Univerzita Pardubice

Fakulta elektrotechniky a informatiky

# Android aplikace pro správu úkolů

Tomáš Křičenský

Bakalářská práce

2019

**Univerzita Pardubice**
**Fakulta elektrotechniky a informatiky**
Akademický rok: **2018/2019**

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Tomáš Křičenský** |
| Osobní číslo: | **I16108** |
| Studijní program: | **B2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Název tématu: | **Android aplikace pro správu úkolů** |
| Zadávající katedra: | **Katedra informačních technologií** |

Zásady pro vypracování:

Cílem BP je naprogramovat mobilní aplikaci pro Android umožňující správu úkolů. V teoretické části autor popíše základní principy programování Android aplikací a popíše používané moderní technologie RxJava a Kotlin. Při návrhu aplikace bude použit návrhový vzor MVVM. Praktická část bude vytvořena ve vybraném jazyku, bude umožňovat připojení ke vzdálené databázi a bude obsahovat: vytvořenou aplikaci na správu úkolů v prostředí Android, její popis, diagram tříd, rozbor jednotlivých obrazovek, ukázku kódu.

Rozsah grafických prací:

Rozsah pracovní zprávy: **30**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

**KARANPURIA, Rashi a Aanand Shekhar ROY Kotlin Programming Cookbook: Explore more than 100 recipes that show how to build robust mobile and web applications with Kotlin, Spring Boot, and Android, Birmingham, United Kingdom: Packt Publishing Limited, 2018. ISBN 178847214**
**NIELD, Thomas Learning RxJava Birmingham, United Kingdom: Packt Publishing Limited, 2018. ISBN 1787120422**
**STROUD, Adam Android database best practices Boston, MA: Addison-Wesley, 2016. ISBN 978-013-4437-996**

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

Datum zadání bakalářské práce: **31. října 2018**
Termín odevzdání bakalářské práce: **12. května 2019**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 20. března 2018

Prohlašuji:

Tuto práci 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., autorský zákon, 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. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 1. 5. 2019

Tomáš Křičenský

## Poděkování

Tímto bych chtěl poděkovat své vedoucí Ing. Soně Neradové, Ph.D. za odbornou pomoc a cenné rady, které mi pomohly při zpracování této práce. Poděkování patří i mé rodině za veškerou podporu.

## ANOTACE

Bakalářská práce se zabývá návrhem a implementací mobilní aplikace na správu úkolů pro operační systém Android. Aplikaci je možné používat na více zařízeních díky synchronizaci se vzdálenou databází. Teoretická část pojednává o principech programování Android aplikací, o moderních technologiích RxJava a Kotlin a představí použitý návrhový vzor Model-View-ViewModel. V praktické části je popsána vytvořená aplikace včetně rozboru jednotlivých obrazovek.



## KLÍČOVÁ SLOVA

Správa úkolů, Android, mobilní aplikace, Kotlin, MVVM, RxJava.


## TITLE

Android Application for Task Management


## ANNOTATION

This Bachelor's thesis deals with the development of a task management application for Android operating system. The application can be used on multiple devices thanks to synchronisation with a remote database. The theoretical part is focused on the principles of Android application development, modern technologies such as RxJava and Kotlin, and architectural pattern Model-View-ViewModel. The practical section then describes the complete application and offers an in-depth description of the application screens.



## KEYWORDS

Task management, Android, mobile application, Kotlin, MVVM, RxJava.

# OBSAH

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM ZKRATEK

| | |
|---|---|
| API | Application Programming Interface |
| BSON | Binární JSON |
| JSON | JavaScript Object Notation |
| MVVM | Model-View-ViewModel |
| NoSQL | Not Only SQL |
| REST | Representational State Transfer |
| Rx | Reactive Extensions |
| SDK | Software development kit |
| SQL | Structured Query Language |
| UUID | Universal Unique Identifier |
| XML | Extensible Markup Language |

# ÚVOD

Mobilní zařízení dnes vlastní skoro každý. Z tohoto důvodu stoupá poptávka po mobilních aplikacích a je potřeba, aby existovali zkušení vývojáři, kteří budou schopni tyto aplikace vytvářet. Cílem této práce je vytvoření profesionálního návrhu a implementace aplikace na správu úkolů. Tato aplikace je dostatečně složitá pro demonstraci zvolené architektury, ale stále je dostatečně přehledná pro rychlé pochopení zvolených principů programování a použitých technologií. Vytvořená aplikace podporuje přihlášení pomocí účtu Google. Správa úkolů přihlášeného uživatele je synchronizována mezi více zařízeními díky vzdálené databázi, se kterou aplikace komunikuje.

Při návrhu aplikace byl použit návrhový vzor Model-View-ViewModel. Pro pomoc s implementací aplikací pomocí tohoto návrhového vzoru vydala společnost Google několik užitečných knihoven, které tato práce využívá. Dále bylo nutné navrhnout vhodnou strukturu balíčků pro snadnou orientaci v kódu a jednoduché přidaní nových obrazovek nebo funkcí. Jako databáze byla zvolena NoSQL databáze Cloud Firestore, která podporuje synchronizaci mezi více zařízeními a také práci v offline režimu.

Při implementaci byl použit programovací jazyk Kotlin, který byl v roce 2017 prohlášen oficiálním programovacím jazykem pro vývoj Android aplikací. Po tomto prohlášení nabyl Kotlin velké popularity mezi vývojáři. Kotlin byl pro implementaci vybrán proto, že kód psaný v Kotlinu je mnohem kratší a jazyk samotný nabízí mnoho inovativních funkcí v porovnání s Javou. Tyto rozdíly budou dále rozebrány v teoretické části práce. Druhou velmi důležitou technologií je RxJava, která zjednodušuje práci s asynchronním prováděním kódu. RxJava (nebo její obdoba) se v praxi velmi často používá a její znalost patří mezi vysoce ceněné schopnosti na trhu práce.

První kapitola popisuje účel vytvořené aplikace, mapuje existující řešení a seznamuje se základními principy programování Android aplikací. Ve druhé kapitole jsou popsány nejdůležitější použité technologie. Třetí kapitola se zaměřuje na popis navrženého řešení včetně struktury databáze a diagramu tříd. V této kapitole je také popsána navržená struktura projektu a je vysvětleno, jak bylo dosaženo okamžité synchronizace obsahu. V poslední kapitole jsou rozebrány jednotlivé obrazovky vytvořené aplikace a je popsán způsob jejich realizace.

# 1 ZMAPOVÁNÍ SOUČASNÉHO STAVU

## 1.1 Existující řešení

Před zahájením vývoje bylo potřeba prozkoumat existující aplikace na správu úkolů. V obchodě s Android aplikacemi Google Play je takových aplikací poměrně hodně. Přestože je podobných aplikací mnoho, každá z nich dokáže být unikátní. Některé aplikace nabízí líbivý design a animace, další zase pokročilé funkce nebo podporu více platforem.

První podobnou aplikací je aplikace „Úkoly“ od společnosti Google. Tato aplikace je velmi jednoduchá, kromě běžných funkcí si uživatel si může vytvořit několik různých seznamů úkolů. Vzhled aplikace je také velmi jednoduchý, spoléhá se na bílou barvu a drží se doporučení pro vzhled Android aplikací. Aplikace spoléhá na synchronizaci úkolů pomocí přihlášení k účtu Google. Tato synchronizace není okamžitá, ale pro jednoduché úkony je postačující. Jako většina aplikací od společnosti Google je i tato aplikace přeložena do mnoha jazyků (včetně češtiny).

Druhou sledovanou aplikací je aplikace „Ike“. Tato aplikace je úplně jiná než aplikace „Úkoly“. Jak již její logo napovídá, tato aplikace rozděluje úkoly na 4 skupiny podle jejich důležitosti a požadavku na včasné provedení. Při používání se snaží ohromit uživatele zajímavým ovládáním a extravagantními animacemi. Tyto animace uživatele, ale po nějaké době omrzí a začnou spíše zdržovat. Uživatel si může vybrat z několika předdefinovaných barevných stylů a může si přejmenovat jednotlivé kategorie. Tato aplikace není dostupná v českém jazyce.

Poslední a zároveň nejpoužívanější (více než 10 milionů stažení na Google Play[1]) aplikací, popsané v této kapitole, je aplikace „Any.do“. Ze zmíněných aplikací nabízí „Any.do“ nejvíce možností pro uživatele. Nabízí běžný seznam úkolů, ale každý úkol také může být rozdělen na dílčí úkoly. Pomocí aplikace jdou spravovat nejen úkoly, ale také nákupní seznamy nebo události v kalendáři. Aplikace je přeložena do několika cizích jazyků (včetně češtiny), změny barev uživatelského rozhraní nebo i napojení na Amazon Alexa[2].

[1] Dle informací z webu Google Play, ze dne 17. 3. 2019.

[2] Amazon Alexa je virtuální asistent vyvinutý společností Amazon.

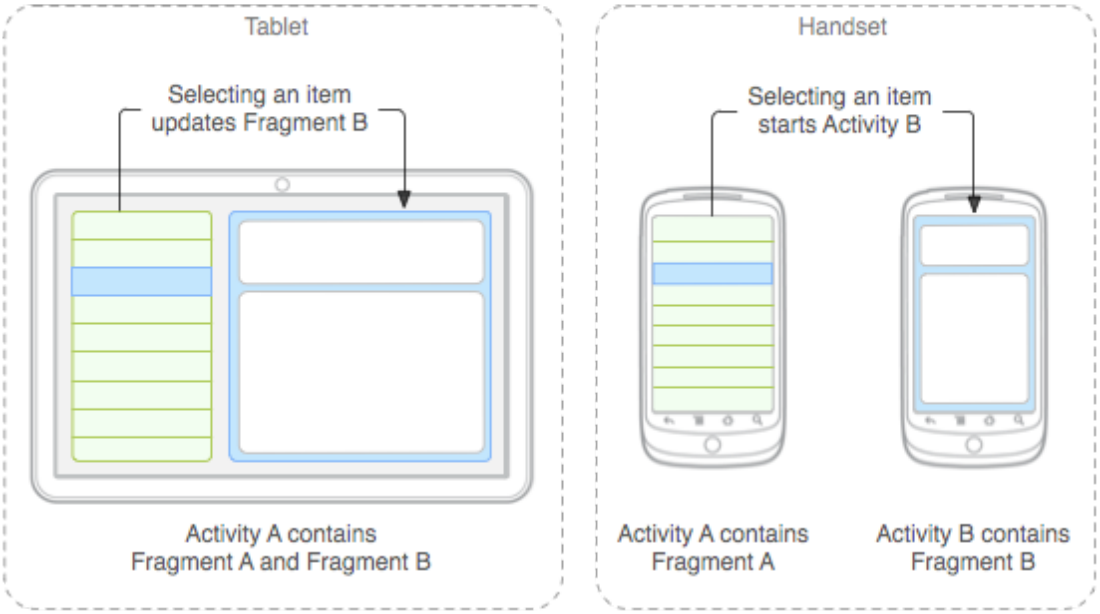
## 1.2 Účel aplikace

Po prozkoumání existujících řešení bylo jasné, že není možné v rámci této práce vytvořit aplikaci s nejvíce pokročilými funkcemi. Ukázalo se, že není nutné nabídnout uživatelům mnoho složitých funkcí, ale je možné je oslovit pomocí zajímavého designu a řešení „co prostě funguje“.

