defaultValue = ""
                }
            )
        ) { entry ->
            BackHandler(true) {
                // Close App.
                activity?.finish()
            }

            MyBusinessCardsScreen(
                googleAuthUiClient = googleAuthUiClient,
                navActions = navActions,
                cardIdToAdd = entry.arguments?.getString(CARD_ID_ARG) ?: ""
            )
        }
    }
}
```

3 SPRÁVA VIZITEK

Když nový uživatel poprvé spustí aplikaci, je vyzván k tomu, aby se přihlásil a to z toho důvodu, aby bylo možné určit, kdo bude aplikaci používat. Jedinou aktuálně dostupnou metodou přihlášení je přihlášení přes Google účet. Přihlášený účet si uživatel může změnit a díky tomu je pak schopen si například oddělit svoje osobní kontakty od těch pracovních. Poté, co se uživatel přihlásí, může plně využívat aplikaci. V aplikaci si může vytvořit vlastní digitální vizitku, kterou bude moci sdílet s jinými uživateli, nebo jen získávat vizitky jiných uživatelů.

3.1 Vytváření a úprava

Poté, co se uživatel úspěšně přihlásí do aplikace, má možnost si vytvořit svoji vlastní vizitku. Na první obrazovce, která je nazvána „My Business Cards“, stačí kliknout na kruhový avatar v pravém horním rohu obrazovky s obrázkem z jeho Google účtu. Tím se uživatel dostane na obrazovku „My Profile“. Zde má možnost buď se odhlásit pomocí tlačítka „Sign Out“ v dolní části obrazovky, nebo kliknout na položku „My Card“, která ho přenese na obrazovku „My Business Card“, kde si může vytvořit nebo upravit svoji digitální vizitku.

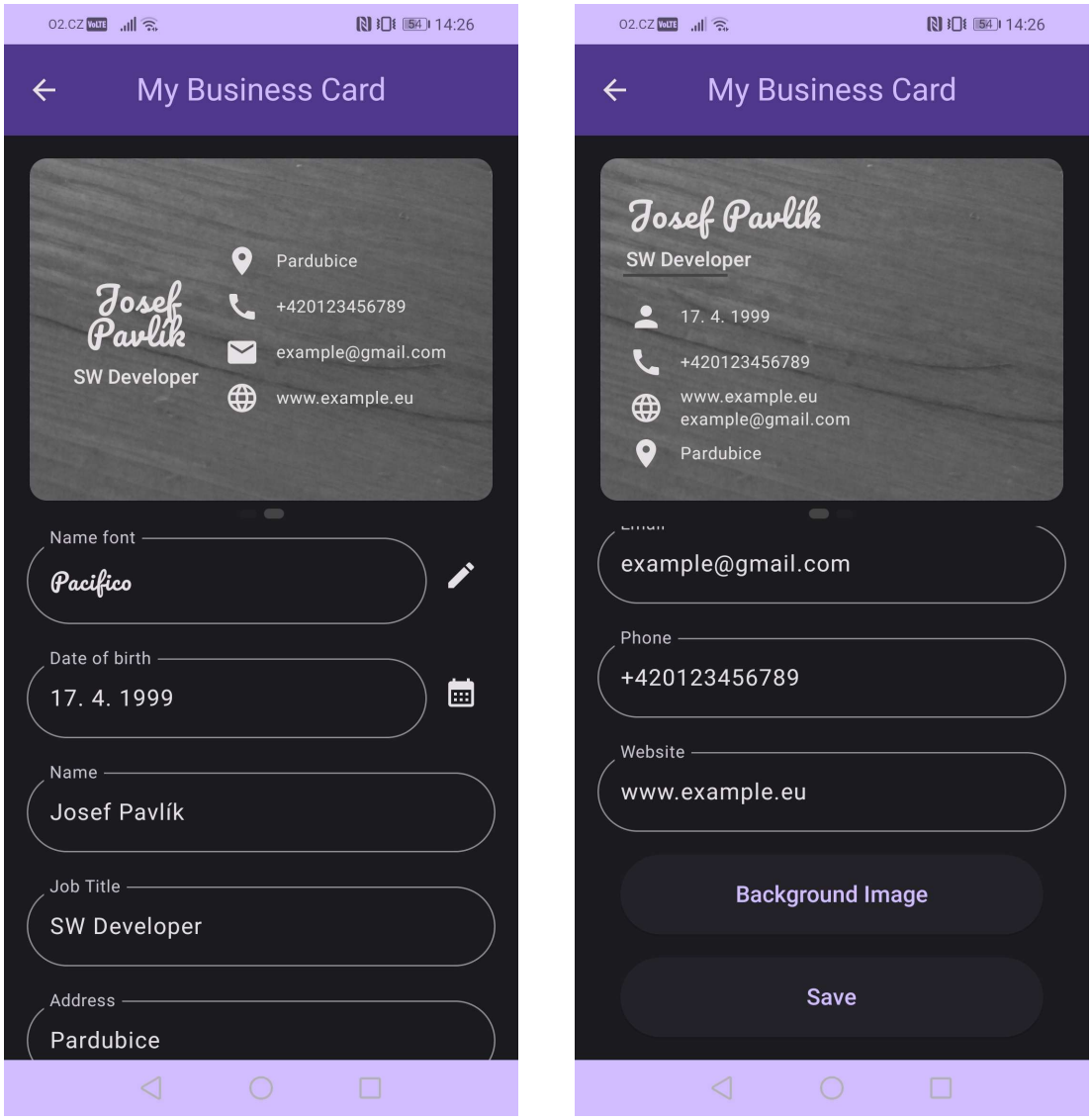
Pro vytvoření vizitky je nutné vyplnit všechna pole s kontaktními informacemi. Na této obrazovce je k dispozici náhled vizitky, který ukazuje, jak bude výsledná vizitka vypadat. Kliknutím na náhled a posunutím doleva nebo doprava může uživatel vybírat mezi různými rozloženími vizitky podle svých preferencí.

Následující pole jsou určena pro zadání konkrétních informací. Při rolování dolů zůstává náhled vizitky stále viditelný, aby se uživatel nemusel stále vracet nahoru. První pole slouží pro změnu fontu jména uživatele na vizitce. Font může buď zůstat základní, tedy „Dosis“, nebo ho uživatel může změnit pomocí tlačítka „Edit“ vpravo, kde dostane na výběr z šesti fontů.

Dalším polem je datum narození, které lze upravit pomocí tlačítka „Calendar“ vpravo. Kliknutím na toto tlačítko si uživatel může vybrat datum svého narození v kalendáři a potvrdit ho tlačítkem „OK“.

Následuje pole „Name“, kde uživatel zadává svoje jméno a příjmení, „Job Title“, kde zadává svoji pracovní pozici. Dalšími poli jsou „Address“ pro adresu, „Phone“ pro telefonní číslo a „Website“ pro odkaz na webovou stránku.

V dolní části obrazovky se nachází dvě tlačítka „Background Image“ a „Save“. Tlačítko „Save“ slouží k uložení aktuálně vytvořené vizitky, ale pouze v případě, že jsou všechna pole správně vyplněna. Důležité je zde taktéž zmínit, že pozadí vizitky je volitelné. K nastavení pozadí vizitky slouží tlačítko „Background Image“, kde si uživatel může upravit pozadí své vizitky dle potřeby.



Obrázek 4: Obrazovka „My Business Card“. Zdroj vlastní.

Pokud si uživatel bude chtít vytvořit svoji vizitku, ale nevyplní všechna pole správně, bude po kliknutí na tlačítko „Save“ upozorněn, aby vyplnil nebo upravil hodnoty polí tak,

aby byly validní. Pokud nebudou všechny potřebné informace vyplněny správně, vizitka nebude vytvořena.

Když dojde k operaci uložení, uživatel je o jejím výsledku informován pomocí dialogového okna, které buď potvrdí úspěch operace, nebo zobrazí chybovou hlášku. Tato dialogová okna jsou také použita pro informování uživatele při načítání nebo při jiných událostech na této obrazovce.

Nakonec bych chtěl dodat, že jakákoli změna hodnot v polích nebo úprava rozložení vizitky či pozadí vyvolá zobrazení dialogového okna, pokud se uživatel pokusí opustit obrazovku bez uložených změn. Opustit obrazovku lze pomocí horní navigace v aplikaci nebo spodní systémové navigace. Toto okno upozorní uživatele, že existují neuložené změny, a zeptá se, zda chce pokračovat bez uložení.

3.1.1 Pozadí vizitky

Uživatel má možnost mít na své vizitce své vlastní pozadí dle svých preferencí a stylu. Tuto možnost může využít tak, že si vybere fotografii z galerie svého zařízení nebo provede pořízení nové fotografie přímo v aplikaci. Pokud uživatel nechce použít vlastní fotografii jako pozadí, může tuto možnost úplně vynechat a zvolit prázdné pozadí.

V případě, že uživatel vybírá fotografii z galerie nebo pořizuje novou fotografii, je nutné zajistit, aby její rozměry odpovídaly poměru stran vizitky, který je 73:52. Pro dosažení optimálního vzhledu je nezbytné provést případné oříznutí fotografie.

Kromě toho má uživatel také možnost libovolně otočit fotografii pouze o násobky 90 stupňů a případně ji převést do odstínů šedi. Tyto úpravy však nejsou povinné a jsou plně volitelné v souladu s estetickými preferencemi uživatele a s tím, jakého vzhledu vizitky si přeje dosáhnout. Tímto způsobem si tedy může každý uživatel vytvořit vizitku, která nejen vystihuje jeho profesionální identitu, ale také reflektuje jeho osobní styl a kreativitu.

Oprávnění aplikace

Pro získání přístupu k fotografiím z galerie zařízení nebo k pořízení nových snímků pomocí fotoaparátu je nezbytné, aby uživatel udělil aplikaci příslušná oprávnění. Bez těchto oprávnění není možné využívat tyto funkce aplikace. Pokud uživatel odmítne udělit aplikaci potřebná oprávnění, nemůže plně využívat všechny dostupné možnosti aplikace. Je důležité si uvědomit, že v různých verzích Androidu se požadovaná oprávnění mohou lišit.

Všechna nezbytná oprávnění, která aplikace potřebuje, musí být specifikována v XML souboru „AndroidManifest“. Požádání o tato oprávnění se provádí prostřednictvím dialogového okna, které aplikace zobrazí ve vhodný moment. Uživatel může poté rozhodnout, zda oprávnění udělí či nikoliv.

Oříznutí pozadí

Po výběru fotografie z galerie zařízení nebo pořízení snímku fotoaparátem je uživatel přesunut na obrazovku určenou k oříznutí fotografie. Tento krok je nezbytný pro zajištění odpovídajícího poměru stran fotografie pozadí vizitky. Na této obrazovce jsou k dispozici následující prvky: Uživatel se může pomocí šipky zpět v horní části obrazovky nebo prostřednictvím spodní systémové navigace vrátit zpět a zrušit celý proces tvorby nového pozadí vizitky. Dále se zde v pravém horním rohu aplikace nachází tlačítko pro otáčení, které umožňuje otáčet fotografii o násobky devadesáti stupňů. Vedle tlačítka pro otáčení se nachází tlačítko „Crop“, které slouží k provedení oříznutí vybrané oblasti. Oblast k oříznutí lze vybrat pomocí obdélníku, který lze přemisťovat pomocí prstů. Tento obdélník má pevný poměr stran 73:52 a umožňuje pouze horizontální orientaci.

Pro tuto funkcionalitu byla použita knihovna Android Image Cropper, která je dostupná ve veřejném repozitáři `https://github.com/CanHub/Android-Image-Cropper`. Použití této knihovny je velmi intuitivní a pro svůj chod nevyžaduje žádné další oprávnění aplikace. Pro implementaci je nejprve nutné inicializovat spouštěč (launcher) pro kontrakt, který usnadňuje předávání dat mezi jednotlivými aktivitami. Existuje sada předdefinovaných tříd kontraktů, které je možné použít, ale je možné si definovat i vlastní. Tato knihovna poskytuje vlastní kontrakt *CropImageContract()*, který umožňuje získat cestu k oříznuté fotografií. Pro spuštění aktivity stačí nad spouštěčem zavolat funkci *launch()*. Tato funkce přijímá jako argument třídu *CropImageContractOptions()*, ve které se specifikuje cesta ke zdrojové fotografii, a třída *CropImageOptions()*, která umožňuje upravit chování ořezávání fotografie. V argumentech třídy *CropImageOptions()* lze například určit pevný poměr stran, tvar výřezu s orientací, konkrétní poměr stran, apod. Po dokončení oříznutí lze ve spouštěči definovat další kroky, které mají být provedeny podle toho, jak tato operace dopadla. Ukázku tohoto kódu lze vidět níže.

