Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Aplikace Android - teorie a praxe

Tomáš Nekolný

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Tomáš Nekolný

Informatika

Název práce

Aplikace Android – teorie a praxe

Název anglicky

Android Applications – theory and practice

Cíle práce

Cílem bakalářské práce je charakterizovat problematiku vývoje aplikací pro operační systém Android. Ten se skládá z následujících dílčích cílů:
- charakteristika architektury platformy Android
- zmapování technologií, které lze použít k vývoji aplikací pro Android
- návrh metodiky, pomocí které bude vytvořena aplikace v programovacím jazyce Java.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů, a také na praktických zkušenostech se systémem Android. Pomocí této metodiky bude vytvořena jednoduchá aplikace. Na základě syntézy teoretických a praktických poznatků budou formulovány doporučení a závěr bakalářské práce.

Doporučený rozsah práce

30 – 40 stran

Klíčová slova

Aplikace Android, architektura platformy Android, programovací jazyk Java

---

Doporučené zdroje informací

DIMARZIO, J. F. Programujeme hry pro Android 4. Brno:Computer press a.s. . 2012, 310str. . ISBN: 978-80-251-3754-3.

GRANT, A. Android 4. Brno: Computer Press a.s. 2013. 656 str. .ISBN: 978-80-251-3782-6.

HERODEK, M. Android jednoduše. Brno: Computer press a.s. 2013. 128str. ISBN: 978-80-251-4118-2.

LACKO, Ľ. Mistrovství – Android, Computer Press, a.s. 2017. 648 str. ISBN: 9788025148754

LACKO, Ľ. Vývoj aplikací pro Android, Computer Press, a.s. 2015. 472 str. ISBN: 978-80-251-4347-6

MEIER, R. Professional Android Application Development. 2. vydání. Indianapolis: Wrox. 2010. 576 str. ISBN 0470565527.

MURPHY, M. L. Android 2, Průvodce programováním mobilních aplikací. Brno: Computer Press, a.s.. 2011. 369 str. ISBN 978-80-251-3194-7.

VÁVRŮ J., UJBÁNYANI M. Programujeme pro Android. Praha: Grada a.s. 2013. 256 str. ISBN 978-80-247-4863-4.

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Eva Kánská

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 31. 10. 2017

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2017

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 04. 02. 2018

Čestné prohlášení

Prohlašuji, že jsem bakalářskou práci Aplikace Android – teorie a praxe vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15. března 2018 _____

Poděkování

Děkuji vedoucí bakalářské práce Ing. Evě Kánské za vedení této práce a své rodině za podporu během jejího vypracování

Aplikace Android – teorie a praxe

Abstrakt

Bakalářská práce se zaměřuje na vývoj aplikací pro operační systém Android. Teoretická část se zabývá architekturou této platformy, dostupnými nástroji pro vývoj a vývojem aplikace za pomoci oficiálních nástrojů. V rámci praktické části byla vytvořena aplikace s názvem Habit tracker, jejíž cílem je pomoci uživateli s vytvářením požadovaných návyků. Vývojem této aplikace bylo demonstrováno použití nástrojů a technologií popsaných v teoretické části.

Klíčová slova: Aplikace Android, architektura platformy Android, programovací jazyk Java

Android Applications – theory and practice

Abstract

The bachelor thesis is focused on development of Android applications. Theoretical part describes architecture of this platfom, available development tools and development of appliaction with official tools. Within the practical part of the thesis, application called Habit tracker was developed. Purpose of the application is to help user to create desired habits. Usage of tools and technologies described in theoretical part was demonstrated by development of this appliaction.

Keywords: Android application, Architecture of Android platform, Java programming language

Obsah

1 Úvod

Odvětví mobilních telefonů se rychle vyvíjí a jednoduché telefony jsou nahrazovány tzv. chytrými telefony (smartphony), které disponují pokročilým operačním systémem. Ve Spojených státech Amerických se podíl smartphonů, z celkového počtu mobilních telefonů, mezi lety 2005 a 2015 zvýšil z 2% na 79%. (1) V roce 2017 v České republice pak smartphone použije alespoň jednou za měsíc téměř 65% obyvatelstva. (2) Celosvětově nejrozšířenějším operačním systémem pro mobilní telefony je Android. Ten v roce 2017 dosáhl počtu 2 miliard aktivních zařízení. (3)

Tato práce se zaměřuje na vývoj aplikací pro operační systém Android. Aplikace, softwarové programy pro mobilní zařízení, umožňují rozličná využití telefonu. Každý uživatel si tak své zařízení může, instalací různých aplikací, uzpůsobit podle svých potřeb. Koncem roku 2017 bylo v nabídce Google Play, což je distribuční služba, 3 a půl milionu aplikací. (4)

K porozumění této problematice je třeba seznámit se s operačním systémem jako takovým, jeho architekturou, technologiemi a nástroji potřebnými k vývoji. Tímto se zabývá teoretická část bakalářské práce. Praktická část se věnuje vytvoření nativní aplikace pro Android s použitím programovacího jazyka Java a vývojového prostředí Android Studio.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem bakalářské práce je charakterizovat problematiku vývoje aplikací pro operační systém Android. Ten se skládá z následujících dílčích cílů:

- charakteristika architektury platformy Android
- zmapování technologií, které lze použít k vývoji aplikací pro Android
- návrh metodiky, pomocí které bude vytvořena aplikace v programovacím jazyce Java.

2.2 Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů, a také na praktických zkušenostech se systémem Android. Pro vypracování práce je nutné seznámit se s architekturou systému, vývojovými nástroji a především pak s architekturou aplikací pro tento systém. Jako odborné zdroje budou použity publikace zaměřené na vývoj aplikací a oficiální dokumentace systému.

Na základě teoretických poznatků bude vytvořena jednoduchá aplikace. Při jejím vývoji tak budou získány praktické zkušenosti s vývojem aplikací na této platformě. Na základě syntézy teoretických a praktických poznatků budou formulovány doporučení a závěr bakalářské práce.

3 Teoretická východiska

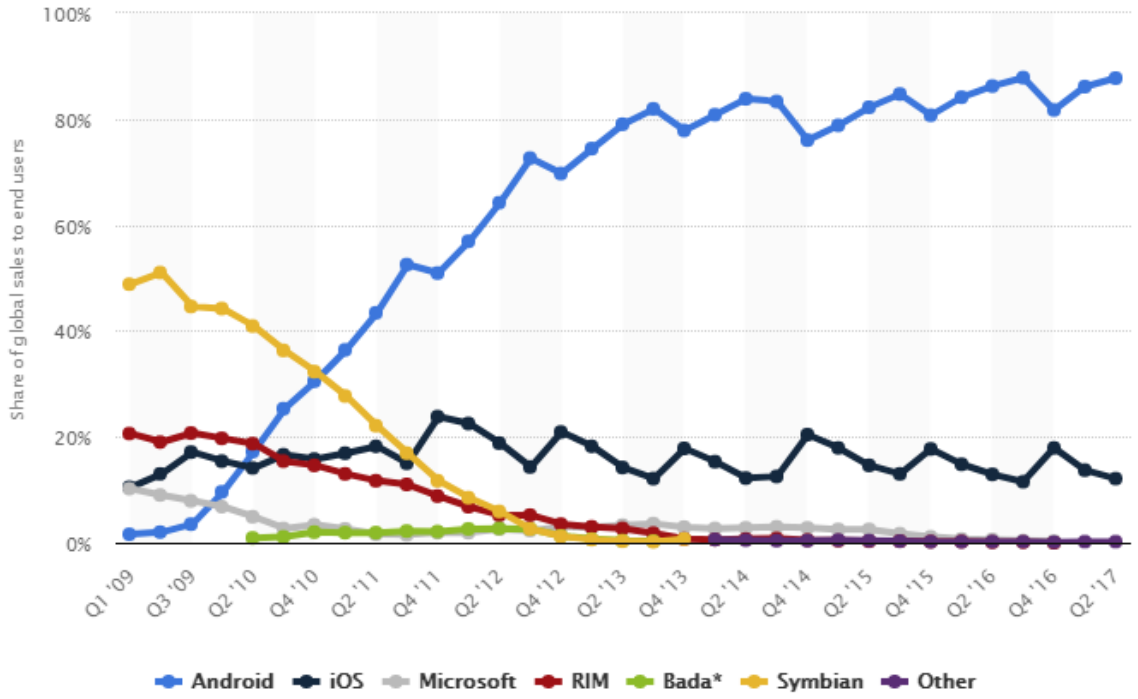
3.1 Operační systém Android

Android je operační systém (dále jen OS) pro mobilní zařízení vyvíjený společností Google a jedná se o nejrozšířenější mobilní OS na světě. (5) (6)

3.1.1 Historie

Společnost Android, Inc. byla založena v říjnu roku 2003. Původní záměrem firmy bylo vytvoření OS pro digitální kamery. Vzhledem k velikosti tohoto trhu, však došlo ke změně zaměření na chytré telefony. (7)

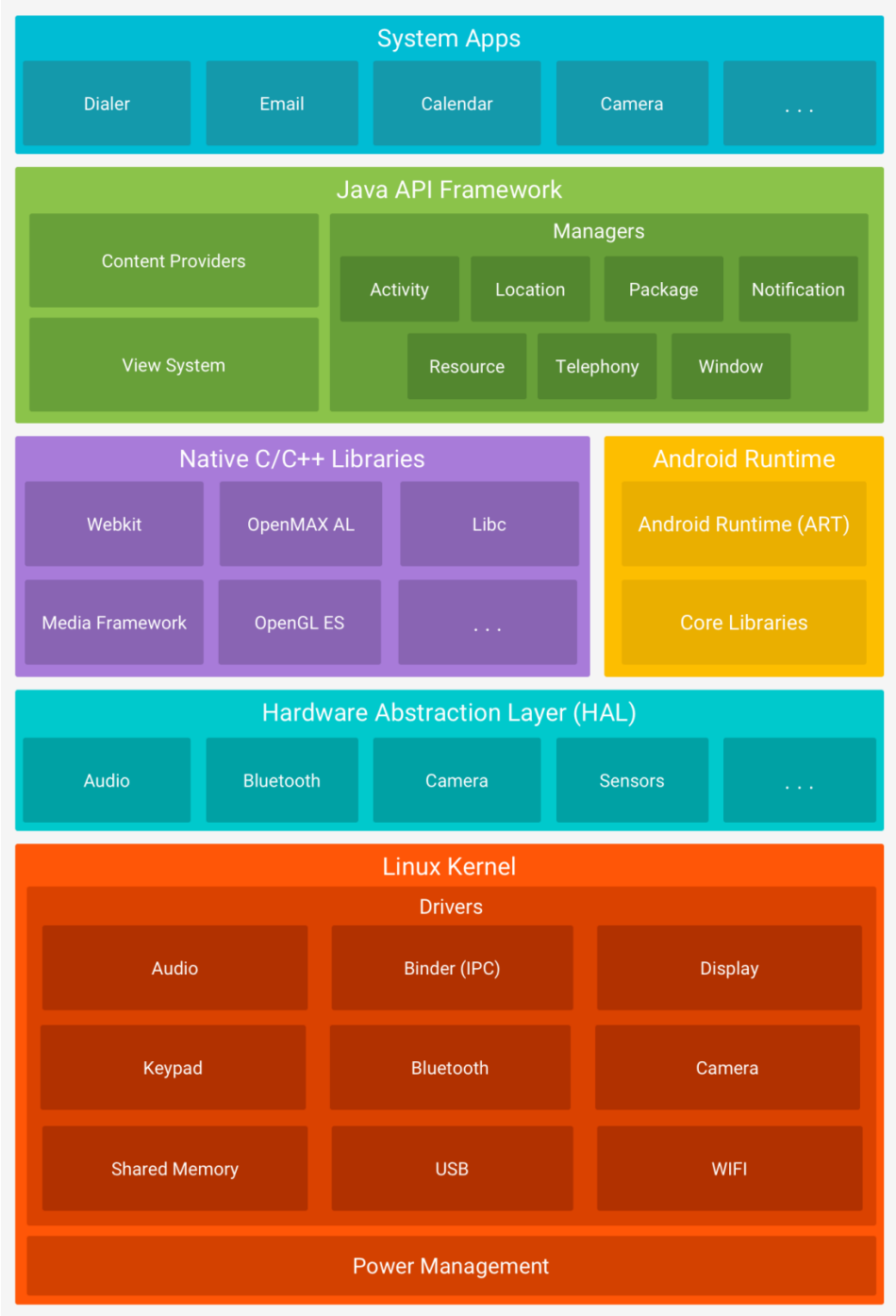
Po akvizici společností Google, Inc. v roce 2005 byl vyvinut OS založený na Linuxovém jádru. První telefon s tímto systémem, HTC Dream, (8) byl uveden na trh v říjnu 2008. Během dvou let se stal Android mobilním OS s největším podílem světového trhu. Růst zastoupení tohoto systému na světovém trhu pokračoval i v následujících letech. Ve třetím kvartálu roku 2016 mělo 88% všech prodaných mobilních zařízení OS Android. (6)



Obrázek 1 Graf zobrazující podíl jednotlivých mobilních OS na trhu, zdroj: (6)

3.1.2 Architektura platformy Android

Android je open source platforma založená na Linuxovém jádru. Skládá se ze šesti vrstev uvedených na následujícím diagramu. Ve většině dostupných zdrojů se hovoří pouze o pěti vrstvách a neuvádí se HAL, výrobce sám jej však jako samostatnou vrstvu uvádí. (9) (10) (11)



Obrázek 2: Architektura platformy Android, zdroj: (10)

Linux kernel

Linux kernel neboli Linuxové jádro tvoří nejnižší vrstvu architektury Androidu. Zajišťuje přímou interakci s hardwarem, řízení procesů, správu paměti, správu napájení či zabezpečení systému. (9) (10) (11)

Hardware Abstraction Layer

Hardware Abstraction Layer (HAL), neboli vrstva abstrakce hardwaru, poskytuje rozhraní vyšší vrstvě Java API Framework, které umožňuje přístup k hardwarovým funkcím. HAL tak činí použití hardwaru nezávislým na jednotlivých hardwarových produktech. (9) (10) (12) (13)

Native C/C++ Libraries

Vrstva knihoven psaných v C nebo C++. Ty jsou prostředníkem mezi Linuxovým jádrem a vrstvou Java API Framework, ta dále poskytuje přístup aplikacím k některým z těchto knihoven. Jedná se například o funkcionalitu související s databází SQLite, grafikou (OpenGL ES), zobrazováním webových stránek (Webkit) a další. (9) (10) (11)

Android Runtime

Android Runtime je běhové prostředí, které se skládá ze dvou částí. První částí je Core Libraries, knihovny poskytující většinu funkcionality programovacího jazyku Java, včetně některých novinek z Java 8. Druhou částí je pak samotný Android Runtime (ART), kterým je od verze Android 5.0 nahrazen Dalvik Virtual Machine (DVM). Hlavím rozdílem mezi DVM a ART je přístup ke kompilaci. DVM využívá metodu Just-In-Time, ke kompilaci dochází při spuštění aplikace, ART umožňuje kompilovat již při instalaci (Ahead-Of-Time). Jde tedy o zrychlení spouštění aplikace a snížení spotřeby energie za cenu většího využití úložiště. Každá spuštěná aplikace má vlastní instanci ART. (9) (10) (14)

Java API Framework

Knihovny psané v programovacím jazyce Java. Tuto vrstvu využívají systémové aplikace pro přístup k základním službám systému. Jsou určené také pro vývojáře, kteří je využijí pro tvorbu vlastních aplikací. Jde například o View System, který spravuje prvky grafického uživatelského rozhraní, Activity Manager, jenž má na starost životní cyklus aplikace, nebo o správce notifikací. (9) (10) (11)

System Apps

Poslední vrstvou jsou systémové aplikace. Jde například o program pro zasílání a příjem SMS, program umožňující telefonování, kalendář, navigaci, klávesnici atd. (10)

3.2 Vývojové nástroje

K vytvoření aplikace pro Android je nutné využít některé z vývojových nástrojů, ať už se jedná o nástroje vytvořené, či oficiálně podporované přímo Googlem nebo nástroje třetích stran. Použití oficiálních vývojových nástrojů se v teoretické rovině věnuje následující podkapitola a v praktické kapitola 4.