Cílem práce je vytvoření Android aplikace, která zaujme uživatele svým designem, bude se dobře používat a bude implementována dle návrhového vzoru Model-View-ViewModel[3]. Hlavní funkcí aplikace je synchronizace všech uživatelských akcí (úprava úkolů, změna stylu, ...) v reálném čase mezi všemi zařízeními přihlášenými pomocí stejného účtu Google. Přestože se aplikace pro realizaci synchronizace připojuje ke vzdálené databázi, lze aplikaci plnohodnotně používat bez přístupu k internetu.

## 1.3 Programování Android aplikací

Nativní aplikace pro platformu Android jsou vyvíjeny pomocí vývojového prostředí Android Studio. Toto prostředí bylo přímo vyvinuto pro vývoj Android aplikací a je tedy tou nejlepší volbou. Aplikace je možné vyvíjet pomocí jazyků Java nebo Kotlin. V těchto jazycích se píše funkční kód aplikace. Vzhled jednotlivých obrazovek se zapisuje pomocí jazyka XML (Extensible Markup Language). [1]




Obrázek 1: Příklad využití fragmentů. [2]

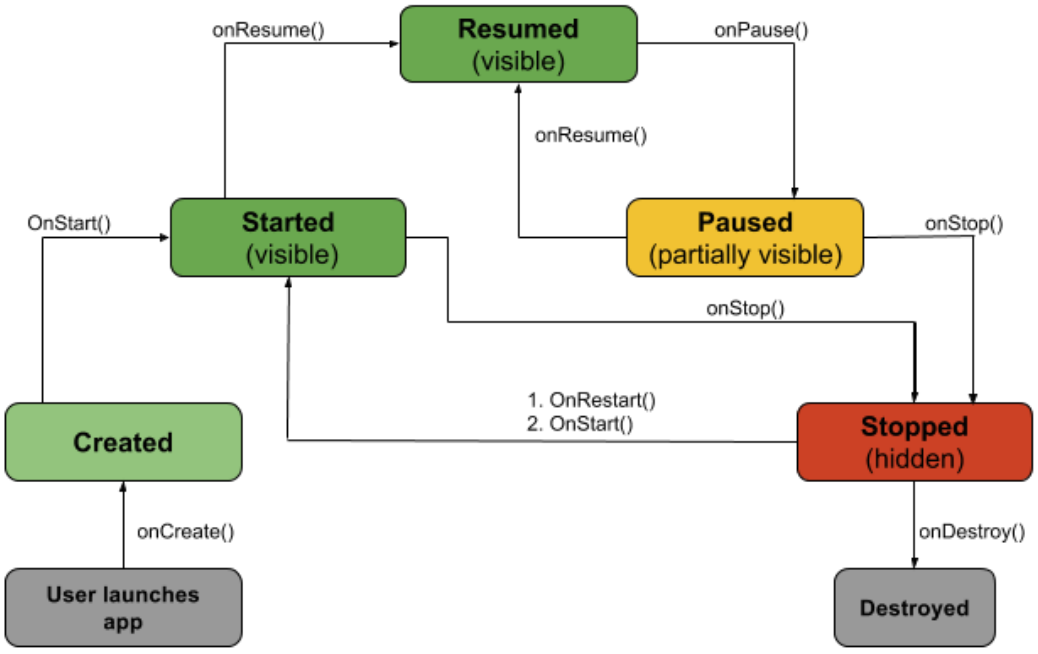
[3]Tento návrhový vzor je popsán v kapitole 3.2.

Nejdůležitější komponentou Androidu je takzvaná aktivita. Každá Android aplikace obsahuje alespoň jednu aktivitu. Aktivita zobrazuje uživatelské rozhraní aplikace a vždy se roztáhne přes celou velikost obrazovky. Každá obrazovka v aplikaci (například přihlašovací obrazovka) může být samostatná aktivita, ale také je možné používat v celé aplikaci pouze jednu aktivitu a v ní střídat jednotlivé obrazovky pomocí fragmentů. [1]

Fragment také zobrazuje uživatelské rozhraní, ale nemusí vždy zabrat celou obrazovku. Fragmenty můžeme zobrazovat vedle sebe nebo i nad sebou. Jdou také jednoduše vyměňovat na jedné pozici v rámci aktivity a tím zjednodušit celou navigaci v aplikaci. Obrázek 1 ukazuje využití fragmentů pro optimalizaci aplikace pro tablety.

Při práci s aktivitami je velmi důležité znát jejich životní cyklus. Třída aktivity tedy obsahuje několik metod, které jsou volány při změně živnotního cyklu. Pokud je potřeba na tyto změny v aplikaci reagovat, stačí překrýt vybrané metody vlastní implementací.

Nejdůležitější metodou životního cyklu je metoda „onCreate“. Tato metoda je volána při startu aktivity a je možné v ní například zobrazit uživatelské rozhraní a začít poslouchat na možné kliknutí na tlačítka a podobně. Druhou velmi důležitou metodou je metoda „onPause“, která je volána v případě, že aktivita již nebude v popředí. Více metod životního cyklu a přechody mezi stavy aktivity ukazuje obrázek 2. [1]




Obrázek 2: Životní cyklus aktivity. [3]

Fragmenty mají také životní cyklus, který se ale liší od životního cyklu aktivity. Životní cyklus fragmentu obsahuje například metody pro obsluhu těchto událostí: připojení fragmentu k aktivitě, vytvoření fragmentu, vytvoření aktivity, vytvoření uživatelského rozhraní, odpojení fragmentu a jiné. Neznalost životního cyklu může vést k častým pádům aplikace, například při snaze upravovat uživatelské rozhraní poté, co bylo zrušeno pro uvolnění prostředků. [1]

Další důležitou součástí Android projektu je složka „res“. Tato složka obsahuje všechny obrázky, texty, barvy, rozměry a mnoho dalšího. Většina těchto informací se zapisuje pomocí jazyka XML. Zdroje se tak zapisují proto, aby bylo možné je jednoduše nahradit pro různé konfigurace mobilního zařízení. Tímto způsobem lze tedy definovat překlad aplikace do jiného jazyka nebo také použití alternativních rozměrů na různě velkých zařízeních. [1]

Při programování Android aplikací je také nutné si zvolit, kterou nejnižší verzi Androidu bude aplikace podporovat. Starší verze často nepodporují určité funkce a pokud verzi již nepoužívá mnoho uživatelů nevyplatí se pro ni aplikaci optimalizovat. Nejnižší verze Androidu, kterou podporuje aplikace vytvořená v této práci je Android 5.0 (verze API číslo 21[4]).

[4]V kódu se verze Androidu označují celými čísly, které se liší od jejich veřejného číselného označení.

# 2 POUŽITÉ TECHNOLOGIE

## 2.1 Kotlin

Pro vývoj této aplikace byl zvolen programovací jazyk Kotlin. Většina existujících Android aplikací je dnes stále napsána v Javě, ale poté co v roce 2017 Google prohlásil Kotlin oficiálně podporovaným jazykem, zažil Kotlin velký nárůst v oblibě. Po tomto oznámení jsem se začal o Kotlin zajímat i já a již brzy jsem začal přepisovat aplikaci, na které jsem pracoval právě do Kotlinu. Takový krok je v podstatě velmi jednoduchý, protože v rámci jedné aplikace můžete libovolně kombinovat oba programovací jazyky. Kotlin vás ani neomezuje ve využívání knihoven určených pro Javu. Tuto vzájemnou kompatibilitu umožňuje to, že je Kotlin kompilován do Java bytekódu. [4]

### 2.1.1 Bezpečnost typů hodnot

Kotlin je stejně jako Java staticky typový jazyk. To znamená, že všechny datové typy musí být známé již při kompilaci a díky tomu lze předejít chybám za běhu programu, způsobených neplatnými datovými typy. Přesto se v Kotlinu nemusí vždy přímo definovat datový typ proměnných, protože ho dokáže kompilátor často odvodit z pravé části výrazu. Kotlin vás vede i k využívání neměnných datových složek, které se definují pomocí klíčového slova „val“ (proměnné se deklarují pomocí velmi podobného klíčového slova „var“). Při využití neměnných datových složek v podmínkách pro kontrolu datového typu, kompilátor automaticky pozná datový typ a v následujících příkazech je s ním možné přímo pracovat, protože s kontrolou typu automaticky proběhne i přetypování se zachováním stejného jména proměnné. [4]

Další velmi důležitou vlastností Kotlinu je bezpečnost null hodnot. Každá proměnná má pevně specifikováno, zda může obsahovat hodnotu null nebo nemůže. Pokud může tuto hodnotu obsahovat, musí být její datový typ definován s otazníkem na konci a není možné na ní provolávat metody pomocí tečky. Pro práci s null datovými typy se používá operátor „.?“. Dalším operátorem, který usnadňuje práci s těmito proměnnými je takzvaný Elvis operátor („?:“). Tento operátor je dobré využít, pokud chceme určit výchozí hodnotu ve výrazu, kde bychom pracovali s proměnnou, která může obsahovat hodnotu null. [4]

### 2.1.2 Funkce rozšíření

Jako mnoho dalších programovacích jazyků podporuje Kotlin takzvané funkce rozšíření. Pomocí těchto funkcí je možné jednoduše přidat nové funkce k existujícím třídám bez použití dědičnosti. Java tento typ funkcí nepodporuje, ale právě díky Kotlinu je možné s nimi pracovat i při vývoji Android aplikací. [4]

```
fun Boolean.toLayoutVisibility(): Int {
    return if (this) {
        View.VISIBLE
    } else {
        View.GONE
    }
}
```

Obrázek 3: Ukázka zápisu funkce rozšíření pro Boolean (převod na viditelnost).

