Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Mobilní aplikace pro podporu sportovního tréninku

Jan Hrubý

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jan Hrubý

Informatika

Název práce

Mobilní aplikace pro podporu sportovního tréninku

Název anglicky

Mobile application for sport training support

Cíle práce

Bakalářská práce se zabývá návrhem a vývojem mobilní aplikace pro vedení tréninkových plánů určené pro systém Android. Hlavním cílem je analyzovat slabá místa stávajících aplikací určených pro tyto účely a na jejich základě navrhnout a implementovat novou aplikaci, která bude tato slabá místa odstraňuje. Dílčím cílem bude popsat použité technologie a postupy.

Metodika

Metodika řešení této bakalářské práce je založena na analyticko-syntetickém přístupu. Bude provedena analýza funkčnosti stávajících aplikací určených k podpoře vedení sportovního tréninku. Na základě zjištěných poznatků bude provedena identifikace slabých míst těchto aplikací. Následně bude navržena aplikace, která bude sloužit k vedení tréninkových plánů a bude reflektovat analýzu slabin existujících aplikací.

Aplikace bude implementována pro mobilní zařízení s OS Android v jazyce Java s pomocí nástroje Android Studio. Po dokončení bude aplikace otestována a bude demonstrován postup jejího zveřejnění v systému Google Play.

Výsledná aplikace bude zhodnocena, zkušenosti z jejího návrhu a implementace budou shrnuty a budou nastíněny případná další možná budoucí rozšíření aplikace.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Java, Android, Android Studio, Mobilní aplikace, Tréninkový zápisník

---

Doporučené zdroje informací

ALLEN, G. *Android 4 : průvodce programováním mobilních aplikací.* Brno: Computer Press, 2013. ISBN 978-80-251-3782-6.

Android Developers [online]. [cit. 2016-04-01]. Dostupné z: http://developer.android.com/sdk/index.html

Java Help Center – Mobile and Embedded Java [online]. [cit. 2016-04-01]. Dostupné z: http://java.com/en/download/faq/index_mobile.xml

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

Oracle [online]. [cit. 2016-04-01]. Dostupné z: http://www.oracle.com/technetwork/java/index.html

SCHILDT, H. Java 7. Brno: Computer Press, 2012. ISBN 978-80-251-3748-2

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 01. 03. 2017

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Mobilní aplikace pro podporu sportovního tréninku“ jsem vypracoval(a) 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(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 7.3.2017 _____

Jan Hrubý

Poděkování

Rád(a) bych touto cestou poděkoval(a) panu Ing. Jiřímu Brožkovi, Ph.D. za odborné vedení a cenné rady, při pravidelných konzultacích. Rád bych také poděkoval pánům Ing. Michalu Stočesovi a Ing. Janu Masnerovi za jejich rady v rámci předmětu „Vývoj aplikací pro mobilní zařízení“. Dále bych chtěl poděkovat rodině za úlevu od domácích prací v době psaní této práce.

Mobilní aplikace pro podporu sportovního tréninku

Souhrn

Tato bakalářská práce se popisuje vývoj aplikace pro operační systém Android za pomoci Android Studia.

V teoretické části rozebírá platformu Android, její historii a architekturu. Dále je v této části popsáno použité vývojové prostředí, použitý programovací jazyk a základní komponenty aplikace. Například aktivity, fragmenty a jejich životní cyklus. Závěrem teoretické části je demonstrována publikace aplikace na Google Play.

Praktická část se z počátku soustředí na analýzu již existujících aplikací. Poznatky z této analýzy jsou promítnuty v následném návrhu databáze, struktury i uživatelského rozhraní aplikace, kterými se praktická část dále zaobírá. V návrzích jsou popsány jednotlivé třídy a wireframy. Po návrzích následuje část zaměřená na implementaci, kde jsou popsány vytvořené třídy, metody a použité ovládací prvky. Závěrem praktické části bylo provedeno testování aplikace.

Klíčová slova: Java, Android, Android Studio, Mobilní aplikace, Tréninkový zápisník

Mobile application for sport training support

Summary

This thesis describes the development of applications for the operating system Android, with the help of Android Studio.

The theoretical part discusses the Android platform, its history and architecture. Further on, this section describes the used development environment, the programming language and the basic components of the application. For example, activities, fragments and their life cycle. At the end of the theoretical section is an illustration of an application publication on Google Play.

At the beginning, the practical part is focused on the analysis of the existing applications. The knowledge earned during the analysis is reflected in the design of the database, the application structure and the user interface, which are the next points of the practical part. In the drafts are described the individual classes and wireframes. The drafts are followed by a section focused on implementation, which describes the created classes, methods and used controls. The end of the practical part consists of testing of the created application.

Keywords: Java, Android, Android Studio, Mobile application, Training notes

Obsah

Seznam obrázků

Odkazovaný seznam obrázků

Seznam ukázek zdrojového kódu

Odkazovaný seznam ukázek zdrojového kódu

1 Úvod

Každý sportující člověk, který se chce svému sportu věnovat trochu více, potřebuje mít nějaký plán. Plán tréninku. Je tedy obvyklé, že si lidé berou papíry s rozepsaným tréninkem, nebo mobilní zařízení s aplikací, kde takový trénink mají napsaný. Nejlepším příkladem pro sport, kde se toto využívá, je fitness. Každý sport má však svá rozdílná specifika, různé cviky, požadavky či intenzitu, na které dnešní aplikace, které podporují plánování tréninku, nejsou uzpůsobeny.

Aplikace, která je zde popsána, je použitelná pro každý sport. Sportovec má možnost přidat si vlastní cviky, specifické pro svůj sport, ze kterých pak může poskládat tréninkový plán, který může kdykoliv modifikovat podle svých potřeb.

2 Cíl práce a metodika

2.1 Cíl práce

Práce se zaměřuje na návrh mobilní aplikace pro podporu a vedení tréninkových plánů, se kterou si bude uživatel moci sepsat svůj tréninkový plán nezávisle na tom, jaký sport provozuje.

Dílčím cílem je provést analýzu slabých míst stávajících aplikací, na jejímž základě bude navržena a implementována nová aplikace, která tato slabá místa odstraňuje.

Hlavním cílem je popsat návrh logické struktury, návrh uživatelského rozhraní a následnou implementaci aplikace.

2.2 Metodika

Metodika řešení této bakalářské práce je založena na analyticko-syntetickém přístupu. Bude provedena analýza funkčnosti stávajících aplikací určených k podpoře vedení sportovního tréninku. Na základě zjištěných poznatků bude provedena identifikace slabých míst těchto aplikací. Následně bude navržena aplikace, která bude sloužit k vedení tréninkových plánů a bude reflektovat analýzu slabin existujících aplikací.

Aplikace bude implementována pro mobilní zařízení s OS Android v jazyce Java s pomocí nástroje Android Studio. Po dokončení bude aplikace otestována a bude demonstrován postup jejího zveřejnění v systému Google Play. Bakalářská práce se bude skládat z dvou částí:

První část bude zaměřena na teoretickou stránku práce. Bude zde představena platforma Android a důležité změny, které nastaly v určitých verzích. Dále zde bude popsána architektura Androidu a vývojové prostředí, ve kterém lze pro tuto platformu vyvíjet. Hlavní část této kapitoly se bude sestávat z detailnějšího popisu komponent aplikace, které jsou nezbytné pro tvorbu aplikací pro OS Android.