3.2.1 Oficiální vývojové nástroje

Android SDK

Android SDK (Software Development Kit) je sada vývojářských nástrojů k vytváření aplikací pro Android. Obsahuje potřebné knihovny, debugger, emulátor, dokumentaci. S každou novou verzí Androidu je vydána také nová verze SDK, aby bylo možné v aplikacích využívat nejnovějších vlastností systému. Vyvíjet aplikace pro Android lze za pomocí libovolného textového editoru, JDK (Java Development Kit) a Android SDK. (15)

Android Debug Bridge

Android Debug Bridge, zkráceně ADB, je součástí Android SDK. Jde o nástroj příkazového řádku, umožňující komunikaci počítače se zařízením běžícím na systému Android. ADB se skládá ze tří částí. Client, spuštěný na počítači, kterým lze posílat příkazy. Daemon, proces na pozadí, který zpracovává příkazy na zařízení s Androidem. Server, který zajišťuje komunikaci mezi clientem a daemonem. Jde o proces spuštěný na počítači, na kterém je client. Aby bylo možné ADB použít, je nutné v nastavení Androidu povolit USB debugging. Následně je možné za pomoci tohoto nástroje do zařízení instalovat aplikace, pracovat s logem, kopírovat soubory, vytvářet zálohu zařízení a poté ji použít k obnově, nebo také spustit na Androidu shell, pomocí, kterého lze spouštět aktivity, ukončovat procesy, vysílat záměry, pořizovat snímky obrazovky a mnoho dalšího. (16)

IDE

IDE neboli integrované vývojové prostředí je program vývoj softwaru, který usnadňuje psaní kódu a umožňuje program ladit. Oficiálním IDE pro Android bylo Eclipse, nástroj třetí strany, s pluginem Android Development Tools. V prosinci roku 2014

však byla vydána první stabilní verze IDE Android Studio, které je vyvíjené přímo společností Google. (17)

Programovací jazyky

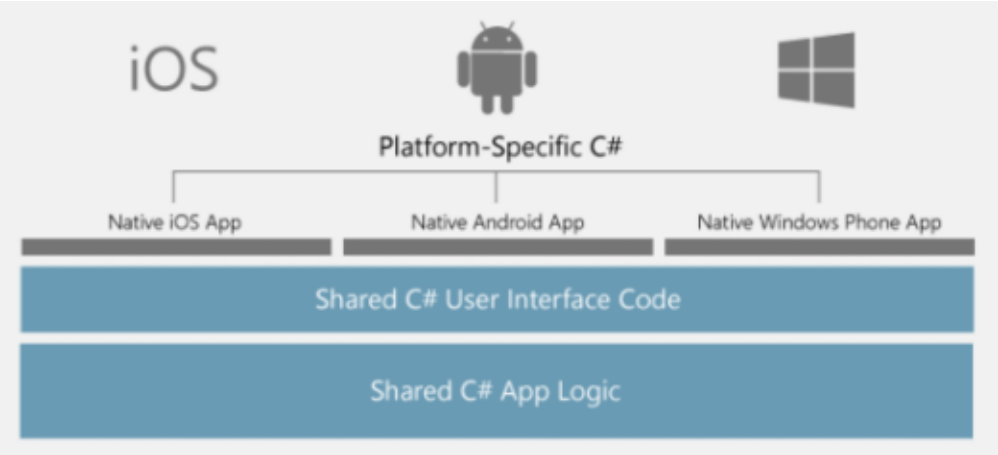
Oficiálním programovacím jazykem pro Android je od počátku Java. Od května 2017 se k ní připojil jazyk Kotlin, který je s Javou plně kompatibilní a aplikace tak mohou být z části psané v Javě a z části v Kotlinu. Mezi výhody Kotlinu oproti Javě patří například možnost použití lambda výrazů nebo nenullovatelné typy. (9) (10) (18) (19)

Android NDK

Android Native Development Kit umožňuje použít pro tvorbu aplikací pro Android kód, psaný v programovacím jazyce C nebo C++. Této možnosti by nemělo být využito z důvodu, že vývojář preferuje tento jazyk, ale měla by jít buď o znovupoužití existující knihovny, psané v tomto jazyce, nebo za účelem lepšího výkonu aplikace. (20)

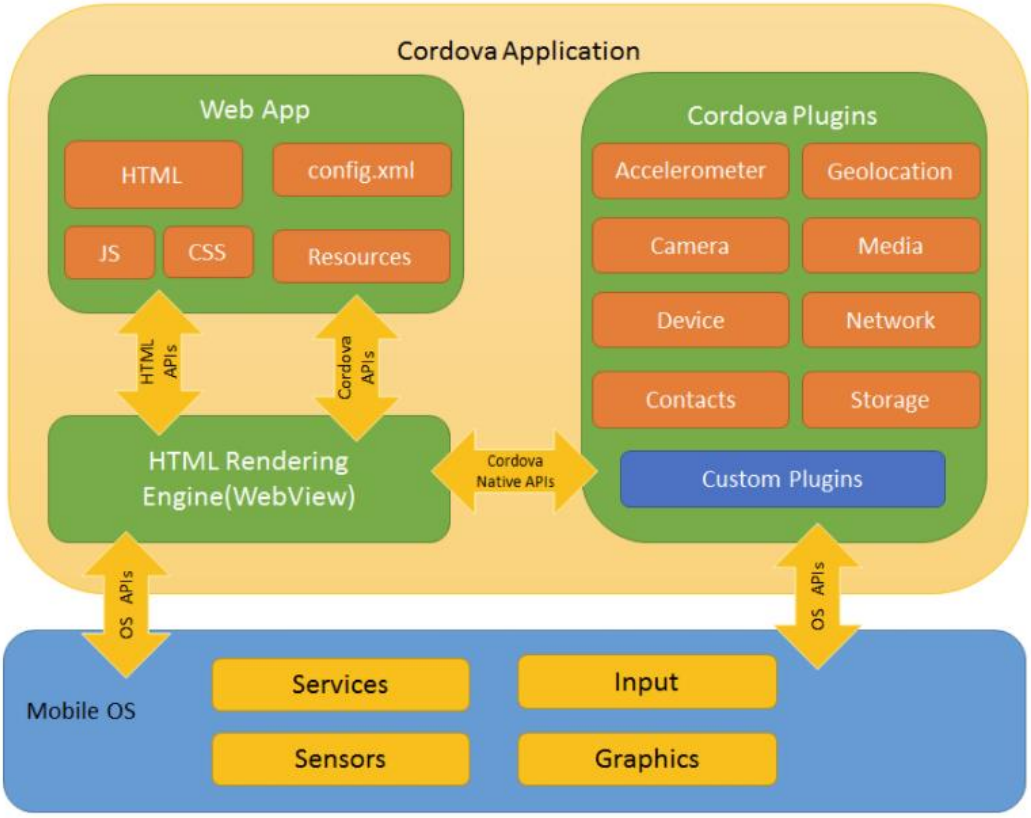
3.2.2 Alternativní vývojové nástroje

Pro vývoj Android aplikace lze použít také celou řadu jiných IDE než je Android Studio a jiných programovacích jazyků než Java či Kotlin. Pro vývoj v Javě lze použít například vývojová prostředí Eclipse či NetBeans. Microsoft ve svém Visual Studiu nabízí možnost vyvíjet v programovacím jazyce C# za pomoci platformy Xamarin. Ta umožňuje multiplatformní vývoj pro mobilní OS, aplikace pro Android, iOS a Windows Phone tak mohou sdílet více než 75% kódu. Ačkoli tomu tak dříve nebylo, Microsoft technologii Xamarin vlastní. Visual studio nabízí také Tools for Apache Cordova.



Obrázek 3: Architektura aplikace za využití Xamarin, zdroj: (19)

Apache Cordova je open-source Framework pro vývoj mobilních aplikací s použitím webových technologií HTML, CSS a JavaScript. Za pomoci Cordovy lze vytvářet multiplatformní aplikace.



Obrázek 4: Architektura aplikace vytvořené s Apache Cordova, zdroj: (20)

Za zmínku stojí také AIDE – IDE for Android, které umožňuje aplikace programovat přímo v telefonu či tabletu s Androidem. (21) (22) (23) (24) (25) (26)

3.3 Vývoj aplikace za použití jazyka Java a IDE Android Studio

3.3.1 Projekt

Po založení nového projektu, Android Studio vytvoří strukturu souborů potřebných pro vývoj aplikace.

Moduly

Aplikace se může skládat z jednoho či více modulů, které mohou být nezávisle sestaveny a testovány. Při vytvoření nového modulu lze zvolit z následujících možností: Android app module, obsahující zdrojový kód, soubory pro aplikaci a její manifest. Ten po sestavení vytváří soubor Android Package Kit (APK), instalační soubor aplikace. Library module obsahuje znovupoužitelný kód, který lze importovat do ostatních modulů či jiných aplikací. Strukturou je stejný jako app module, po sestavení však nevytváří soubor APK, ale Android Archive (AAR) či Java Archive (JAR). AAR může obsahovat všechny typy souborů, které se v Android projektu mohou nacházet, zatímco v JAR mohou být pouze zdrojové kódy v jazyce Java. Poslední variantou je Google Cloud module, který je určený pro vývoj backendu na Google Cloud. (27)

Soubory v projektu

Struktura Android projektu je následující:

- Název-modulu/
    - `build/` - soubory vytvořené při sestavení
    - `libs/` - soukromé knihovny modulu
    - `src/` - obsahuje veškeré zdrojové soubory
        - `androidTest/` - zdrojový kód pro testy, využivající uživatelské rozhraní aplikace
        - `main/` - zdrojové soubory sdílené pro všechny buildy aplikace. Mezi nimi AndroidManifest.xml.
            - `java/` - zdrojový kód aplikace v jazyce Java
            - `jni/` - nativní kód (NDK)
            - `gen/` - Java soubory generované Android studiem
            - `res/` - XML soubory s definicí uživatelského rozhraní, obrázky využité v aplikaci, definice textových řetězců

        - `assets/` - Soubory, které by měli být zkompilovány do výsledného APK v nezměněné formě.
    - `test/` - unit testy
  - `build.gradle` – konfigurační soubor buildu, obsahující specifika pro daný modul
- `build.gradle` – build konfigurace společná pro všechny moduly

Zdroje: (9) (27)

3.3.2 Architektura aplikace

Základními komponentami, které tvoří aplikaci, jsou:

- Aktivity (`Activity`)
- Služby (`Service`)
- Broadcast Receivers (`BroadcastReceiver`)
- Záměry (`Intent`)
- Content providers (`ContentProvider`)

Zdroje: (9) (11)

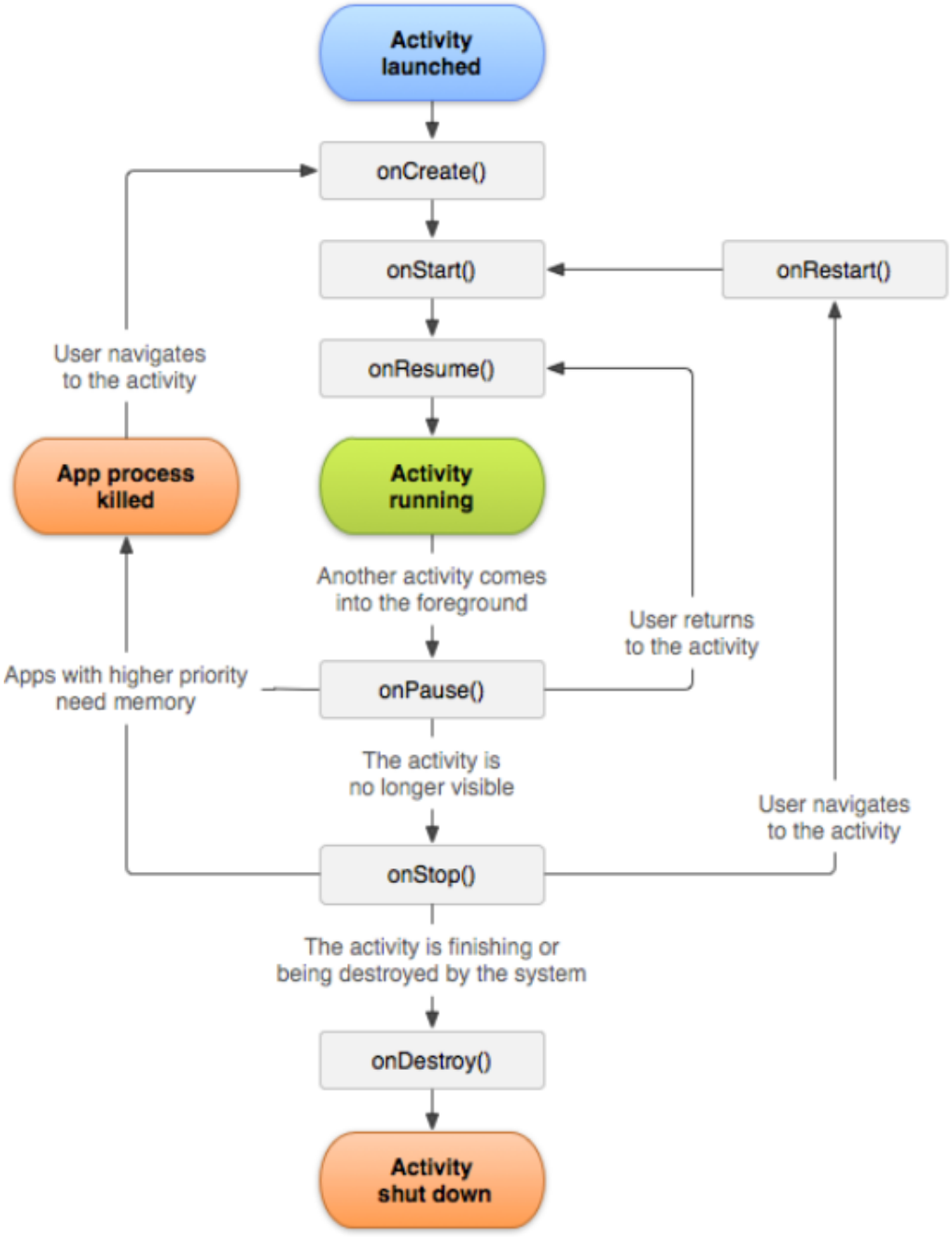
Activity

`Activity` neboli aktivita umožňuje uživateli vykonat úlohu, jako například vyplnění formuláře nebo odeslání zprávy. Je to třída, která zobrazí grafické uživatelské rozhraní a zachytává interakce uživatele přes toto rozhraní. Aplikace se může skládat z jedné nebo více aktivit, které si mezi sebou mohou předávat informace.

Jedna z aktivit je vždy definována, jako hlavní, ta je spuštěna po zapnutí aplikace. Při spuštění nové aktivity, je předchozí aktivita pozastavena, ale zůstává uložena v zásobníku (Back stack) pod nově spuštěnou aktivitou, která se dostává do popředí. Aktivita bývá nejčastěji zobrazena přes celou obrazovku, může být ale také ve formě okna nebo vnořena do jiné aktivity. Android rozhoduje o tom, kdy bude instance aktivity vytvořena, odsunuta do pozadí, či zničena. Je třeba počítat s tím, že k odstranění instance může dojít kdykoliv. Například pokud začnou docházet systémové zdroje. (9) (10) (14)

Životní cyklus aktivity

Při běhu aplikace se instance aktivit dostávají do různých fází svého životního cyklu. Ten je definovaný metodami, které se spouští v definovaný moment. Jedná se o metody `onCreate`, `onStart`, `onResume`, `onPause`, `onStop` a `onDestroy`. Životní cyklus aktivity je znázorněn na následujícím diagramu. (9) (10)



Obrázek 5: Diagram zobrazující životní cyklus aktivity, zdroj: [6]

Fáze životního cyklu aktivity

Aktivita na popředí

Aktivita na popředí je zobrazena na displeji a má fokus, to znamená, že může interagovat s uživatelem. Jde o aktivity ve stavu Running nebo Resumed. Aktivita na popředí je vždy první v Back stacku. Android se takovou aktivitu snaží udržet v běhu i za cenu odstranění instancí jiných aktivit. (9) (11)

Pozastavená aktivita

Pozastavená aktivita je částečně viditelná (např. překrytá dialogovým oknem), ztrácí fokus, nereaguje tedy na vstupy od uživatele. Taková aktivita může být systémem odstraněna pouze v případě akutního nedostatku paměti pro aktivitu na popředí. (9) (11)

Zastavená aktivita

Zastavená aktivita není viditelná, zůstává uložena v Back stacku, ale stává se kandidátem na odstranění v případě potřeby uvolnit paměť. Při zastavení aktivity je důležité uložit data a současný stav uživatelského rozhraní. (11) (13)

Metody definující přechody mezi jednotlivými stavy:

`onCreate` - Metoda se aktivuje při spuštění aktivity, v jejím těle se vytváří proměnné potřebné k běhu aktivity, vytváří se uživatelské rozhraní, které však zatím není vidět. Po dokončení této metody systém zavolá metodu `onStart` a ihned poté `onResume`.

`onStart` a `onResume` - Zde se realizuje přechod aktivity z pozadí do popředí, zobrazuje se grafické uživatelské rozhraní. Metoda `onStart` je volána při předchozím zatavení aktivity, `onResume` následuje po `onStart` nebo je volána při přesunu aktivity z pozadí do popředí.

`onPause` – Používá se při přechodu aktivity na pozadí, doporučuje se uložit změny údajů, se kterými aktivita pracovala.

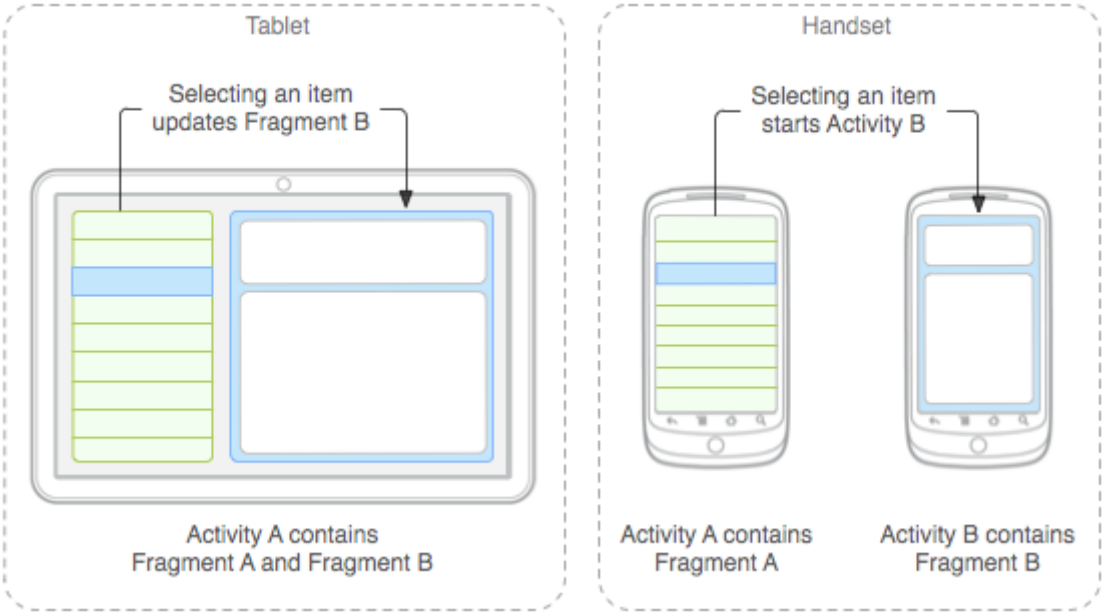
`onStop` - Tato metoda se používá při zastavení aktivity z jiného důvodu, než nedostatku paměti. Aktivita totiž zůstává back stacku a může být opět zobrazena na popředí.