Tyto funkce je dobré použít pro kód, který by se v projektu často opakoval. Funkce rozšíření je možné často použít i v případě, kde byste chtěli v Javě použít takzvanou „utility“ třídu. Kotlin má ve své standardní knihovně definováno mnoho funkcí, které vám při programování usnadní práci. Například při práci s kolekcemi se vám často může hodit funkce „filter“, která vybere prvky dle zadaného predikátu. [4]

## 2.2 RxJava

Název technologie sice napovídá, že byla tato aplikace napsána v Javě. To ale není pravda. Tato technologie má ve jméně slovo Java pouze proto, že je to vydání určené pro běh na Java Virtual Machine (a je tedy vhodná i pro Kotlin). Existuje i varianta určená přimo pro Kotlin (RxKotlin), která ale obsahuje pouze funkce rozšíření nad standardní knihovnou RxJava.

RxJava je určená pro takzvané reaktivní programování. Využívá technik funkcionálního programování pro asynchronní zpracování událostí. K tomu využívá návrhový vzor observer (pozorovatel). Velmi usnadňuje asynchronní operace jako komunikace přes REST API. V této práci byla RxJava využita pro zpracovávání dat z databáze při změně. Díky tomu dokáže aplikace dynamicky reagovat na změny ve všech připojených zařízeních stejného uživatele. [5]

Na obrázku 4 je ukázka kódu využívající RxJavu. Tato ukázka představuje způsob, jakým lze sledovat změny úkolů z databáze. Nejdříve je potřeba získat z „todo repository“ takzvaný Observable objekt. Na tomto objektu je možné nastavit na jakém vlákně budou probíhat určité operace pomocí funkcí „subscribeOn“ a „observeOn“. V ukázkovém kódu je tedy nastaveno, že se mají operace odehrávat na vstupně-výstupním vlákně, a že samotná data mají již přicházet na vlákně hlavním, ze kterého lze aktualizovat uživatelské rozhraní. Samotná inicializace poslouchání dat se provádí pomocí funkce „subscribe“. Tato funkce přijímá dva parametry typu „Consumer“, které jsou volány v případě vyskytnutí události. Tyto parametry byly v příkladě nahrazeny lambda výrazem. První parametr se jmenuje „onNext“ a je provoláván při každé změně sledovaného seznamu úkolů. Druhý parametr se jmenuje „onError“ a je provoláván v případě, že dotaz skončí chybou. Nakonec je výsledek volání „subscribe“ uložen do takzvané „Disposable“. Díky tomu je možné později přestat pozorovat změny dat, pokud přestane být aplikace na popředí a podobně.

```
private fun observeTodos() {
    todosDisposable = todoRepository.getAllTodos()
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe({ it: List<Todo>!
                viewModelData = viewModelData.copy(todos = it)
            }, { error ->
                logger.e(error)
            })
}
```

Obrázek 4: Ukázka zápisu pro sledování změn pomocí RxJava Observable.

## 2.3 Vzdálená databáze

Jako vzdálená databáze byla použita dokumentová NoSQL databáze Cloud Firestore. Tato databáze běží na serverech společnosti Google, tudíž je možné s ní přes internet synchronizovat obsah na zařízeních uživatelů aplikace. Synchronizace probíhá v reálném čase, takže pokud upravíme nějaký úkol na jednom zařízení, budeme moci okamžitě vidět změny i na druhém zařízení stejného uživatele. Ačkoli je databáze na serveru, je možné s ní na webu a mobilních zařízeních pracovat i offline. Nevýhodou offline režimu je, že nepodporuje transakce, které ale nakonec nebyly pro implementaci aplikace potřeba. [6]

Jak již bylo řečeno Cloud Firestore je dokumentová databáze. Tyto dokumenty jsou ukládány do takzvaných kolekcí, které by se daly přirovnat k tabulkám z relačních databázových systémů. Dokumenty v jedné kolekci, ale nemusí mít stejnou datovou strukturu a mohou obsahovat i další vnořené kolekce. [7] Přestože není konzistence struktury dokumentů v kolekci vyžadována, všechny kolekce využité v této práci mají jednotnou strukturu. Díky tomu se s nimi z aplikace mnohem lépe pracuje.

Každý dokument v kolekci musí mít unikátní identifikátor. Jako identifikátor je možné si zvolit téměř jakoukoliv textovou hodnotu. Tato hodnota nesmí obsahovat lomítko a musí být menší než 1 500 bajtů. V databázi Cloud Firestore tento identifikátor ale vůbec nemusí být součástí struktury dokumentu. Na dokumenty se tedy odkazuje pomocí takzvané cesty k dokumentu. Cesta k dokumentu obsahuje název kolekce a unikátní identifikátor dokumentu. [7]

# 3 NÁVRH APLIKACE

## 3.1 Analýza a návrh použité databáze

Před začátkem návrhu struktury databáze bylo potřeba vybrat databázi. Při výběru databáze byl kladen důraz na dva hlavní požadavky. Tím nejdůležitějším byla možnost připojit se k databázi vzdáleně z mobilní aplikace bez nutnosti programování logiky na vlastním serveru.

Druhým důležitým požadavkem byla schopnost synchronizovat část databáze ze serveru na mobilní zařízení. Díky této vlastnosti je možné aplikaci používat i bez přístupu k internetu. Vybraná databáze musí být schopna řešit konflikty mezi verzí úkolu uloženou na serveru a verzí, kterou uživatel upravoval bez přístupu k internetu.

Těmto požadavkům vyhovovaly hned dvě různé databáze. První z nich byla databáze MongoDB Mobile. MongoDB je dokumentová NoSQL databáze, která ukládá data ve formátu BSON (binární JSON). A dokáže synchronizovat data se vzdálenou databází pomocí MongoDB Stitch. Tato synchronizace je ale stále ve vývoji a je k dispozici pouze beta verze. [8], [9]

Druhou vyhovující databází se ukázal být Cloud Firestore (nebo i jeho předchůdce Firebase Realtime Database). Cloud Firestore je také dokumentová NoSQL databáze. Cloud Firestore je vzdálená databáze, která má možnost se automaticky synchronizovat do lokálního úložiště a dokáže tedy fungovat i bez přístupu k internetu. Zároveň podporuje synchronizaci změn v databázi v reálném čase. [7]

Nakonec byla pro implementaci vybrána databáze Cloud Firestore. Tato databáze byla vybrána z důvodu přívětivějšího SDK pro vývoj a také jednoduššího prvnotního nastavení. Další velkou výhodou bylo, že je tato databáze vyvíjena společností Google a tudíž půjde jednoduše integrovat s ostatními nástroji od Googlu, které byly v této práci využity.

### 3.1.1 Kolekce „todos“

Jak již bylo zmíněno, v databázi Cloud Firestore jsou data ukládána do kolekcí místo klasických tabulek. V kolekci jsou uloženy všechny úkoly. I když databáze nevyžaduje striktní dodržování struktury dat v kolekci, všechny kolekce využité v této práci dodržují danou strukturu.

Každý dokument v této kolekci obsahuje položku „id“, ve které je uložen unikátní identifikátor úkolu (ten je obsažen i v samotné cestě k dokumentu). Tento identifikátor je generován na straně aplikace a je to UUID (univerzální unikátní identifikátor), které se do databáze ukládá jako text. Druhou nejdůležitější položkou dokumentu této kolekce je „userId“. Je datového typu string a jak již název napovídá, obsahuje ID uživatele, kterému úkol patří. Toto ID je v aplikaci získáno z Google účtu přihlášeného uživatele. Dále tato kolekce obsahuje textové položky „title“ a „description“. Tyto položky obsahují název, respektive popis úkolu. Poslední 2 datové položky v této kolekci jsou typu Timestamp. První z nich se jmenuje „dueDate“ a obsahuje datum, do kdy chce uživatel tento úkol splnit. Druhou položkou typu Timestamp je „completedAt“. Tato datová položka obsahuje informace o tom, kdy byl úkol splněn. Pokud úkol ještě splněn nebyl, obsahuje hodnotu null. Držení informace o datu splnění úkolu je důležité pro schopnost mazat splněné automaticky po uplynutí několika dní. Všechny položky ze struktury „todos“ a přiřazené datové typy jsou zmíněné v tabulce 1.

Tabulka 1: Struktura dokumentu v kolekci „todos“.

| **Název položky** | **Datový typ** |
|---|---|
| id | string |
| userId | string |
| title | string |
| description | string |
| dueDate | timestamp |
| completedAt | timestamp |

### 3.1.2 Kolekce „settings“

Druhou a zároveň poslední kolekcí, která byla v práci využita, je kolekce „settings“. Tato kolekce obsahuje informace o vlastním nastavení aplikace pro každého uživatele. Obsahuje tedy pouze dvě datové položky. První z nich se jmenuje „autoRemoveTodosAfter“ a definuje počet dní, po kterém se mají automaticky mazat splněné úkoly. Druhou položkou v této kolekci je „selectedStyleName“. Tato položka je datového typu string, ale reálně může obsahovat pouze 3 různé hodnoty. Tyto hodnoty odpovídají 3 možným barevným verzím aplikace: „reddish“ (červená), „bluish“ (modro-zelená) a „yellowish“ (žlutá). Pro spojení nastavení v databázi se správným uživatelem se využívá dříve zmíněná cesta k dokumentu. Tato cesta tedy obsahuje ID uživatele.

Tabulka 2: Struktura dokumentu v kolekci „settings“.

| **Název položky** | **Datový typ** |
|---|---|
| autoRemoveTodosAfter | number |
| selectedStyleName | string |

## 3.2 Model-View-ViewModel

Pro vývoj této aplikace byl zvolen návrhový vzor Model-View-ViewModel. Stejně jako u ostatních návrhových vzorů je zde potřeba dodržet princip oddělení odpovědností. Ten říká, že každá část aplikace má dělat pouze jednu věc a má ji dělat dobře. Architektura rozděluje zodpovědnost do tří hlavních vrstev: model, view a ViewModel.

