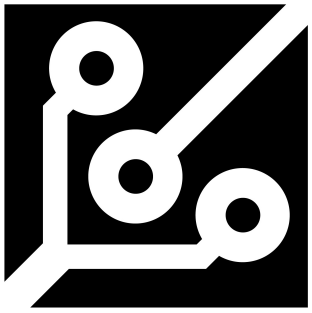
Take Your Pill

mobilní aplikace

SPŠE V Úžlabině



Vojtěch Hořánek

I4.D

15.03.2021

„Prohlašuji, že jsem tuto práci vypracoval samostatně a použil jsem literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů informací.“

V Praze dne ....

podpis autora

Anotace

Předložená práce je mobilní aplikace pro systém Android, která slouží k připomínání užití léků. V aplikaci je implementovaná historie připomínání a užívání léků, statistiky a grafy. Uživatel snadno získá přehled, jaké a kdy léky vynechává a může si nastavit opakované připomínání, aby již lék nezmeškal. Design aplikace je udělán tak, aby odpovídal material designu a jeho nejnovějším trendům. Aplikace je napsána v jazyce Kotlin a využívá moderních knihoven a technologií.

Anotation

The presented work is a mobile application for the Android operating system which reminds its users to take their pills. History, statistics, and graphs are implemented in the application. The user can effortlessly get an overview of which pills at what time did they miss and can set repeating reminders, so they do not forget them the next time. The application follows the material design guidelines and its latest trends. Kotlin was used as the programming language and utilizes modern libraries and technologies.

Obsah

Úvod

Cílem této práce bylo vytvořit aplikaci, která uživatelům usnadní pravidelné užívání léků jejich připomínáním, sledováním historie a statistik. Pro každý lék lze nastavit, aby se připomínal pouze určitý počet dní nebo aby se připomínal v cyklu X dní aktivní Y dní neaktivní.

Práce se skládá z mobilní aplikace pro systém android. Aplikace je navržena co nejjednodušeji a rozdělena do dvou hlavních sekcí: léky a historie. V sekci „léky“ uživatel nalezne léky, které si do aplikace přidal. V seznamu léků je zobrazen jejich název, popis, barva, fotografie, časy připomínek a příjem. V sekci „historie“ může uživatel sledovat užití svých léku, zobrazí se mu kompletní historie (včetně kdy a jestli si lék vzal, kdy mu byla poslána připomínka a kolik prášků si vzal) a grafy zobrazující souhrnné informace. Součástí práce je i propagační a informační plakát.

Kapitola 1

Návrh aplikace

Design aplikace jsem navrhoval před samotnou implementací, nutno ale dodat, že při implementaci prošel design několika iteracemi. Aplikaci jsem původně koncipoval jako jedinou hlavní obrazovku, kde by se uživateli ukázali všechny důležité informace. Postupem času se toto řešení ukázalo jako nevhodné a nepraktické, zvolil jsem proto více tradiční postup a to rozdělení aplikace do tří přehledných sekcí: *Léky*, *Historie* a *Nastavení*. Každá obrazovka obsahuje velký nadpis a teprve potom samotný obsah. Mimo spodních dialogů[1] není nadpis nijak ohraničen, pouze je odsazen. Změny mezi různými obrazovkami doprovázejí animace, které jsou implementovány podle Material Designu. Aplikace díky těmto animacím vypadá svižněji.

Celé rozhraní jsem upravil pomocí vlastních stylů, které vycházejí ze stylů Material Design [12]. Pro některé prvky aplikace, jmenovitě titulky a tlačítka, jsem použil písmo *Jost* [10]. Ikony použité v aplikaci jsou z knihovny Material Design Icons [13], ikonu aplikace jsem získal od Austina Andrewse [1] a grafika léků je dostupná na GitHubu pod názvem *material-icons* [18]. Nechybí ani podpora světlého a tmavého designu, který se dá přepínat v nastavení (VOJTO TODO REFERENCE).

[1] tím myšleno BottomSheetDialogFragment

Kapitola 2

Implementace aplikace

Při vytváření aplikace jsem vycházel ze zadání a využil jsem vlastních znalostní a zkušeností. Na naprogramování aplikace jsem použil programovací jazyk Kotlin. Zvolil jsem ho proto, že je preferovaný společností Google a oproti jazyku Java má mnoho výhod. Mnoho Android knihoven vychází právě pro Kotlin a tak mi jeho použití ulehčilo mnoho práce při programování. Jmenovitě knihovny z rodiny Android Jetpack [2] jsem použil hojně.