`onDestroy` - Metoda volána při ukončení životního cyklu aktivity, ať už jde o záměrný konec pomocí metody `finish` či je aktivita ukončena systémem z důvodu uvolnění paměti.

Zdroje: (9) (10) (28)

Fragmenty

Fragmenty byly přidány do Androidu ve verzi 3.0 v reakci na příchod tabletů. Představují řešení, jak vytvářet uživatelská rozhraní vhodná jak pro malé, tak velké displeje. Fragment nabízí podobnou funkcionalitu jako aktivita, je ale vždy zobrazen v jejím rámci a životní cyklus fragmentu je tedy navázán na životní cyklus dané aktivity. V rámci jedné aktivity může být zobrazeno více fragmentů a zároveň lze jeden fragment znovupoužít ve více aktivitách. Na obrázku níže je znázorněno typické využití fragmentů, struktura typu master-detail, kde master je seznam objektů a detail obsahuje podrobnosti o vybraném objektu. Na velkém displeji jsou zobrazeny oba fragmenty v rámci jedné aktivity. V případě, že toto zobrazení velikost displeje neumožňuje, kliknutí na objekt v seznamu otevírá novou aktivitu, obsahující fragment s detailem. (10) (13)



Obrázek 6: Příklad použití fragmentů, zdroj: (10)

Services

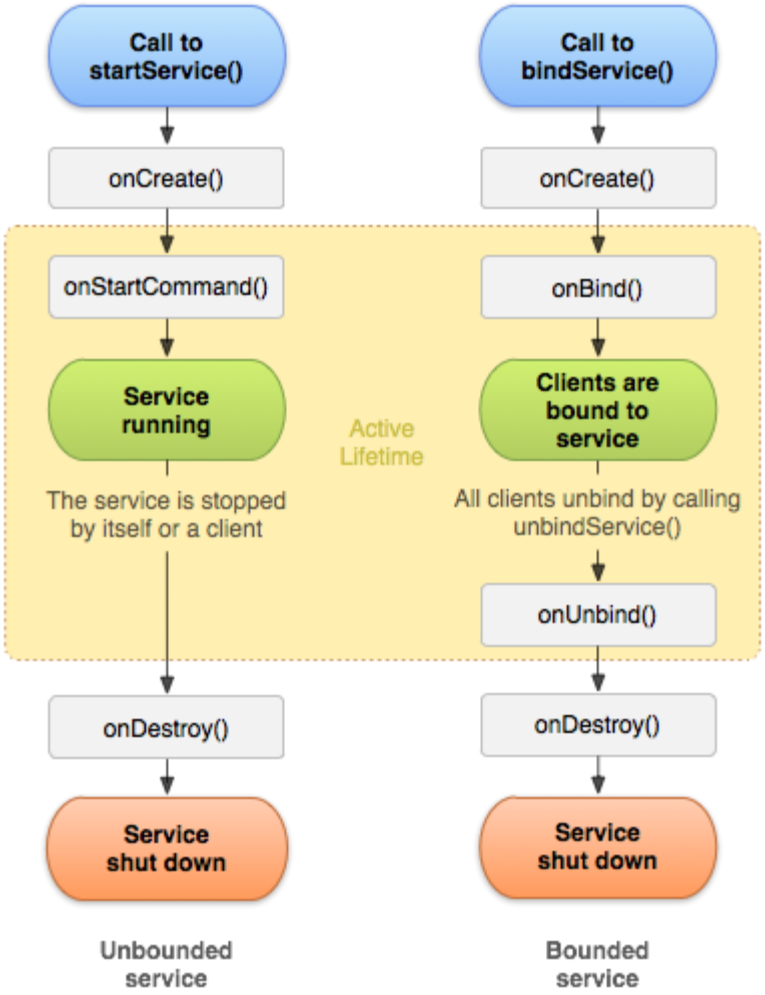
`Service` neboli služba je komponenta, která může asynchronně, paralelně s hlavním vláknem, vykonávat déle trvající operace. Aby byla služba spuštěna v novém vlákně, musí zajistit programátor, není tomu tak automaticky. Služby nedisponují uživatelským rozhraním, vykonávají operace, jako například přehrávání hudby na pozadí nebo čtení a zápis do souborů. Pokud by taková déle trvající operace nebyla řešena asynchronně, aktivita na popředí by nereagovala. Po pěti vteřinách neaktivity by Android zobrazil chybové hlášení s možností tuto aktivitu ukončit. (9) (10) (14)

Životní cyklus služby

Služba má dva základní stavy:

- **Spuštěná služba** – Služba je vytvořena ve chvíli, kdy jiná komponenta zavolá metodu `startServie` a běží až do té doby, kdy se sama ukončí metodou `stopSelf` nebo je ukončena jinou komponentou, pomocí metody `stopService`. Služba běží na pozadí i v případě, že byla aktivita, ze které byla spuštěna, zastavena.
- **Vázaná služba** – Vázaná služba je vytvořena, když jiná komponenta (klient) zavolá metodu `bindService`. Klient může se službou komunikovat prostřednictvím interfejsu `IBinder` a spojení může ukončit metodou `unbindService`. Služba běží jen tehdy, kdy je k ní nějaká komponenta připojena. Jakmile se odpojí poslední klient, služba je ukončena.

Tyto stavy nejsou zcela oddělené, klient se může vázat i k službě, která již byla spuštěna pomocí `startService`. V takovém případě metody `stopService` a `stopSelf` službu nezastaví a běží až do doby, kdy se odpojí poslední klient. (9) (10)



Obrázek 7: Diagram zobrazující životní cyklus služby, zdroj: [10]

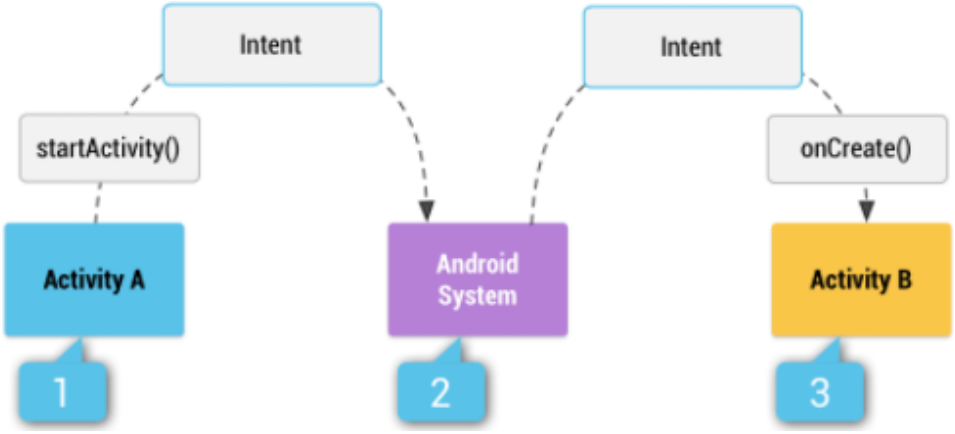
Broadcast Recievers

Broadcast recievers jsou objekty poslouchající na pozadí a reagují na události, které jsou zastoupeny objekty typu `Intent` (záměr). Broadcast receiver má registrovaný určitý záměr, jakmile takový záměr zachytí, reaguje dle definice v metodě `onReceive`. (13)

Intents a Intent Filters

Intents neboli záměry umožňují vzájemnou komunikaci mezi komponentami aplikací nebo mezi aplikací a Android OS. Záměr oznamuje, co má aktivita v úmyslu. Pokud je záměr implicitní (nedefinuje, která aktivita jej má obsloužit), systém vyhledá všechny aktivity, které jej mohou splnit. Pokud je aktivit nalezeno více, uživatel může zvolit, která záměr zpracuje. Opakem je explicitní Intent, který přesně definuje, jakou akci má kdo vykonat. Intents umožňují také přenos údajů mezi komponentami, a to za pomoci

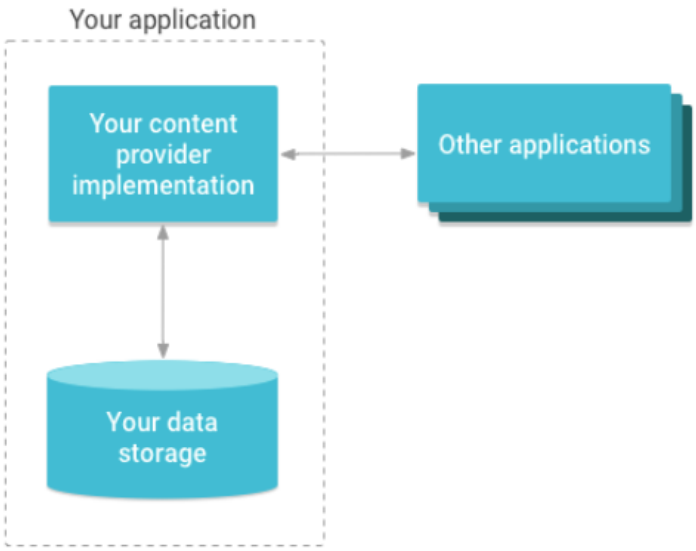
třídy `Bundle`. Intent Filter je definován v manifestu aplikace, říká, jaké druhy záměrů je schopna aplikace obsloužit. (9) (29)



Obrázek 8: Zpracování implicitního záměru, Zdroj: [10]

Content providers

V operačním systému Android jsou veškeré zdroje aplikace privátní, jiné aplikace k nim nemají přístup. Pokud aplikace potřebuje pracovat s daty jiné aplikace nebo poskytovat svá data, musí být definován content provider, poskytovatel obsahu, skrze který jsou data sdílena. (9) (10) (13)



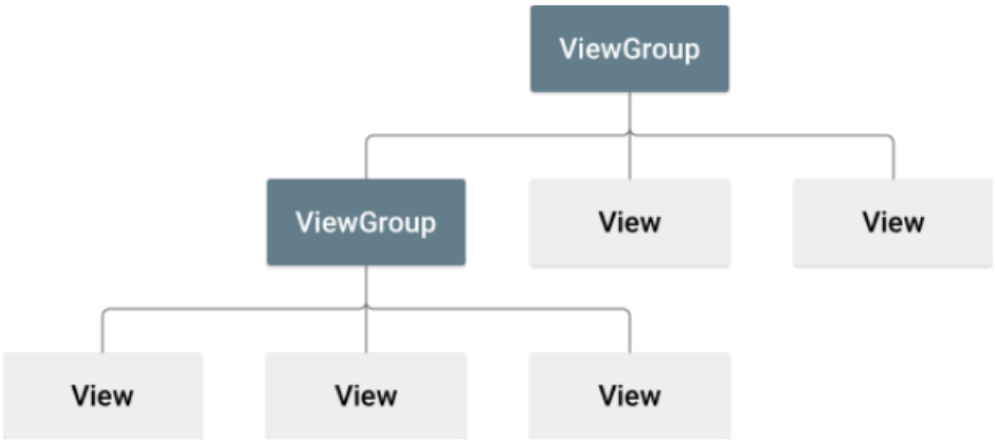
Obrázek 9: Diagram znázorňující funkci content provideru, zdroj: (10)

3.3.3 Aplikační manifest

Aplikační manifest je soubor `AndroidManifest.xml`. Tento soubor poskytuje systému Android informace o aplikaci. Jde například o název balíčku aplikace, který sklouží jako jedinečný identifikátor. Manifest popisuje jednotlivé komponenty, ze kterých se aplikace skládá: aktivity, services, broadcast receivers a content providers. V manifestu je také deklarováno jaké permissions aplikace vyžaduje nebo pro jaké verze Androidu je určena. (10) (11)

3.3.4 Grafické uživatelské rozhraní

Veškeré prvky grafického uživatelského rozhraní (GUI) jsou potomky třídy `View` nebo `ViewGroup`. `View` je objekt, který na obrazovku vykreslí prvek, se kterým může uživatel interagovat. `ViewGroup` je objekt, který může obsahovat další objekty `View` nebo `ViewGroup`.



Obrázek 10: Hierarchie prvků grafického uživatelského rozhraní, zdroj: (10)

Android nabízí dva způsoby, jak GUI vytvářet:

1. Definovat prvky GUI pomocí značkovacího jazyka XML
2. Pomocí kódu aplikace za jejího běhu

Výhodou definice pomocí XML je lepší oddělení aplikační logiky a vzhledu aplikace. Je tak možné definovat odlišná rozložení prvků pro zařízení s různou velikostí či orientaci displeje (na výšku/portrait nebo na šířku/landscape). Názvy elementů v XML odpovídají názvům tříd a názvy atributů, až na drobné výjimky, názvům metod. Definice v XML

se ukládá do adresáře `res/layout` a aby byla v aplikaci skutečně použita, je třeba ji nastavit v kódu aplikace pomocí metody `setContentView`. (9) (10) (11)

Layouts

Layouts a jsou odvozeny od třídy `ViewGroup`. Jde o kontejnery, které definují vizuální strukturu aktivit, fragmentů nebo widgetů aplikace. Každý typ Layoutu nabízí rozdílné řešení, jak rozmístění jednotlivých prvků definovat. (9) (10)

LinearLayout

`LinearLayout` uspořádává vnořené prvky do jednoho řádku či sloupce podle parametru `orientation`. Pomocí atributu `layout_weight` lze definovat, jak velký prostor bude v daném řádku/sloupci zabírat jednotlivý vnořený prvek. `layout_weight` je váha neboli priorita daného prvku, čím vyšší číslo, tím je vyšší priorita. Pokud jsou v layoutu například tři prvky, z nichž jeden má váhu 0 a zbylé dva váhu 1, prvek s nulovou váhou zabere prostor daný jeho obsahem a zbylé dva prvky se podělí o zbytek volného místa. `Layout_gravity` určuje, jak bude prvek zarovnán (do středu či k jedné ze stran). (9) (10) (14)

GridLayout

`GridLayout` je byl do Androidu přidán s API 14 (Android 4.0). Jedná se o obdobu `LinearLayout` s tím rozdílem, že lze definovat sloupce a řádky zároveň. Pomocí `layout_columnSpan` a `layout_rowSpan` lze určit přes kolik buněk tabulky bude daný prvek umístěn. (9) (10)

RelativeLayout

`RelativeLayout` umožňuje uspořádat vnořené prvky pomocí definice jejich polohy vzhledem k tomuto layoutu a definice polohy vzhledem k ostatním vnořeným prvkům. Při použití `RelativeLayout` tak odpadá potřeba používat vnořené `ViewGroup`, je však třeba dbát opatrnosti, aby mezi vazbami nevznikl rozpor. Polohu vůči rodičovskému layoutu lze určit atributy nabývající hodnoty `true` nebo `false`. Například `layout_alignParentLeft` – zarovná levý okraj prvku s levým okrajem layoutu, `alignParentBottom` – zarovná spodní okraje, `layout_centerInParent` – vycentruje prvek uprostřed layoutu. Atributům, které definují polohu k jinému vnořenému prvku, je třeba jako hodnotu nastavit `id` daného prvku. Jde například o `layout_bellow` – horní

okraj prvku zarovná k spodnímu okraji daného prvku, `layout_alignRight` – zarovná pravé okraje prvků, nebo `layout_toRightOf` – umístí levý okraj k pravému okraji prvku se zadaným `id`. (10) (13)

ConstraintLayout

`ConstraintLayout` stejně jako `RelativeLayout` umožňuje určovat pozici prvku vztahem k jinému prvku, konkrétně vtahem mezi jejich hranami. Například pro vycentrování prvku horizontálně je třeba připojit jeho pravou hranu k pravé a levou k levé hraně rodičovského prvku. Výhodou tohoto layoutu je snadná tvorba pomocí editoru Android studia. (10)

CoordinatorLayout

`CoordinatorLayout` slouží pro koordinaci vnořených prvků jako například rozbalovací aplikační lišty a zbytku uživatelského rozhraní. (13)

FrameLayout

`FrameLayout` je určený k zobrazení jednoho vnořeného prvku, lze však také použít k zobrazení více prvku, které se v tomto případě zobrazí přes sebe. Jejich pozici lze určit pomocí `layout_gravity`. (9) (10) (29)

ScrollView

`ScrollView` je odvozeno od `FrameLayout` a řeší situaci, kdy je třeba zobrazit obsah přesahující velikost displeje, umožňuje totiž posouvat obsah ve svislém směru. Pro vodorovné posouvání lze použít `HorizontalScrollView`. `ScrollView` umožňuje vnoření pouze jednoho prvku, Řešením však může být vnoření `ViewGroup` do, kterého lze umístit další prvky. (9)

Layouty vytvářené Adapterem

Pokud je obsah, který je třeba zobrazit dynamický a není předem definovaný, lze využít některého z potomků třídy `AdapterView`, které jsou posouvatelné. Třída `Adapter` funguje jako prostředník mezi zdrojem dat a layoutem. Jde například o `GridView`, kde jsou prvky uspořádány do mřížky, využitím může být například zobrazení více obrázků v galerii. `ListView`, s uspořádáním obsahu do jednoho sloupce pod sebe, se používá pro zobrazení různých seznamů.