```
1 val imageCropLauncher = rememberLauncherForActivityResult(
2     contract = CropImageContract()
3 ) { result ->
```

```
    if (result.isSuccessful) {
        result.uriContent?.let { uri ->
            backgroundImageUri = uri
        }
    } else {
        // Something went wrong.
    }
}
val cropOptions = CropImageContractOptions(
    uri = sourceImageUri,
    cropImageOptions = CropImageOptions(
        aspectRatioX = 292,
        aspectRatioY = 208,
        cropShape = CropImageView.CropShape.RECTANGLE_HORIZONTAL_ONLY,
        fixAspectRatio = true,
    )
)

imageCropLauncher.launch(cropOptions)
```

Pozadí v odstínech šedi

Tlačítko „Background Image“ neslouží jen pro výběr fotografie z galerie zařízení nebo pro pořízení nového snímku fotoaparátem. Po jeho stisknutí se otevře dolní panel, kde je možné najít také položku „Apply Grayscale“, ale pouze v případě, že je nějaké pozadí nastaveno. Tato položka umožňuje po kliknutí převést pozadí vizitky do odstínů šedé barvy.

Proces úpravy barev začíná načtením obrázku z pozadí a jeho převodem na bitmapový obrázek. Poté je vytvořena nová instance bitmapového obrázku, která má stejné rozměry v pixelech jako původní obrázek, a slouží jako cílová pro upravené barvy. Samotná úprava probíhá postupně pixel po pixelu: Každý pixel ze zdrojového obrázku je konvertován do odstínů šedi a následně zapsán do výsledné bitmapy. Jakmile je úprava všech pixelů dokončena, výsledný obrázek je uložen ve formátu JPEG a použit jako nové pozadí vizitky.

Odebrání pozadí

Poslední položkou v dolním panelu, který se zobrazí po stisknutí tlačítka „Background Image“, je položka „Delete“. Tato položka se však zobrazí pouze v případě, pokud je aktuálně nějaké pozadí nastaveno. Slouží k úplnému odstranění pozadí vizitky, ať už se jedná o pozadí v původních barvách nebo ve stupních šedi.

Chování této funkce je následující: Pokud si uživatel vytváří novou vizitku nebo upravuje již existující, ale předtím neměl nastavené žádné pozadí, tak stisknutím tlačítka „Delete“ dojde pouze k odstranění reference na dané pozadí a k odstranění souboru z mezipaměti. Následné stisknutí tlačítka „Save“ žádné operace s ohledem na pozadí neprovede.

V případě, že již existuje nějaké pozadí, stisknutím tlačítka „Delete“ se provede stejná operace jako v předchozím případě. Skutečná změna nastává, až když uživatel přejde do fáze, kdy chce uložit provedené změny pomocí tlačítka „Save“.

V této fázi je nutné provést aktualizaci dat a řešit následující: Aby bylo možné sdílet pozadí vizitek mezi uživateli, musí být tyto obrázky někde uloženy. Pro tento účel se v aplikaci využívá služba Cloud Storage od Firebase, která umožňuje ukládat soubory.

Pokud by se při každé úpravě pozadí vizitky uživatele nahrávalo do tohoto úložiště, docházelo by k hromadění nepoužívaných souborů. Proto je to v aplikaci implementováno tak, že po stisknutí tlačítka „Save“ se nejprve odstraní staré pozadí z úložiště a až poté se uloží nové. Tímto způsobem je dosaženo efektivnějšího využití dostupných zdrojů.

3.2 Ukládání dat vizitek

Uchovávání uživatelských dat je klíčovou součástí této aplikace. Uživatelská data musí být vždy a snadno dostupná. Z tohoto důvodu bylo použito několik služeb poskytovaných Firebase. Mezi tyto služby patří Cloud Firestore, Cloud Storage, Realtime Database a Authentication. Pro uživatele je klíčová zejména služba Authentication, která zajišťuje bezpečné přihlášení a ochranu jejich účtů.

Celý proces začíná již při prvním přihlášení uživatele do aplikace. Pomocí služby Authentication je uživatel identifikován a poté jsou o něm získány základní informace, jako je emailová adresa, uživatelské jméno a unikátní identifikátor účtu. Tyto údaje jsou následně použity k vytvoření uživatelského profilu, který je základem pro další interakci s aplikací.

Každý uživatel má možnost si vytvořit svoji vlastní vizitku nebo získávat vizitky od ostatních uživatelů. Implementace této funkcionality je zajištěna pomocí Cloud Firestore databáze, která uchovává data ve dvou hlavních kolekcích: „users“ a „businessCards“. V kolekci „users“ jsou v dokumentech uloženy informace o jednotlivých uživatelích, obsahující jejich uživatelské jméno a seznam vizitek, které získali od ostatních uživatelů. Kolekce „businessCards“ pak obsahuje samotné vizitky spolu s kontaktními informacemi a odkazy na obrázky pozadí, které jsou uchovávány v Cloud Storage.

Poslední službou je Realtime Database, která umožňuje uchovávat události, které si uživatelé mezi sebou naplánovali. Zde se nachází kolekce „events“. Záznamy v této kolekci obsahují informace, jako jsou unikátními identifikátory vizitek uživatelů, popisy událostí a časy, na které je tyto události naplánovány.

Díky těmto službám jsou data uživatelů přístupná odkudkoli a kdykoli, pokud je uživatel připojen k síti.

3.2.1 Použití Cloud Firestore

Následující kód slouží k uložení vizitky uživatele do databáze. Funkce *saveMyBusinessCard()* přijímá datovou třídu *BusinessCardDto* obsahující informace o nově vytvořené vizitce uživatele. Nejprve se zde získává identifikátor aktuálně přihlášeného uživatele pomocí služby Authentication. Následně se provede nahrání obrázku pozadí vizitky do úložiště Cloud Storage. Poté se provede uložení všech informací o vizitce, včetně osobních údajů a URL adresy obrázku, do kolekce „businessCards“ v Cloud Firestore databázi.