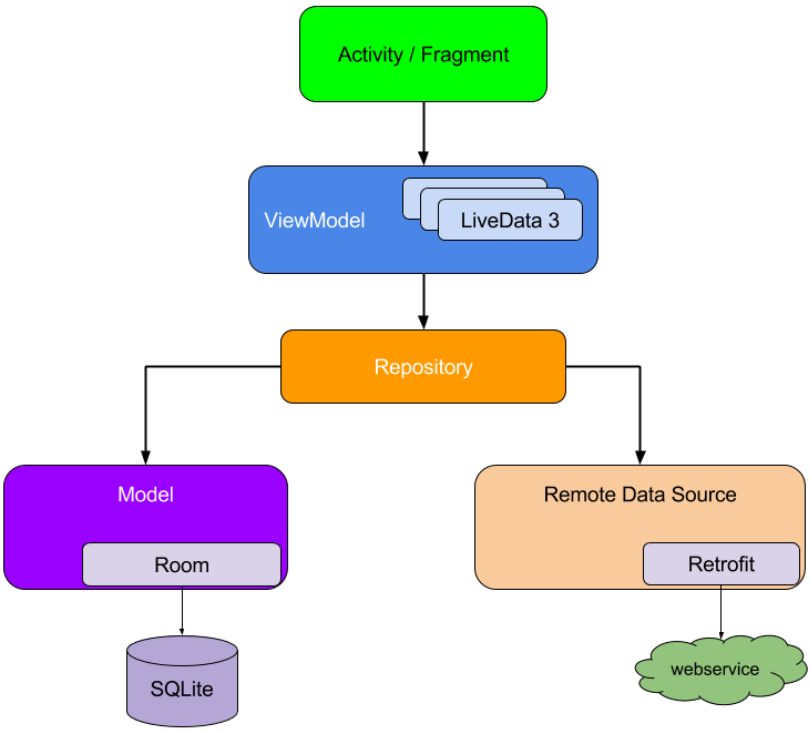
Model řeší, jakým způsobem budou uložena data a jak k nim může ViewModel přistupovat. [10]

View vrstva pozoruje data vystavena ViewModelem, aby je mohla zobrazit a při jejich změně je vykreslí do uživatelského rozhraní. View dále posílá uživatelem vyvolané události pro zpracování do ViewModelu. View by nemělo obsahovat žádnou speciální logiku a mělo by pouze provádět svých několik jednoduchých činností. [10]

ViewModel zpracovává vstup ze svého view a pomocí vnitřní logiky rozhoduje, co se stane dál. Také komunikuje s model vrstvou, aby mohl připravit data, která budou poté zobrazena pomocí view. Data k vykreslení neposílá view přímo. ViewModel vlastně ani

neví, že nějaké view existuje. Pouze vždy vystaví nová data do pozorovatelné proměnné, kterou view sleduje a při každé změně dat vykreslí nový stav uživatelského rozhraní. [10]

Obrázek 5 ukazuje jasně rozdělené role komponent v MVVM architektuře. Každá z komponent si drží referenci pouze na tu následující (na obrázku směrem dolů). To například znamená, že view (aktivita nebo fragment) neví nic o modelu (repository) a komunikuje pouze s ViewModelem. Směrem nahoru komponenty komunikují pomocí návrhového vzoru observer (pozorovatel). Pro tuto komunikaci mezi view a ViewModelem byly v této práci použity LiveData. LiveData umožňují pozorovat změny dat a zároveň si automaticky hlídají životní cyklus komponenty, která chce změny sledovat. [10]




Obrázek 5: Vizualizace MVVM architektury pro Android [11]

## 3.3 Struktura projektu

Před začátkem implementace aplikace bylo potřeba vymyslet strukturu balíčků a tříd. Bylo tedy určeno, že se bude zdrojový kód rozdělovat do balíčků dle obrazovek, které kód primárně využívají. V praxi to například znamená, že třídy, které jsou potřeba pro správnou funkčnost přihlašovací obrazovky, budou umístěny do stejného balíčku.

Tento balíček společné funkcionality se dále dělí na několik vnořených balíčků podle toho do jaké vrstvy architektury MVVM dané třídy patří. Vnořené balíčky mohou být následující: „di“, „repository“, „view“ a „viewmodel“.

Balíček „di“ obsahuje třídy, které tato obrazovka potřebuje pro správnou funkčnost vkládání závislostí[5]. Většinou tento balíček obsahuje pouze moduly pro framework Dagger 2. V těchto modulech je specifikováno mapování poskytovaných instancí objektů na rozhraní, které implementují.

Druhém vnořeným balíčkem je balíček „repository“. Tento balíček obsahuje to, co je v MVVM architektuře součástí vrstvy model. Obsahuje tedy třídy typu „repository“, alternativní zdroje dat a jejich rozhraní.

Dalším důležitým balíčkem je balíček „view“. Ten obsahuje vše, co se týká uživatelského rozhraní. Vždy tedy obsahuje fragment pro danou obrazovku. Také může obsahovat jednotlivé položky, které se budou vykreslovat v seznamu nebo i pomocné třídy pro interakci a vykreslování uživatelského rozhraní.

Poslední balíček, který je běžně součástí této struktury, je balíček „viewmodel“. Tento balíček obsahuje vše, co souvisí s ViewModelem z architektury MVVM. Obsahuje tedy samotný ViewModel, továrnu, která ho umí vytvořit a několik dalších potřebných tříd. Těmito třídami jsou typicky různé datové modely nebo i speciální mapovací třída, která převádí čistá data na data, která fragment pouze zobrazí a nebude je vůbec muset zpracovávat.

Vizualizaci tříd ze struktury balíčků pro přihlašovací obrazovku ukazuje diagram tříd v příloze A. Tento diagram obsahuje pouze třídy vytvořené v rámci této práce.

## 3.4 Podpora více jazyků

Jedním z požadavků na tuto aplikaci byla možnost používání aplikace v několika různých jazycích. Výchozím jazykem aplikace je angličtina. Tento jazyk bude použit pokud uživatel bude mít nastavený jazyk systému na jiný jazyk než ten, který aplikace explicitně podporuje. Dalšími jazyky, které aplikace podporuje jsou čeština a slovenština. Přestože je aplikace přeložena do těchto jazyků, tak se název aplikace nepřekládá.

Aby bylo možné přeložit aplikaci do dalších jazyků je nutné všechny texty zapisovat do souboru „strings.xml“. Každý z takto definovaných textů má jasně určené ID.

[5] Anglicky dependency injection.

Při tvorbě aplikace je nutné se na všechny texty odkazovat právě podle jejich ID. Díky tomu nejsou texty nikde pevně definované je snadné je vyměnit za jiné v závislosti na konfiguraci telefonu.

Výchozí texty je tedy potřeba definovat ve složce „res/values“ v souboru „strings.xml“. V případě této aplikace jsou tu tedy definovány všechny anglické texty. Texty, které se mají ukazovat na telefonech se systémem v českém jazyce je potřeba definovat ve složce „res/values-cs“ a slovenské texty ve složce „res/values-sk“. Stejným způsobem by bylo snadné přidat i další překlady aplikace.

## 3.5 Okamžitá synchronizace obsahu

Velmi důležitou funkcí této aplikace je synchronizace obsahu na více zařízeních. Díky správné volbě databáze se obsah synchronizuje v reálném čase. To znamená, že pokud se uživatel přihlásí do aplikace pomocí stejného Google účtu na dvou zařízeních, tak na obou těchto zařízeních po jakékoliv změně okamžitě uvidí stejný obsah.

```
private fun observeSettings() {
    stopObservingSettings()
    settingsDisposable = settingsRepository.observeSettings()
            .subscribe({ settings ->
                if (settings.selectedStyle != viewModelData.lastStyle) {
                    events.value = Event(UpdateStyle)
                }

                if (settings.autoRemoveTodosAfter < viewModelData.lastRemoveTodosAfter) {
                    todoRepository.removeCompletedTodos(settings.autoRemoveTodosAfter).subscribe({
                        logger.d( message: "Successfully removed old completed todos.")
                    }, { error ->
                        logger.e(error)
                    }).bindToViewModel()
                }

                viewModelData = viewModelData.copy(
                        lastStyle = settings.selectedStyle,
                        lastRemoveTodosAfter = settings.autoRemoveTodosAfter
                )
            }, { it: Throwable!
                logger.e(it)
            })
}
```

Obrázek 6: Ukázka kódu pro sledování změn nastavení.

Při načítání seznamu úkolů z databáze se aplikace rovnou přihlásí o to, že chce být při každé změně dat notifikována. Takže jakákoliv změna stávajících úkolů nebo i vytvoření nového úkolu, který má přiděleno ID stávajícího uživatele, dostane aplikace ke zpracování nový seznam úkolů. Tento seznam aplikace připraví pro vykreslení a poté jen notifikuje adaptér, který data vykresluje o změnách. Díky tomu se při každé změně nemusí překres-

lovat celý seznam, ale vždy se pouze překreslí, přesune nebo smaže pouze záznam, kterého se daná akce týká.

Překreslování pouze změn předchází zbytečnému překreslování celého obsahu a tím šetří výkon. Toho bylo možné dosáhnout pomocí třídy „DiffUtil“, které je součástí podpůrné knihovny pro Android od společnosti Google. Tato třída dokáže vypočítat rozdíly mezi 2 seznamy dat pomocí třídy „DiffUtil.Callback“, která byla v aplikaci implementována. O těchto rozdílech pak pouze notifikuje adaptér pro „RecyclerView“, který vykresluje seznam úkolů. Díky tomu, že je adaptér notifikován o změnách, je možné automaticky přehrát animace pro přesunutí úkolu nebo i smazání úkolu.

Kdyby nebylo využito třídy „DiffUtil“, tak by se při každé změně jakéhokoliv úkolu muselo znovu vytvořit uživatelské rozhraní pro všechny položky seznamu a poté by ještě všechny musely být znovu naplněny daty. A to i v případě, že by se změnil v celém seznamu pouze jeden úkol. Takový přístup by vůbec nebyl vhodný pro aplikaci, která potřebuje vykreslovat změny v seznamu v reálném čase.