`AdapterView` je od verze Androidu 5 nahrazeno jeho pokročilejší verzí `RecyclerView`. Ta je šetří operační paměť telefonu. Obzvláště pro větší množství zobrazených prvků, protože vytvoří pouze počet prvků, který je viditelný na obrazovce. Při scrollování jsou pak vytvořené prvky recyklována a jsou na ně navázána nová data. (10)

Widgets

Widgets jsou prvky grafického uživatelského rozhraní. Jedná se o potomky třídy `View`. Slovo widget je používáno také pro widget aplikace, což je prvek uživatelského rozhraní, který lze umístit přímo na plochu telefonu a umožňuje tak přímý přístup k funkcím aplikace.

Mezi nejpoužívanější widgety patří:

- `TextView` - prvek sloužící k zobrazení textu.
- `EditText` - umožňuje uživateli zadávat text.
- `ImageView` – prvek pro zobrazení obrázku
- `Spinner` – umožňuje výběr jedné položky ze seznamu. Je přizpůsoben malé ploše dotykového displeje telefonu.
- `Button` - tlačítko
- `CheckBox` – dvoustavový prvek, který je možné přepínat mezi hodnotou `true` (pravda) a `false` (nepravda).
- `RadioButton` – je třeba používat minimálně dva prvky tohoto typu, vznikne tak přepínač - vždy může být vybrán pouze jeden.
- `ProgressBar` – prvek indikující průběh činnosti. Může být konečný, kdy jeho pomocí může uživatel určit, v jakém stadiu se činnost nachází nebo nekonečný, ze kterého tuto informaci vyčíst nelze.

Zdroje: (28) (13)

Material design

Material design je způsob jakým navrhovat uživatelské rozhraní, byl představen v roce 2014 společností Google. Tento design vytváří dojem, že jsou jednotlivé prvky umístěné v trojrozměrném prostoru, mohou se překrývat a vrhají stíny. (13) (13) (30)

Obrázek 11: Material Design, zdroj: (30)

3.3.5 Distribuce aplikace

Hotovou aplikaci je možné distribuovat buď přímým zasláním balíčku APK uživateli, nebo pomocí obchodu Play či alternativním obchodu s aplikacemi. Android chrání uživatele před instalací potenciálně nebezpečných aplikací tím, že uživatel musí povolit instalaci aplikací z neznámých zdrojů. Těmi jsou všechny aplikace z jiného zdroje než je obchod Play. (9) (31)

Google Play

Google Play je distribuční služba, nabízející, mimo jiné také aplikace. Tato služba vznikla sloučením Google Music a Android Market v roce 2012. (32) Právě Android market byl do té doby hlavní distribučním kanálem pro aplikace, tato část Google Play se nyní nazývá Google Play Store. Aby mohl vývojář svou aplikaci na Google Play zveřejnit, musí se zaregistrovat jako vývojář a zaplatit registrační poplatek ve výši 25 dolarů. (9)

4 Vlastní zpracování – aplikace Habit tracker

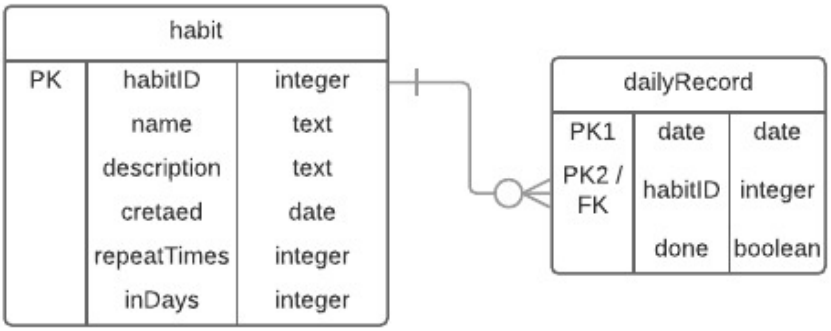
V rámci této práce byla vytvořena aplikace pro Android s názvem Habit tracker, česky „hlídač návyků“. Jelikož český název nezní příliš atraktivně, není v aplikaci přeložen. Habit tracker umožňuje uživateli zadat návyky, které si chce osvojit, a sledovat úspěšnost jejich dodržování. Po dvaceti až třiceti pravidelných opakování si uživatel činnost zautomatizuje a skutečně se stává jeho návykem, nemusí se nadále do činnosti nutit. (33)

4.1 Databáze a datový model

K uchování dat slouží v aplikaci databáze SQLite, která je součásti operačního systému. SQLite nenabízí datové typy pro datum nebo typ `boolean`, obsahující hodnotu pravda nebo nepravda. Z tohoto důvodu jsou v navržené databázi, datumové hodnoty uchovávány ve formě textu ve formátu „RRRR/MM/DD“ a hodnoty typu `boolean` ve formě čísla, kde 0 představuje stav nepravda a 1 stav pravda. Pro lepší názornost jsou však v ERD (Entity-Relationship Diagram) typy `date` (datum) a `boolean` použity.

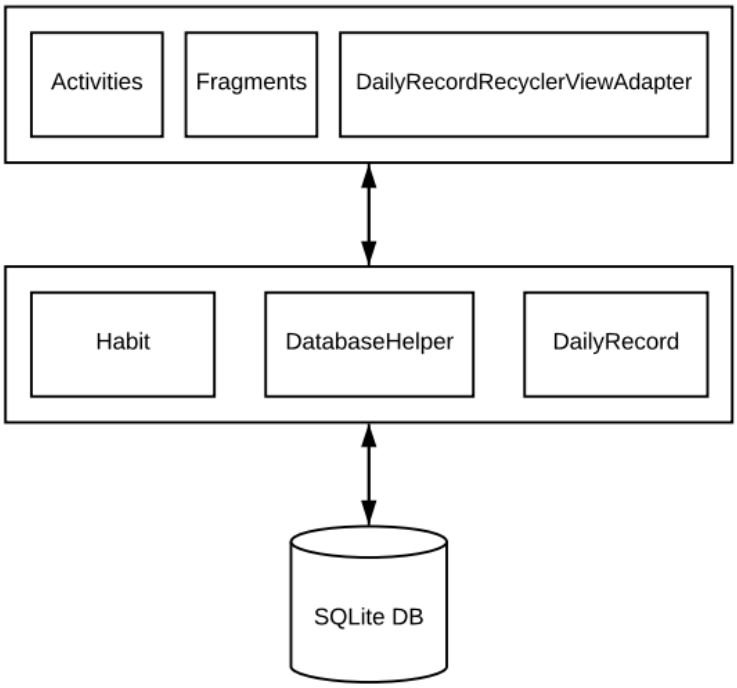
4.1.1 Návrh databáze

Pro účely aplikace Habit tracker je dostačující jednoduchý datový model obsahující dvě entity. Těmi jsou `habit` (návyk) a `dailyRecord` (denní záznam). Primárním klíčem entity `habit` je jeho unikátní ID, které je databází při přidání záznamu automaticky inkrementováno, dále obsahuje textové atributy `name` (jméno), `description` (popis), `created` (vytvořen) a číselné atributy `repeatTimes` („kolikrát“) a `inDays` („ve dnech“). Atribut `created` obsahuje datum vytvoření návyku, `repeatTimes` a `inDays` pak slouží k uchování informace o požadované frekvenci plnění, například pětkrát za sedm dní. Denní záznam je identifikován datem a ID návyku, v databázi tak nemůže existovat více záznamů pro jeden den a návyk. ID návyku (`habitID`) je také cizím klíčem a slouží k provázání těchto dvou entit. Posledním atributem entity `dailyRecord` je `done`, který uchovává informaci o tom, zda byl daný den návyk splněn nebo ne.



Obrázek 12: Entity-Relationship Diagram navržené databáze, zdroj: vlastní zpracování

V projektu jsou definované třídy, odpovídající těmto entitám, `Habit` a `DailyRecord`, které obsahují přístupové metody ke svým atributům. Třída `Habit` obsahuje také třídní konstantu typu `String ARG_ITEM_ID`, která slouží jako klíč pro předávání id návyku v rámci aplikace pomocí třídy `Bundle`. `DailyRecord` obsahuje třídní konstanty typu `int DONE_TRUE = 1` a `DONE_FALSE = 0`, definující možné hodnoty pro atribut `done`.



Obrázek 13: Diagram znázorňující přístup k databázi skrze datovou vrstvu aplikace, zdroj: vlastní zpracování

4.1.2 Třída DatabaseHelper

Přístup k databázi umožňuje ostatním třídám v projektu třída `DatabaseHelper`, která je potomkem třídy `SQLiteOpenHelper`. Ta umožňuje vytvoření databáze a práci s verzemi pro případ změny struktury databáze v aktualizaci aplikace. Jednotlivé operace s databází v přístupových metodách jsou realizované pomocí metod třídy `SQLiteDatabase`.

Metody třídy DatabaseHelper

- `public void onCreate(SQLiteDatabase db)` - překrytá metoda rodičovské třídy, vytváří tabulky s návyky a denními záznamy.
- `public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)` - překrytá metoda rodičovské třídy, definuje co se má stát v případě navýšení verze databáze v rámci aktualizace.
- `public long addHabit(Habit habit)` – vytvoří nový záznam pro návyk, obsahující hodnoty argumentu, instance třídy `Habit`. Metoda vrací ID vytvořeného návyku.
- `public Habit getHabit(int id)` – vrací instanci třídy Habit vytvořené ze záznamu v databázi, který byl nalezen pomocí zadaného ID.
- `public List<Habit> getAllHabits()` – vrací list všech návyků uložených v databázi.
- `public void updateHabit(Habit habit)` - upraví záznam návyku podle argumentu.
- `public void deleteHabit(Habit habit)` – smaže daný návyk z databáze
- `public void addDailyRecord(DailyRecord record)` - vytvoří nový denní záznam, obsahující hodnoty argumentu, instance třídy `DailyRecord`.
- `public DailyRecord getDailyRecord(String date, int habitId)` – vrací instanci třídy `DailyRecord` na základě daného data a ID návyku.
- `public List<DailyRecord> getDailyRecords(String date)` - Vrací list instancí třídy `DailyRecord` pro dané datum.
- `public List<DailyRecord> getDailyRecords(int habitId)` - Vrací list instancí třídy `DailyRecord` pro daný návyk.

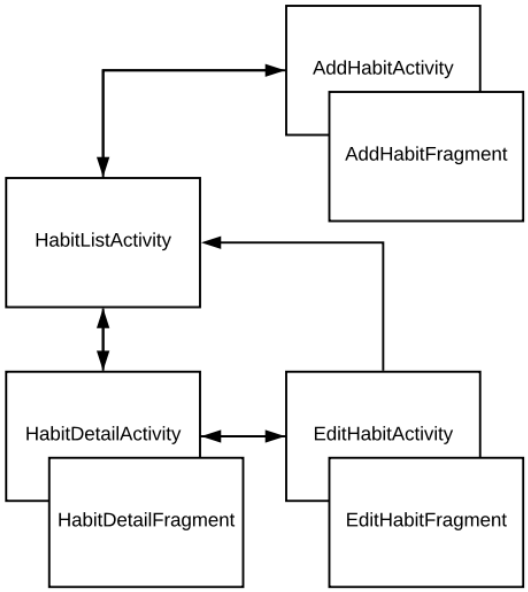
- `public void updateDailyRecord(DailyRecord record)` - upraví denní záznam podle argumentu.
- `public void deleteDailyRecords(Habit habit)` – smaže všechny denní záznamy pro daný návyk.
- `public void createDailyRecords()` - Metoda projde v cyklu všechny existující návyky, pro každý z nich iteruje data od aktuálního k datu vytvoření návyku a pokusí se vyhledat příslušný denní záznam. Pokud neexistuje, vytvoří jej. Pokud záznam již existuje, zbývající data již nejsou iterována a přechází se následující návyk.

Implementace metody `createDailyRecords()`:

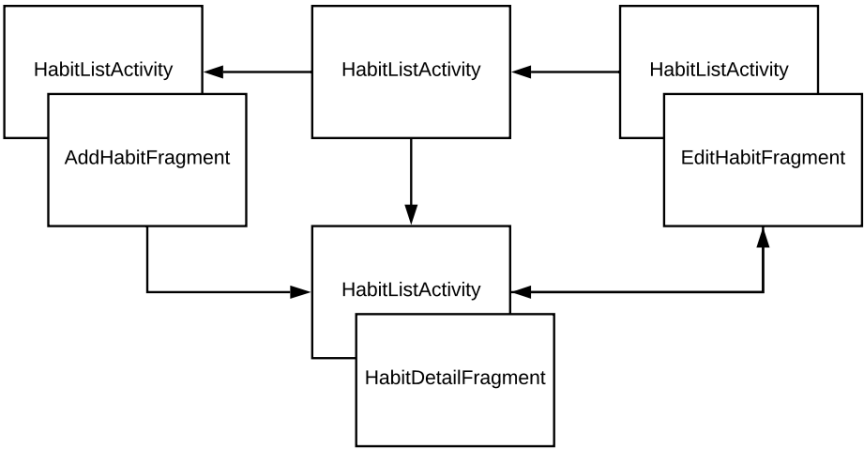
```
public void createDailyRecords() {
    List<Habit> habits = getAllHabits();
    LocalDate currentDate = new LocalDate();
    for (Habit habit : habits) {
        LocalDate dateOfCreation;
        dateOfCreation = LocalDate.parse(habit.getDateOfCreation(),
                DATE_FORMATTER);

        for (LocalDate date = currentDate; date.isAfter(dateOfCreation);
             date = date.minusDays(1)) {
            DailyRecord dailyRecord =
                    getDailyRecord(DATE_FORMATTER.print(date), habit.getId());
            if (dailyRecord == null) {
                addDailyRecord(new DailyRecord(DATE_FORMATTER.print(date),
                        habit.getId(), DailyRecord.DONE_FALSE));
            } else {
                break;
            }
        }
    }
}
```

4.2 Struktura aplikace

Struktura aplikace byla navržena tak, aby uživatelské rozhraní vyhovovalo jak telefonům, tak i tabletům. Toho bylo dosaženo pomocí struktury master-detail, kde na telefonech jsou pro různé akce otevírány nové aktivity a na tabletech dochází pouze k výměně zobrazeného fragmentu v rámci hlavní aktivity, kterou je `HabiListActivity`. Zbylé aktivity slouží pouze jako kontejner pro zobrazení příslušného fragmentu, který nese funkcionalitu. `AddHabitFragment` umožňuje přidání nového návyku, `EditHabitFragment` jeho editaci a `HabitDetailFragment` zobrazuje podrobností o daném návyku.



Obrázek 15: Diagram, znázorňující přechody mezi aktivitami na zařízení s malým displejem, zdroj: vlastní zpracování



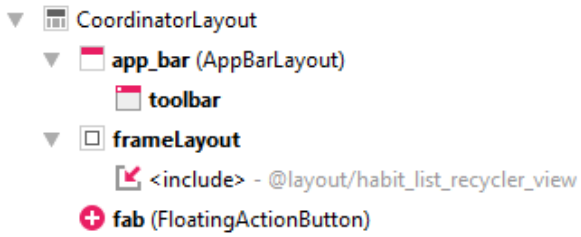
Obrázek 14: Diagram, znázorňující přechody mezi stavy aplikace na zařízení s velkým displejem, zdroj: vlastní zpracování

4.2.1 Denní přehled návyků

Denní přehled návyku představuje hlavní obrazovku aplikace tvořenou aktivitou `HabitListActivity`.