Uživatelské rozhraní aplikace je napsáno v jazyce XML. Pro jeho manipulaci jsem použil knihovny *ViewBinding* [9] a *DataBinding* [5]. Pro snadnější práci s ViewBindingy jsem využil knihovnu *FragmentViewBindingDelegate-KT* [19][20], která mi zpřístupnila „delegát“ `byviewBinding()` s jehož pomocí jsem mohl layout inicializovat (a zahodit) jen jednou řádkou kódu.

Při samotném vývoji jsem používal vývojové prostředí *Android Studio* [11] a emulátor *Android Emulator*.

Aplikace je napsána tak, aby odpovídala architektuře **Model-View-ViewModel**. Znamená to, že každá obrazovka má svůj *ViewModel* a každá datová sekce má svůj *Repozitář*. Tyto třídy jsou odděleny od samotných fragmentů a aktivit. Pro získávání dat asynchronně používá aplikace třídu `LiveData` [7].

ViewModel je třída obsahující data a metody pro její fragment/aktivitu, která má vlastní životní cyklus. Díky ViewModelu data přežijí změnu konfigurace jako například otočení obrazovky.

Repozitář (repository) je třída, která shromažďuje data z různých zdrojů a nabízí je ve vhodné formě ostatním třídám (například může data uchovat v mezipaměti). V této aplikaci přistupuje repozitář pouze do databáze a ve většině případů přímo volá metody implementované v databázové vrstvě.

Aby se aplikace snáze programovala a byla více škálovatelná, rozhodl jsem se použít tvz. „automated dependency injection“ [4]. Doporučená knihovna pro automated dependency injection `Hilt` [6] je sice v beta verzi, avšak já jsem se ji rozhodl využít, jelikož mně u předchozích projektů fungovala bezproblémově.

Automated dependency injection (česky automatizované vkládání závislostí) je technika, která zajistí, že třídy, které k činnosti potřebují ostatní třídy, tyto třídy dostanou automaticky a v jakémkoli podporovaném kontextu. Když například repozitář potřebuje ke své funkci třídu, která přistupuje do databáze, automated dependency injection zařídí, že tuto třídu automaticky dostane a programátor ji nemusí explicitně vytvářet. V případě hiltu je možné využít i vytváření *singletonů*, což zajistí, že instance třídy existuje v aplikaci pouze jednou a jiné třídy dostanou právě tuto instanci.

2.1 Uživatelské rozhraní

Hlavní obrazovka aplikace je rozdělena na tři části: *Léky*, *Historie* a *Nastavení*. Uživatel mezi těmito sekcemi přepíná pomocí prvku `BottomNavigationView`. Do sekce *Historie* a na obrazovku *Přidat lék* se může uživatel dostat i pomocí zkratky[1] z domovské obrazovky. Celá aplikace obsahuje pouze tři aktivity: MainActivity, AboutActivity a AppIntroActivity. Všechny ostatní obrazovky jsou implementovány jako fragmenty a pro jejich navigaci byla použita knihovna *Navigation* [8].

2.1.1 Úvodní obrazovka

Pro přidání úvodní obrazovky jsem použil knihovnu *material-intro* [17]. Tato knihovna se postará o všechen layout a logiku úvodní obrazovky. Má jednoduché API a tak jediné, co jsem do aplikace přidal, byla aktivita `AppIntroActivity` dědící ze třídy `IntroActivity` a v ní přidal slidy pomocí metody `addSlide()`. Obrázky ve slidech jsem vyfotil v android emulátoru a upravil v programu *GIMP*. Úvodní obrazovka se spustí pouze když uživatel

[1] zktratky využívají App Shortcuts API

aplikaci spustí poprvé. Toto je zajištěno tak, že do trvalé paměti aplikace[1] se po ukončení této aktivity uloží proměnná `firstRun` s hodnotou `false`. Snímek obrazovky výsledku lze vidět na straně 13 (2.1a).