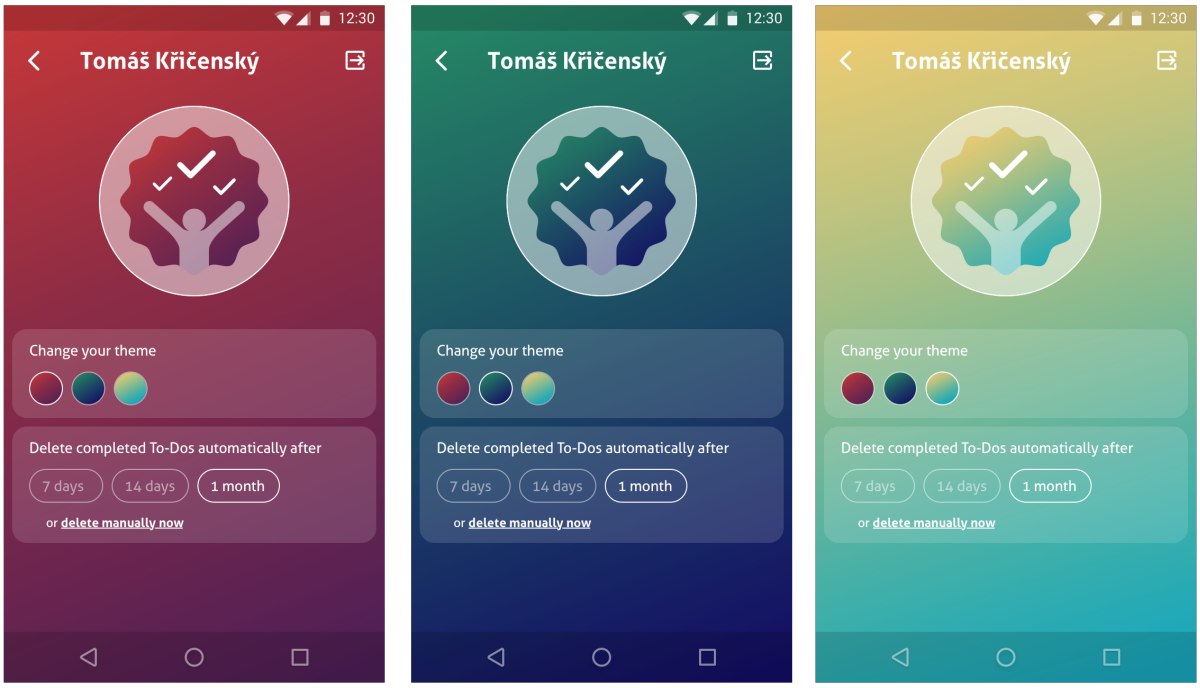
### 3.5.1 Změna vzhledu aplikace

Okamžitá synchronizace zobrazených informací neprobíhá pouze na seznamu úkolů, ale samozřejmě probíhá i na profilu uživatele. Zde se synchronizují veškerá uživatelská nastavení. Vždy tedy tato obrazovka zobrazuje aktuální vybrané hodnoty. Při automatické změně vybraných hodnot si ale aplikace musí dávat pozor, aby nezpůsobila další aktualizaci hodnot v databázi, která by znovu způsobila potřebu vykreslení dat. Tím by mohlo jednoduše dojít k nekonečnému cyklu. Z toho důvodu aplikace reaguje na změny vybraných hodnot pouze, když tyto hodnoty vybral přímo uživatel na tomto zařízení.

Synchronizováno není pouze nastavení, ale také samotný vzhled aplikace. Tudíž při každé detekované změně vzhledu je potřeba překreslit okno aplikace. Nezáleží přitom na tom, zda byla změna stylu způsobena tímto zařízením nebo zda se o změně aplikace dozvěděla díky synchronizaci databáze.

Na změny vzhledu, ale reaguje aplikace na každé obrazovce kromě té přihlašovací. Na přihlašovací obrazovce není možné reagovat na změny vzhledu, protože aplikace není ještě spojená s žádným účtem Google. Na všech ostatních obrazovkách se barvy mění okamžitě po detekci změny nastavení. Styl bylo možné změnit najednou v celé aplikaci, protože byla aplikace vyvinuta s využitím pouze jedné aktivity (dále popsáno v kapitole

„Navigace v aplikaci“). V případě, že tedy aplikace detekuje změnu nastavení barevného stylu, stačí pouze restartovat jedinou běžící aktivitu a při startu jí nastavit nově vybraný styl jako aktuální. Díky tomu se změní pozadí všech obrazovek a také se nastaví specifické barvy pro každý styl. Tyto specifické barvy ovlivňují například barvu textu v některých tlačítkách nebo i barvu hlavičky aplikace v seznamu spuštěných aplikací. Jak vypadají jednotlivé barevné styly ukazuje obrázek 7.




Obrázek 7: Ukázka různých barevných stylů aplikace.

# 4 POPIS A REALIZACE APLIKACE

Cílem této práce je vytvořit aplikaci pro správu úkolů. Vytvořená aplikace se jmenuje „Happy To-Dos“[6]. Tento název má své opodstatnění. Výchozí barevný vzhled aplikace je uklidňující a může navozovat šťastné myšlenky při plnění úkolů. Vzhled si může uživatel sám vybrat ze 3 předdefinovaných možností, ale všechny barevné varianty se drží stejného stylu. Na pozadí všech obrazovek je tedy barevný přechod z levého horního rohu do pravého dolního.

## 4.1 Navigace v aplikaci

Celá aplikace byla vytvořena s využitím jediné aktivity. Všechny obrazovky jsou tedy pouze fragmenty, které se střídají (nebo i zobrazují současně) v jedné aktivitě. Pro realizaci navigace v aplikaci byla využita knihovna z Android Architecture Components od Google. Tato knihovna byla vytvořena přímo pro navigaci mezi fragmenty v rámci jedné aktivity.

Bylo tedy nutné definovat všechny možné navigační stavy. To znamená u každé obrazovky zapsat do kterých dalších obrazovek z ní může uživatel navigovat. Tyto stavy se zapisují do XML souboru a jejich vizualizaci ukazuje příloha B.

První obrazovkou, kterou uvidí uživatel po prvním spuštění aplikace je přihlašovací obrazovka. Z té je možné navigovat po úspěšném přihlášení na seznam úkolů. Ze seznamu úkolů je možné přejít na profil uživatele pomocí ikony v pravém horním rohu obrazovky. Po kliknutí na tlačítko „Přidat úkol“ přejde uživatel ze seznamu úkolů na obrazovku pro přidání úkolu. Poslední navigační možností je kliknutí na samotný úkol v seznamu. Po této akci uživatel přejde na obrazovku pro úpravu existujícího úkolu.

Z obrazovek pro přidání a úpravu úkolů je možné se vrátit zpět na seznam úkolů pomocí křížku v levém horním rohu obrazovky. Z obrazovky pro přidání úkolu je také možné se přesunout zpět po úspěšném přidání úkolu pomocí tlačítka „Přidat“.

Poslední obrazovkou, ze které je možné někam odejít, je profil uživatele. Z profilu je možné se vrátit na seznam úkolů pomocí „šipky“ v levém horním rohu. Také je možné se z aplikace odhlásit pomocí ikony v pravém horním rohu. Po odhlášení se uživatel vrátí na přihlašovací obrazovku.

[6] Česky „Šťastné úkoly“ – název se ale nepřekládá.

### 4.1.1 Vlastní navigátor

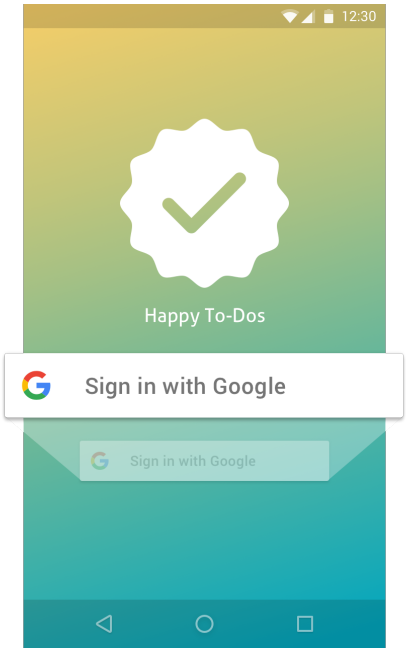
Při běžné navigaci se pouze nahradí obsah okna novým fragmentem, ale u několika obrazovek (přidání a úprava úkolu) bylo nutné, aby se zobrazovaly nad aktuální obrazovkou a obsah pod nimi zůstal z části viditelný. Z tohoto důvodu bylo nutné implementovat vlastní navigační logiku pomocí nového navigátoru.

Každý navigátor dokáže navigovat do fragmentů definované pomocí jednoho tagu. U tohoto navigátoru byl využit vlastní tag „blurred_fragment“. Tento navigátor tedy nenahradil obsah v hlavním okně, ale způsobil rozmazání hlavního obsahu a zobrazil nový fragment nad ním.

Při navigaci z jedné z těchto speciálních obrazovek se překrytí hlavního obsahu skryje a jeho rozmazání se zruší. Poté je tedy znovu zobrazen stejný obsah, který byl předtím vidět rozmazaný.

## 4.2 Přihlášení uživatele

První obrazovka, kterou uživatel po spuštění aplikace uvidí, je přihlašovací obrazovka. Na této obrazovce je bílé logo aplikace a také její název. Pod názvem aplikace je známé tlačítko pro přihlášení pomocí účtu Google. Pokud by přihlášení z nějakého důvodu selhalo, zobrazí se pod přihlašovacím tlačítkem chybová hláška spolu s kódem chyby. Následující obrázek ukazuje vzhled přihlašovací obrazovky.