```
fun saveMyBusinessCard(businessCard: BusinessCardDto) {
    googleAuthUiClient.getSignedInUser()?.userId?.let { id ->
        uploadImageToStorage(
            businessCard.backgroundImage.toUri()
        ) { imageUrl, fileName ->
            val businessCardToSave = hashMapOf(
                "ownerId" to id,
                "style" to businessCard.style,
                "dateOfBirth" to Timestamp(
                    Date.from(
                        businessCard.dateOfBirth.atZone(
                            ZoneId.systemDefault()
                        ).toInstant()
                    )
                ),
                "name" to businessCard.name,
                "nameFont" to businessCard.nameFont,
                "jobTitle" to businessCard.jobTitle,
                "address" to businessCard.address,
                "email" to businessCard.email,
                "phone" to businessCard.phone,
                "website" to businessCard.website,
                "backgroundImage" to imageUrl,
                "backgroundImageFileName" to fileName
            )
            val businessCardRef = db.collection("businessCards")
                .document(businessCard.id)

```

```
            businessCardRef.set(businessCardToSave)
                .addOnSuccessListener { /* Success */ }
                .addOnFailureListener { /* Error */ }
        }
    }
}
```

3.2.2 Použití Cloud Storage

Následující kód slouží k nahrání obrázku pozadí vizitky do úložiště. Funkce *uploadImageToStorage()* přijímá URI obrázku k uložení a callback funkci. Pokud je URI obrázku platné, vygeneruje se nový název souboru a tento soubor, který se nachází na cestě *imageUri*, se uloží do Cloud Storage. Po úspěšném uložení se provede callback funkce, které je předána URL adresa, kde je soubor uložen v Cloud Storage, a jméno nového souboru.

```
private fun uploadImageToStorage(
    imageUri: Uri,
    callback: (String, String) -> Unit
) {
    val uriString = imageUri.toString()
    if (uriString.isNotBlank()) {
        val imageFileName = UUID.randomUUID().toString() + ".jpg"
        val imageRef = storage.reference.child("images/$imageFileName")

        imageRef.putFile(imageUri)
            .addOnSuccessListener { _ ->
                imageRef.downloadUrl
                    .addOnSuccessListener { imageUrl ->
                        callback(imageUrl.toString(), imageFileName)
                    }
            }
            .addOnFailureListener { e ->
                callback("", "")
            }
    } else {
        callback("", "")
    }
}
```

3.2.3 Bezpečnost uživatelských dat

Bezpečnost uživatelských dat je nezbytnou součástí každé moderní aplikace. V rámci této aplikace muselo být zajištěno to, že pouze vlastník vizitky má možnost si ji upravit, zatímco ostatní uživatelé ji mohou pouze prohlížet. Toto je zajištěno pomocí služby Au-

thentication, která umožňuje uživatelům se pohodlně a bezpečně přihlašovat do aplikace prostřednictvím svého Google účtu.

Po úspěšném přihlášení má uživatel plnou kontrolu nad svými daty. Pouze on má oprávnění k úpravě informací na své vizitce, což je klíčový prvek důvěry, který aplikace poskytuje svým uživatelům.

Výhoda integrace služby Authentication ve Firebase spočívá v možnosti zabezpečit přístup k datům napříč různými službami. Každá služba v rámci Firebase poskytuje možnost definovat bezpečnostní pravidla. Tato pravidla určují, kdo má právo na provádění konkrétních operací a k jakým datům mají uživatelé přístup. Tímto způsobem jsou data chráněna před neoprávněným použitím či zneužitím.

Více informací o bezpečnostních pravidlech služeb Firebase lze nalézt v kapitole 1.5.

3.3 Sdílení

Sdílení informací je klíčovou součástí moderních aplikací, které se zaměřují na snadnou a efektivní výměnu dat mezi uživateli, a to jak online, tak offline na místě. Tato aplikace se primárně zaměřuje na přenos dat na krátkou vzdálenost bez nutnosti použití internetu. Pro tuto funkcionalitu využívá dvě hlavní technologie: NFC a QR kódy, které umožňují přenos dat i pro zařízení, která NFC nepodporují.

I přes to, že tyto technologie nemusí být ideální pro přenos dat na krátké vzdálenosti, tato aplikace slouží jako prostředek pro výzkum možnosti využití těchto technologií. Cílem je zjistit, zda je možné efektivně využít NFC a QR kódy pro sdílení informací.

Implementace NFC a QR kódů v této aplikaci umožňuje uživatelům snadno sdílet své vizitky s ostatními bez zdlouhavých procesů. Aplikace se snaží být co nejintuitivnější, aby uživatelé mohli rychle a jednoduše využívat možnosti sdílení.

V následujících kapitolách se detailněji zaměřím na implementaci NFC a QR kódů v této aplikaci. Popíši jejich funkce, výhody a možnosti, které poskytují uživatelům pro efektivní sdílení svých vizitek.

3.3.1 NFC v režimu emulace karty

Jak již bylo zmíněno v kapitole 1.6.2, platforma Android podporuje několik režimů provozu NFC. Tato aplikace využívá režim HCE (Host-based Card Emulation) pro sdílení vizitek, kdy zařízení funguje jako fyzická karta.

HCE na platformě Android pracuje jako služba provádějící operace na pozadí, aniž by bylo nutné interakce uživatele. V případě, že je vyžadováno potvrzení, služba zobrazí odpovídající uživatelské rozhraní.

Důležité je si však uvědomit, že na pozadí zařízení může být spuštěno několik služeb využívajících HCE. Je nutné identifikovat, s kterou službou je třeba pracovat. K tomu slouží AID (Application ID), což je identifikátor specifikující konkrétní aplikaci.

Implementace v aplikaci

Pro fungování HCE se využívá třídy *HostApduService*. V této třídě se implementují dvě klíčové funkce *processCommandApdu()* a *onDeactivated()*, které definují jak se daná služba bude chovat. Kromě těchto funkcí je možné implementovat i další funkce podle potřeby.

Funkce *processCommandApdu()* je volána při obdržení APDU (Application Protocol Data Unit) příkazu, na který pak následně reaguje. V tomto případě tato funkce čeká na příkaz *SELECT*, na který odpoví odesláním unikátního identifikátoru vizitky uživatele.

Další implementovanou funkcí je funkce *onStartCommand()*, která je spuštěna při každém novém spuštění této služby. Skrze tuto funkci jsou předávána data určená ke sdílení.

Poslední funkce *onDeactivated()* je volána, pokud AID v APDU příkazu se neshoduje s AID definovaným v metadatech služby, nebo pokud dojde k přerušení komunikace.

Aby bylo vůbec možné tuto službu spustit, je třeba prvně specifikovat aplikaci oprávnění pro její spuštění a hardware, který vyžaduje ke svému chodu. Dalším krokem je pak definování služby samotné v XML souboru „AndroidManifest“, kde se definuje služba s podporovanými AID.

Potřebný hardware a oprávnění aplikace lze specifikovat v XML souboru „AndroidManifest“ následovně:

```
1 <uses-feature
2     android:name="android.hardware.nfc.hce"
3     android:required="true" />
4
5 <uses-permission android:name="android.permission.NFC" />
```

Definice služby v XML souboru „AndroidManifest“ vypadá následovně:

```
<service
    android:name=".app.MyHostApduService"
    android:exported="true"
    android:permission="android.permission.BIND_NFC_SERVICE">
    <intent-filter>
        <action android:name="android.nfc.cardemulation
        .action.HOST_APDU_SERVICE" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <meta-data
        android:name="android.nfc.cardemulation.host_apdu_service"
        android:resource="@xml/apduservice" />
</service>
```

Metadata HCE služby v XML souboru „apduservice“ vypadají následovně:

```
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/service_description"
    android:requireDeviceUnlock="false">
    <aid-group
        android:category="other"
        android:description="@string/subscription">
        <aid-filter android:name="F0123123123123" />
    </aid-group>
</host-apdu-service>
```

Implementace třídy *HostApduService* může být následující:

```
class MyHostApduService : HostApduService() {

    private var cardId: String = ""

    companion object {
        private const val AID = "F0123123123123"
        private val STATUS_SUCCESS = byteArrayOf(
            0x90.toByte(), 0x00.toByte()
        ) // 9000 - Command successfully executed (OK).
        private val STATUS_ERROR_FILE_NOT_FOUND = byteArrayOf(
            0x6A.toByte(), 0x82.toByte()
        ) // 6A82 - File not found.
        private const val SELECT_APDU_HEADER_HEX = "00A40400"
        val SELECT = NfcUtils.hexStringToByteArray(
            SELECT_APDU_HEADER_HEX +
                    String.format("%02X", AID.length / 2) +
                    AID
        )
    }
```

```
20
21      override fun onStartCommand(
22          intent: Intent, flags: Int, startId: Int
23      ): Int {
24          if (intent.hasExtra("cardId")) {
25              cardId = intent.getStringExtra("cardId") ?: "Unknown"
26          }
27
28          return START_STICKY
29      }
30
31      override fun processCommandApdu(
32          commandApdu: ByteArray?, extras: Bundle?
33      ): ByteArray {
34          if (SELECT.contentEquals(commandApdu)) {
35              val id = "cardId:$cardId".toByteArray(charset("UTF-8"))
36              val result = ByteArray(id.size + STATUS_SUCCESS.size)
37
38              System.arraycopy(id, 0, result, 0, id.size)
39              System.arraycopy(
40                  STATUS_SUCCESS, 0, result, id.size, STATUS_SUCCESS.size
41              )
42
43              return result
44          }
45
46          return STATUS_ERROR_FILE_NOT_FOUND
47      }
48
49      override fun onDeactivated(reason: Int) { }
50  }
```

Službu lze spustit následovně:

```
1  fun startService(context: Context, cardId: String) {
2      val intent = Intent(context, MyHostApduService::class.java)
3      intent.putExtra("cardId", cardId)
4      context.findActivity().startService(intent)
5  }
```

Službu lze pozastavit následovně:

```
1  fun stopService(context: Context) {
2      val intent = Intent(context, MyHostApduService::class.java)
3      context.findActivity().stopService(intent)
4  }
```

3.3.2 NFC čtečka

Dalším režimem, který platforma Android podporuje pro NFC, je režim čtení (NFC Reader Mode). Tento režim umožňuje zařízení číst různé typy NFC značek. V této aplikaci lze tohoto režimu využít k získání vizitky jiného uživatele prostřednictvím NFC.

Pro implementaci NFC čtečky je nejprve získán NFC adaptér pomocí metody *getDefaultAdapter()*. Poté je vytvořena instance *ReaderCallback*, která definuje akce, které se mají provést při zaznamenání NFC značky. Navíc je zde implementována možnost aktivovat a deaktivovat režim čtení pomocí NFC čtečky přímo v aplikaci.

Následující kód ukazuje použití NFC čtečky v aplikaci pro získání vizitek jiných uživatelů:

```
val activity = context.findActivity()
val nfcAdapter = NfcAdapter.getDefaultAdapter(activity)
val callback = NfcAdapter.ReaderCallback { tag ->
    val isoDep = IsoDep.get(tag)

    try {
        isoDep?.connect()

        val response = isoDep?.transceive(MyHostApduService.SELECT)
        val stringResponse = String(response ?: ByteArray(0), Charsets.UTF_8)

        when {
            stringResponse.length >= 2 -> {
                val stringWithoutLastTwoCharsOfStatus = stringResponse
                    .substring(0, stringResponse.length - 2)

                if (stringWithoutLastTwoCharsOfStatus.startsWith("cardId:")) {
                    stringWithoutLastTwoCharsOfStatus.substring(7) // cardId
                }
            }
        }
    } catch (e: Exception) {
        // Error.
    } finally {
        isoDep?.close()
    }
}

nfcAdapter?.enableReaderMode(
    activity,
    callback,
    NfcAdapter.FLAG_READER_NFC_A or NfcAdapter.FLAG_READER_SKIP_NDEF_CHECK,
    null
)
```