Druhá část je již zaměřena na praktickou stránku práce, kde jsou při implementaci použity postupy systémového inženýrství.

Výsledná aplikace bude zhodnocena, zkušenosti z jejího návrhu a implementace budou shrnuty a budou nastíněna případná další možná budoucí rozšíření aplikace.

3 Teoretická východiska

3.1 Platforma Android

Android je open-source platforma na bázi Linuxu, která dovoluje vytvářet hry a aplikace pomocí programovacího jazyka Java. Je primárně určená pro mobilní zařízení, tedy pro chytré telefony, tablety, navigace a fotoaparáty. Dokáže se přizpůsobit různým rozlišením displeje, proto se platforma Android poslední dobou hojně objevuje i v televizích, palubních deskách automobilů či v dalších zařízeních.

Největší výhodou je otevřenost platformy, tedy možnost úprav i ze strany uživatelů. Tato výhoda však jde ruku v ruce s jistou nevýhodou, kdy díky otevřenosti je na trhu spousta aplikací pochybné kvality. Mobilní zařízení i tablety na této platformě vyrábí velké množství firem (např. HTC, Samsung, SONY), což je zárukou dynamického vývoje nových zařízení, a tedy i aplikací. [1]

Obrázek 1: Logo Android (zdroj: https://upload.wikimedia.org/wikipedia/commons/thumb/d/d7/Android_robot.svg/2000px-Android_robot.svg.png)

3.2 Historie Androidu

Je známo, že Android je majetkem Googlu. Avšak nebylo tomu tak od začátku. Začátkem roku 2003 byla 4 společníky založena společnost Android Inc., kterou v polovině roku 2005 Google odkoupil jako začínající společnost, a udělal z ní svojí dceřinou firmu. [2][3]

3.3 Verze Androidu

V září 2008 byl v USA uveden na trh první telefon s operačním systémem **Android 1.0**. V únoru 2009 Google vydal update k prvotní verzi, který měl odstraňovat nalezené chyby v systému. Update nesl název **Android 1.1**. Koncem dubna 2009 byla vydána první verze se jménem **Android 1.5 Cupcake**. Počínaje touto verzí začal Google všechny verze pojmenovávat podle cukrovinek. Cupcake již uměla přehrávat videa. Poslední verzí první generace byl **Android 1.6 Donut**, který s sebou přinesl zásadní změnu - podporu různých rozlišení a poměrů stran obrazovky. [1][2][3]

Druhou generaci odstartovala ke konci října 2009 verze **Android 2.0/2.1 Eclair**, která přinesla podstatné změny v designu uživatelského prostředí, podporu standardů HTML 5 a také hardwarová vylepšení. Další verze **Android 2.2 Froyo**, která přišla na svět koncem května 2010, znovu oživila uživatelské rozhraní, ale také přinesla webový prohlížeč s optimalizací JavaScriptu, podporou Flash 10 a možností instalovat aplikace na paměťovou kartu. Také vznikla řada zařízení Nexus – zařízení, na kterých je od výroby pouze čistý systém podporovaný přímo Googlem. 6. prosince 2010 byla vydána poslední verze pro druhou generaci a to **Android 2.3 Gingerbread**. Tato verze měla vylepšenou správu napájení, komunikaci přes NFC a pár dalších menších změn. [1][3]

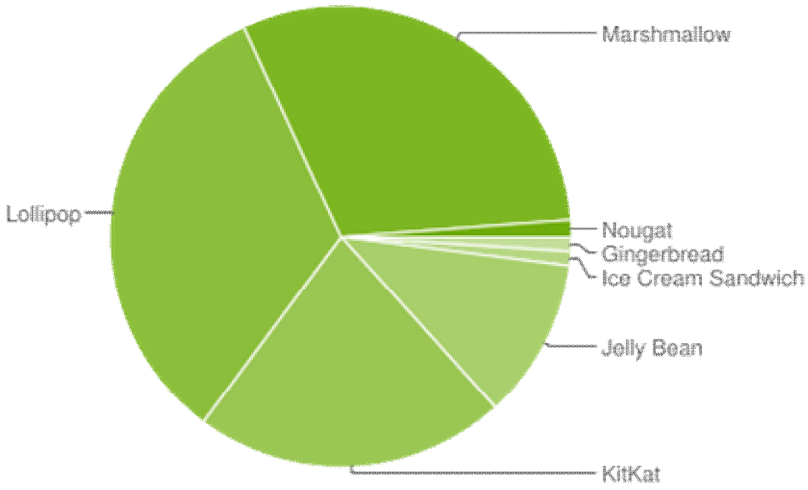
Třetí generace **Android 3.0 Honeycomb**, na trhu od 22 února 2011, se moc neosvědčila, šlo spíše o slepou vývojovou větev. Šlo o systém, který byl pouze pro tablety, ten s sebou však přinesl i komponentu Action Bar a hardwarovou akceleraci. [1][3]

Čtvrtá generace se primárně ubírala cestou zlepšení uživatelského rozhraní. První verze této generace **Android 4.0 Ice Cream Sandwich** byla uvedena v polovině října 2011. Tato verze odstraňuje rozdíly mezi verzemi pro tablety a mobilní telefony. Zásadní změna vzhledu, vznik vzhledu „Holo“. Mezi další novinky patřily dále aplikace na obrazovce uzamčení, nedávno spuštěné aplikace, nebo možnost vytváření adresářů. Dále 9. července 2012 přišly na svět verze **Android 4.1/4.2/4.3 Jelly Bean**. Tyto verze již nebyly tolik přelomové jako jejich předchůdce Ice Cream Sandwich, ale vylepšily např. aktualizace aplikací, vylepšení notifikací, možnost přepínání uživatelských účtů nebo vznik Chytrých karet Google. Po roční odmlce byla vydána verze **Android 4.4 KitKat**. Největší změnu v této verzi zaznamenala vizuální stránka systému, ve které se odehrálo nejvíce změn. [1][2][3]

Pátá generace, která byla vypuštěna na trh v listopadu roku 2014, nesla název **Android 5.0/5.1 Lollipop**. Tato verze s porovnáním předešlých verzí přinesla asi největší změnu v designu. Přišla se standardem, který se stal alfou a omegou, a nesl název „Material Design“. Dále velká změna této verze je nahrazení virtuálního stoje Dalvik, novým ART (Android Runtime). [1][2][3]

Ve druhé polovině roku 2015 byla na trh uvedena šestá generace **Android 6.0 Marshmallow**. V této verzi nešlo už tolik o redesign jako u předešlých dvou generací, ale zaměřuje se spíše na vylepšení systému. Mezi tyto vylepšení patří např. možnost nastavení oprávnění k aplikacím, správce operační paměti, nový režim úspory baterie (režim spánku) anebo podpora snímače otisků prstů. [3]

V polovině roku 2016 vydal Google nejnovější verzi Android 7.0 Nougat. Nejvýraznější změnou je propracovanější lišta s rychlými nastaveními a notifikace. Novinkou této verze je víceoknová podpora (multi-window support). Tato novinka dovoluje uživateli mít otevřené dvě aplikace najednou. [3]