Uživatelské rozhraní

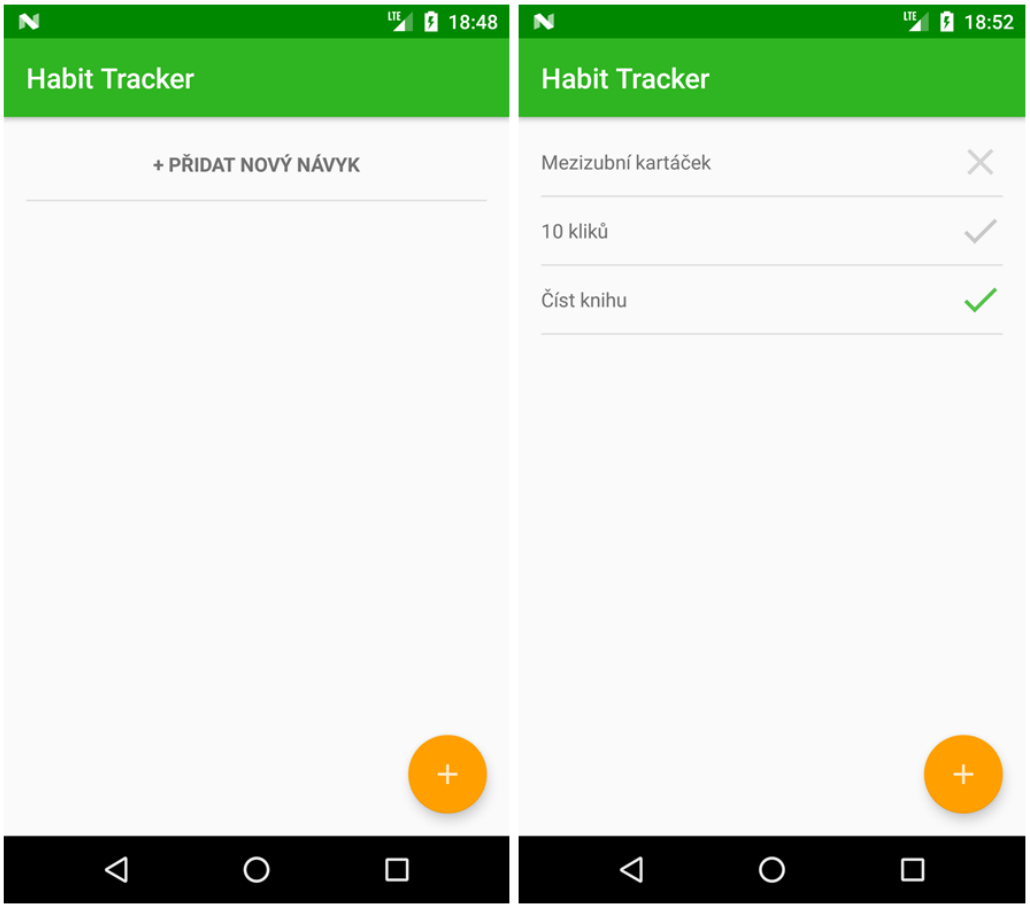
Soubor definující uživatelské rozhraní `activity_habit_list.xml` má následující strukturu:



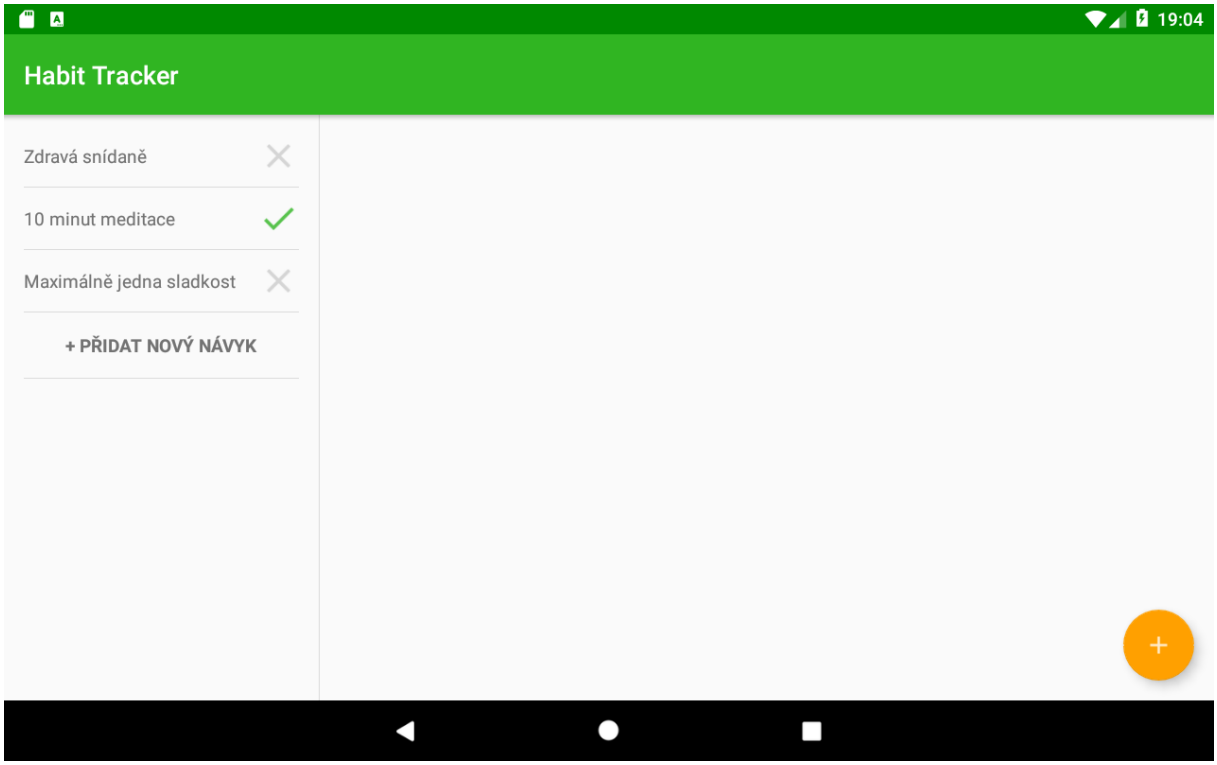
Obrázek 16: Struktura uživatelského rozhraní v souboru `activity_habit_list.xml`, zdroj: vlastní zpracování

Rodičovským prvkem je `CoordinatorLayout`. Obsahuje `Toolbar` (aplikační lištu) zobrazující název aplikace, tlačítko `FloatingActionButton` a `FrameLayout`, do kterého je pomocí elementu `include` vložen layout ze souboru `habit_list_recycler_view.xml`. Je tomu tak z důvodu znovupoužití tohoto layoutu v jiné části aplikace. Soubor `activity_habit_list.xml` se v projektu vyskytuje dvakrát, jednou ve standardní složce layout a podruhé ve složce layout-w900dp, kde je definice pro zařízení s širokým displejem. Zbylé soubory s definicemi layoutu jsou v projektu jen jednou a jsou použity pro všechna zařízení. Definice `activity_habit_list.xml` pro široké displeje se odlišuje pouze tím, že prvek `FrameLayout` má nastavenou pevnou šířku a je vnořen do `LinearLayout` s horizontální orientací, ve kterém se nachází další prvek `FrameLayout`, sloužící jako kontejner pro fragmenty. Soubor `habit_list_recycler_view.xml` obsahuje definici prvku `RecyclerView`, pomocí kterého je v aplikaci zobrazen seznam denních záznamů návyků. Definice řádku pro denní záznam návyku je v souboru `habit_row.xml`, definice pro speciální řádek, představující tlačítko pro přidání nového návyku, v `habit_list_button.xml`. V obou případech jde o `LinearLayout`, kde standardní řádek obsahuje `ImageView`, sloužící pro zobrazení ikony

stavu splnění daného návyku, a `TextView`. Řádek, sloužící jako tlačítko obsahuje pouze `TextView`.



Obrázek 17: Uživatelské rozhraní `HabitListActivity` na telefonu. Vlevo: před vytvořením návyků, vpravo: obrazovka s již vytvořenými návyky, zdroj: vlastní zpracování



Obrázek 18: Uživatelské rozhraní `HabitListActivity` na tabletu, zdroj: vlastní zpracování

Funkcionalita a implementace

onCreate

V metodě `onCreate` se nejprve do prvku `Toolbar` nastaví zobrazení názvu aplikace, poté se definuje funkce tlačítka `FloatingActionButton` (dále FAB) na volání metody `openAddHabit` a nastaví se jeho ikona na „+“. Dále se ukládá informace o tom, zda je použit layout pro široká zařízení do proměnné `mTwoPane`, vytváří instance třídy `DatabaseHelper`, pomocí které je získán list záznamů pro dnešní datum. Ten je nastaven adaptérem `HabitListRecyclerViewAdapter` do prvku `RecyclerView`.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_habit_list);

    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    toolbar.setTitle(getTitle());

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setImageResource(R.drawable.ic_add);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            openAddHabit();
        }
    });

    if (findViewById(R.id.habit_fragment_container) != null) {
```

```
            mTwoPane = true;
        }

        database = new DatabaseHelper(this);

        RecyclerView recyclerView = findViewById(R.id.habit_list);
        assert recyclerView != null;
        mAdapter = new HabitListRecyclerViewAdapter(this,
                database.getDailyRecords(DatabaseHelper.DATE_FORMATTER.print(new
LocalDate())));
        recyclerView.setAdapter(mAdapter);
        recyclerView.addItemDecoration(new DividerItemDecoration(this,
                DividerItemDecoration.VERTICAL));
}
```

onResume

V metodě `onResume` je volána metoda třídy `DatabaseHelper` `createDailyRecords`, která vytváří záznamy pro dnešní den a pro dny předchozí, pokud již neexistují. Tvorba záznamů je součástí metody `onResume` a nikoli `onCreate` proto, aby byly nové záznamy vytvořeny i v případě, kdy je aplikace před půlnocí poslána na pozadí a po půlnoci opět do popředí. V `onResume` je také volána metoda adaptéru `reloadRecords` s nově získaným listem návyků z databáze, aby byl seznam aktuální v případě, že se uživatel navrátil z aktivity pro přidání či editaci návyku. K přepsání záznamů dochází také v metodě `onAttachFragment`. Tím je ošetřena tatáž situace při použití layoutu pro tablety.

openAddHabit

Implementace metody openAddHabit, volané tlačítkem FAB, je následující. Pokud je použit layout pro tablety, je do prvku s ID `habit_fragment_container` vložen `AddHabitFragment`, pokud je použit běžný layout, spustí se aktivita `AddHabitActivity` pomocí třídy `Intent`.

```
private void openAddHabit() {
    if (mTwoPane) {
        AddHabitFragment fragment = new AddHabitFragment();
        getSupportFragmentManager().beginTransaction().replace(
                R.id.habit_fragment_container, fragment).commit();
    } else {
        Intent theIntent = new Intent(getApplication(), AddHabitActivity.class);
        startActivity(theIntent);
    }
}
```

DailyRecordRecyclerViewAdapter

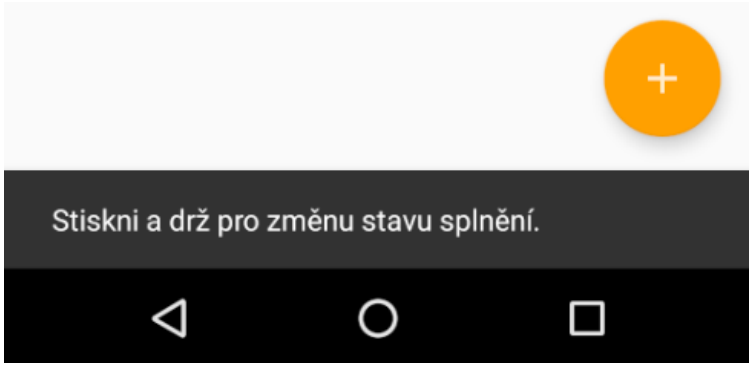
`DailyRecordRecyclerViewAdapter` není součástí `HabitListActivity`, je však předkem třídy `HabitListRecyclerViewAdapter`, jenž je vnitřní třídou `HabitListActivity` a je použita pro vytvoření denního seznamu návyků. Protože se v detailu návyku nachází seznam, který je v mnoha ohledech podobný tomu tomuto,

bylo společné chování vyčleněno do `DailyRecordRecyclerViewAdapter`. Funkcionalita seznamu je definována především v metodě `onBindViewHolder`. Ta je volána ve chvíli, kdy se data navazují na zobrazené views (řádky), tedy při scrollování seznamem nebo při jeho vytvoření. Tato metoda je implementována následovně:

```
@Override
public void onBindViewHolder(final ViewHolder holder, final int position) {
    if (position != mRecords.size()) {
        final DailyRecord record = mRecords.get(position);

        final int done = record.getDone();
        Habit habit = database.getHabit(record.getHabitId());
        int times = habit.getTimes();
        int inDays = habit.getInDays();

        int doneInDays = 0;
        LocalDate recordDate;
        recordDate = LocalDate.parse(record.getDate(), database.DATE_FORMATTER);


 for (int i = 1; i <= inDays; i++) {
            DailyRecord dailyRecord = database.getDailyRecord(
                    database.DATE_FORMATTER.print(recordDate), habit.getId());
            if (dailyRecord == null) {
                break;
            }
            if (dailyRecord.getDone() == DailyRecord.DONE_TRUE) {
                doneInDays++;
            }
            recordDate = recordDate.minusDays(1);
        }

        if (done == DailyRecord.DONE_FALSE) {
            if (doneInDays >= times) {
                holder.mDoneView.setImageResource(R.drawable.ic_action_check);
                doneByFrequency++;
            } else {
                holder.mDoneView.setImageResource(R.drawable.ic_action_x);
                doneFalse++;
            }
            holder.mDoneView.setColorFilter(mParentActivity.getResources()
                    .getColor(R.color.colorGrey));
        } else if (done == DailyRecord.DONE_TRUE) {
            holder.mDoneView.setImageResource(R.drawable.ic_action_check);
            holder.mDoneView.setColorFilter(mParentActivity.getResources()
                    .getColor(R.color.colorPrimary));
            doneTrue++;
        }

        holder.itemView.setTag(record);

        holder.mDoneView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Snackbar.make(holder.itemView, mParentActivity.getResources()
                        .getString(R.string.habit_row_onclick),
                        Snackbar.LENGTH_LONG).show();
            }
        });

        holder.mDoneView.setOnLongClickListener(new View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                int doneNew;
                if (done == DailyRecord.DONE_FALSE) {
                    doneNew = DailyRecord.DONE_TRUE;
                } else {
                    doneNew = DailyRecord.DONE_FALSE;
                }
```

```
                record.setDone(doneNew);
                database.updateDailyRecord(record);
                reloadRecords(mRecords);
                return true;
            }
        });
    }
}
```

Podmínka `if (position != mRecords.size())`, obalující obsah této metody zajišťuje, že následně popsané nastavení nebude provedeno, pokud je řádkem tlačítko pro přidání nového záznamu. Pokud jde o běžný řádek, získá se z databáze návyk a informace o požadované frekvenci jeho dodržování. V cyklu se projdou záznamy pro tento návyk za předchozí dny, jejichž počet je dán hodnotou `inDays` a sečte se počet dní, kdy byl návyk dodržen. Na základě toho se řádku nastaví stav ve formě ikonky, kde zelená fajfka značí, že byl zvyk dodržen. Šedá fajfka, že nebyl dodržen dnes, ale v předchozích dnech dle definované frekvence a křížek značí, že návyk nebyl dodržen dnes ani v dostatečném počtu předchozích dní. Dále je definováno chování pro kliknutí na tuto ikonku. Zobrazí se prvek `Snackbar` nesoucí informaci o tom, že pro změnu stavu je třeba ikonku podržet. Při podržení se do databáze uloží změna hodnoty `done` (splněno) a dojde k aktualizaci zobrazeného seznamu.



Obrázek 19: Prvek `Snackbar` zobrazený po kliknutí na ikonu stavu denního záznamu návyku, zdroj: vlastní zpracování

HabitListRecyclerViewAdapter

`HabitListRecyclerViewAdapter` rozšiřuje funkcionalitu svého předka o vložení tlačítka pro přidání nového záznamu. To je v uživatelském rozhraní pro tablety zobrazeno vždy a na telefonech pouze v případě, že není vytvořen žádný návyk. Důvodem přidání tohoto tlačítka je fakt, že jakmile je v rozhraní pro tablety zobrazen některý z fragmentů, tlačítko FAB mění svou funkcionalitu podle daného fragmentu. Uživatel by tak musel pro přidání druhého a každého dalšího návyku aplikaci vypnout a opět zapnout. Druhou

změnou oproti `DailyRecordRecyclerViewAdapter` je nastavení chování při kliknutí na řádek seznamu. Tím je v případě tabletu nahrazení aktuálního fragmentu, fragmentem `HabitDetailFragment`, v případě telefonu spuštění aktivity `HabitDetailActivity`.