2.1.2 Domovská obrazovka

Domovská obrazovka neboli sekce „Léky“ (`HomeFragment`) je fragment obsahující pouze `RecyclerView` (dále jen recycler) a `ExtendedFloatingActionButton` (dále jen FAB). FAB se při posunutí recycleru zmenší. Zvětší se až když seznam posuneme na začátek.

Pro zobrazení dat v recycleru je potřeba mít `RecyclerAdapter`. Tato třída se stará o zobrazování dat s příslušným `ViewHolderem`. Adaptér, který je nastavený na tomto recycleru se jmenuje `AppRecyclerAdapter`. Každý *ViewHolder* pro třídy *Pill* obsahuje kartu, jejichž layout je definován v souboru `item_pill.xml`. Na kartě se zobrazují všechny potřebné informace o léku: název, popis, barva, fotografie, připomínky a příjem. Pokud uživatel nepotvrdil nejnovější připomínku v posledních 30 minutách, zobrazí se na kartě i výzva k potvrzení. Po kliknutí na kartu se otevře detail příslušného léku. Snímek obrazovky sekce „Léky“ lze vidět na straně 13 (2.1a).

ViewHolder (doslovný překlad „držitel pohledu“) je třída, která se stará o zobrazení jedné položky v RecyclerView Adaptéru. Má za úkol nastavit layout tak, aby odpovídal vstupním datům (např. jednomu léku). Třídu si musíme definovat sami pro každý typ položky, které chceme zobrazovat.

AppRecyclerAdapter je `RecyclerAdapter` založený na `ListAdapter`. Tento adaptér se používá pro většinu seznamů v aplikaci, jelikož umožnuje přidat titulek a zobrazit prázdný stav. Toto je dosaženo přepsáním metody `submitList` a dosazením speciálních položek (`HeaderItem` a `EmptyItem`). Adaptér podporuje třídy, které dědí z `BaseModel`, jmenovitě `Pill`, `HistoryPillItem`, `HeaderItem` a `EmptyItem`.

2.1.2.1 Detail léku

Obrazovka léku (`DetailsFragment`) je implementována jako fragment s layoutem `fragment_details.xml`. Na obrazovce lze vidět název léku, jeho popis a fotografii (pokud tyto položky má), připomínky a příjem. Při dlouhém podržení na fotografii se fotografie zobrazí

[1] trvalou pamětí je myšleno úložiště SharedPrefs

v plné velikosti. Pokud obrazovku otevřeme z oznámení, nebo má lék nepotvrzenou připomínku v posledních 30 minutách, zobrazí se nad titulkem karta vyznívající k potvrzení této připomínky. Na spodní části obrazovky jsou tlačítka „smazat“, „historie“ a „upravit“. Tlačítko „smazat“ otevře dialog, kde uživatel může zvolit, zda chce smazat pouze lék a zachovat jeho historii, nebo ho smazat i s historií. Dialog je implementovaný ve třídě `DeleteDialog`. Tlačítko „historie“ otevře dialog, ve kterém se zobrazí historie pro tento lék (`HistoryViewDialog`). Více o tomto dialogu naleznete na straně 8. Tlačítko „upravit“ otevře `EditFragment`, kde může uživatel lék upravit. Více o této obrazovce v následující sekci.

2.1.2.2 Nový lék

Obrazovka „Nový lék“ (`EditFragment`) má dvě funkce. Slouží jako obrazovka pro přidávání nového léku a zároveň se používá pro úpravu léku. Titulek obrazovky se mění podle použití. Opět je implementována jako fragment. Pro každý lék je možno nastavit název a popis. Tyto dvě hodnoty se zapisují do prvku `TextInputLayout` z knihovny *material*. Následuje nastavení barvy. Uživatel má na výběr ze sedmi barev: modrá, tmavě modrá, tyrkysová, zelená, žlutá, oranžová a červená. Vybírání barvy je implementováno pomocí prvku `RecyclerView` s atributem `orientation` nastaveným na hodnotu `horizontal`. Jednotlivé barvy jsou definované třídou `PillColor`. Dále si uživatel nastaví připomínky. Při kliknutí na připomínku nebo na tlačítko „přidat připomínku“ se zobrazí `ReminderDialog` kde uživatel může upravit/vytvořit připomínku. U připomínky lze nastavit i množství léku, jaké si v daný čas má uživatel vzít. Pro jeden lék nelze nastavit dvě připomínky se stejným časem, každá připomínka musí mít unikátní čas. V neposlední řade lze léku nastavit příjem. Uživatel si může zvolit, zda lék bere neustále, jen určitý počet dní a nebo v cyklu X dní aktivních, Y dní neaktivních. Tento prvek[1] je implementován v `PillOptionsView`, který dědí z `LinearLayout` a používá layout `layout_pill_options_view.xml`. Veškerá logika výběru příjmu je implementována v této třídě. Jediné, o co se `EditFragment` musí postarat, je získání `ReminderOptions` (vysvětleno v kapitole 2.2.2) z tohoto prvku pomocí metody `getOptions()`.