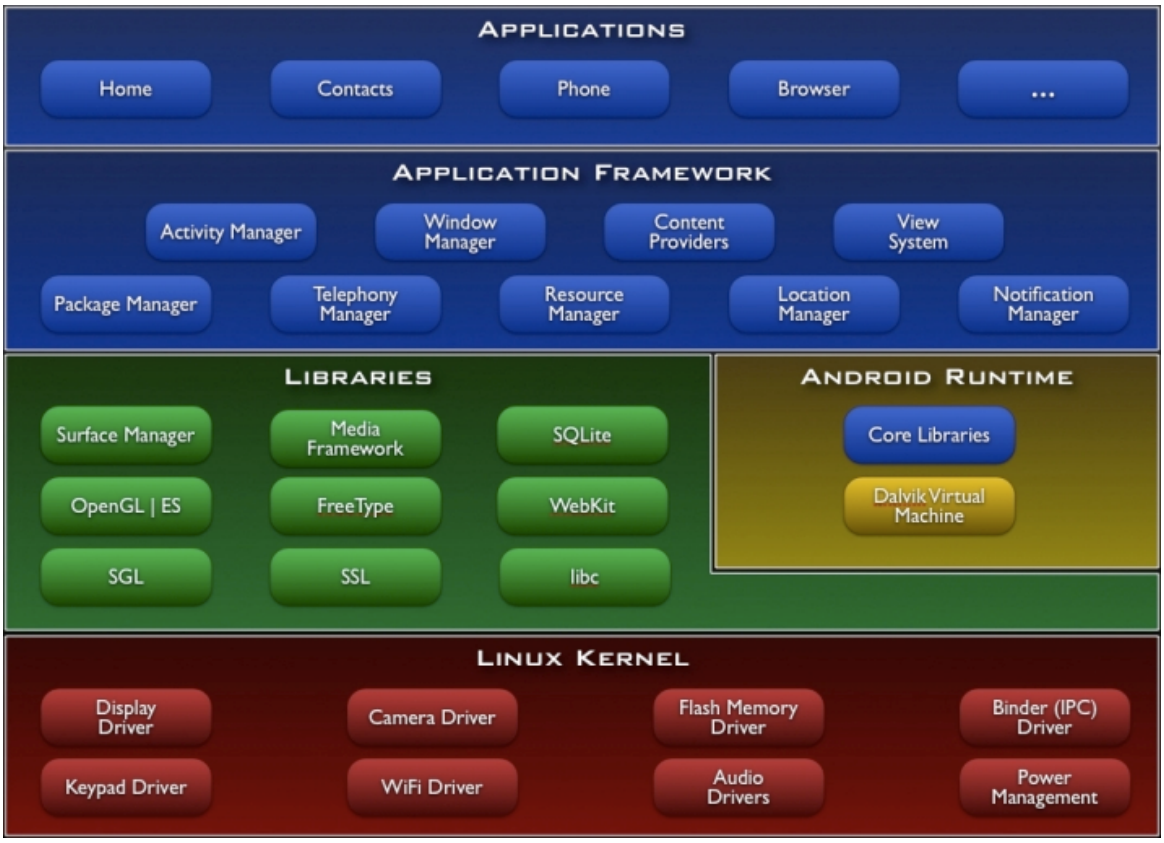


Obrázek 2: Podíl verzí androidu (aktualizováno k datu 6.února 2017) (zdroj: https://developer.android.com/about/dashboards/index.html)

3.4 Architektura

K vývoji aplikací je potřeba znát základní informace o principech operačního systému, pro který bude aplikace vyvíjena. OS Android se skládá z pěti základních vrstev. Nejnižší vrstvou je **Linuxové jádro**, které komunikuje přímo s hardwarem mobilního

zařízení. Další vrstvou jsou **Libraries**. Jedná se o knihovny, které aplikacím poskytují přímý přístup k jednotlivým komponentám Androidu. Tvoří takzvanou mezivrstvu mezi linuxovým jádrem a komponentami vyšších vrstev. Další vrstva **Android Runtime** obsahuje sadu základních knihoven s virtuálním strojem, sloužících ke spouštění zkompilovaných Java souborů. Čtvrtou vrstvou je **Application Framework**, která v aplikacích obsahuje opakovaně použitelný software, např. ikony nebo ovládací prvky. Je to nejdůležitější vrstva pro vývojáře aplikací, jelikož poskytuje aplikacím základní služby systému, se kterými mohou pracovat. Poslední a také nejvyšší vrstvou v architektuře Androidu jsou samotné **Applications**, například kalendář, email, seznamy kontaktů nebo hry. [1]



Obrázek 3:Schéma architektury OS Android (zdroj: http://androidmarket.cz/android/jak-vypada-android-uvnitr-aneb-co-je-rom-kernel-bootloader-a-dalsi/)

3.4.1 Linux Kernel (Linuxové jádro)

Linuxové jádro je základní a také nejnižší vrstvou architektury Androidu. Hlavní funkcí Kernelu je zajišťovat komunikaci mezi hardwarem a softwarem. K tomu slouží drivery (ovladače). Dále například zajišťuje správu procesů, paměti, napájení nebo zajišťuje síťové spojení. Správa a řízení procesů zajišťuje současný běh několika procesů, bez vzájemného ovlivňování. [1][4]

3.4.2 Libraries (Knihovny)

Knihovny jsou vrstva nad jádrem, které poskytují přístup k různým komponentám systému. Nativní knihovny jsou napsané v C/C++. Slouží pro práci s grafikou, videem, zvukem, podporu multidotykového displeje, vykreslování webových stránek, práci s daty nebo také šifrovaný přenos dat. [1][4]

3.4.3 Android Runtime

Vrstva, která obsahuje sadu základních Java knihoven a slouží k běhu aplikací. Aplikace v Androidu jsou psány v Javě, tudíž musí mít virtuální stroj, který kód převede do nativního kódu. O převod se stará Dalvik Virtual Machine. Dalvik umožňuje spouštět soubory s příponou DEX. DEX soubory vznikají kompilací ze souboru JAR a CLASS. Současně může být spuštěno více instancí virtuálního stroje. [1][4]

3.4.4 Application Framework

Nejdůležitější vrstva pro Android vývojáře. Obsahují další knihovny, tentokrát psané v Javě, které tvoří systémové API (aplikační rozhraní). API je soubor funkcí, které umožňují programátorovi pracovat s prvky operačního systému. Jedná se např. o přístup ke grafickým prvkům (tlačítka atd.), kontaktům, možnost pracovat s notifikacemi či spravovat životní cykly aplikací. [1][4]

3.4.5 Applications

Nejvyšší úroveň celé architektury operačního systému Android. Na této úrovni se již nacházejí samotné aplikace (např. Kalendář, Zprávy, Mail), které uživatelé mobilních zařízení používají. [1][4]

3.5 Vývojová prostředí

Pro vývoj aplikací na platformě Android existuje několik vývojových prostředí. Mezi nejznámější prostředí pro vývoj v Javě patří **Android Studio**, **Eclipse** a **App Inventor**. Pro vyvíjení v C# je zde od Microsoftu **Xamarin.**