```
35
36  nfcAdapter?.disableReaderMode(activity)
```

Po navázání spojení s NFC značkou pomocí instance *IsoDep* se odesílá APDU příkaz *SELECT*, který slouží k identifikaci a výběru konkrétní aplikace na NFC značce. Jakmile dorazí odpověď z NFC značky, provádí se její zpracování. V tomto případě se očekává, že tato odpověď obsahuje identifikátor vizitky ve formátu „cardId:XYZ“. V případě, že tato podmínka je splněna, dochází k extrakci samotného identifikátoru vizitky z odpovědi.

3.3.3 Generování QR kódu

Pro generování QR kódu byla využita knihovna ZXing (Zebra Crossing) s otevřeným zdrojovým kódem, dostupná ve veřejném repozitáři `https://github.com/zxing/zxing`. Tato knihovna umožňuje skenování a generování různých druhů čárových kódů.

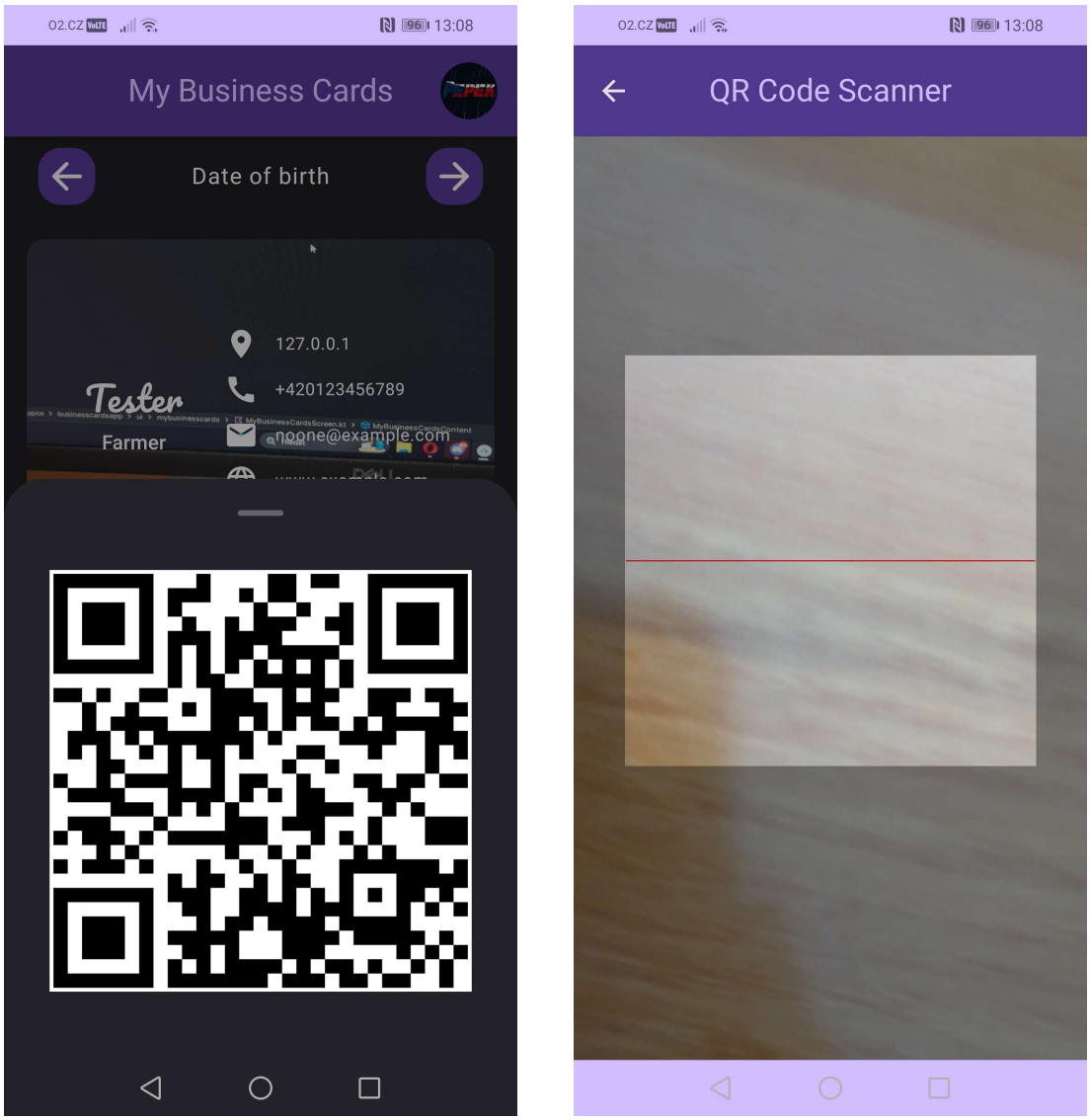
V rámci aplikace jsem tuto knihovnu primárně využil pro generování QR kódu, který slouží ke sdílení vizitky uživatele prostřednictvím unikátního identifikátoru. Jakmile je uživatel přihlášen a má vytvořenou svoji vizitku, může na obrazovce „My Business Cards“ vpravo dole kliknout na tlačítko „+“, čímž se otevře spodní panel. Zde má možnost vybrat volbu „Share Card Using QR Code“, což otevře další spodní panel s vygenerovaným QR kódem pro sdílení vizitky, to lze vidět na obrázku 5 vlevo. Tento QR kód může jiný uživatel naskenovat pomocí aplikace a tak získat vizitku tohoto uživatele.

Proces generování QR kódu spočívá ve vytvoření matice bitů, kde jsou označeny pixely, které mají být vybarveny, na základě konkrétního obsahu. Hodnoty této bitové matice jsou použity pro vytvoření nového bitmapového obrázku, který pak představuje samotný QR kód pro sdílení.

3.3.4 Skenování QR kódu

Aby mohl uživatel získat vizitku jiného uživatele přes QR kód, je potřeba implementovat QR skener. Ten pro svou funkci využívá kameru zařízení k zachycení snímků, které následně zpracovává a extrahuje z nich obsažená data. Existuje několik způsobů skenování, včetně možnosti využití knihovny ZXing, která byla použita i pro generování QR kódů, a dalších podobných knihoven.

V této práci jsem zkoušel dva způsoby skenování QR kódů. Prvním z nich je použití již existujícího QR skeneru, který zpracuje zachycený QR kód a vrátí z něj extrahovaná data. K tomu byla použita knihovna ZXing Android Embedded, která je dostupná ve veřejném repozitáři `https://github.com/journeyapps/zxing-android-embedded`. Jak již název této knihovny napovídá, tato knihovna je postavena na dekodéru z knihovny ZXing. Druhým způsobem je vlastní implementace skeneru s použitím Jetpack knihovny CameraX pro práci s kamerou a rozhraním Vision API z SDK ML Kit.



Obrázek 5: Obrazovka „My Business Cards“: Sdílení s QR kódem. Zdroj vlastní.

V současné verzi aplikace je pro uživatele dostupná pouze verze skeneru využívající knihovnu ZXing Android Embedded. Tuto verzi jsem se rozhodl použít, protože byla jednodušší na implementaci a měla menší prostor pro chyby ve srovnání s druhou variantou.

Nicméně v této kapitole budou představeny obě varianty pro srovnání a lepší porozumění možností konfigurace a implementace.

Když je uživatel přihlášen do aplikace a chce získat vizitku jiného uživatele, může na obrazovce „My Business Cards“ vpravo dole kliknout na tlačítko „+“, čímž se otevře spodní panel. Zde má možnost vybrat volbu „Get Card Using QR Code“, což uživatele přenese na obrazovku „QR Code Scanner“, kterou lze vidět na obrázku 5 vpravo. Na této obrazovce pak stačí jednoduše zamířit zařízením na QR kód jiného uživatele. Po úspěšném zachycení a zpracování QR kódu se uživatel automaticky vrátí zpět na obrazovku „My Business Cards“. Zde bude požádán, zda si přeje tuto vizitku přidat do svého seznamu, nebo zda ji chce zahodit.

Použití knihovny ZXing Android Embedded

Pro použití této knihovny je nezbytné, aby uživatel aplikaci udělil oprávnění k používání fotoaparátu, bez kterého nelze provádět skenování QR kódů. Knihovna nabízí několik způsobů, jak integrovat skener do aplikace. Jedním z nich je vytvoření instance *CompoundBarcodeView* pro zobrazení náhledu skenování. Tato instance je nastavena tak, aby správně řídila skenování a zpracování výsledků. Důležité je však zajistit, aby skener byl aktivní pouze v době, kdy neprobíhá zpracování žádného jiného QR kódu. K tomu lze využít například proměnnou *scanFlag*, která indikuje stav skenování. Nakonec je instance *CompoundBarcodeView* použita v komponentě *AndroidView* pro zobrazení náhledu skenování přímo v uživatelském rozhraní aplikace. Při použití této knihovny je však nutné dbát na správný životní cyklus skeneru v aplikaci, včetně inicializace a ukončení skenování.

```
val context = LocalContext.current
var scanFlag by remember { mutableStateOf(false) }
val compoundBarcodeView = remember {
    CompoundBarcodeView(context).apply {
        val capture = CaptureManager(context as Activity, this)
        capture.initializeFromIntent(context.intent, null)
        capture.decode()
        this.decodeContinuous { result ->
            if (scanFlag) { return@decodeContinuous }
            scanFlag = true
            result.text?.let { qr ->
                if (qr.startsWith("cardId:")) {
                    navActions.navigateToMyBusinessCards(
                        cardId = qr.substring(7)
                    )
                }
```

```
                scanFlag = false
            }
        }
    }
}

DisposableEffect(key1 = Unit) {
    compoundBarcodeView.resume()
    onDispose { compoundBarcodeView.pause() }
}

AndroidView(
    modifier = Modifier,
    factory = { compoundBarcodeView },
)
```

Použití knihovny CameraX s ML Kit Vision API

Podobně jako u knihovny ZXing Android Embedded je zde nezbytné, aby uživatel udělil aplikaci oprávnění k používání fotoaparátu. Hlavním rozdílem je, že v tomto případě je potřeba si vytvořit vlastní skener, což lze realizovat pomocí Jetpack knihovny CameraX.

CameraX nabízí široké možnosti pro práci s fotoaparátem zařízení a umožňuje přenášet obraz přímo do uživatelského rozhraní aplikace, a to pomocí komponenty *PreviewView*. Jednou z hlavních výhod této knihovny je možnost přiřazení analyzátoru pro detekci objektů v obraze pomocí rozhraní *ImageAnalysis.Analyzer*.

Postup implementace začíná výběrem kamery a strategie pro výběr obrázků k analýze. Následně je specifikován analyzátor, který má za úkol provádět detekci objektů. V tomto případě se jedná o třídu *QRCodeAnalyzer*, která implementuje rozhraní *ImageAnalysis.Analyzer* a slouží k detekci QR kódů v obraze. Tento analyzátor funguje tak, že analyzuje obrázek pouze ve vymezené oblasti, kterou definuje obdélník *targetRect*. Je důležité poznamenat, že pozice tohoto obdélníku a jeho velikost se může měnit v závislosti na konkrétním zařízení.

V uživatelském rozhraní aplikace je zobrazen přenesený obraz z fotoaparátu pomocí komponenty *PreviewView*. Nad touto komponentou je pak přidána další vrstva, která určuje oblast, ve které se provádí hledání QR kódů. To umožňuje uživateli snadněji zaměřit QR kód a využívat fotoaparát zařízení ve spojení s ML Kit Vision API pro detekci QR kódů v reálném čase a přímo v aplikaci.

Kód pro detekci a zpracování QR kódů ve vymezené oblasti může vypadat takto:

```
scanner.process(inputImage)
    .addOnCompleteListener {
        if (it.isSuccessful) {
            it.result.let { barcodes ->
                barcodes.forEach { barcode ->
                    barcode.boundingBox?.transform { scaledBound ->
                        scaledBound.offset(-marginX, -marginY)
                        if (targetRect.toRectF().contains(scaledBound)) {
                            barcode.rawValue?.let(onQRCodeScanned)
                        }
                    }
                }
            }
        } else {
            it.exception?.let { exception -> /* Handle error. */ }
        }
        image.close()
    }
```

3.4 Kalendář s událostmi

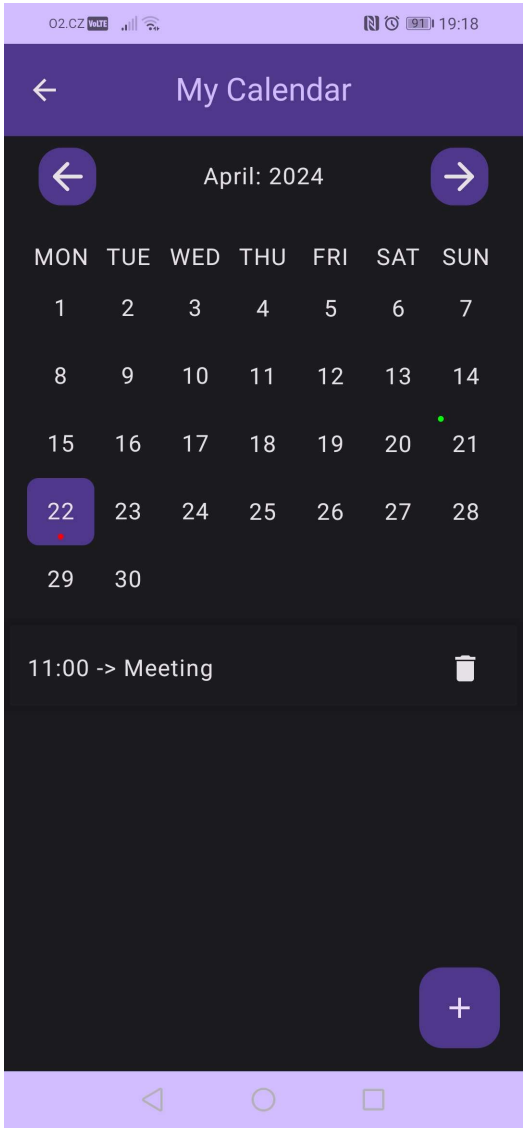
Další obrazovkou aplikace je obrazovka „My Calendar“. K této obrazovce mají přístup pouze přihlášení uživatelé, kteří mají svou vizitku a vizitku jiných uživatelů. Na tuto obrazovku se lze dostat z obrazovky „My Business Cards“, kde uživatel klikne na vizitku jiného uživatele a poté bude přesunut sem. Zde si může naplánovat události s vybraným uživatelem. Tyto události mají mezi sebou sdíleny, takže stačí, když ji vytvoří jeden a druhý ji ihned, díky Firebase Realtime Database, bude mít k dispozici.

Uživatel může vytvořit událost kliknutím na konkrétní den v kalendáři a následně na plovoucí tlačítko v pravém dolním rohu obrazovky s ikonou „+“. Po kliknutí na toto tlačítko se zobrazí dialogové okno, kde uživatel nastaví čas události a napíše její popisek. Událost lze vytvořit kliknutím na tlačítko „OK“, zrušit vytváření lze pomocí tlačítka „Cancel“. Důležité je poznamenat, že události lze vytvořit pouze do budoucna.

Pro přechod mezi měsíci v kalendáři slouží šipky „Vlevo“ a „Vpravo“ v horní části obrazovky. V kalendáři mohou být jednotlivé dny označeny červenou a zelenou tečkou. Červená tečka označuje dny s naplánovanými událostmi, zelená tečka označuje aktuální den.

Poté, co uživatel v kalendáři vybere den, může vidět v seznamu níže události, které jsou na tento den naplánovány. U každé položky seznamu se nachází ikona koše, která umožňuje odstranění události. Událost může odstranit pouze ten, kdo ji vytvořil.

Poslední komponentou na této obrazovce je šipka „Zpět“ vlevo v horní části obrazovky, která uživatele vrátí na obrazovku „My Business Cards“. Tuto obrazovku lze vidět na obrázku 6.



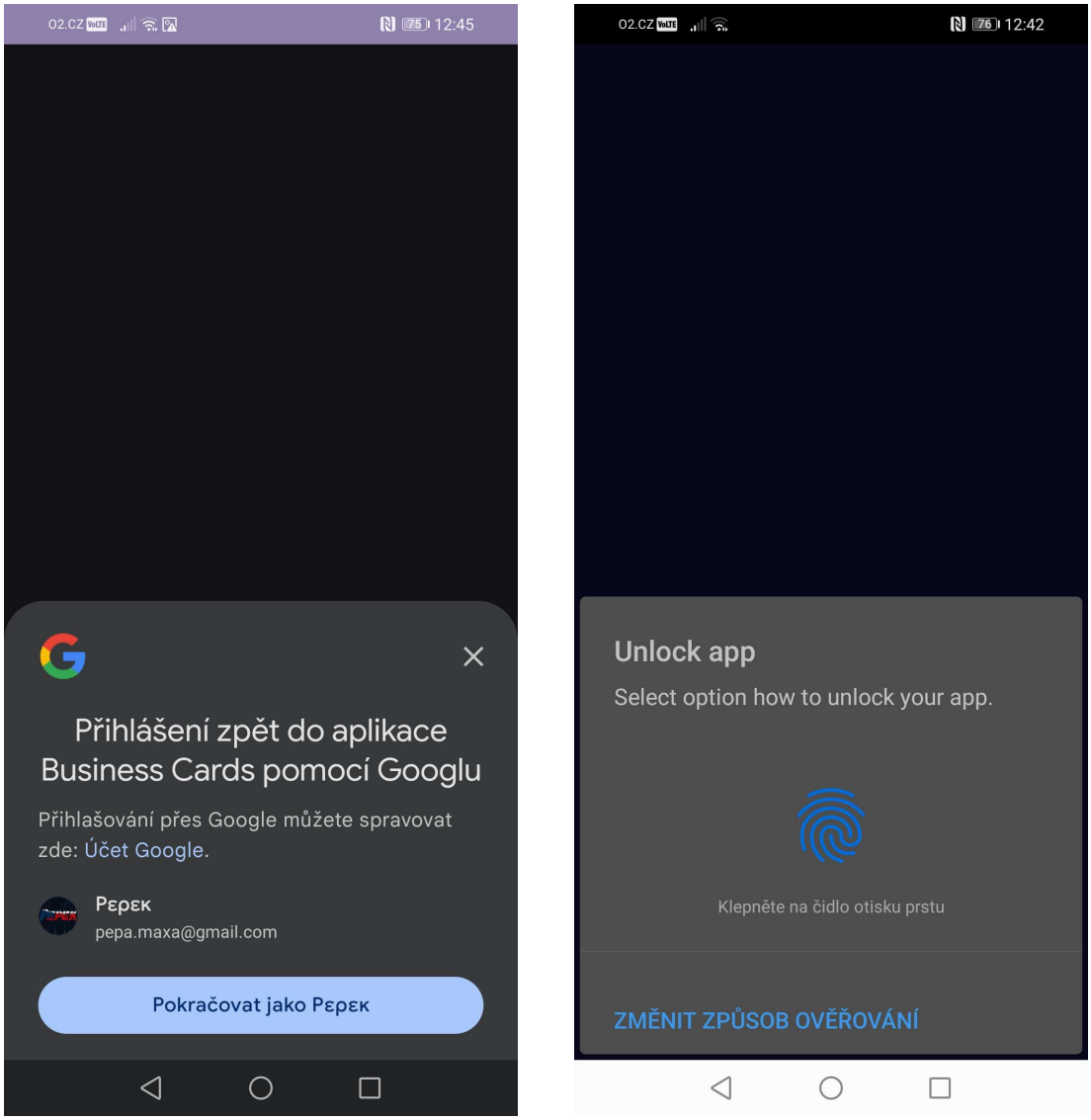
Obrázek 6: Obrazovka „My Calendar“. Zdroj vlastní.

4 ZABEZPEČENÍ APLIKACE

Zabezpečení aplikace je klíčové pro ochranu dat a soukromí uživatelů. Hlavním cílem je zajistit, že pouze oprávnění uživatelé mají přístup k citlivým informacím a že tyto informace jsou spolehlivě chráněny před neoprávněným přístupem.

4.1 Přístup do aplikace

Pro používání aplikace je vyžadováno přihlášení pomocí Google účtu, který je následně propojen se všemi daty uživatele. To lze vidět na obrázku 7 vlevo. Tento krok zajišťuje, že pouze vlastník má přístup ke svým datům.



Obrázek 7: Obrazovka „Sign In“. Zdroj vlastní.

Při opětovném používání aplikace, poté, co uživatel zůstal přihlášen, se musí znovu autentizovat a je mu umožněno si vybrat preferovaný způsob ověření, jako je PIN, heslo nebo otisk prstu, pokud jsou tyto možnosti dostupné na jeho zařízení. To lze vidět na obrázku 7 vpravo.

Pokud zařízení podporuje biometrickou autentizaci a uživatel ji má nastavenou, aplikace mu umožní ji využít k přihlášení. V opačném případě je uživatel vyzván k nastavení biometrické autentizace.

V situaci, kdy zařízení nepodporuje biometrickou autentizaci nebo uživatel nemá nastavený žádný biometrický identifikátor, může použít klasický autentizační proces, který vyžaduje zadání PINu nebo hesla.

Je důležité zdůraznit, že uvedené postupy mohou být ovlivněny verzí Androidu a dostupnými možnostmi na daném zařízení. Pro zajištění maximální úrovně bezpečnosti je také nezbytné, aby uživatel měl aktivní zabezpečení zamykací obrazovky.

5 TESTOVÁNÍ APLIKACE

Důkladné testování aplikace na různých zařízeních a verzích operačního systému Android je nezbytné pro zajištění kvalitního uživatelského zážitku a spolehlivého fungování aplikace. Tento proces umožňuje identifikovat a řešit potenciální problémy a zajistit, že aplikace je použitelná a stabilní napříč širokou škálou zařízení a verzí operačního systému Android.

Aplikace je určena pro Android 10 a výše. Průběžné testování probíhalo primárně na zařízení HONOR s operačním systémem Android 10, které poskytovalo základní prostředí pro vývoj a ladění aplikace. Pro ověření funkcionality sdílení vizitek pomocí NFC a QR kódu bylo využito i zařízení Samsung s operačním systémem Androidem 12, které pak ověřilo i kompatibilitu s novější verzí systému.

Kromě toho byla aplikace distribuována i pro jiné uživatele (testery) pomocí služby Firebase App Distribution. Aby byl proces distribuce a testování co nejjednodušší a automatizovaný, bylo pro to využito GitHub Actions a příslušné workflow. Tato workflow je spouštěna manuálně v případě, když je potřeba vydat novou verzi aplikace pro testování. Jakmile je nová verze vydána, je automaticky zaslán email testerům, kteří mají přístup k aplikaci přes Firebase App Distribution. Tímto způsobem jsou testerům rychle sděleny informace o nové verzi, kterou mohou okamžitě otestovat.

ZÁVĚR

Cílem této diplomové práce bylo vytvoření nativní mobilní aplikace pro platformu Android, která bude umožňovat vytvářet a uchovávat vizitky uživatelů. Aplikace měla umožňovat jak vytváření, tak úpravu vizitek, a to včetně možnosti nastavení vlastního pozadí. Dále měla umožňovat synchronizaci dat v reálném čase a sdílení vizitek mezi uživateli pomocí technologií NFC a QR kódů přímo v aplikaci.

V rámci této práce bylo úspěšně dosaženo všech výše stanovených cílů. Mobilní aplikace umožňuje uživateli vytvoření a upravení vizitky spolu s nastavením vlastního pozadí a to výběrem obrázku z galerie zařízení, nebo pořízením fotografie přímo v aplikaci. Data jsou pro uživatele dostupná vždy aktuální díky službám Firebase. Kromě toho jsou data uživatele zabezpečena tak, že si je může upravit pouze on sám. Důležité je taktéž zmínit, že aplikace je zabezpečena před neoprávněným vstupem a to pomocí otisku prstu, PINu, nebo hesla, a to podle toho, co má uživatel nastavené na zamykací obrazovce. Aplikace také umožňuje sdílení vizitek mezi uživateli pomocí NFC a QR kódů. Pro zajištění této funkcionality byla implementována čtečka NFC a HCE (Host Card Emulation). Uživatelé si také mohou mezi sebou přímo v aplikaci naplánovat události. Pro používání aplikace je nutné se přihlásit pomocí Google účtu, který pak umožňuje používání aplikace na různých zařízeních se stejnými daty.