```
private class HabitListRecyclerViewAdapter extends DailyRecordRecyclerViewAdapter {
    HabitListRecyclerViewAdapter(HabitListActivity parent,
                                 List<DailyRecord> items) {
        super(parent, items);
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        if(viewType == R.layout.habit_row){
           return super.onCreateViewHolder(parent, viewType);
        }
        else {
            return new ViewHolder(LayoutInflater.from(parent.getContext())
                    .inflate(R.layout.habit_list_button, parent, false));
        }
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        super.onBindViewHolder(holder, position);

        if(position == mRecords.size()) {
            holder.itemView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                   openAddHabit();
                }
            });
        }
        else {
            final Habit habit = database
                    .getHabit(mRecords.get(position).getHabitId());
            holder.mTextView.setText(habit.getName());
            holder.itemView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    if (mTwoPane) {
                        Bundle arguments = new Bundle();
                        arguments.putInt(Habit.ARG_HABIT_ID, habit.getId());
                        HabitDetailFragment fragment = new HabitDetailFragment();
                        fragment.setArguments(arguments);
                        getSupportFragmentManager().beginTransaction()
                                .replace(R.id.habit_fragment_container,
                                        fragment).commit();
                    } else {
                        Context context = view.getContext();
                        Intent intent = new Intent(context,
                                HabitDetailActivity.class);
                        intent.putExtra(Habit.ARG_HABIT_ID, habit.getId());
                        context.startActivity(intent);
                    }
                }
            });
        }
    }

    @Override
    public int getItemViewType(int position) {
        return (position == mRecords.size())
                ? R.layout.habit_list_button : R.layout.habit_row;
    }

    @Override
    public int getItemCount() {
        if (mTwoPane || mRecords.size() == 0) {
            return mRecords.size() + 1;
```

```
            }
            return super.getItemCount();
        }
    }
```

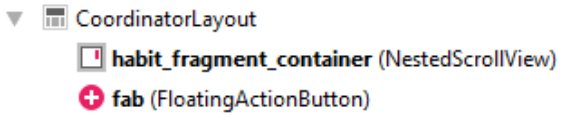
4.2.2 Přidání a editace návyku

Přidání a editace návyku jsou natolik podobné úkony, že mohou velkou část kódu sdílet.

Uživatelské rozhraní

Aktivity

Definice uživatelského rozhraní aktivit `AddHabitActivity` a `EditHabitActivity` je uložena v souboru `activity_habit_properties.xml` a má následující strukturu.

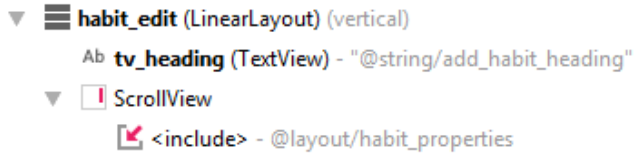


Obrázek 20: Struktura uživatelského rozhraní v souboru `activity_habit_properties.xml`, zdroj: vlastní zpracování

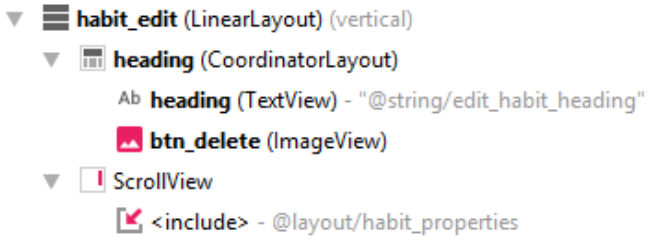
Prvek `NestedScrollView` zde slouží jako kontejner pro fragment.

Fragmenty

Oba fragmenty jsou vytvořeny pomocí `LinearLayout` s vertikální orientací a obsahují `scrollView`, které pomocí elementu `include` používá rozhraní definované v souboru `habit_properties.xml`. `EditHabitFragment` obsahuje navíc prvek `ImageView` s ikonou odpadkového koše, který slouží jako tlačítko pro smazání návyku.



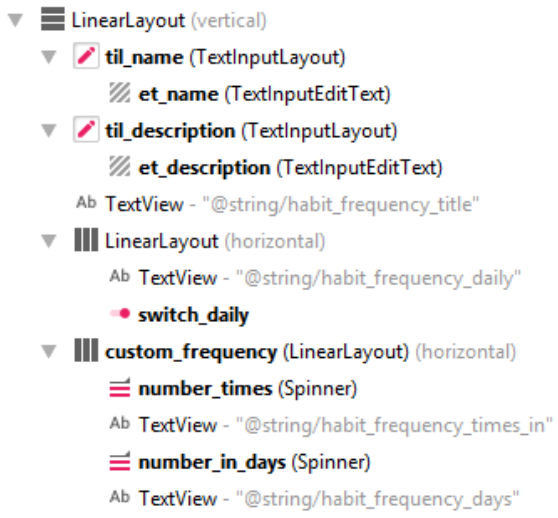
Obrázek 21: Struktura uživatelského rozhraní v souboru `fragment_add_habit.xml`, zdroj: vlastní zpracování



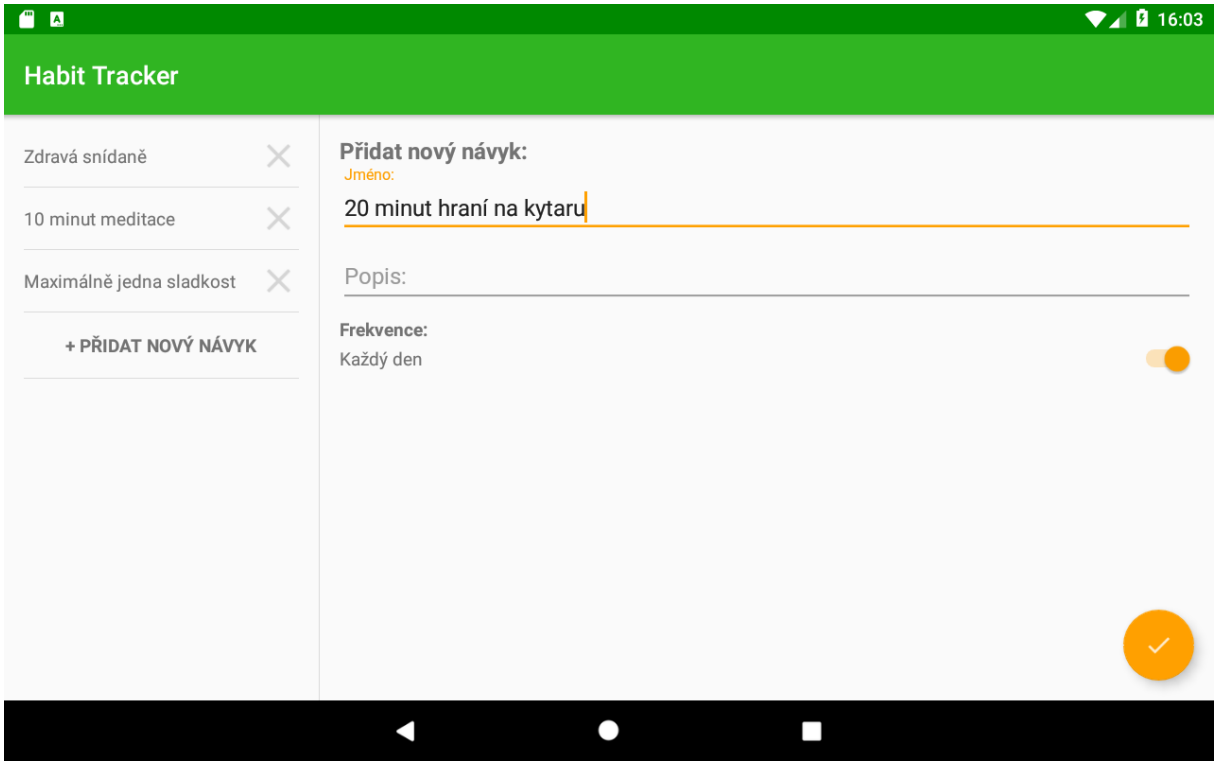
Obrázek 22: Struktura uživatelského rozhraní v souboru `fragment_edit_habit.xml`, zdroj: vlastní zpracování

habit_properties.xml

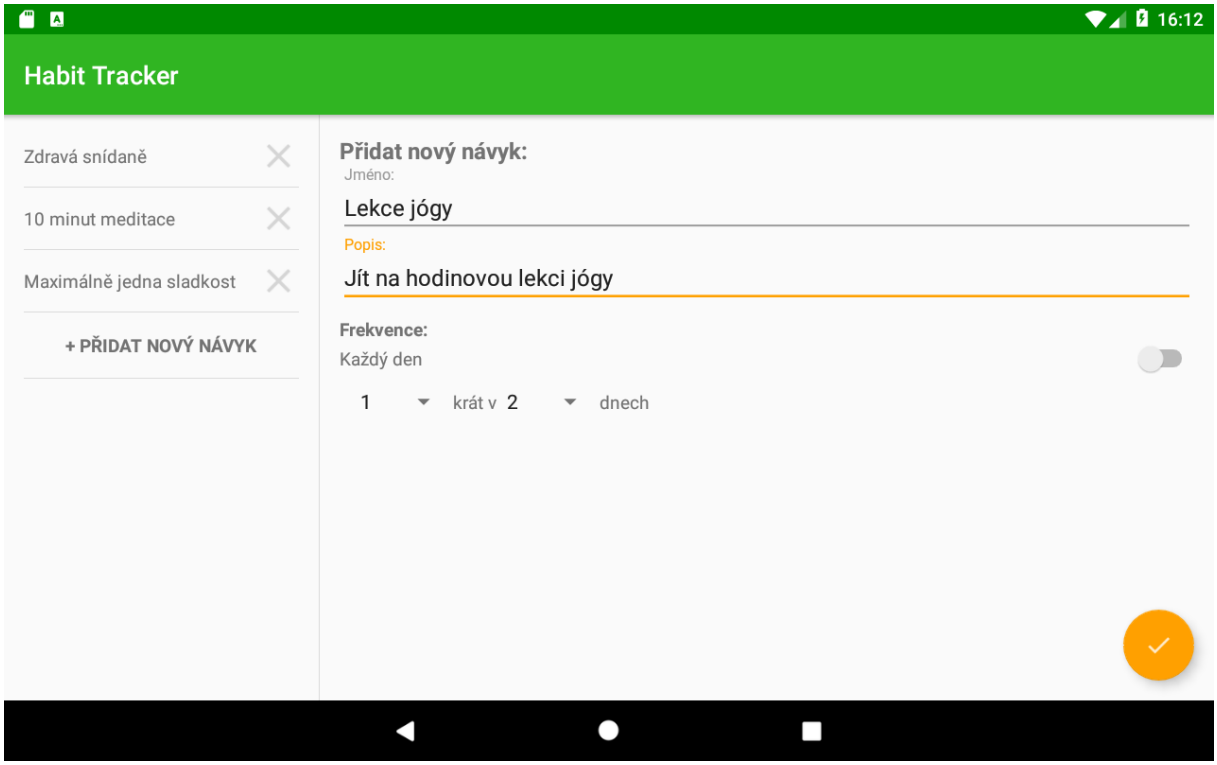
V tomto souboru jsou definována pole pro zadání názvu a popisu návyku, realizované pomocí prvku `TextInputLayout`. Frekvence opakování lze definovat pomocí přepínače (`Switch`), pokud je zapnutý, požadovaná frekvence je každý den. V případě, že uživatel přepne do druhé polohy, zobrazí se možnost zvolit, kolikrát za kolik dní chce daný návyk dodržovat. Výběr počtu dnů je řešen prvkem `Spinner`.



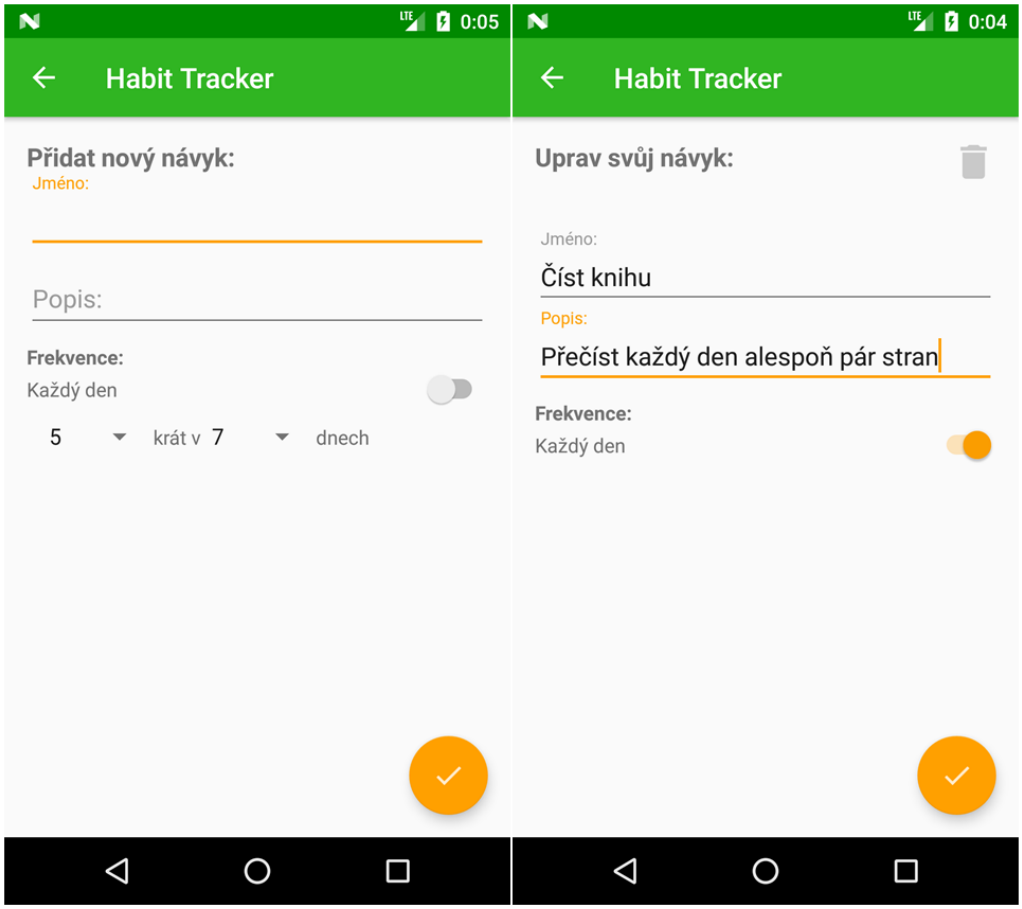
Obrázek 23: Struktura uživatelského rozhraní v souboru `habit_properties.xml`, zdroj: vlastní zpracování



Obrázek 24: Uživatelské rozhraní pro přidání nového návyku na tabletu, zdroj: vlastní zpracování



Obrázek 25: Uživatelské rozhraní pro editaci návyku na tabletu, zdroj: vlastní zpracování



Obrázek 26: Uživatelské rozhraní pro přidání nového návyku (vlevo) a editace návyku (vpravo) na telefonu, zdroj: vlastní zpracování

Funkcionalita a implementace

Aktivity `AddHabitActivity` a `EditHabitActivity`, použité pouze v případě zařízení s menším displejem, obsahují pouze metodu `onCreate`. V té je nastaveno zobrazení navigačního prvku (šipky) v aplikační liště, která vede na místo, ze kterého byla daná aktivita spuštěna. V případě přidání nového návyku, `HabitListActivity` a v případě editace, `HabitDetailActivity`. Jedinou další úlohou v této metodě je zobrazení příslušného fragmentu. V případě editace se nejprve musí získat ID návyku, které bylo aktivitě předáno pomocí záměru, a použít jej jako argument instance fragmentu `EditHabitFragment`.

HabitProperitesFragment

`HabitPropertiesFrament` je předkem tříd `AddHabitFragment` a `EditHabitFragment`, obsahuje jejich společnou funkcionalitu.

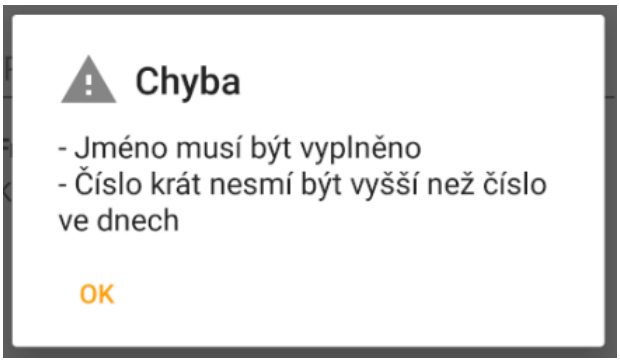
V metodě `onCreate` se vytváří instance třídy `DatabaseHelper` a do proměnné `mTwoPane` se ukládá informace o tom, zda byl použit layout pro tablet nebo telefon – pokud je zobrazenou aktivitou `HabitListActivity`, hodnota `mTwoPane` je `true`.

V metodě `onCreateView` jsou nalezeny prvky uživatelského rozhraní a ukládají se do proměnných, které jsou později použity v potomcích této třídy. Dále je nastaveno chování přepínače pro denní opakování – když je přepínač zapnutý zobrazí se možnost zadat vlastní frekvenci opakování, pokud je přepínač vypnutý, tato možnost se skryje. Také se v této metodě definuje obsah prvků `Spinner`. Tím jsou celá čísla od jedné do třiceti, což je rozsah poskytující dostatečnou variabilitu frekvence opakování.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState, View rootView) {
    editTextName =  rootView.findViewById(R.id.et_name);
    editTextDescription = rootView.findViewById(R.id.et_description);

    switchRepeatDaily = rootView.findViewById(R.id.switch_daily);
    final View customFrequency = rootView.findViewById(R.id.custom_frequency);
    switchRepeatDaily.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            if (isChecked) {
                customFrequency.setVisibility(View.GONE);
            } else {
                customFrequency.setVisibility(View.VISIBLE);
            }
        }
    });

    spinnerTimes = rootView.findViewById(R.id.number_times);
    spinnerInDays = rootView.findViewById(R.id.number_in_days);
```

```
    Integer [] items = new  Integer [30];
    for (int i = 0; i < 30; i++){
        items[i] = i +1;
    }
    ArrayAdapter<Integer> adapter = new ArrayAdapter<Integer>(getActivity(),
android.R.layout.simple_spinner_item, items);
    spinnerTimes.setAdapter(adapter);
    spinnerInDays.setAdapter(adapter);

    return rootView;
}
```

Poslední metodou této třídy je `checkValuesAndGetHabit`, ta zajišťuje kontrolu zadaných hodnot a pokud jsou v pořádku, vytvoří z nich instanci třídy Habit, kterou tato metoda vrací. Nejprve se zkontroluje zaškrtnutí přepínače pro denní opakování. Pokud je zapnutý, nastaví se jako hodnota v prvcích `Spinner` číslo 1. Poté je ověřeno, že má návyk nastavené jméno a že hodnoty nastavené pro opakování dávají smysl (počet opakování musí být nižší nebo stejný jako celkový počet dní). V případě, že kontrola neproběhne úspěšně, je zobrazen dialog na následujícím obrázku, vytvořený pomocí třídy `AlertDialog`.