K léku lze přidat i fotografii. Prvek na výběr fotografie je implementován v `ImageChooserView`. Tato třída dědí z `LinearLayout` a používá layout definovaný v `layout_image_chooser.`

[1] prvkem je myšlen prvek uživatelského rozhraní, neboli `view`

`xml`. Pokud lék již nějakou fotografii obsahuje, prvek zobrazí tlačítko na její odstranění. Při výběru fotografie je použita knihovna *EasyPermissions* [14] pro zajištění potřebných oprávnění a upravená verze knihovny *imagepicker* [16]. Knihovnu jsem upravil tak, aby respektovala vzhled aplikace. Zaprvé již nepoužívá standardní `AlertDialog`, nýbrž `BottomSheetDialog` a také vzhled tohoto dialogu byl upraven, aby odpovídal všem ostatním dialogům v aplikaci. Uživatel si může zvolit, zda chce fotografii vybrat z galerie, nebo chce vyfotit fotografii novou. Po vybrání/vyfocení se uživateli ukáže obrazovka, kde může fotografii upravit. Pro úpravu fotografie jsem použil knihovnu *uCrop* [21], kterou jsem také upravil. Oproti originální verzi se liší v použití prvků na navigaci, ikonách, podporou automatického tmavého vzhledu a sladěním do vzhledu aplikace. Knihovna uživateli umožní fotografii oříznout, otočit a škálovat. Pro plynulejší úpravy jsem zvolil „native“ verzi knihovny, knihovna tak využívá kód napsaný v C++, který je rychlejší a optimalizovaný pro jednotlivé architektury, avšak přidá k velikosti aplikace cca 1.5 MB.

Po nastavení všech hodnot může uživatel lék uložit pomocí `FAB` tlačítka. Aplikace lék vloží do databáze (pokud již existuje, tak jej aktualizuje) a naplánuje jeho následující připomínku.

PillColor je třída, používaná pro ukládání barvy léku. Obsahuje atributy `resource` a `isChecked`. Atribut `resource` ukládá `id` barvy uložené v *App resources* [3]. Atribut `isChecked` vyjadřuje, zda je barva vybraná. Tento atribut se používá k zobrazování seznamu barev a zjištění, jakou barvu uživatel vybral.

2.1.3 Historie

Druhá hlavní sekce aplikace se nazývá „historie“ a uživatel zde může pozorovat kdy mu přišli připomínky, kdy a jestli si lék vzal a jaké množství si vzal. V této sekci jsou dostupné i tři koláčové grafy, které uživateli vizuálně ukazují jeho statistiky. Sekce historie je implementována jako fragment, který je prvky `ViewPager2` a `TabLayout` rozdělen na dvě části. Tento fragment (`HistoryFragment`) neobsahuje žádnou logiku ohledně historie, pouze se stará o nastavení výše zmíněných prvků.

ViewPager2 a **TabLayout** jsou prvky, které rozdělují obrazovku na dva panely, které jsou přepínatelné pomocí posouvání. `ViewPager2` je nová verze `ViewPager`, která interně využívá `RecyclerView` jehož položky jsou fragmenty. `TabLayout` pouze zobrazuje záložky nad panely a dovoluje jejich přepínání pomocí stisku.