Aplikace byla navržena tak, aby bylo možné ji jednoduše spravovat a rozšiřovat o další funkcionality. Pro dosažení tohoto cíle byl využit návrhový vzor MVVM. Kromě toho jsou v aplikaci použity nejnovější technologie, které se pro vývoj mobilní aplikací pro platformu Android využívají, a to především sada nástrojů Jetpack Compose. Taktéž byl použit i rámec Koin, který je jedním z nejpoužívanějších rámců pro přidělování závislostí a je napsán v jazyce Kotlin. Ačkoliv se může zdát, že používání nejnovějších nástrojů a technologií může být pro vývoj aplikací tím nejlepším řešením, opak je pravdou. Nové technologie s sebou přinášejí mnohdy dost problémů, jelikož mohou být pořád ve vývoji. To v praxi znamená, že se jejich implementace může často měnit, nebo některé funkcionality mohou být úplně odebrány. Výsledek této práce a její implementace však představuje významný přínos pro mé budoucí projekty v oblasti vývoje mobilních aplikací.

POUŽITÁ LITERATURA

[1] MIXON, Erica. What is Android OS? *TechTarget* [online]. USA, Newton, Massachusetts: TechTarget, 19. 1. 2024 [cit. 2024-04-26]. Dostupné z: `https://www.techtarget.com/searchmobilecomputing/definition/Android-OS`.

[2] MANCHANDA, Amit. 7 Android Programming Languages to Build Your Next Mobile App. *Net Solutions* [online]. Indie, Chandigarh: Net Solutions, 2. 5. 2022 [cit. 2024-04-26]. Dostupné z: `https://www.netsolutions.com/hub/mobile-app-development/android/programming-languages`.

[3] ECKEL, Bruce a Svetlana ISAKOVA. *Atomic Kotlin*. USA, Crested Butte: MindView LLC, 2020. ISBN 978-0-9818725-4-4.

[4] Google. Kotlin overview. *Android Developers* [online]. USA, Mountain View: Google, 1. 3. 2023 [cit. 2024-04-26]. Dostupné z: `https://developer.android.com/kotlin/overview`.

[5] JEMEROV, Dmitry a Svetlana ISAKOVA. *Kotlin in Action*. USA, Shelter Island, New York: Manning Publications Co., 2017. ISBN 978-1-61729-329-0.

[6] Google. Android's Kotlin-first approach. *Android Developers* [online]. USA, Mountain View: Google, 5. 4. 2024 [cit. 2024-04-26]. Dostupné z: `https://developer.android.com/kotlin/first`.

[7] kmDev. Understanding Jetpack Compose in Android: A Comprehensive Guide. *Medium* [online]. USA, San Francisco: Medium, 12. 8. 2023 [cit. 2024-04-26]. Dostupné z: `https://medium.com/@mkcode0323/understanding-jetpack-compose-in-android-a-comprehensive-guide-c3176fcc4745`.

[8] NIKOLOV, Lazar. Getting Started with Jetpack Compose. *Sentry: Product Blog* [online]. USA, San Francisco: FUNCTIONAL SOFTWARE, INC., 15. 2. 2023 [cit. 2024-04-26]. Dostupné z: `https://blog.sentry.io/getting-started-with-jetpack-compose/`.

[9] GHITA, Catalin. *Kickstart Modern Android Development with Jetpack and Kotlin*. UK, Birmingham: Packt Publishing Ltd., 2022. ISBN 978-1-80181-107-1.

[10] Google. State and Jetpack Compose. *Android Developers* [online]. USA, Mountain View: Google, 18. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://developer.android.com/develop/ui/compose/state.

[11] Google. Jetpack Compose: Material Design 3. *Material Design* [online]. USA, Mountain View: Google, [cit. 2024-04-26]. Dostupné z: https://m3.material.io/develop/android/jetpack-compose.

[12] Google. Material Design 3 in Compose. *Android Developers* [online]. USA, Mountain View: Google, 18. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://developer.android.com/jetpack/compose/designsystems/material3.

[13] The Coding Montana. Mastering Layouts in Jetpack Compose: A Comprehensive Guide with Examples. *Medium* [online]. USA, San Francisco: Medium, 19. 1. 2024 [cit. 2024-04-26]. Dostupné z: https://medium.com/@thecodingmontana/mastering-layouts-in-jetpack-compose-a-comprehensive-guide-with-examples-a053e156155e.

[14] Google. Compose layout basics. *Android Developers* [online]. USA, Mountain View: Google, 18. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://developer.android.com/develop/ui/compose/layouts/basics.

[15] HUMZA KHAN, Muhammed. Koin: A Practical Guide to Dependency Injection in Android. *Medium* [online]. USA, San Francisco: Medium, 4. 6. 2023 [cit. 2024-04-26]. Dostupné z: https://medium.com/@humzakhalid94/koin-a-practical-guide-to-dependency-injection-in-android-143cdab1756b.

[16] Koin. Android ViewModel. *Koin* [online]. Francie, Toulouse: Kotzilla, © 2024 [cit. 2024-04-26]. Dostupné z: https://insert-koin.io/docs/reference/koin-android/viewmodel.

[17] Koin. Definitions. *Koin* [online]. Francie, Toulouse: Kotzilla, © 2024 [cit. 2024-04-26]. Dostupné z: https://insert-koin.io/docs/reference/koin-core/definitions/.

[18] Koin. Start Koin on Android. *Koin* [online]. Francie, Toulouse: Kotzilla, © 2024 [cit. 2024-04-26]. Dostupné z: https://insert-koin.io/docs/reference/koin-android/start/.

[19] Koin. Injecting in Android. *Koin* [online]. Francie, Toulouse: Kotzilla, © 2024 [cit. 2024-04-26]. Dostupné z: https://insert-koin.io/docs/reference/koin-android/get-instances/.

[20] Google. Firebase Authentication. *Firebase* [online]. USA, Mountain View: Google, 16. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/auth.

[21] Google. Understand Firebase Security Rules for Cloud Storage. *Firebase* [online]. USA, Mountain View: Google, 16. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/storage/security.

[22] Google. Firestore. *Firebase* [online]. USA, Mountain View: Google, 25. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/firestore.

[23] Google. Supported data types. *Firebase* [online]. USA, Mountain View: Google, 25. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/firestore/manage-data/data-types.

[24] Google. Get realtime updates with Cloud Firestore. *Firebase* [online]. USA, Mountain View: Google, 25. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/firestore/query-data/listen.

[25] Google. Cloud Storage for Firebase. *Firebase* [online]. USA, Mountain View: Google, 16. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/storage.

[26] Google. About Cloud Storage buckets. *Google Cloud* [online]. USA, Mountain View: Google, 24. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://cloud.google.com/storage/docs/buckets.

[27] Google. Firebase Realtime Database. *Firebase* [online]. USA, Mountain View: Google, 16. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/database.

[28] Google. Connect your App to Firebase. *Firebase* [online]. USA, Mountain View: Google, 17. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/database/android/start.