Obrázek 27: Chybový dialog při zadávání nebo editaci návyku, zdroj: vlastní zpracování

Implementace metody `checkValuesAndGetHabit`:

```
protected Habit checkValuesAndGetHabit() {
    String name = editTextName.getText().toString();
    String description = editTextDescription.getText().toString();
    int times = Integer.parseInt(spinnerTimes.getSelectedItem().toString());
    int inDays = Integer.parseInt(spinnerInDays.getSelectedItem().toString());

    Habit habit = null;

    if (switchRepeatDaily.isChecked()) {
        times = 1;
        inDays = 1;
    }

    if (name.length() == 0 || times > inDays) {
        String errorMessage = "";
        if (name.length() == 0) {
            errorMessage = getResources().getString(R.string.error_name_empty)
                    + "\n";
```

```
        }

        if (times > inDays) {
            errorMessage = errorMessage + getString(R.string.error_incorrect_numbers);
        }

        new AlertDialog.Builder(getContext())
                .setTitle(getResources().getString(R.string.error_title))
                .setMessage(errorMessage)
                .setIcon(R.drawable.ic_action_warning)
                .setNeutralButton(android.R.string.ok,
                null).show();
    } else {
        habit = new Habit(name, description,
                database.DATE_FORMATTER.print(new LocalDate()),
                times, inDays);
    }

    return habit;
}
```

AddHabitFragment

V metodě `onCreate` třídy `AddHabitFragment` se pouze nastavuje funkce tlačítka FAB na vložení nového návyku do databáze (volání metody `addHabit`). V `onCreateView` se použije správná definice uživatelského rozhraní. `addHabit` nejprve volá svého předka `checkValuesAndGetHabit` a získá tím instanci třídy Habit. Pokud není rovna hodnotě `null`, vloží tento návyk do databáze a poté v případě telefonu spouští aktivitu `HabitListActivity` a v případě tabletu nahrazuje tento fragment, fragmentem `HabitDetailFragment`, kterému předává ID vytvořeného návyku.

Implementace třídy `AddHabitFragment`:

```
public class AddHabitFragment extends HabitPropertiesFragment {

    public AddHabitFragment() {

    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        FloatingActionButton fab = getActivity().findViewById(R.id.fab);
        fab.setImageResource(R.drawable.ic_action_check);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                addHabit();
            }
        });
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_add_habit,
                container, false);
        return super.onCreateView(inflater, container,
                savedInstanceState, rootView);
    }
```

```
 private void addHabit() {
        Habit habit = checkValuesAndGetHabit();

        if (habit != null) {
            int habitId = (int) database.addHabit(habit);
            habit.setId(habitId);

            database.addDailyRecord(new DailyRecord(
                    database.DATE_FORMATTER.print(new LocalDate()),
                    habit.getId(), DailyRecord.DONE_FALSE));

            if (mTwoPane) {
                Bundle arguments = new Bundle();
                arguments.putInt(Habit.ARG_HABIT_ID, habit.getId());
                HabitDetailFragment fragment = new HabitDetailFragment();
                fragment.setArguments(arguments);
                getActivity().getSupportFragmentManager().beginTransaction()
                        .replace(R.id.habit_fragment_container, fragment)
                        .commit();
            } else {
                Intent intent = new Intent(getActivity(),
                        HabitListActivity.class);
                startActivity(intent);
            }
        }
    }
}
```

EditHabitFragment

V `onCreate` je získáno ID upravovaného návyku pomocí metody `getArguments()` a poté, na základě ID, instance třídy `Habit` pro tento návyk. Dále je nastavena funkce tlačítka FAB na volání metody `updateHabit`. V `onCreateView` se vyplní hodnoty upravovaného návyku a nastaví funkce prvku `ImageView` s ikonou odpadkového koše na volání metody `deteleHabit`.

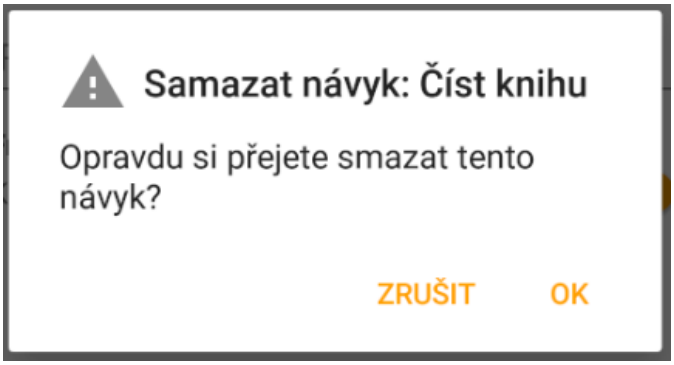
Implementace metody `onCreateView`:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_edit_habit,
            container, false);
    super.onCreateView(inflater, container, savedInstanceState,
            rootView);

    editTextName.setText(habit.getName());
    editTextDescription.setText(habit.getDescription());

    int times = habit.getTimes();
    int inDays = habit.getInDays();
    if (times == inDays) {
        switchRepeatDaily.setChecked(true);
    } else {
        switchRepeatDaily.setChecked(false);
        spinnerTimes.setSelection(times - 1);
        spinnerInDays.setSelection(inDays - 1);
    }

    buttonDelete = rootView.findViewById(R.id.btn_delete);
    buttonDelete.setOnClickListener(new View.OnClickListener() {
        @Override
```

```
        public void onClick(View v) {
            deleteHabit();
        }
    });

    return rootView;
}
```

Metoda `deleteHabit` zobrazuje dialog, vytvořený pomocí třídy `AlertDialog`, po jehož potvrzení dojde k vymazání návyku a všech jeho denních záznamů z databáze. Následně je spuštěna aktivita `HabitListActivity`.

```
private void deleteHabit() {
    new AlertDialog.Builder(getContext())
            .setTitle(getResources().getString(R.string.delete_dialog_title)
                    + " " + habit.getName())
            .setMessage(R.string.delete_dialog_text)
            .setIcon(R.drawable.ic_action_warning)
            .setPositiveButton(android.R.string.yes,
                    new DialogInterface.OnClickListener() {

                public void onClick(DialogInterface dialog,
                                    int whichButton) {
                    database.deleteHabit(habit);
                    database.deleteDailyRecords(habit);
                    Intent intent = new Intent(getActivity(),
                            HabitListActivity.class);
                    startActivity(intent);
                }
            })
            .setNegativeButton(android.R.string.no, null).show();
}
```



Obrázek 28: Dialog zobrazený po kliknutí na ikonu pro smazání návyku, zdroj: vlastní zpracování

Metoda `updateHabit` je obdobou metody `addHabit` fragmentu `AddHabitFragment`. Liší se pouze použitím metody pro úpravu záznamu v databázi namísto jeho přidání, a také otevírá detail upraveného návyku i v případě rozhraní pro menší zařízení.

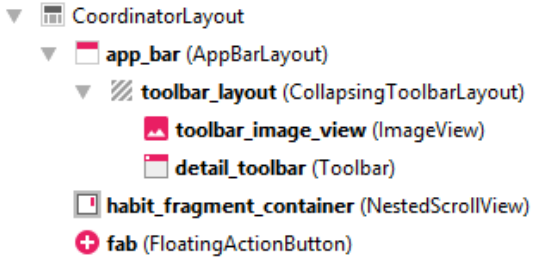
4.2.3 Detail návyku

Detail návyku, zobrazený po kliknutí na návyk v `HabitListActivity`, zobrazuje podrobnosti a historii záznamu pro dny minulé.

Uživatelské rozhraní

activity_habit_detail.xml

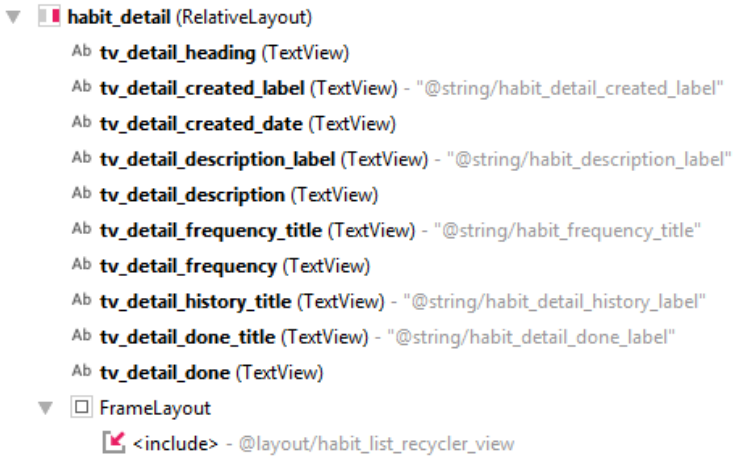
Tato definice uživatelského rozhraní disponuje rozbalovací aplikační lištou, obsahující ikonu se symbolem zaškrtnutí. Pokud je lišta sbalená, ikona není vidět. Aby bylo tohoto efektu dosaženo, byl jako rodičovský prvek použit `CoordinatorLayout`. Dále se zde nachází kontejner pro fragment (`NestedScrollView`) a FAB, které je uchyceno na předělu aplikační lišty a kontejneru.



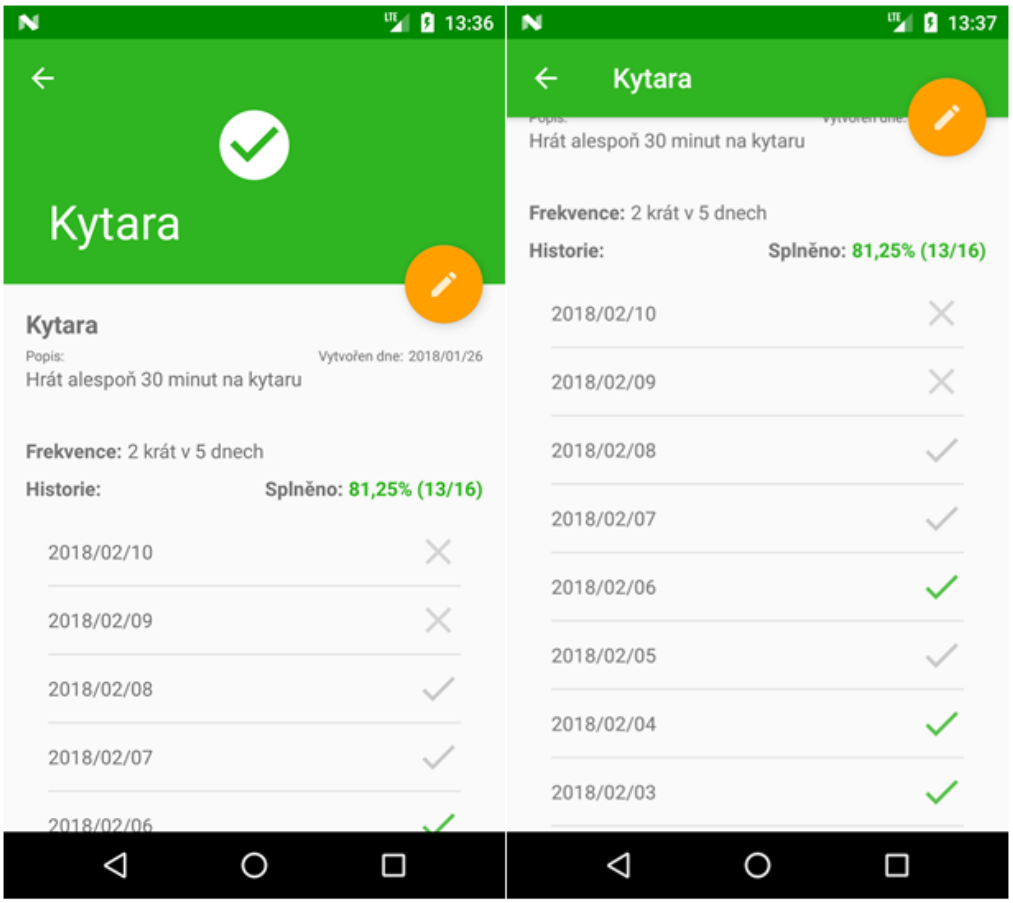
Obrázek 29: Struktura uživatelského rozhraní v souboru `activity_habit_detail.xml`, zdroj: vlastní zpracování

fragment_habit_detail.xml

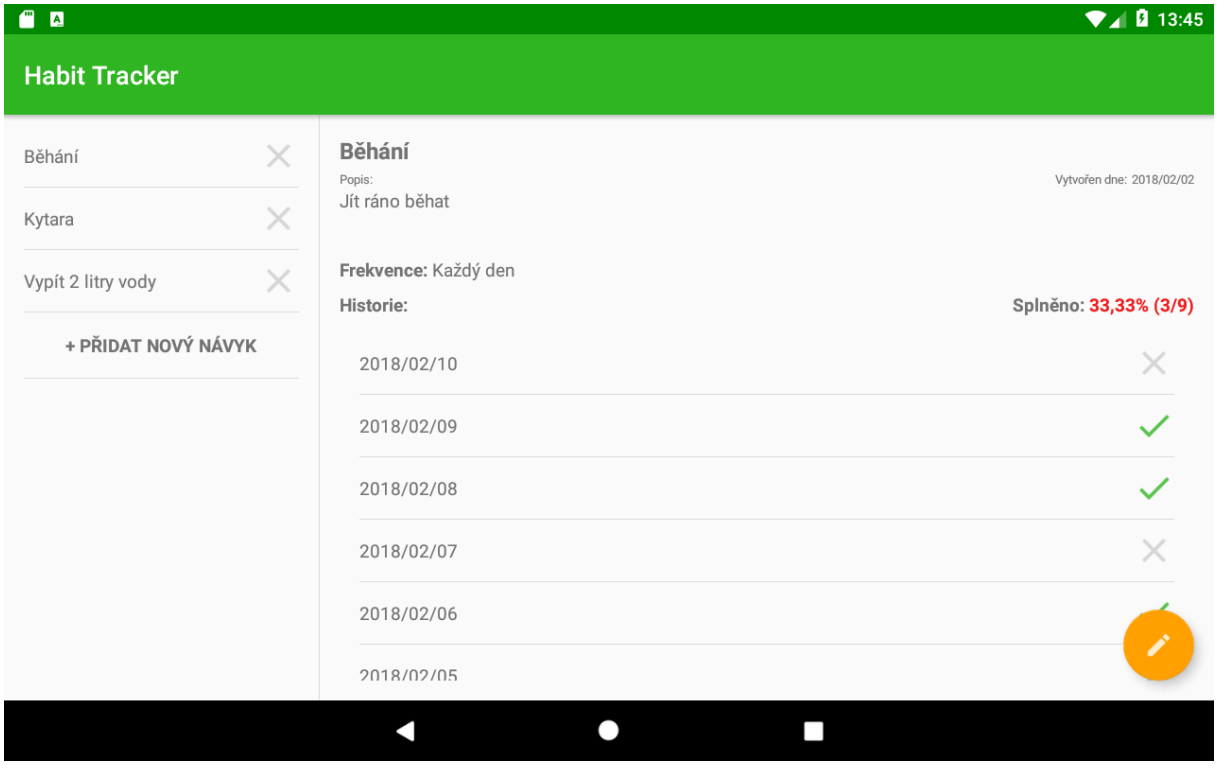
Rodičovským prvkem v tomto souboru je `RelativeLayout`, který obsahuje 10 prvků `TextView` a `FrameLayout`. Do tohoto `FrameLayout` je pomocí `include` vložen `RecyclerView`, definovaný souborem `habit_list_recycler_view.xml`.



Obrázek 30: Struktura uživatelského rozhraní v souboru `fragment_habit_detail.xml`, zdroj: vlastní zpracování



Obrázek 31: Uživatelské rozhraní detailu návyku na telefonu. Vpravo: rozbalená aplikační lišta, vlevo: sbalená aplikační lišta, zdroj: vlastní zpracování



Obrázek 32:Uživatelské rozhraní detailu návyku na tabletu, zdroj: vlastní zpracování

Funkcionalita a implementace

HabitDetailActivity

Obdobně jako u editace návyku, obsahuje tato aktivita pouze nastavení navigačního prvku v aplikační liště a předání ID návyku, vkládanému fragmentu.

```
public class HabitDetailActivity extends AppCompatActivity {

    int habitId;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_habit_detail);

        Toolbar toolbar = (Toolbar) findViewById(R.id.detail_toolbar);
        setSupportActionBar(toolbar);

        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
        }

        habitId = getIntent().getIntExtra(Habit.ARG_HABIT_ID, -1);

        if (savedInstanceState == null) {
            Bundle arguments = new Bundle();
            arguments.putInt(Habit.ARG_HABIT_ID, habitId);
            HabitDetailFragment fragment = new HabitDetailFragment();
            fragment.setArguments(arguments);
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.habit_fragment_container, fragment)
                    .commit();
        }
    }
}
```

HabitDetailFragment

Vnitřní třída tohoto fragmentu, `HabitDetailRecyclerViewAdapter`, je odvozena od `DailyRecordRecyclerViewAdapter` a funkcionalitu svého rodiče rozšiřuje pouze o nastavení data denního záznamu a volání metody `fillStatistic` uvnitř metod `onBindViewHolder` a `reloadRecords`. tato vnitřní třída slouží jako adaptér pro zobrazené `RecyclerView`.