Pro tuto práci bylo zvoleno Android Studio. Jedná se o vývojové prostředí, vyvíjené společnostmi Google a JetBrains. Přestože produkty JetBrains jsou placené, Android Studio je zcela zdarma. Je postaveno nad Community verzí prostředí IntelliJ IDEA, díky čemuž získává stejné možnosti práce s kódem (např. navigace v kódu, refractoring, našeptávání), které značně usnadňují práci. Design se v Android Studiu může navrhovat v návrhovém módu nebo přímo v XML. V návrhovém módu má k dispozici rozsáhlé a responzivní IDE. [5]

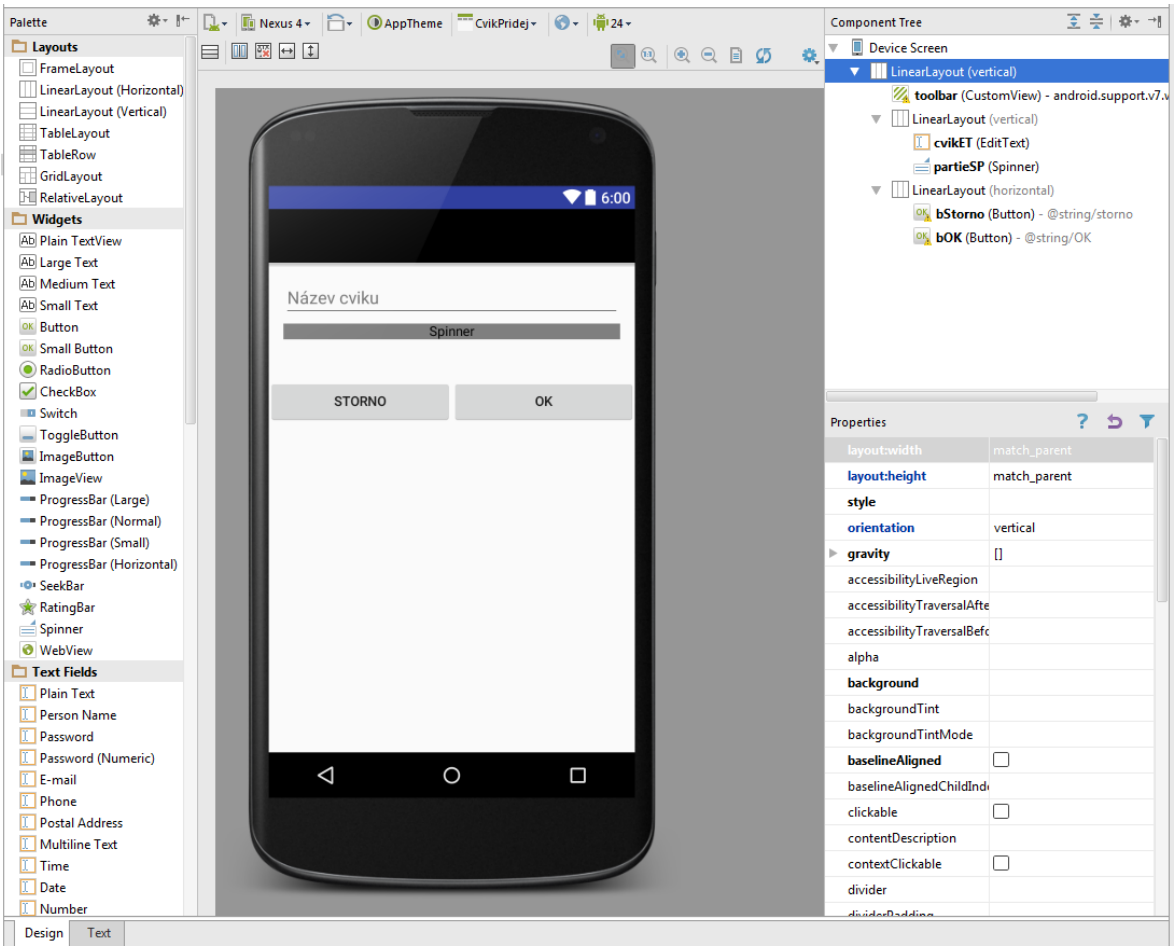
Android Studio dále obsahuje buildovací nástroj Gradle, který je popsán v kapitole **3.6.5 Gradle - build**. Studio také obsahuje emulátory, které jsou volně konfigurovatelné (Android API, rozlišení, velikost displeje, paměť atd.).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context="com.hrubos.treninkovy_planovac.CvikPridej">

   <android.support.v7.widget.Toolbar
      android:id="@+id/toolbar"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:background="?attr/colorPrimary"
      android:elevation="6dp"
      android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
      app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>