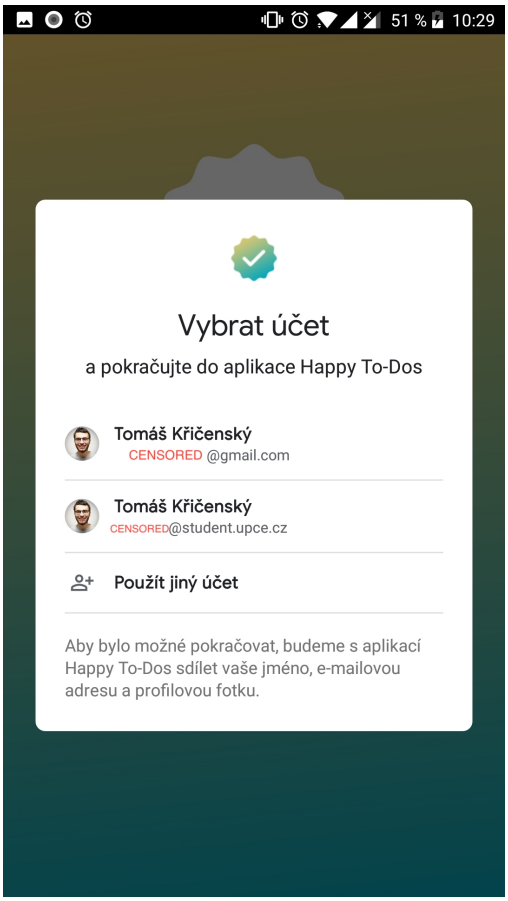

Obrázek 8: Přihlašovací obrazovka se zvýrazněným přihlašovacím tlačítkem.

Pokud se uživatel jednou do aplikace přihlásí, tak se už nemusí nikdy přihlašovat znovu (pokud se sám neodhlásí), protože si aplikace pamatuje naposledy přihlášeného uživatele. Tato funkce o hodně zrychlí a zpříjemní používání aplikace.

### 4.2.1 Google Sign in

Jako přihlašovací služba bylo zvoleno přihlášení pomocí účtu Google. Je to pro uživatele příjemnější než si pamatovat další e-mail a heslo. A také odpadá zdlouhavé psaní hesla v aplikaci. Dalším dobrým důvodem pro volbu externího poskytovatele přihlášení je, že aplikace vůbec nezná hesla uživatelů a nemusí je tedy nikde ukládat a zabezpečovat.

Google byl zvolen jako poskytovatel přihlášení, protože každý uživatel Androidu, který si aplikaci stáhne z Google play již Google účet má a nebude se muset složitě registrovat a přihlásí se tak jedním kliknutím. Pokud by se stalo, že uživatel Google účet nemá, tak si může přímo v aplikaci vytvořit účet nový. Pokud by naopak měl uživatel účtů několik, může si jednoduše vybrat, který chce použít, pomocí výběru z přihlašovacího dialogu. Tento dialog ukazuje obrázek 9.

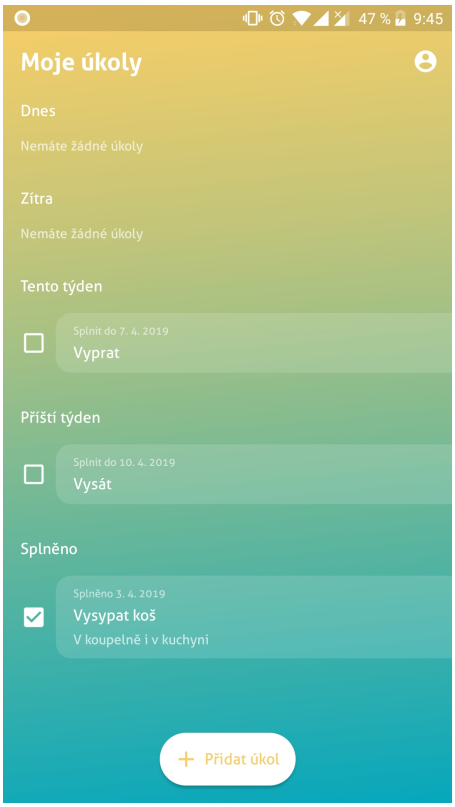



Obrázek 9: Dialog pro výběr účtu pro přihlášení.

## 4.3 Seznam úkolů

Hlavní obrazovkou celé aplikace je obrazovka se seznamem úkolů. Na této obrazovce jsou vypsány všechny úkoly přihlášeného uživatele. Všechny vypsané úkoly jsou seřazeny podle času do kdy mají být splněny. První v seznamu jsou ty úkoly, které mají nejdřívější datum plánovaného splnění.

Tyto úkoly jsou rozdělené do různých sekcí podle času, kdy mají být splněny. Pokud v sekci není žádný úkol, tato sekce je stále v aplikaci zobrazená spolu s hláškou, že v sekci nejsou k dispozici žádné úkoly. Možné sekce pro jednotlivé úkoly jsou následující: „Pozadu“, „Dnes“, „Zítra“, „Tento týden“, „Příští týden“ a „Později“. Názvy sekcí jsou jasně pochopitelné a není potřeba je dále vysvětlovat. Sekce „Pozadu“ a „Později“ nejsou vždy viditelné jako všechny ostatní, ale pokud v nich nejsou žádné úkoly, tak se uživateli vůbec neukazují.



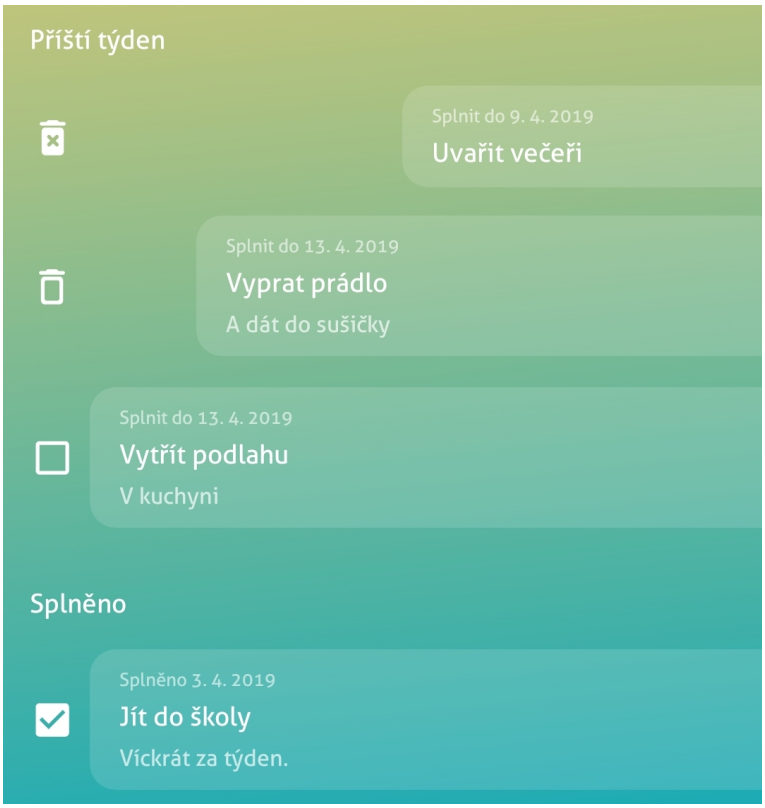

Obrázek 10: Hlavní obrazovka aplikace – seznam úkolů.
Obrázek v plné velikosti ukazuje příloha C.

Speciální sekcí je sekce „Splněno“. Jak již název napovídá, tak jsou v této sekci vypsány všechny splněné úkoly. Tyto úkoly nejsou seřazeny podle data předpokládaného splnění úkolu, ale podle reálného data splnění.

U každého úkolu je vypsán jeho název. Pokud má úkol vyplněný i popis, vypíše se pod titulkem. Popis úkolu může být libovolně dlouhý, a proto se v seznamu vypíše maximálně na 3 řádky. Nad titulkem je vypsáno datum, kdy by měl být úkol splněn. U splněných úkolů je vypsáno datum, kdy byl úkol opravdu splněn. Vlevo vedle titulku je zaškrtávací políčko. Zaškrtnutím tohoto políčka se nastaví úkol jako splněný.

### 4.3.1 Mazání úkolů

Úkoly je samozřejmě i možné mazat i pokud nejsou splněny. Úkoly lze tedy smazat pomocí přetažení karty s úkolem zleva doprava. Při tomto pohyhu se plynule animuje zaškrtávací políčko do tvaru popelnice. Tato animace naznačí uživateli, že se jedná o mazání úkolu. Při přetáhnutí úkolu přes 50 % šířky obrazovky popelnice znovu změní tvar. Tentokrát se změní na vyplněnou ikonku popelnice. Touto změnou aplikace uživateli naznačí, že úkol bude smazán, pokud uživatel přestane tahat s úkolem a zvedne prst. Všechny zmíněné animace fungují i při pohybu prstu opačným směrem. Zaškrtávací tlačítko se také plynule animuje vždy a nezáleží tedy na tom, zda je zaškrtnuto.




Obrázek 11: Koncové stavy animace mazání úkolu.

Těchto animací bylo dosaženo pomocí třídy „AnimatedVectorDrawable“, která je standardní součástí androidu od verze 5.0. Tato třída umožňuje animace vektorových obrázků. V této práci bylo využito schopnosti animovat cestu, která definuje tvar obrázku do jiné cesty. Tyto cesty ale musí být kompatibilní a to většinou u rozdílných obrázků nebudou. Z toho důvodu jsem využil online převodník „Shape Shifter“[7], který vygeneroval pro všechny animované obrázky kompatibilní cestu.

Každá z animací se definuje ve vlastním souboru a neexistuje možnost animaci provést v opačném směru. Tudíž pro animaci mezi dvěma stavy a zpět jsou potřeba definovat hned dva soubory s animacemi. Animovat mezi sebou můžeme pouze obrázky. Takže když uživatel začne tahat s úkolem do strany, je nutné vyměnit funkční zaškrtávací políčko pouze za obrázek, který již lze animovat do tvaru popelnice.

## 4.4 Přidání úkolu

Obrazovka pro přidání úkolu je zobrazena přímo nad seznamem úkolů. Seznam úkolů je pod ní částečně vidět, ale je rozmazaný. Díky rozmazání neodvádí pozadí příliš pozornost a pouze přidává zajímavý efekt. Tuto obrazovku ukazuje obrázek 12.

Při otevření obrazovky se označí textové pole pro zadání názvu úkolu a zobrazí se klávesnice. Klávesnice posune zbytek uživatelského rozhraní nahoru, takže uživatel stále na všechno vidí. Pod textovým polem jsou dvě ikony, které fungují jako tlačítka.