[29] Google. Choose a database: Cloud Firestore or Realtime Database. *Firebase* [online]. USA, Mountain View: Google, 25. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://firebase.google.com/docs/firestore/rtdb-vs-firestore.

[30] GUILLERMO GÓMEZ TORRES, Juan. How to distribute your application with Firebase App Distribution. *Medium* [online]. USA, San Francisco: Medium, 14. 10. 2021 [cit. 2024-04-26]. Dostupné z: https://medium.com/google-developer-experts/how-to-distribute-your-application-with-firebase-app-distribution-8b70db0d389a.

[31] LÓPEZ-MAÑAS, Enrique. GitHub Actions for Android developers. *Medium* [online]. USA, San Francisco: Medium, 11. 2. 2021 [cit. 2024-04-26]. Dostupné z: https://medium.com/google-developer-experts/github-actions-for-android-developers-6b54c8a32f55.

[32] Intel. What Is Bluetooth® Technology? *Intel: Data Center Solutions, IoT, and PC Innovation* [online]. USA, Santa Clara, CA: Intel, 16. 10. 2023 [cit. 2024-04-26]. Dostupné z: https://www.intel.com/content/www/us/en/products/docs/wireless/what-is-bluetooth.html.

[33] Bluetooth. Bluetooth Technology Overview. *Bluetooth® Technology Website* [online]. USA, Kirkland, WA: Bluetooth SIG, Inc., 26. 5. 2021 [cit. 2024-04-26]. Dostupné z: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/.

[34] AVSystem. Everyone is using Bluetooth Low Energy – should you? *AVSystem: Shaping the world of connected devices* [online]. Polsko, Kraków: AVSystem, 30. 4. 2021 [cit. 2024-04-26]. Dostupné z: https://www.avsystem.com/blog/linkyfi/bluetooth-low-energy-ble/.

[35] Google. Bluetooth overview. *Android Developers* [online]. USA, Mountain View: Google, 4. 1. 2024 [cit. 2024-04-26]. Dostupné z: https://developer.android.com/develop/connectivity/bluetooth.

[36] TARDI, Carla. Near Field Communication (NFC) Definition. *Investopedia* [online]. USA, New York City: Dotdash Meredith, 31. 7. 2023 [cit. 2024-04-26]. Dostupné z: https://www.investopedia.com/terms/n/near-field-communication-nfc.asp.

[37] P. SABELLA, Robert a John PAUL MUELLER. *NFC For Dummies*. USA, Hoboken, NJ: John Wiley & Sons, Inc., 2016. ISBN 978-1-119-18292-4.

[38] HIGGINS, Malcolm. NFC Security: 10 security risks you Need to know. *NordVPN* [online]. Panama, Panama City: NordVPN, 10. 8. 2023 [cit. 2024-04-26]. Dostupné z: https://nordvpn.com/blog/nfc-security/.

[39] Google. Near field communication (NFC) overview. *Android Developers* [online]. USA, Mountain View: Google, 3. 1. 2024 [cit. 2024-04-26]. Dostupné z: https://developer.android.com/develop/connectivity/nfc.

[40] GAPCHENKO, Artem. Do you use NFC HCE API in Android? *Medium* [online]. USA, San Francisco: Medium, 9. 5. 2020 [cit. 2024-04-26]. Dostupné z: https://medium.com/swlh/why-you-need-to-be-very-careful-while-working-with-nfc-hce-apis-in-android-9bde32cc7924.

[41] Google. Host-based card emulation overview. *Android Developers* [online]. USA, Mountain View: Google, 3. 1. 2024 [cit. 2024-04-26]. Dostupné z: https://developer.android.com/develop/connectivity/nfc/hce.

[42] HAYES, Adam. Quick Response (QR) Code: Definition and How QR Codes Work. *Investopedia* [online]. USA, New York City: Dotdash Meredith, 29. 2. 2024 [cit. 2024-04-26]. Dostupné z: https://www.investopedia.com/terms/q/quick-response-qr-code.asp.

[43] RODRIGUE, Erin. What is a QR Code + How Does It Work? Everything Marketers Should Know. *HubSpot Blog* [online]. USA, Cambridge, Massachusetts: HubSpot, Inc., 23. 12. 2021 [cit. 2024-04-26]. Dostupné z: https://blog.hubspot.com/blog/tabid/6307/bid/16088/everything-a-marketer-should-know-about-qr-codes.aspx.

[44] LANXON, Andrew. How to Scan a QR Code With Any Android Phone. *CNET* [online]. USA, San Francisco, CA: CNET, 8. 12. 2022 [cit. 2024-04-26]. Do-

stupné z: https://www.cnet.com/tech/mobile/how-to-scan-a-qr-code-with-an-android-phone/.

[45] FERNANDES, Blair. How to Add QR Code Generation and Scanning in an Android App. *Medium* [online]. USA, San Francisco: Medium, 21. 5. 2023 [cit. 2024-04-26]. Dostupné z: https://blair49.medium.com/how-to-add-qr-code-generation-and-scanning-in-your-android-app-7baff15bd7c6.

[46] Google. ML Kit. *ML Kit* [online]. USA, Mountain View: Google, 17. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://developers.google.com/ml-kit/guides.

[47] Google. Barcode scanning. *ML Kit* [online]. USA, Mountain View: Google, 17. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://developers.google.com/ml-kit/vision/barcode-scanning.

[48] Google. ML Kit Analyzer. *Android Developers* [online]. USA, Mountain View: Google, 5. 1. 2024 [cit. 2024-04-26]. Dostupné z: https://developer.android.com/media/camera/camerax/mlkitanalyzer.

[49] Digital Trends Staff. What Is Wi-Fi Direct? Here's Everything You Need to Know. *Digital Trends: Tech News, Reviews, Deals, and How-To's* [online]. USA, Portland, Oregon: Digital Trends Media Group, 1. 11. 2021 [cit. 2024-04-26]. Dostupné z: https://www.digitaltrends.com/computing/what-is-wi-fi-direct/.

[50] Google. Create P2P connections with Wi-Fi Direct. *Android Developers* [online]. USA, Mountain View: Google, 18. 4. 2024 [cit. 2024-04-26]. Dostupné z: https://developer.android.com/develop/connectivity/wifi/wifi-direct.

[51] KUMAR, Sudhanshu. Coil Compose: Loading and Caching Images in Compose. *Medium* [online]. USA, San Francisco: Medium, 11. 11. 2023 [cit. 2024-04-26]. Dostupné z: https://medium.com/@sudhanshukumar04/coil-compose-loading-and-caching-images-in-compose-ebd7b25820c0.

SEZNAM PŘÍLOH

PŘÍLOHA A

```yaml
name: Deploy to Firebase

on:
  workflow_dispatch:
    inputs:
      release_notes:
        type: string
        required: true
        default: 'Manual Release Build'
        description: 'Release Notes'

jobs:
  lint:
    name: Building and distributing app
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'

      - name: Setup Gradle
        uses: gradle/gradle-build-action@v2

      - name: Make gradlew executable
        run: chmod +x ./gradlew

      - name: Execute Gradle command -> assembleRelease
        run: ./gradlew assembleRelease

      - name: Upload Artifact to Firebase App Distribution
        uses: wzieba/Firebase-Distribution-Github-Action@v1
        with:
          appId: ${{secrets.FIREBASE_APP_ID}}
          serviceCredentialsFileContent: ${{secrets.CREDENTIAL_FILE_CONTENT}}
          groups: testers
          file: app/build/outputs/apk/release/app-release-unsigned.apk
          releaseNotes: ${{inputs.release_notes}}
```

PŘÍLOHA B

Obsahuje zdrojový kód mobilní Android aplikace. Projekt je přiložen v archivu s názvem PavlikJ_SpravaVizitek_SN_prilohaDP_2024.zip. Tento projekt lze spustit ve vývojovém prostředí Android Studio.