</LinearLayout>
```

Zdrojový kód 1: Android Studio ukázka XML



Obrázek 4: Android Studio návrhový mód (zdroj: vlastní)

3.6 Programovací jazyk

K vývoji aplikace byl použit programovací jazyk Java - jeden z nejpopulárnějších a nejpoužívanějších objektově orientovaných jazyků. Jeho syntaxe je velmi podobná jazykům C nebo C++. Java je interpretovaný jazyk. Program se nepřekládá přímo do strojového kódu, ale do Java bajtkódu. Bajtkód je následně interpretován virtuálním strojem Javy (Java Virtual Machine - JVM). Díky tomuto procesu se Java stává na platformě nezávislým jazykem, jelikož je bajtkód nezávislý na architektuře počítače či zařízení. O správu paměti se stará automatický garbage collector, který zabezpečuje odstranění již nepoužívaných objektů z paměti. [1]

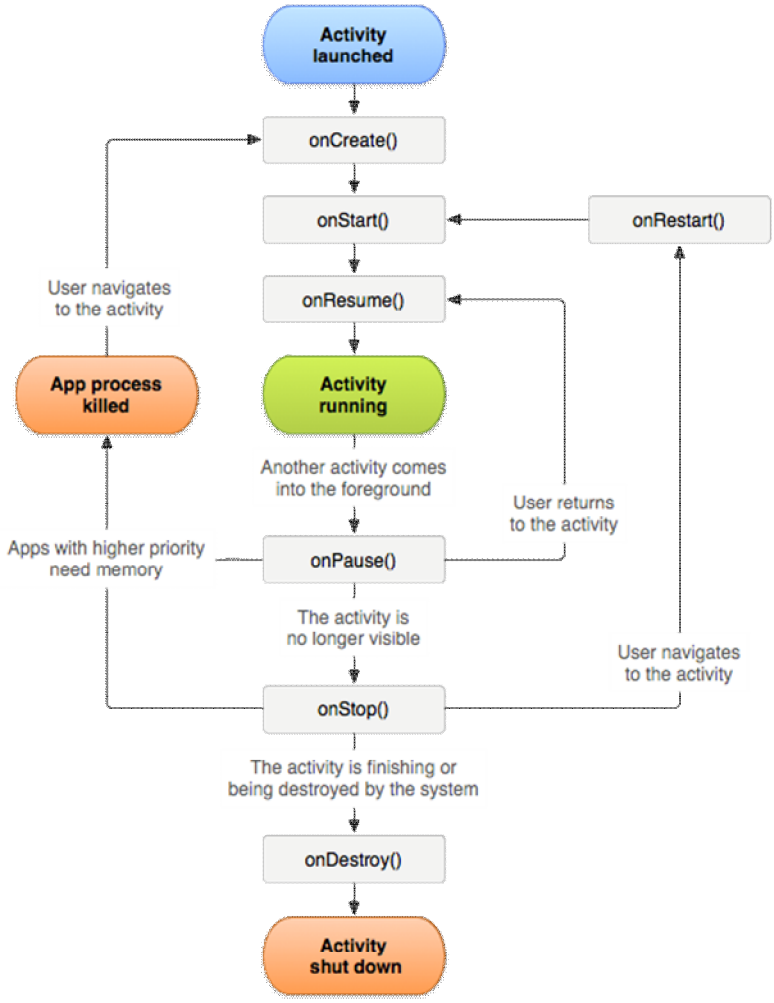
3.7 Základní komponenty aplikace

3.7.1 Aktivita a její životní cyklus

Aktivita je základním prvkem aplikace. Je to hlavní třída, která se po spuštění aplikace zobrazí uživateli. Aktivit může být v aplikaci zcela libovolný počet. Aktivity umožňují uživatelům interakci s aplikací. Přes grafické rozhraní (GUI) mohou uživatelé přijímat informace od aplikace a zároveň jí ovládat (Button, ListView atd.). Každá aktivita by měla zastávat určitou úlohu. Vyplnění formuláře, seznam výsledků, menu s výběrem a podobné. [1][6][7]

Android aplikace nemá žádný klasický vstupní bod, jakým u většiny ostatních aplikací bývá statická metoda *main()*, která se po spuštění aplikace spustí a vytvoří všechny potřebné instance. Takto Android nefunguje. Android sám vytváří instance (nejen) aktivit, stará se o jejich zničení, i co v průběhu života mají dělat. To zajišťuje pomocí životního cyklu aktivit. Každá aktivita má 7 metod životního cyklu, s nimiž může dosáhnout těchto stavů: [6][8]

- **Běžící aktivita na popředí** – Úspěšně spuštěná aktivita běžící na popředí, viditelná pro uživatele, který s aktivitou pracuje. Aktivita dostává informace o uživatelském vstupu.
- **Pozastavená aktivita** – Pozastavená aktivita je vidět, ale je částečně překrytá jinou aktivitou (upozornění na hovor, dialog o plném nabití baterie nebo v případě této aplikace aktivitou pro přidání nového cviku/tréninku). Takto pozastavená aktivita nedostane žádný uživatelský vstup. Uživatel se k ní nedostane. Aktivita může být obnovena do běžícího stavu, anebo v případě kritického nedostatku místa může být aktivita ukončena.
- **Zastavena** – Aktivita je zastavená v okamžiku, kdy není vidět, ale ještě není zničen její objekt. Má stále dostatek místa v paměti, a systém předpokládá, že by se k aktivitě chtěl uživatel ještě vrátit. Nepřijímá žádné uživatelské vstupy. [7][8]



Obrázek 5: Životní cyklus aktivit (zdroj: https://developer.android.com/reference/android/app/Activity.html)

Průběh života aplikace řídí metody životního cyklu. Tyto metody mají přesně dané pořadí, v jakém budou spouštěny. Také mají danou situaci, ve které se budou spouštět. K dosažení stavů životního cyklu aktivity v určitý moment, je nutné přepsat jednu nebo více metod životního cyklu. Tyto metody jsou vzájemně propojené. [1][7]

onCreate()

Poprvé je metoda *onCreate()* aktivována při spuštění aktivity. Přijímá parametr savedInstanceState, objektu Bundle. Objekt Bundle obsahuje uložený stav této aktivity, pokud byla v minulosti již spuštěna. Pokud nebyla, hodnota Bundle je null. V těle metody se na pozadí vytváří uživatelské rozhraní pomocí vytvoření ContentView. Vytvoří se instance objektů a inicializují potřebné proměnné. Metoda *onCreate()* se spouští ještě

za dalších okolností. Pokud již byla aktivita spuštěna a následně pozastavena nebo ukončena, a uživatel se k ní znovu vrátil. Aktivita je spuštěná, ale systém si vynutil změnu zdrojů (otočení displeje, změna jazyku UI a podobné). Po zavolání této metody je aktivita stále neviditelná, zastavená a nepřijímá uživatelské vstupy. [1][6][7]

onStart()

Metoda *onStart()* je volána ihned po *onCreate()*. Po zavolání *onStart()* se provedou všechny úkony, potřebné k zobrazení aktivity. Například spustí vlákno pro pravidelné stahování dat internetu do aplikace. Následuje volání metody *onResume()* pro přenesení aktivity na popředí. [1][6][7]

onResume()

Po zavolání metody *onResume()* se aktivita přesune do popředí, a stane se viditelná pro uživatele. Začne přijímat uživatelské vstupy. Metoda *onResume()* se volá vždy, když aplikace přechází z pozadí do popředí. Aplikace zůstává v tomto stavu do té doby, dokud uživatel nepřenese svou pozornost na jinou aktivitu, nebo neukončí aplikaci. Po této metodě vždy přichází metoda *onPause(),* která dále rozhoduje, v jakém stavu se bude aktivita nacházet. [1][6]

onPause()

Metoda *onPause()* se volá při přechodu aktivity na pozadí aplikace. Tento okamžik nastává při spuštění nové aktivity. V této metodě je vhodné automaticky ukládat údaje, které byly změněny při práci s aktivitou. Jedná se například o změny v souborech, dokumentech nebo v databázi. Tato metoda je používána také při potřebě uvolnění systémových zdrojů nebo jiných zdrojů, které mají negativní vliv na baterii zařízení. Obvykle po pozastavení aktivity znovu nastupují *onResume()* nebo *onCreate()*. Třetí možností je zavolání metody *onStop()*. [1][6][7]

onStop()

Metoda *onStop()* je volána při potřebě zastavení a zneviditelnění aktivity. Nikdy není volána kvůli nedostatku paměti. Při tomto volání se zruší vše, co bylo inicializování při *onStart()*. Pro budoucí obnovení zůstávají inicializované komponenty z *onCreate(),*