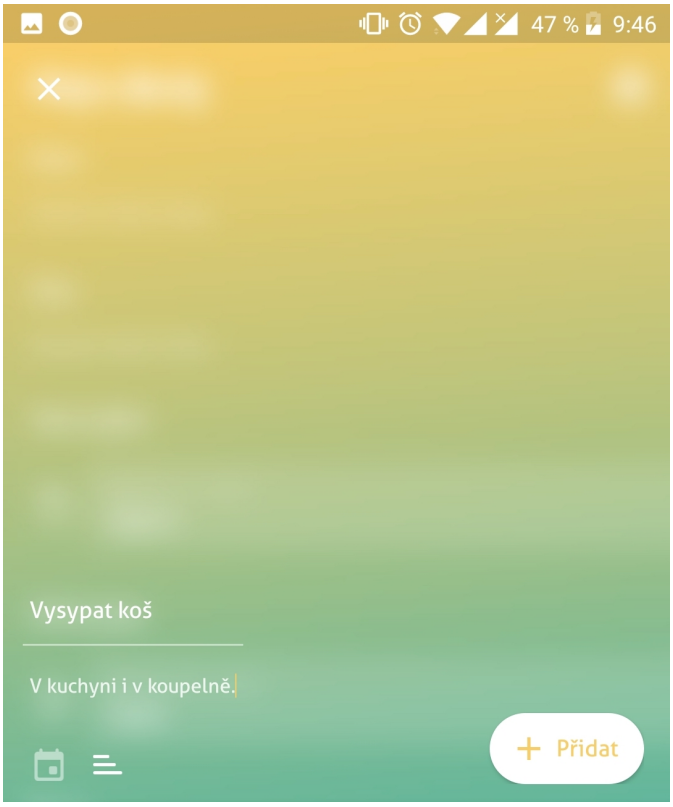
Po kliknutí na ikonu kalendáře se objeví dialog pro výběr data, do kdy má být úkol splněn. V tomto dialogu nelze vybrat datum z minulosti a je tedy možné naplánovat úkoly pouze na dnešek a nebo do budoucnosti. Tento dialog je možné i opustit bez změny výběru pomocí kliknutí mimo oblast dialogu. Také je možné zrušit dříve vybrané datum pomocí tlačítka „Zrušit výběr“. Po úspěšném výběru data se zvolené datum zobrazí vedle ikony kalendáře a ikona změní barvu na čistě bílou.

Vedle ikony kalendáře se nachází ještě ikona tří čar. Po kliknutí na tuto ikonu odjede textové pole s názvem úkolu nahoru a pod ním se ukáže pole pro vyplnění popisu. Narozdíl od pole pro název lze do tohoto pole psát delší text (na více řádků). Podobně jako ikona kalendáře i tato ikona mění barvu dle toho, zda je pole pro zadání popisu úkolu viditelné.

Posledním důležitým prvkem na této obrazovce je tlačítko „Přidat“. Jak již název napovídá, tak po kliknutí na toto tlačítko bude přidán úkol do seznamu. Při úspěšném přidání

[7]Dostupný na adrese `https://www.shapeshifter.design`.

úkolu se také tato obrazovka zavře a uživatel se vrátí zpět na seznam úkolů. Na toto tlačítko, ale nejde kliknout kdykoliv, protože úkol musí projít validací. Úkol je validní v případě, že název obsahuje alespoň jeden viditelný znak. Pokud úkol při ukládání neobsahuje vyplněné datum, tak se automaticky nastaví datum předpokládaného splnění na dnešní den.



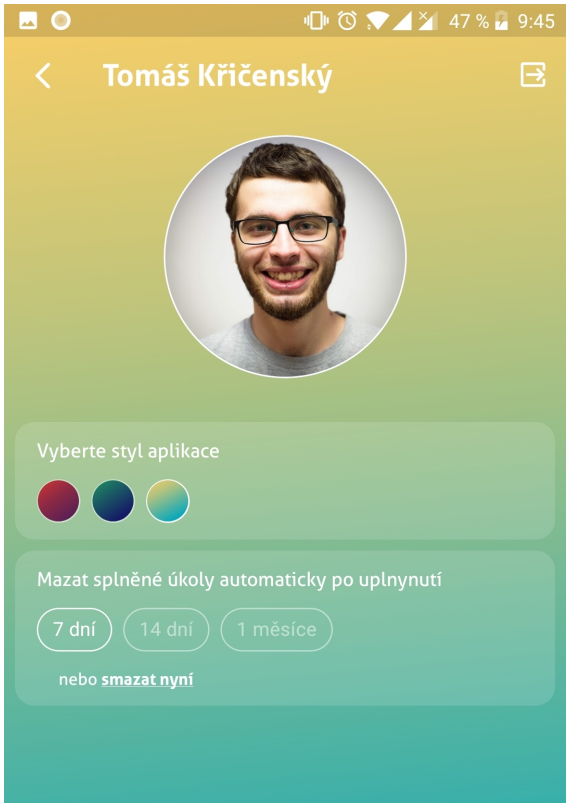

Obrázek 12: Obrazovka pro přidání úkolu.

## 4.5 Úprava úkolu

Obrazovka pro úpravu vypadá a funguje stejně jako obrazovka pro přidání úkolu. Hlavním rozdílem je, že na této obrazovce není tlačítko „Přidat“. Protože zde není tlačítko přidat, funguje ukládání na této obrazovce jinak. Při pokusu o opuštění obrazovky (křížkem v levém horním rohu nebo systémovým tlačítkem zpět) se pokusí aplikace vyplněné údaje uložit do databáze. To ale nemusí nutně uspět, protože uživatel mohl smazat název úkolu, který je pro uložení povinný. V případě, že název úkolu není vyplněn, tak se při opuštění obrazovky změny neuloží.

## 4.6 Profil uživatele

Poslední obrazovkou v aplikaci je profil uživatele. Na této obrazovce jsou vypsány informace o přihlášeném uživateli. Nahoře v titulku stránky je vypsáno plné jméno uživatele a hned pod jménem je v kruhu jeho fotka. Pokud přihlášený uživatel nemá na svém Google účtu nahranou fotografii, zobrazí se zástupný obrázek barevně přizpůsobený uživatelskému rozhraní. Uživatelské rozhraní profilu ukazuje obrázek 13 a zástupný obrázek v různém barevném provedení ukazuje obrázek 7.




Obrázek 13: Obrazovka profil uživatele.

Z profilu může uživatel odejít zpět na seznam úkolů pomocí ikony v levém horním rohu obrazovky. Také se uživatel může odhlásit kliknutím na ikonu v pravém horním rohu. Při odhlášení neproběhne pouze navigace na přihlašovací obrazovku, ale také aplikace zapomene veškeré citlivé informace o přihlášeném uživateli. Takže po restartu aplikace už nebude naposledny přihlášený uživatel automaticky přihlášen do aplikace. Jediná informace, která se odhlášením nesmaže, je vybraný vzhled aplikace. Tudíž i přihlašovací obrazovka zůstane barevná dle předvolby posledního přihlášeného uživatele.

Tato obrazovka ale neslouží pouze pro zobrazení informací o uživateli. Také zde může uživatel měnit různá nastavení, která se týkají pouze jeho. Prvním nastavením je změna barevného stylu aplikace. Uživatel si může vybrat jeden ze tří předdefinovaných stylů. Každý z těchto stylů je v nastavení prezentován malým kruhem, který má na pozadí stejný barevný přechod jako bude mít aplikace po výběru této možnosti. Kolem vybraného stylu je vykreslen bílý rámeček. Ve výchozím nastavení je aplikace ve žluto-zeleném barevném stylu.

Druhým nastavením na této obrazovce je po jaké době se mají automaticky mazat splněné úkoly. Tato možnost existuje proto, že by nebylo vhodné splněné úkoly uživateli rovnou mazat. Mohlo by se stát, že uživatel některý úkol splní omylem a chtěl by ho ještě vrátit mezi nesplněné. Ale také by nebylo dobré, aby si aplikace všechny splněné úkoly archivovala. Prvním důvodem je, že splněné úkoly jsou vypsané v seznamu úkolů a nebylo by dobré zbytečně načítat z databáze mnoho splněných úkolů. Druhým důvodem je šetření místa na serveru. Kdyby bylo uživatelům poskytnuto neomezené úložiště, mohlo by brzo dojít k zahlcení databáze. Z těchto důvodů si může uživatel vybrat po jaké době se mu budou mazat splněné úkoly. Má tedy na výběr z následujících možností: 7 dní, 14 dní, nebo 1 měsíc. Ve výchozím nastavení se splněné úkoly mažou po uplynutí sedmi dní.

Poslední možností na této stránce je možnost okamžitě smazat všechny splněné úkoly. Po kliknutí na text „smazat nyní“ se zobrazí potvrzovací dialog, ve kterém se uživatel ještě může rozhodnout, zda chce opravdu smazat všechny splněné úkoly. Tento dialog lze zavřít kliknutím do ztmaveného pozadí vedle dialogu. Také si uživatel může vybrat že ve skutečnosti nechce smazat všechny splněné úkoly a tím pouze skrýt dialog. Při výběru kladné možnosti se okamžitě provede smazání všech splněných úkolů právě přihlášeného uživatele. Pokud by uživatel nechtěl mazat úkoly všechny, musel by je mazat klasickým způsobem na obrazovce se seznamem všech úkolů.

# ZÁVĚR

Cílem této práce bylo vytvoření Android aplikace pro správu úkolů. Tato aplikace měla být implementována pomocí návrhového vzoru Model-View-ViewModel a měla komunikovat se vzdálenou databází, aby ji bylo možné používat na více zařízeních současně.

Všech vytyčených cílů bylo dosaženo. Všechny obrazovky aplikace byly implementovány dle daného návrhového vzoru a kód byl logicky strukturován do balíčků pro snadnou rozšiřitelnost práce o další funkce nebo i nové obrazovky.

Pro možnost používání aplikace na více zařízeních jednoho uživatele bylo využito přihlášení pomocí účtu Google. Díky využití stejného účtu na více zařízeních bylo možné ukázat uživateli vždy pouze jeho úkoly. Schopnosti zobrazovat na všech zařízeních jednoho uživatele stejné úkoly bylo dosaženo díky komunikaci se vzdálenou databází Cloud Firestore. Část této databáze je také duplikována do lokálního úložiště a díky tomu je možné používat aplikaci i bez přístupu k internetu.