2.1.3.1 Přehled

Sekce „přehled“ spadá pod sekci „historie“ a je implementována jako fragment. Tento fragment se zobrazuje jako první položka v prvku `ViewPager2` v sekci „historie“. V tomto fragmentu je pouze prvek `RecyclerView`, který zobrazuje jak veškerou historii, tak historii pro jednotlivé léky. V seznamu se zobrazí karty s názvem a barvou léku a statistiky ukazující kolikrát byl lék připomenut, kolikrát byl potvrzen a kolikrát byl zmeškán. Po kliknutí jednu z karet se zobrazí `HistoryViewDialog`. V tomto dialogu má uživatel k dispozici seznam připomínek, které lék obdržel (v případě vybrání karty „Všechny léky“ se zobrazí seznam všech připomínek). Pro každou připomínku může uživatel rozbalit kontextové menu a provést jednu ze čtyř akcí. Uživatel může změnit stav potvrzení připomínky (potvrzená x nepotvrzená), může změnit čas potvrzení (pokud je připomínka potvrzena), může změnit množství, které si v daný čas vzal nebo měl vzít a v neposlední řadě může tuto připomínku smazat. Pokud by uživatel chtěl smazat všechny připomínky pro daný lék, může stisknout tlačítko smazat nacházející se v titulku dialogu.

Pro tento seznam jsem zvolit odlišný způsob odlišení různých typů položek. Jelikož jsem potřeboval při zobrazení všech léku zobrazit i název léku a v případě první připomínky dne zobrazit i datum, zvolil jsem pouze jeden `ViewHolder`, který v sobě obsahuje všechny prvky. Toto se liší od způsobu odlišení různých položek v `AppRecyclerView`, kde je využito ViewHolderů několik. Layout jednotlivých položek je definovaný v `item_history.xml`. Jednotlivé prvku jsou skryty a zobrazeny podle potřeby a layout je udělán tak, aby vypadal konzistentně při všech typech zobrazení.

Další specialitou tohoto seznamu je aktualizování okolních připomínek při odstraňování. Toto zajistí správné zobrazení data a předělové čáry.

2.1.3.2 Grafy

Druhou sekcí historie je fragment `HistoryChartFragment`, který se stará o zobrazení grafů. Fragment obsahuje kartu se třemi koláčovými grafy, touto kartou lze posouvat pro zobrazení i těch grafů, které se na obrazovku nevešly. Pro grafy jsem využil knihovnu *MPAndroidChart* [15] dovolující vytvoření velkého množství typů grafů. Já jsem použil jen graf koláčový, protože ostatní grafy by v této aplikaci nedávaly smysl. Všechny grafy v aplikaci jsou procentuální a knihovna si procenta vypočítá sama, aplikace počítá pouze

počet položek, nikoliv jejich procentuální množství. Pokud se v grafu zobrazují jednotlivé léky (první a druhý graf), barva výseče souhlasí s barvou léku.

Graf 1 Prvním grafem je graf s názvem „všechny připomínky“. Graf zobrazuje procentuální poměr připomínek jednotlivých léků. Data se získávají následujícím způsobem:

```
val allEntries = ArrayList<PieEntry>()
val pillsHistory = history.groupBy { it.pillId }.values
pillsHistory.forEach { pillHistory ->
    val pill = getPill(pillHistory.first().pillId)
    allEntries.add(PieEntry(pillHistory.size, pill.name))
}
```

Tento kód vypočítá délku každé skupinu historie indikovanou podle ID léku a přidá ji do seznamu používaném pro vykreslení grafu.

Graf 2 Druhý graf na obrazovce pod názvem „vynechané připomínky“ zobrazuje procentuální poměr vynechaných připomínek pro jednotlivé léky. Pokud neexistují žádné vynechané připomínky, graf se nezobrazí. Data pro tento graf se získávají následovně (kód upraven pro snazší orientaci):

```
val missedEntries = ArrayList<PieEntry>()
val pillsHistory = history.groupBy { it.pillId }.values
pillsHistory.forEach { pillHistory ->
    val pill = getPill(pillHistory.first().pillId)
    val pillMissed = pillHistory.count { !it.hasBeenConfirmed }
    if (pillMissed > 0) {
        missedEntries.add(PieEntry(pillMissed, pill.name))
    }
}
```

Tento kód vypočítá počet nepotvrzených připomínek každé skupinu historie indikovanou podle ID léku a pokud je větší než nula, přidá jej do seznamu používaném pro vykreslení grafu.