které si aktivita zanechala uložené, stejně tak jako stav View objektů v layoutech. Po zastavení aktivity je možné ji znovuobnovit pomocí *onRestart(),* zničit a vše znovu inicializovat pomocí *onCreate(),* nebo zavolat metodu *onDestroy()*. [1][6][7]

onRestart()

Metoda *onRestart()* znovuobnoví zastavenou aktivitu. Volá *onStart()* a po ní následně *onResume()*. Aktivita se tímto postupem znovu dostane do popředí aplikace. [6]

onDestroy()

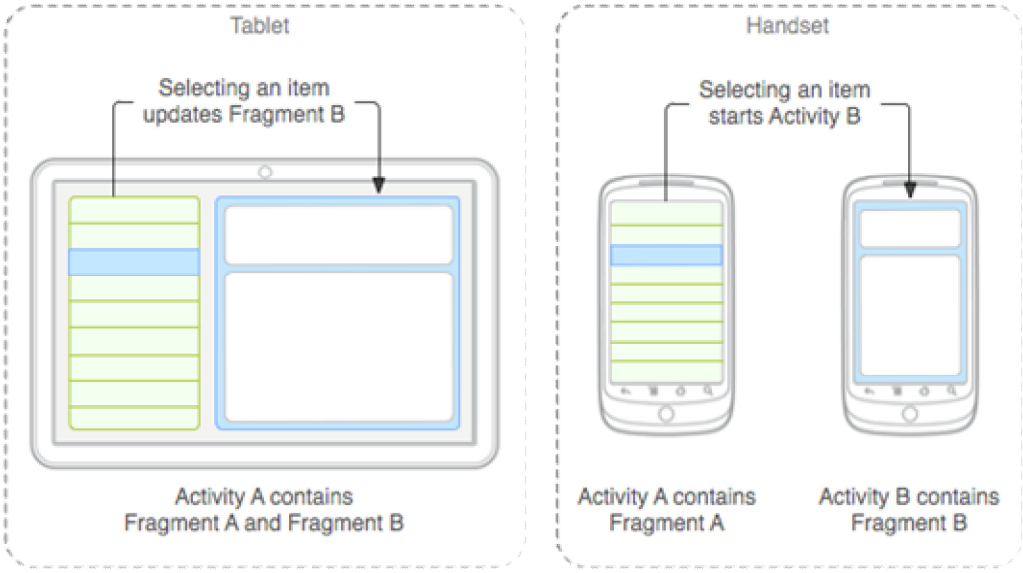
Metoda volána těsně před zničením aktivity. Jedná se o poslední volání, které aktivita obdrží. *onDestroy()* může být zavolána metodou *finish(),* nebo je zničena frameworkem Androidu dočasně. V této metodě je potřeba uklidit vše, co nebylo uklizeno v *onStop(),* tedy to, co zůstalo po *onCreate(),* například běžící vlákna. [1][6]

3.7.2 Fragment a jeho životní cyklus

Fragmenty jsou nedílnou součástí Androidu. Byly přidány ve verzi 3.0 pro podporu uživatelského rozhraní u zařízení s velkými displeji. Jedná se o komponenty, které běží v aktivitě. Fragment musí být vždy součástí aktivity. V rámci jedné aktivity může být kombinováno několik fragmentů. Fragment má svůj trochu odlišný životní cyklus oproti aktivitě, ale vždy je podřízen stavům své aktivity, ve které se nachází. Fragment může být přímo vložen do layoutu aktivity, avšak není to nutnost. Každý fragment může mít vlastní layout, který bude definovat jeho uživatelské rozhraní. Kód aplikace může být díky fragmentům zapouzdřen a použit vícekrát. Převážně se využívá pro zobrazování obsahu s různými velikostmi displeje. Zejména se jedná o rozdíly ve velikosti displeje u tabletu a mobilního zařízení. [1][9][10]

Dobrým příkladem pro použití fragmentů je zobrazování informací typu master-detail. Seznam objektů bude vypsán v jednom fragmentu, a detailnější informace k vybranému objektu se zobrazí v druhém. U malých displejů (jedná o většinu mobilních zařízení) bude každý fragment zobrazen zvlášť ve vlastní aktivitě. U velkých displejů (tabletů) bude pro stejné zobrazení stačit, když budou oba fragmenty zobrazeny najednou vedle sebe, v rámci jedné aktivity. [1][9][10]

Fragmenty mohou být do aktivity přidány dvěma způsoby. Staticky a programově. Statický způsob je vložení fragmentu do layoutu aktivity. Programový způsob přidání fragmentu umožňuje pomocí objektu FragmentManager. [1]



Obrázek 6: Využití fragmentů (zdroj: https://developer.android.com/guide/components/fragments.html)

Životní cyklus fragmentu je do značné míry ovlivněn životním cyklem aktivity. Díky závislosti na stavech aktivity, se při pozastavení aktivity, v každém jejím fragmentu aktivuje metoda *onPause()*. Díky této závislosti životní cyklus fragmentu disponuje stejnými metodami jako aktivita. Přesto fragment obsahuje pár dalších metod svého životního cyklu.

onAttach()

Metoda *onAttach()* se se volá při spojení fragmentu s aktivitou, která je mu předána jako argument. Je volána ještě před zavoláním metody *onCreate(),* ve které se stejně jako u aktivity inicializují proměnné. [1]

onCreateView()

Po *onCreate()* se volá metoda *onCreateView(),* která v případě, že fragment obsahuje uživatelské rozhraní, vrací View, které bude následně fragment v uživatelském rozhraní reprezentovat. V opačném případě (neobsahuje UI) vrací null. [1][9]

onActivityCreated()

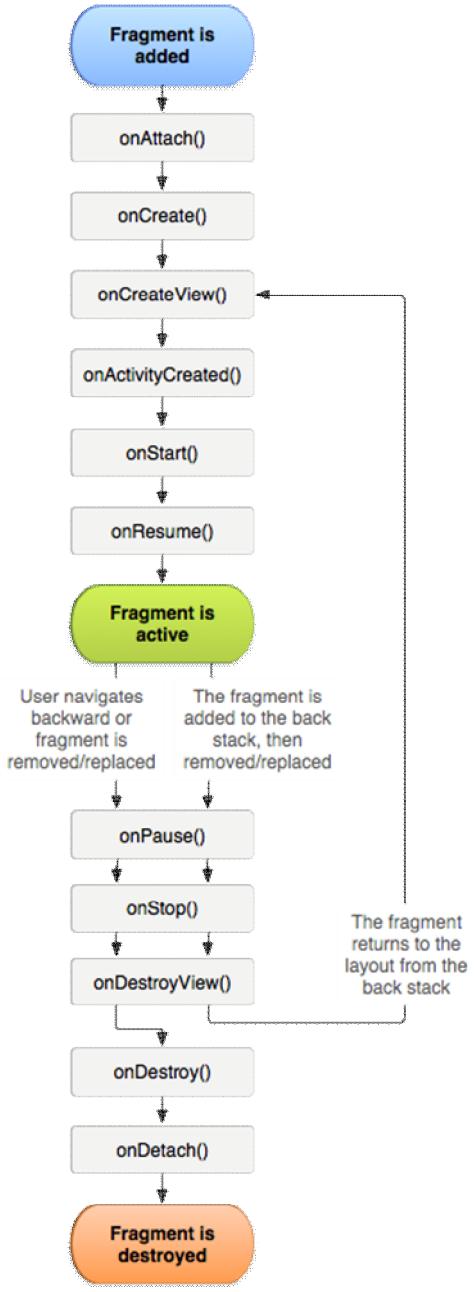
Metoda je volána po *onCreateView()* a po skončení metody *onCreate()* hostitelské aktivity, tedy po vytvoření této aktivity. Jestliže byl fragment přidán do aktivity programově, aktivita již byla dávno pomocí *onCreate()* vytvořena, volá se *onActivityCreated()* bezprostředně po skončení *onCreateView()*. [1][7]

onDestroyView()

Tato metoda je volána po metodě *onStop(),* při odstraňování fragmentu z uživatelského rozhraní. Metoda *onDestroyView()* je používána pro úklid všeho, co bylo v *onCreateView()* vytvořeno. [1][7]

onDetach()

Metoda *onDetach()* je volána pro zrušení spojení mezi fragmentem a aktivitou. Je tedy opakem metody *onAttach()*. Volá se po zničení, tedy po metodě *onDestroy()*. [1][7]



Obrázek 7: Fragment životní cyklus (zdroj: https://developer.android.com/guide/components/fragments.html)

3.7.3 Intent

Jedná se o jednoduchý, avšak důležitý objekt, který definuje vstupy a výstupy aktivit. Intent v překladu znamená záměr, a je také odvozen od slova intention, což znamená úmysl. Jedná se tedy o asynchronní zprávu, přenášející požadovanou informaci.

Umožňuje vzájemnou komunikaci mezi vázanými komponentami aplikace. Intent systému v podstatě oznamuje co má běžící aktivita v úmyslu udělat. Intent se dá použít pro mnoho akcí. Jednou z nejpoužívanějších je spuštění jiné aktivity, které předá parametr pomocí metody *putExtra()*. Intent se rozdělují na dva typy, podle upřesnění způsobu provedení. [1][11][12]

Explicitní Intent přímo ví, s jakou třídou má pracovat a co přesně má systém spustit. První předávaný parametr je instance aktuální třídy/komponenty. Druhý parametr definuje třídu, která se má po odeslání Intentu spustit. Na druhém řádku Intent spouští metodu, která předává hodnotu mezi komponentami. Předávané hodnoty jsou mapovány pomocí textových řetězců. Třetí řádek obsahuje metodu, která spouští aktivitu s požadovaným záměrem. [1][11][12]