Do budoucna by bylo možné aplikaci rozšířit o další funkce, jakými jsou třeba implementace upozornění na úkoly nebo přetahování úkolů mezi sekcemi. Dalším možným rozšířením by byla možnost sdílet úkoly s dalšími lidmi a mít možnost se společně podílet na jejich řešení. Tyto funkce nejsou pro aplikaci v této fázi stěžejní, ale představovaly by příjemné rozšíření existující funkcionality.

# POUŽITÁ LITERATURA


[1] DUTSON, Phil. *Android development patterns: best practices for professional developers*. Boston: Addison-Wesley, 2016. ISBN 978-0-133-92368-1.

[2] Fragments. In: *Android Developers* [online]. [cit. 2019-04-07]. Dostupné z: https://developer.android.com/guide/components/fragments

[3] HOWARD, Joe a Namrata BANDEKAR. Activity lifecycle pyramid. In: *Raywenderlich.com* [online]. 26 Jul 2017 [cit. 2019-04-07]. Dostupné z: https://www.raywenderlich.com/500-introduction-to-android-activities-with-kotlin

[4] JEMEROV, Dmitry a Svetlana ISAKOVA. *Kotlin in action*. Shelter Island, NY: Manning Publications Co., 2017. ISBN 978-161-7293-290.

[5] NURKIEWICZ, Tomasz a Ben CHRISTENSEN. *Reactive programming with RxJava: creating asynchronous, event-based applications*. Sebastopol, CA: O'Reilly Media, 2016. ISBN 978-1-491-93165-3.

[6] Cloud Firestore. *Firebase* [online]. 31 Jan 2019 [cit. 2019-04-07]. Dostupné z: https://firebase.google.com/docs/firestore/

[7] KUMAR S, Ashok. *Mastering Firebase for Android Development: Build real-time, scalable, and cloud-enabled Android apps with Firebase*. Birmingham: Packt Publishing, 2018. ISBN 978-1788624718.

[8] MongoDB Mobile: The embedded database from MongoDB. *MongoDB* [online]. [cit. 2019-04-07]. Dostupné z: https://www.mongodb.com/products/mobile

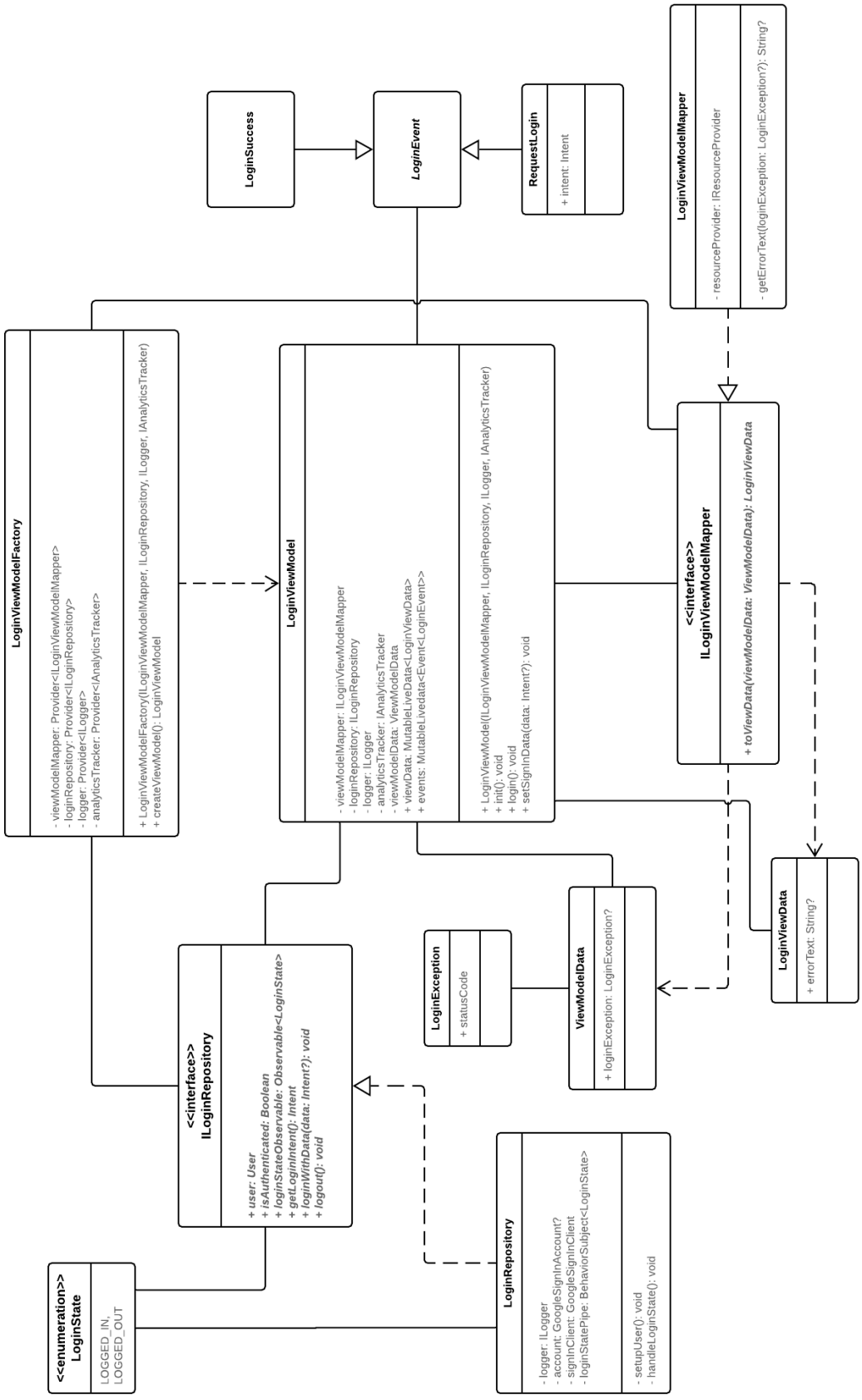
[9] MongoDB Stitch: The serverless platform from MongoDB. *MongoDB* [online]. [cit. 2019-04-07]. Dostupné z: https://www.mongodb.com/cloud/stitch

[10] MAINKAR, Prajyot. *Expert Android Programming: Master skills to build enterprise grade Android applications*. Birmingham: Packt Publishing, 2017. ISBN 978-1-78646-895-6.

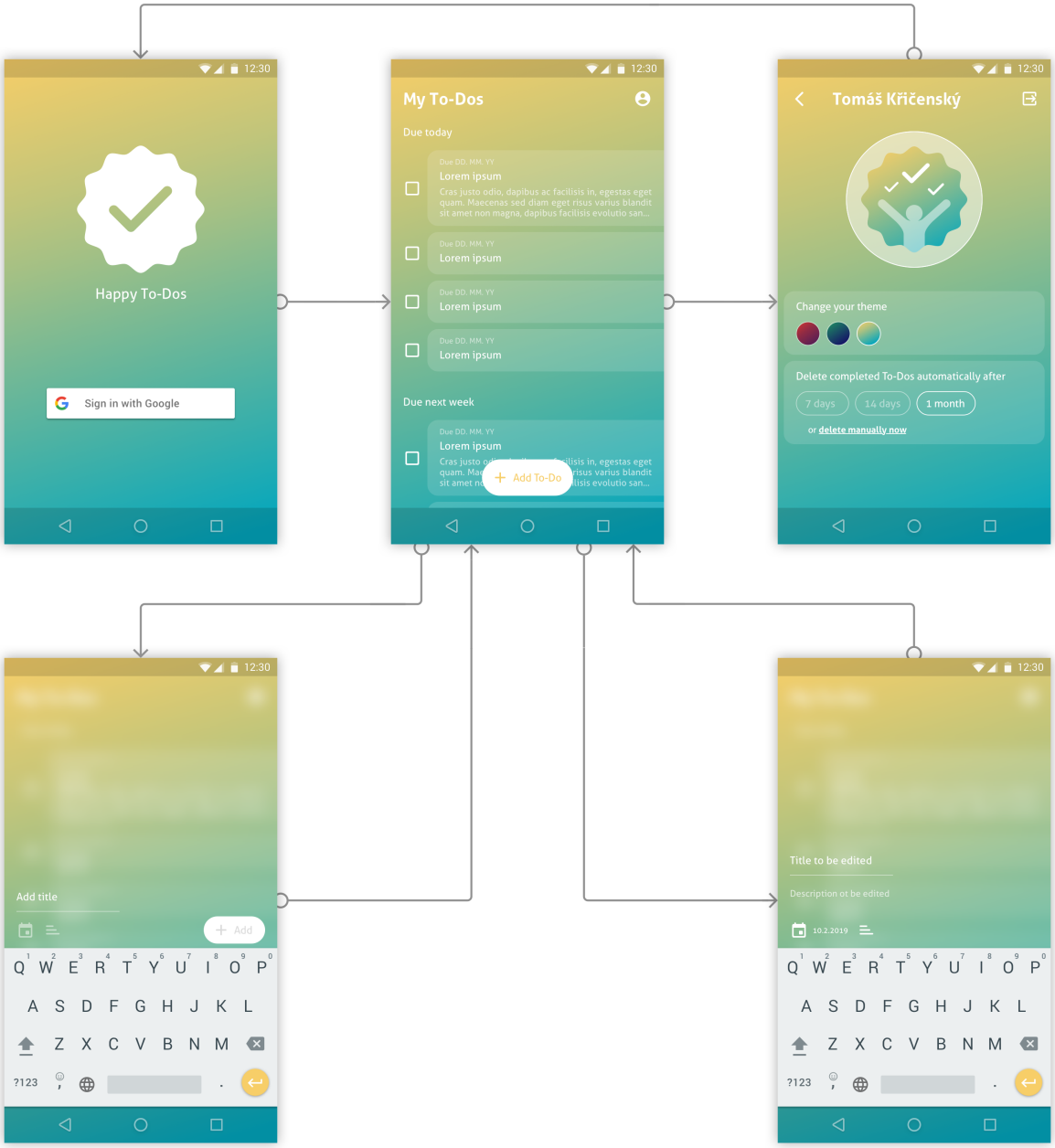
[11] Final architecture. In: *Android Developers* [online]. [cit. 2019-04-07]. Dostupné z: https://developer.android.com/jetpack/docs/guide

# SEZNAM PŘÍLOH

# PŘÍLOHA A – DIAGRAM TŘÍD



# PŘÍLOHA B – NAVIGACE V APLIKACI



# PŘÍLOHA C – SEZNAM ÚKOLŮ