```
private class HabitDetailRecyclerViewAdapter extends DailyRecordRecyclerViewAdapter {

    HabitDetailRecyclerViewAdapter(Activity parent, List<DailyRecord> items) {
        super(parent, items);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        super.onBindViewHolder(holder, position);
        holder.mTextView.setText(mRecords.get(position).getDate());
        fillStatistic(doneTrue, doneFalse, doneByFrequency, textViewDone);
    }

    @Override
    public void reloadRecords(List<DailyRecord> records) {
        super.reloadRecords(records);
```

```
        fillStatistic(doneTrue, doneFalse, doneByFrequency, textViewDone);
    }
```

V `onCreate` se ukládá informace o tom, zda byl použit layout pro tablety podle spuštěné aktivity. Vytváří se zde instance třídy DatabaseHelper, pomocí které je na základě předaného ID návyku získána příslušná instance třídy Habit. Dále se mění nadpis v aplikační liště na název návyku a nastavuje se funkce tlačítka FAB na otevření editace návyku. To je řešeno záměrem spouštějícím příslušnou aktivitu nebo nahrazením fragmentu, dle použitého uživatelského rozhraní.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

if (getActivity() instanceof HabitListActivity) {
        mTwoPane = true;
    }

    database = new DatabaseHelper(getContext());
    if (getArguments().containsKey(Habit.ARG_HABIT_ID)) {
        habit = database.getHabit(getArguments()
                .getInt(Habit.ARG_HABIT_ID));

        Activity activity = this.getActivity();
        CollapsingToolbarLayout appBarLayout = (CollapsingToolbarLayout)
                activity.findViewById(R.id.toolbar_layout);
        if (appBarLayout != null) {
            appBarLayout.setTitle(habit.getName());
        }

        FloatingActionButton fab = getActivity().findViewById(R.id.fab);
        fab.setImageResource(R.drawable.ic_edit);

        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mTwoPane) {
                    Bundle arguments = new Bundle();
                    arguments.putInt(Habit.ARG_HABIT_ID, habit.getId());
                    final EditHabitFragment fragment = new EditHabitFragment();
                    fragment.setArguments(arguments);
                    getActivity().getSupportFragmentManager().beginTransaction()
                            .replace(R.id.habit_fragment_container,
                                    fragment).commit();
                } else {
                    Intent intent = new Intent(getActivity(),
                            EditHabitActivity.class);
                    intent.putExtra(Habit.ARG_HABIT_ID, habit.getId());
                    startActivity(intent);
                }
            }
        });
    }
}
```

V metodě `onCreateView` se v případě, že byla zobrazena v předcházející obrazovce, skrývá klávesnice, která nemá v tomto fragmentu žádné využití. Do zobrazených `TextView` se vyplní následující údaje o návyku: jméno, popis, datum vytvoření a frekvence opakování. `TextView` pro zobrazení úspěšnosti plnění je pouze nalezeno a plní se pomocí metody `fillStatistics` volané adaptérem. Nakonec

se v `onCreateView` získá seznam všech denních záznamů pro daný návyk a za pomoci adaptéru se jím naplní `RecyclerView`.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_habit_detail, container, false);

    View view = getActivity().getCurrentFocus();
    if (view != null) {
        InputMethodManager imm = (InputMethodManager) getActivity()
                .getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
    }

    if (habit != null) {

        TextView textViewName = rootView.findViewById(R.id.tv_detail_heading);
        textViewName.setText(habit.getName());

        TextView textViewDescription = rootView.findViewById(R.id.tv_detail_description);
        textViewDescription.setText(habit.getDescription());

        TextView textViewCreated = rootView.findViewById(R.id.tv_detail_created_date);
        textViewCreated.setText(habit.getDateOfCreation());

        TextView textViewFrequency = rootView.findViewById(R.id.tv_detail_frequency);
        if (habit.getTimes() == habit.getInDays()) {
            textViewFrequency.setText(getText(R.string.habit_frquency_every_day));
        } else {
            textViewFrequency.setText(habit.getTimes() + " "
                    + getString(R.string.habit_frequency_times_in) + " "
                    + habit.getInDays() + " " + getString(R.string.habit_frequency_days));
        }

        textViewDone = rootView.findViewById(R.id.tv_detail_done);

        RecyclerView recyclerView = rootView.findViewById(R.id.habit_list);
        assert recyclerView != null;
        records = database.getDailyRecords(habit.getId());
        mAdapter = new HabitDetailRecyclerViewAdapter(getActivity(), records);
        recyclerView.setAdapter(mAdapter);
        recyclerView.addItemDecoration(new DividerItemDecoration(getActivity(),
                DividerItemDecoration.VERTICAL));
    }
    return rootView;
}
```

V metodě `onResume` se z databáze získá aktuální seznam návyků a nastaví do zobrazeného `RecyclerView`.

```
@Override
public void onResume() {
    super.onResume();
    records = database.getDailyRecords(habit.getId());
    mAdapter.reloadRecords(records);
}
```

`FillStatistics` vypočítá procentuální úspěšnost plnění návyku z následujících hodnot. `doneTrue` – počet dní, kdy byl návyk skutečně splněn, `doneByFrequency` – počet dní, kdy návyk skutečně splněn nebyl, ale byl splněn dostatečným počtem opakování (dle definované frekvence) v předchozích dnech a `doneFalse` – počet dní, kdy nebyl návyk splněn vůbec. Tyto hodnoty jsou počítány adaptérem a této metodě pouze předány jako argument. Po výpočtu je nastavena textová reprezentace úspěšnosti a obarvena na zeleno pokud je splněno nad 80%, oranžovo pokud nad 40% a pod touto hodnotu jsou procenta v červené barvě.

```
private void fillStatistic(int doneTrue, int doneFalse, int doneByFrequency,
                           TextView doneLabel) {
    int doneAll = doneTrue + doneByFrequency + doneFalse;
    int done = doneTrue + doneByFrequency;

    double percentage = (double) done / doneAll * 100;
    doneLabel.setText(new DecimalFormat("#.##").format(percentage)
            + "% (" + done + "/" + doneAll + ")");

    if (percentage > 80) {
        doneLabel.setTextColor(getResources().getColor(R.color.colorGreen));
    } else if (percentage > 40) {
        doneLabel.setTextColor(getResources().getColor(R.color.colorOrange));
    } else {
        doneLabel.setTextColor(getResources().getColor(R.color.colorRed));
    }
}
```

5 Zhodnocení

V této práci byla vytvořena aplikace pro OS Android s názvem Habit Tracker. Při jejím vývoji byl kladen důraz na vhodné využití obrazovky zařízení, na kterém bude aplikace spuštěna. Výsledkem jsou dvě odlišné varianty uživatelského rozhraní, přičemž tyto varianty sdílí co nejvíce zdrojového kódu a definice uživatelského rozhraní.

Uživatelské rozhraní aplikace je sestaveno z různých prvků typu ViewGroup, nabízejících odlišné přístupy k určování pozice vnořených prvků, jako například LinearLayout, FrameLayout či RelativeLayout. Jednu z nejdůležitějších částí aplikace tvoří seznam denních záznamů návyku, který je řešen pomocí RecyclerView. Mezi nejčastěji využívané prvky typu View patří: TextView, EditText, ImageView a FloatingActionButton

Aplikaci tvoří aktivity HabitListActivity, AddHabitActivity, EditHabitActivity a HabitDetailActivity. HabitListActivity obsahuje seznam návyků vytvářený pomocí třídy odvozené od RecyclerViewAdapter. Zbylé aktivity slouží pouze jako kontejner pro fragmenty, které nesou funkcionalitu. Těmi jsou: AddHabitFragment, umožňující přidání nového návyku, EditHabitFragment, sloužící k editaci a HabitDetailFragment, který zobrazuje podrobnosti a historii návyku. Pro práci s časem se používá třída LocalDate z knihovny Joda Time. Tato knihovna by mohla být v budoucnu odstraněna, jelikož její obdoba, která je součástí Java 8, je dostupná od API 26 (Android 8.0).

Aplikace využívá k ukládání dat databázi SQLite, se kterou se pracuje pomocí tříd SQLiteOpenHelper a SQLiteDatabase. Tento přístup vyžaduje definování příkazů v jazyce SQL. V případě dalšího rozšiřování aplikace by stálo za zvážení použití některé z ORM (Object-relational mapping) knihoven, které umožňují k databázi přistupovat objektově a psaní SQL tak není nutné.

6 Závěr

Teoretická část práce se zabývá operačním systémem Android, jeho architekturou a nástroji pro vývoj aplikací pro tuto platformu. Především pak oficiálním vývojovým prostředím, Android Studiem, a Android SDK, což je balík nástrojů a knihoven, které obsahují základní třídy tvořící aplikaci.

Tyto teoretické poznatky byly využity v praxi k vytvoření aplikace za použití zmíněného vývojového prostředí, programovacího jazyka Java, ve kterém byla naprogramována a značkovacího jazyka XML, který byl použit pro definici jejího grafického uživatelského rozhraní. To bylo navrženo s ohledem na zařízení s různou velikostí displeje. Pro ukládání dat je použita databáze SQLite, která je součástí OS Android. Aplikace nese název Habit tracker a je nástrojem pro vytváření a udržování návyků. Může se jednat například o návyk pravidelně cvičit nebo konzumovat méně alkoholu.

Funkcionalitu aplikace tvoří vytvoření návyků, u kterých lze určit vlastní frekvenci opakování či ponechat opakování každý den. Pro tyto návyky jsou pak automaticky, po zapnutí aplikace, vytvořeny záznamy s daty od vytvoření návyku až po současnost, pokud již neexistují. Na hlavní obrazovce jsou zobrazeny pouze záznamy pro dnešní den. Ty jsou prezentovány řádky seznamu, které obsahují název návyku a symbol představující stav splnění. Podržením symbolu lze tento stav měnit z hodnoty nesplněn na hodnotu splněn a naopak. Kliknutím na tento řádek je zobrazen detail návyku, obsahující informaci o celkové úspěšnosti jeho plnění a záznamy z předchozích dní, jejichž stav lze také měnit. Je tomu tak, aby bylo možné záznamy doplnit v případě, že uživatel aplikaci několik dní nepoužil nebo upravit, pokud v minulosti zadal chybný údaj. Všechny parametry návyku, kromě dat vytvoření, lze upravovat nebo je možné návyk smazat, včetně všech jeho záznamů.

Do budoucna by bylo vhodné umožnit návyk archivovat, aby se pro něj dále nevytvářely nové záznamy. Aktuálně je nutné pro dosažení takového stavu, aby uživatel návyk smazal, včetně celé jeho historie. Další možností rozšíření by pak mohlo být definování různých typů návyku. Například určené číslem či časem a jejich maximální

či minimální hodnotou. Bylo by tak možné vytvořit například návyky: „maximálně 2 sladkosti za týden“ nebo „alespoň 2 hodiny četby během tří dní“.

7 Seznam použitých zdrojů

1. U.S. Smartphone Penetration Surpassed 80 Percent in 2016. *ComScore.* [Online] 3. únor 2017. [Citace: 4. listopad 2018.] https://www.comscore.com/Insights/Blog/US-Smartphone-Penetration-Surpassed-80-Percent-in-2016.

2. Top 50 countries by smartphone users and penetration. *newzoo.* [Online] [Citace: 4. Listopad 2017.] https://newzoo.com/insights/rankings/top-50-countries-by-smartphone-penetration-and-users/.

3. Google I/O Keynote (Google I/O '17) | Google devloper. *YouTube.* [Online] 17. květen 2017. [Citace: 14. listopad 2017.] https://www.youtube.com/watch?v=Y2VF8tmLFHw.

4. Number of available applications in the Google Play Store from December 2009 to December 2017. *Statista.* [Online] [Citace: 4. Únor 2018.] https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

5. Smartphone OS Market Share, 2017 Q1. *IDC.* [Online] [Citace: 4. Únor 2018.] http://www.idc.com/promo/smartphone-market-share/os.

6. Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017. *Statista.* [Online] [Citace: 4. Únor 2018.] https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

7. Before it took over smartphones, Android was originally destined for cameras. *The Verge.* [Online] 16. Duben 2013. [Citace: 14. Listopad 2017.] https://www.theverge.com/2013/4/16/4230468/android-originally-designed-for-cameras-before-smartphones.

8. První telefon s Androidem zítra oslaví pět let! *Svět Androida.* [Online] 22. Září 2013. [Citace: 19. Listopad 2017.] https://www.svetandroida.cz/prvni-telefon-s-androidem-zitra-oslavi-pet-let-201309.

9. **LACKO, Ľ.** *Vývoj aplikací pro Android.* Brno : Computer Press, 2015. ISBN 978-80-251-4347-6.

10. Developer Guides. *Android Developers.* [Online] https://developer.android.com/guide/index.html.

11. **MEIER, R.** *Professional AndroidTM 2 Application Development.* Indianapolis : Wiley Publishing,Inc., 2010. ISBN: 978-0-470-56552-0.

12. Android Open Source Project. [Online] [Citace: 16. Prosinec 2017.] https://source.android.com.

13. **LACKO, Ľ.** *Mistrovství Android.* Brno : Computer Press, 2017. ISBN: 9788025148754.

14. **UJBÁNYAI, M.** *Programujeme pro Android.* Praha : Grada Publishing, a.s., 2012. ISBN: 978-80-247-3995-3.

15. Command Line Tools. *Android Developers.* [Online] [Citace: 16. Prosinec 2017.] https://developer.android.com/studio/command-line/index.html.

16. Android Debug Bridge (adb). *Android Studio.* [Online] [Citace: 16. Prosinec 2017.] https://developer.android.com/studio/command-line/adb.html.

17. Android Studio Release Notes. *Android Studio.* [Online] [Citace: 5. Leden 2018.] https://developer.android.com/studio/releases/index.html.

18. Android Developers Blog. [Online] 17. Květen 2017. [Citace: 5. Leden 2018.] https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html.

19. Comparison to Java Programming Language. *Kotlin.* [Online] [Citace: 10. Leden 2018.] https://kotlinlang.org/docs/reference/comparison-to-java.html.

20. NDK. *Android Developers.* [Online] [Citace: 10. Leden 2018.] https://developer.android.com/ndk/guides/index.html.

21. List of IDEs for Android App Development, Which is Best for You? *Tek Eye.* [Online] [Citace: 19. Leden 2018.] https://tekeye.uk/android/list-of-android-app-development-ides.

22. **SINICKI, A.** I want to develop Android Apps – What languages should I learn? *Android Authority.* [Online] 24. Srpen 2017. [Citace: 19. Leden 2018.] https://www.androidauthority.com/develop-android-apps-languages-learn-391008/.

23. Apache Cordova. [Online] [Citace: 19. Leden 2018.] https://cordova.apache.org/.

24. Introduction to Mobile Development. *Xamarin Developers.* [Online] [Citace: 19. Leden 2018.] https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/.

25. Documentation. *Apache Cordova.* [Online] [Citace: 13. Leden 2018.] https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

26. Visual Studio Tools for Xamarin. *Visual Studio.* [Online] [Citace: 19. Leden 2018.] https://www.visualstudio.com/xamarin/.

27. Projects Overview. *Android Developers.* [Online] [Citace: 4. Únor 2018.] https://developer.android.com/studio/projects/index.html.

28. **HORTON, J.** *Android Programming for Beginners.* Birmighmam : Packt Publishing Ltd., 2015. ISBN: 978-1-78588-326-2.

29. **DIMARZIO, J. F.** *Programujeme hry pro Android 4.* Brno : Compeuter Press a.s., 2012. ISBN:978-80-251-3754-3.

30. Material Design. [Online] [Citace: 21. Únor 2018.] https://material.io.

31. Alternative distribution options. *Android Developers.* [Online] [Citace: 12. Prosinec 2017.] https://developer.android.com/distribute/marketing-tools/alternative-distribution.html.

32. **HACHMAN, M.** Google Rebrands Android Market, Google Music as 'Play'. *PC Mag.* [Online] 6. březen 2012. [Citace: 4. Únor 2018.] https://www.pcmag.com/article2/0,2817,2401210,00.asp.

33. **LUDWIG, P.** *Konec prokrastinace.* místo neznámé : Jan Melvil Publishing, 2013. ISBN: 978-80-87270-51-6.

8 Seznam obrázků

9 Seznam použitých termínů

Smartphone - telefon s pokročilým operačním systémem.

Operační systém/OS – základní programové vybavení, umožňující ovládání daného zařízení.

Aplikace – softwarový program pro mobilní zařízení.

Software – programové vybavení.

Platforma – prostředí, ve kterém je software spouštěn. V kontextu této práce je tímto slovem míněn operační systém.

Programovací jazyk – jazyk pomocí, kterého lze zapisovat instrukce pro počítač a vytvářet tak programy.

Linux – operační systém s otevřeným kódem.

Hardware – Fyzické vybavení počítače/mobilního zařízení.

Vývojové prostředí/IDE – Program, ve kterém lze programovat.

Knihovna/library – soubor, obsahující sadu funkcí, který lze použít ve více programech.

Databáze – systém souborů obsahující data.

SQLite – databáze, která je součástí operačního systému Android.

Kompilace – proces, při kterém je programový kód převeden do spustitelné podoby.

Debugger – program pro nalézání chyb v jiných programech.

Emulátor – program umožňující simulaci jiného operačního systému.

Proces – spuštěná instance programu.

Instance – konkrétní objekt.

Log – textový soubor se záznamem činnosti.

Shell – textové uživatelské rozhraní pro Linuxové systémy.

Plugin – program, který nepracuje samostatně, ale jako rozšíření jiného programu.

Modul – část programu.

Backend – část programu, se kterou přímo neinteraguje uživatel.

Cloud – síť serverů, poskytujících nějakou službu. Např. úložiště.

Build – Kompilovaná verze programu.

Uživatelské rozhraní/UI – Prostředek, skrze který uživatel může ovládat stroj.

Grafické uživatelské rozhraní/GUI – uživatelské rozhraní tvořené grafickými prvky

Unit test – automatický testdílčí části programu (jednotky)

Back stack – zásobník uchovávající aktivity.

Interfejs – souhrn informací o objektu, kterým je specifikováno, co o něm ví a jak s ním může pracovat okolí.

Permission – povolení, které musí být aplikaci uděleno, aby měla přístup k určité funkci systému.

Layout – rozmístění prvků. V kontextu této práce může jít také o rodičovský prvek uživatelského rozhraní.

Widget – Může se jednat o část aplikace, kterou lze umístit na plochu systému, nebo jsou tímto termínem označeny jednotlivé grafické prvky, jako například tlačítko.

Entita – libovolný objekt reálného světa zachycený v datovém modelu.

Třída – konstrukční prvek v objektovém programování, definující vlastnosti a metody objektů vytvořených podle této třídy.

Metoda – část kódu, poskytující funkcionalitu.

Scrollovat – posouvat zobrazený obsah.