```
Intent intent = new Intent(getActivity(), DetailActivity.class);
i.putExtra("_vybranyTreninkID",id);
startActivity(i);
```

Zdrojový kód 2: Explicitní intent

Implicitní Intent umožňuje napsat pouze záměr, co by měl systém udělat, avšak ne přesný způsob, jak určitý Intent provést. Musí však obsahovat dostatek informací, aby systém mohl rozhodnout, kterou z dostupných aktivit může použít. V mnoha případech použití implicitního Intentu výrazně zjednoduší práci. Pro praktické provedení systém vyhledá všechny aktivity, které jsou schopny Intent provést a umožní uživateli v dialogu vybrat jednu z aktivit. Vybrané aktivitě je následně Intent předán jako vstup. [1][11][12]

3.7.4 View

Třída View představuje základní stavební kámen pro tvorbu ovládacích prvků uživatelského rozhraní. View (pohled) na obrazovce zabírá obdélníkovou plochu, vykresluje a zpracovává události. Implementuje základní metody a atributy (technické a vzhledové). Technické, které předepisují, jak se dané View vykreslí. Vzhledové manipulují s daným View, např. měněním atributů. [13][14]

Ovládací prvky jako jsou Button, TextView, EditText, ImageView, Spinner, ProgressBar či různé layouty, jsou odvozené od třídy View. Jsou schopné zobrazovat text, obrázky, jiný obsah nebo se chovají jako ovládací prvky. Všechny View v okně jsou

uspořádány v jedné úrovni stromové struktury a mohou být přidány pomocí XML v layoutu nebo pomocí kódu. [13][14]

Po vytvoření stromové struktury je možné provést nad vizuálními prvky běžné operace. Nastavení vlastností, proměnných, či nastavení textu článku v TextView. Dále lze nastavit focus (zaměření) nad určitým prvkem použitým v daném View. Další z velice používaných operací je nastavení listenerů (posluchačů). Listenery se nastavují na podtřídy třídy View, a zachytávají události, které se na určitém prvku stanou. Například při stisknutí tlačítka se pomocí listeneru v kódu tento okamžik odchytí a provede kód či metodu, kterou vývojář specifikoval pro tuto událost. Poslední z nejpoužívanějších operací je nastavení viditelnosti daného View. [14][15]

Jednou z nejdůležitějších podtříd třídy View, je ViewGroup. Jedná se o skupinu pohledů, která může obsahovat další View nebo ViewGroup. Třída umožňuje tvorbu libovolně uspořádané hierarchie prvků uživatelského rozhraní. Jedná se o základní třídu pro layouty, které rozmisťují své potomky. [15][16]

3.7.5 Android manifest

Jedná se o hlavní konfigurační soubor aplikace. Definuje jednotlivé komponenty, nastavuje konfiguraci a oprávnění aplikace. Každá aktivita musí mít v manifestu definované své jméno, případně může definovat svůj popis nebo rodičovskou aktivitu. [1]

Pro konfiguraci aplikace se nastavuje použitý package name (název balíku aplikace) jako unikátní identifikátor. Dále například použitá ikona, téma, název aplikace nebo jaká aktivita se spustí při spuštění aplikace. Každá aplikace musí mít v Manifestu vypsaný seznam všech oprávnění, které potřebuje k činnosti. Mezi ně například patří přístup k internetu, možnost volání či odesílání zpráv. Oprávnění je užitečný nástroj, díky němuž má uživatel přehled o tom, co jaká aplikace může dělat. Při instalaci vždy musí souhlasit s oprávněními instalované aplikace. Jednotlivá nastavení najdeme v souboru AndroidManifest.xml. [16]

3.7.6 Android Support Library

Jedná se o knihovny, díky kterým lze dosáhnou tzv. dopředné kompatibility. Díky těmto knihovnám lze vyvíjet aplikace na nejnovější verzi API a zároveň - podle použitých knihoven - může být výsledná aplikace spustitelná na mnohem starších zařízeních se starším API. Každá knihovna začíná písmenem „v“ a číslem označující verzi knihovny, vypovídající o tom, od kterého API je v aplikaci možné využívat příslušnou knihovnu. Například knihovna „v4 support library“.

3.8 Publikace Aplikace

Pro publikování aplikace je potřeba vybrat si službu, přes kterou se bude aplikace nabízet a šířit. Služeb pro Android existuje několik, např. Google Play, Amazon Appstore, F-Droid anebo GetJar. Pro demonstrování publikace vytvořené aplikace byl vybrán nejnavštěvovanější obchod s Android aplikacemi, a to Google Play.[1]

Pro publikování aplikace je prvním krokem zaregistrování vývojářského účtu. Za registraci je potřeba zaplatit 25$. [1]

Před samotnou publikací je potřeba vytvořit podepsaný soubor APK a certifikát. Ty se v Android Studiu vytvoří pomocí záložky v menu **Build → Generate Signed APK.** Soubor .apk je již možné nahrát na Google Play, nesmí však přesáhnout velikost 50 MB. Pomocí souborů rozšíření lze dále nahrát další soubory, obsahující např. grafiku či rozšíření aplikace. Je důležité vytvořený certifikát a heslo k němu neztratit, jelikož nahranou aplikaci by již nebylo možné aktualizovat pod jiným certifikátem. [1]

Po nahrání je dále potřeba vyplnit údaje o aplikaci v záložce **Záznam o obchodu**. Jedná se o údaje jako je jazyk, název aplikace, její krátký a úplný popis, typ (aplikace nebo hra), kategorie a obrázky, které se budou zobrazovat potencionálnímu uživateli aplikace. [1]

Další záložkou, nezbytnou pro úspěšnou publikaci, je **Cena a distribuce**. Zde se uvádí, zda je aplikace poskytována za poplatek, nebo zdarma. Pokud bude zvolen způsob prodeje aplikace za poplatek, Google si ze stanovené ceny vezme vždy 30%. [1]

Po nahrání aplikace trvá několik dní, než bude aplikace schválena a bude ke stažení na Google Play. [1]

4 Vlastní práce

K vytvoření aplikace bylo zvoleno vývojově prostředí Android Studio. Vlastní práce se nejdříve věnuje analýze již existujících aplikací, návrhu struktury aplikace, uživatelského rozhraní a databáze. Po navržení struktury se zabývá následnou implementací.

U návrhu struktury aplikace je hlavní zaměření na jednotlivé třídy, které jsou zde pro přesnější popsání označeny dále jako aktivity, fragmenty a adaptéry. Při návrhu tříd je u každé popsaná činnost, jakou by měly jednotlivě zastávat. Po návrhu následuje popsání způsobu implementace těchto tříd. Blíže specifikované jsou zde způsoby přidávání, editování i mazání záznamů, způsob implementace komponent jako je ListAdapter i vzájemná spojitost fragmentů a aktivit.