Graf 3 Poslední graf s názvem „potvrzené x vynechané“ zobrazuje procentuální poměr potvrzených a vynechaných připomínek. Data pro poslední graf se získávají tímto způsobem (kód upraven pro snazší orientaci):

```
val totalConfirmed = history.count { it.hasBeenConfirmed }
val totalMissed = history.size - totalConfirmed
val confirmedEntries: ArrayList<PieEntry> = arrayListOf(
        PieEntry(totalConfirmed, R.string.confirmed),
        PieEntry(totalMissed, R.string.missed)
)
```

Výpočet pro tento graf je nejjednodušší, pouze se vypočítá počet potvrzených a počet vynechaných (od velikosti veškeré historie odečtené potvrzené) a tyto hodnoty se přidají do seznamu používaném pro vykreslení grafu.

I když jsem zde v textu kód rozdělil do tří sekcí, v aplikaci je kód na jednom místě (`HistoryChartViewModel`) a využívá pouze jeden cyklus, je tak efektivnější a méně se dotazuje databáze. I když by veškeré získávání dat pro grafy bylo možné přepsat do databázové vrstvy do jazyka SQL, já jsem zvolil tento způsob zpracování dat.

2.1.4 Nastavení

Poslední hlavní sekcí aplikace je sekce „nastavení“. Tato sekce je implementována pomocí dvou fragmentů: `SettingsFragment` a `PreferencesFragment`. První z těchto fragmentů je rodičem toho druhého. Jediné co obsahuje je titulek „Nastavení“ a pod ním prvek `FragmentContainerView` do kterého se vkládá právě druhý z fragmentů (`PreferencesFragment`). Tento druhý fragment nedědí ze třídy `Fragment` jako všechny ostatní fragmenty v této aplikaci, nýbrž ze třídy `PreferenceFragmentCompat`. Díky této třídě nemusíme vymýšlet vlastní layout pro nastavení, pouze definujeme položky v nastavení v souboru `preferences.xml` (název souboru může být jakýkoliv) a tento soubor použijeme při volání metody `setPreferencesFromResource()`, o zbytek se postará třída. Dále můžeme jednotlivé položky nastavení dynamicky manipulovat, nebo můžeme nastavit akce po jejich stisknutí. Já jsem se postaral o následující nastavení položek:

- „Nastavení možností oznámení“ se skryje, pokud aplikace běží na zařízení se systémem starší než Android 8.0 Oreo.
- Po kliknutí na „Nastavení možností oznámení“ se otevře android nastavení s obrazovkou pro správu oznámení pro tuto aplikaci.
- Po kliknutí na „O této aplikaci“ se otevře `AboutActivity`.

- Po kliknutí na „Přidat testovací data“ se do aplikace přidají léky a historie určené na testování a demonstraci aplikace.
- Po vybrání nového vzhledu aplikace se aplikace do tohoto vzhledu přepne.

Aby se fragment (`PreferencesFragment`) správně zobrazil ve fragmentu s titulkem (`SettingsFragment`) bylo nutné na jeho interním prvku[1] `listView` změnit určitá nastavení; jmenovitě vypnutí zobrazení `OVER_SCROLL`[2], vypnutí nastavení `clipToPadding` a přidání dolní mezery 56dp. Tento fragment vlastní i svůj *ViewModel*, v něm je avšak implementováno pouze přidávání testovacích dat sloužících pro demonstraci aplikace.

2.1.4.1 O aplikaci

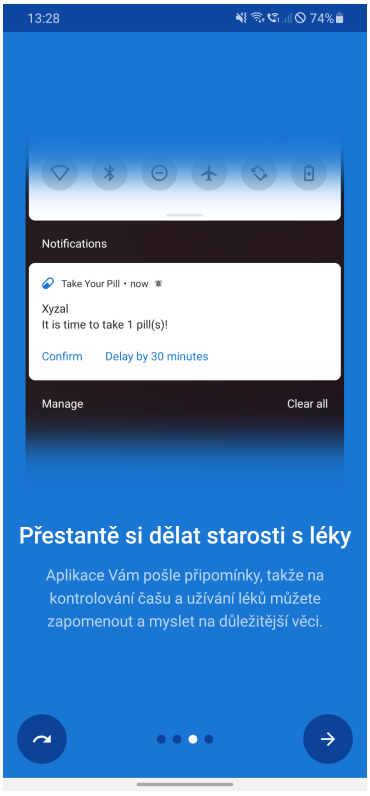
2.2 Programová implementace

2.2.1 Databáze

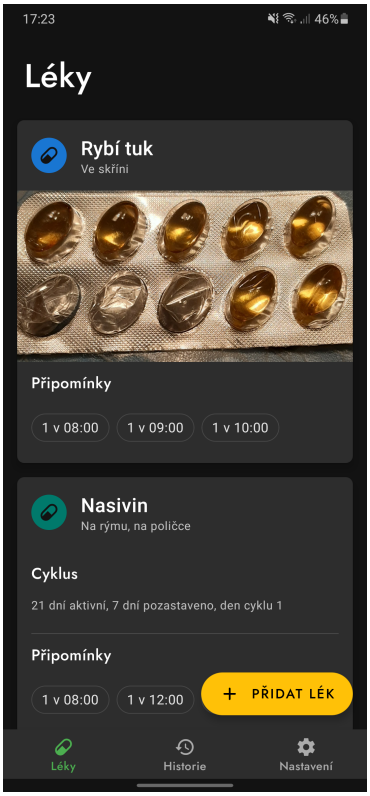
2.2.2 Připomínání

[1] tento prvek nedefinuji já, ale třída `PreferenceFragmentCompat`
[2] zobrazení zpětné vazby pokud uživatel seznam přesune na jeho začátek/konec

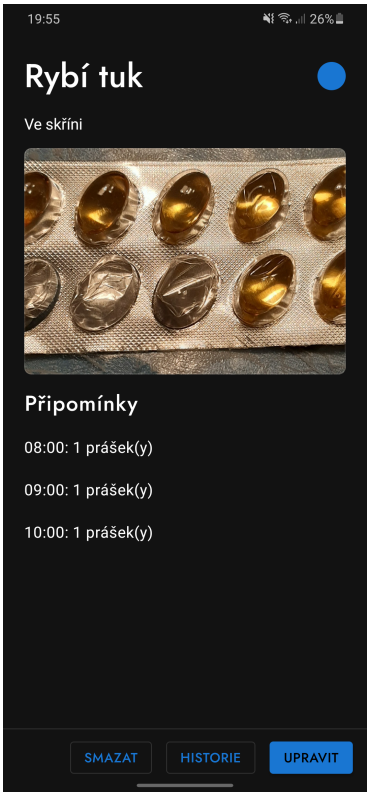
Závěr



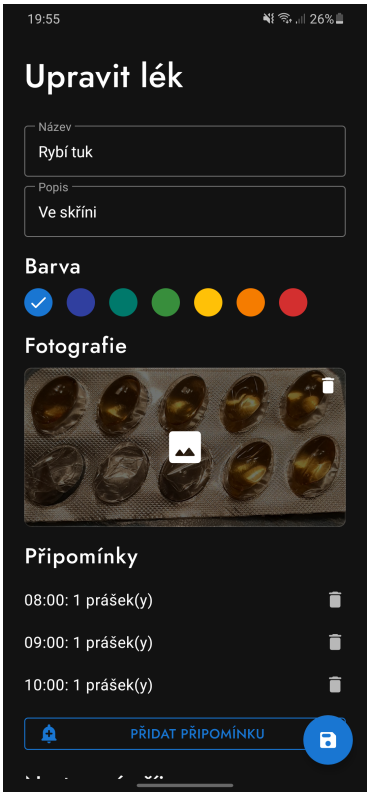
(a) Úvodní obrazovka



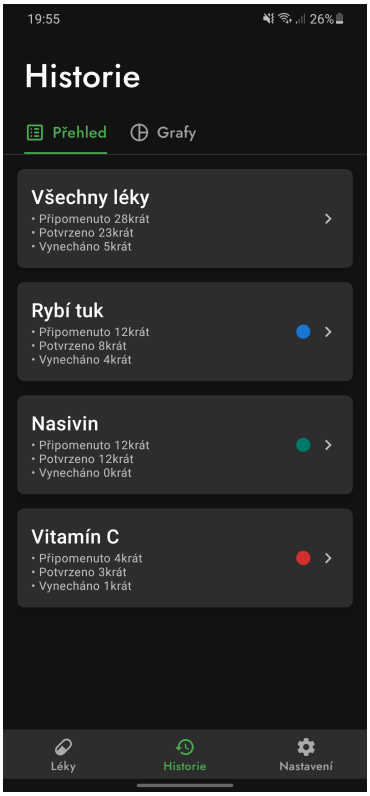
(b) Domovská obrazovka



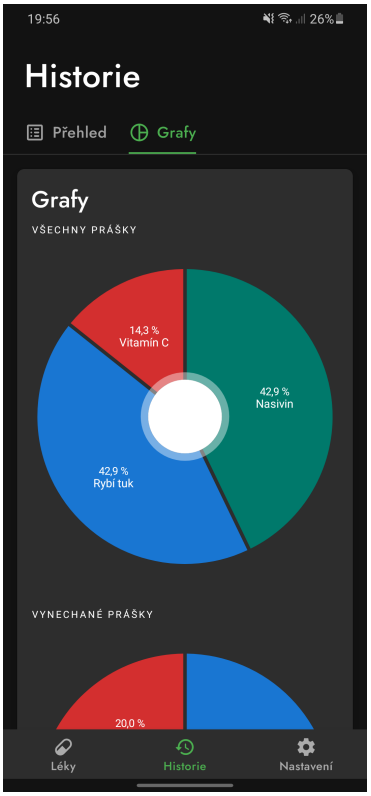
(c) Detail léku



(d) Úprava léku



(e) Přehled historie



(f) Grafy

Obrázek 2.1: Snímky obrazovky z aplikace

Bibliografie

[1] Austin Andrews. *pill*. URL: https://materialdesignicons.com/icon/pill. [cit. 13.03.2021].

[2] Android Developer. *Android Jetpack*. URL: https://developer.android.com/jetpack. [cit. 13.03.2021].

[3] Android Developer. *App resources overview*. URL: https://developer.android.com/guide/topics/resources/providing-resources. [cit. 13.03.2021].

[4] Android Developer. *Automated dependency injection*. URL: https://developer.android.com/training/dependency-injection#automated-di. [cit. 14.03.2021].

[5] Android Developer. *Data Binding Library*. URL: https://developer.android.com/topic/libraries/data-binding. [cit. 13.03.2021].

[6] Android Developer. *Dependency injection with Hilt*. URL: https://developer.android.com/training/dependency-injection/hilt-android. [cit. 14.03.2021].

[7] Android Developer. *LiveData Overview*. URL: https://developer.android.com/topic/libraries/architecture/livedata. [cit. 13.03.2021].

[8] Android Developer. *Navigation*. URL: https://developer.android.com/guide/navigation. [cit. 13.03.2021].

[9] Android Developer. *View Binding*. URL: https://developer.android.com/topic/libraries/view-binding. [cit. 13.03.2021].

[10] Google Fonts. *Jost*. URL: https://fonts.google.com/specimen/Jost. [cit. 13.03.2021].

[11] Google. *Android Studio*. URL: https://developer.android.com/studio. [cit. 13.03.2021].

[12] Google. *Material Design*. URL: https://material.io/design. [cit. 13.03.2021].

[13] Google. *Material Design - Icons*. URL: https://material.io/resources/icons/. [cit. 13.03.2021].

[14] googlesamples. *EasyPermissions*. URL: https://github.com/googlesamples/easypermissions. [cit. 13.03.2021].

[15] Philipp Jahoda. *MPAndroidChart*. URL: https://github.com/PhilJay/MPAndroidChart. [cit. 14.03.2021].

[16] Dhaval Patel. *Image Picker Library for Android*. URL: https://github.com/Dhaval2404/ImagePicker. [cit. 13.03.2021].

[17] Jan Heinrich Reimer. *material-intro*. URL: https://github.com/heinrichreimer/material-intro. [cit. 13.03.2021].

[18] ShimonHoranek. *material-icons*. URL: https://github.com/ShimonHoranek/material-icons. [cit. 13.03.2021].

[19] Gabor Varadi. *FragmentViewBindingDelegate-KT*. URL: https://github.com/Zhuinden/fragmentviewbindingdelegate-kt. [cit. 14.03.2021].

[20] Gabor Varadi. *Simple one-liner ViewBinding in Fragments and Activities with Kotlin*. URL: https://zhuinden.medium.com/simple-one-liner-viewbinding-in-fragments-and-activities-with-kotlin-961430c6c07c. [cit. 14.03.2021].

[21] Yalantis. *uCrop - Image Cropping Library for Android*. URL: https://github.com/Yalantis/uCrop. [cit. 13.03.2021].