Uživatelské rozhraní je nejdříve navrženo podle toho, jaké akce má provádět, jak má vypadat a také jak bude fungovat menu navigace mezi jednotlivými layouty. K jeho navržení byl použit online designer NinjaMock [17]. Po návrhu následuje samotná implementace. Zde je kladen důraz na popis použitých layoutů a dalších ovládacích prvků, které byly pro tvorbu aplikace použity.

Návrh a implementace databáze vychází z předchozích dvou návrhů. Je zde popsán autorův pohled na problematiku návrhu databáze i její následnou implementaci. Dále je zde definována struktura jednotlivých sloupců, tabulek a jejich vztahů.

4.1 Volba verze API

Aplikace je psána pro nejnižší verzi API 19, tedy pro verzi 4.4 KitKat. Tato verze byla vybrána, jelikož stále zaujímá téměř 22% na trhu (podle aktuálních dat, viz obr.2). Cílením aplikace na nižší API by již nepřineslo žádný velký přísun potencionálních uživatelů. I vzhledem ke svému tržnímu zastoupení, je tato verze milníkem, co se týče vylepšení. Jelikož vyšší verze Androidu budou díky zpětné kompatibilitě stále schopny tuto aplikaci bez problému spustit, půjde vytvořená aplikace spustit přibližně na 86,7% všech mobilních zařízeních s operačním systémem Android. [18]

4.2 Analýza

K analýze byly vybrány aplikace z Google Play s podobnou problematikou jako u vyvíjené aplikace. Při porovnávání byly vybírány aplikace, které před stažením působily dojmem komplexnosti, bez zaměření na jeden konkrétní sport. Byly vybrány ty nejstahovanější nebo první zobrazené. Aplikace byly vyhledávány pod hesly „gym notes“, „workout notes“, „training planner“. Jednotlivé aplikace byly analyzovány a porovnány podle funkcionality a následně porovnány s požadavky na vytvářenou aplikaci. Analyzována byla silná i slabá místa. Silná místa byla využita pro inspiraci, nebo byla navržena pro budoucí zlepšení aplikace, jelikož se nejednalo o momentálně klíčové vlastnosti. Slabá místa nalezená při porovnávání aplikací byla při návrhu této aplikace odstraněna.

4.2.1 Gym Workout

Jedná se spíše o aplikaci, poskytující jednotlivé tréninkové plány bez možnosti jakékoliv editace. Zpočátku obsahuje tréninkové plány pro začátečníky, jelikož pokročilejší techniky jsou dostupné pouze po zaplacení. Neexistuje zde možnost tvorby vlastního plánu a následné naplnění cviky. V aplikaci má uživatel možnost si i vytvořit vlastní tréninkový plán na jednotlivé dny. Jednotlivé cviky, které aplikace obsahuje, mají kvalitně zpracovaný popis i s animací, jak cvik správně provádět. Z tohoto důvodu zřejmě aplikace neumožňuje vytvoření vlastního cviku.

Při porovnání s požadavky vytvářené aplikace pro jakýkoliv sport je tudíž nevyhovující. Hlavním využitím této aplikace je spíše komerční poskytování již vytvořených tréninkových plánů a ne tvoření vlastních.

4.2.2 Gym Notes

Na první dojem tato aplikace působila velice dobře. Z počátku se i zdálo, že by mohla vyhovovat požadavků na komplexní aplikaci sloužící pro tvorbu a uchovávání tréninkových plánů. Avšak po krátkém zkoumání aplikace byl nalezen fatální nedostatek. Aplikace neumožňuje tvorbu tréninkového pánu. Slouží pouze k vytvoření určitého cviku,

ke kterému si uživatel píše záznamy, kolikrát a s jakou váhou ho provedl. U každého cviku je statistická analýza a graf znázorňující zlepšení či zhoršení.

Velkým plusem aplikace však je propracovanost jednotlivých cviků. Při tvorbě si uživatel může vybrat z široké škály propracovaných obrázků, které jsou zde na výběr. Je pouze škoda, že nemá možnost přidat obrázek vlastní. K jednotlivému cviku lze snadno přidat různé opakování a použitou váhu, kdy každý tento zápis vytvoří novou pracovní sérii. Jsou však nedomyšlené případy, kdy by si uživatel rád zapsal například uběhlou vzdálenost, intenzitu, či rychlost provedení. Toto bohužel aplikace neumožňuje.

Při porovnání této aplikace s požadavky bylo usouzeno, že je aplikace zaměřena spíše na záznamy k jednotlivým cvikům, pro uživatele posiloven, kteří si rádi vedou záznamy o nazvedané váze. Jedná se spíše o takový fitness zápisník a ne tréninkový plánovač. Je však třeba dodat, že autor této aplikace má dobře zpracované aktivity pro tvorbu a editaci cviků.

4.2.3 NoteSport

Při analýze této aplikace trvalo řadu minut, než byl pochopen životní cyklus aplikace. Hned od začátku se aplikace jevila jako dosti zmatečná a nepřehledná. I přes počáteční problémy byly nalezeny světlé stránky aplikace. Aplikace umožňuje přidání vlastních cviků, kterým při vytváření určí, zda se k nim bude zaznamenávat čas, opakování, či váha. Značně nešťastné je však zakládání nového tréninku. Při jeho tvorbě se totiž musí vybrat cviky, které v něm budou obsaženy, jelikož dále nebude možné další cviky do tréninku přidávat. Dále trénink nelze rozložit na dny. Tedy pro každý tréninkový den by musel být vytvořen nový tréninkový plán.

Aplikace částečně vyhovuje požadavkům tvořené aplikace, avšak působí značně nepřehledně a zmateně. Obsahuje mnoho nedostatků, které by trénujícího uživatele rychle odradily od jejího používání. Posledními nedostatky je zdlouhavost při snaze tvorby cviků či tréninků, jednotlivé cviky není možno nijak filtrovat a než se uživatel dopracuje k nějakému výsledku, musí projít množstvím aktivit a dialogových oken.

4.2.4 FitNotes

Hned při začátku analýzy této aplikace přišlo příjemné překvapení v podobě nápovědy při prvním provádění jakékoliv akce. Aplikace poskytuje možnost tvorby vlastních cviků. Dobře vymyšlené má i filtrování cviků pomocí partií. Funguje na přidávání několika cviků do jednoho tréninkového dne, přiřazeného v kalendáři. Bohužel neexistuje možnost tvorby jednotného tréninkového plánu, například na týden. Výhodou je však kopírování jednotlivých dnů napříč kalendářem. Dobrou vlastností aplikace je vyhledávání.

Tato aplikace se přiblížila požadavků asi nejvíce, avšak má stále mnoho nedostatků. Prvním z nich je nemožnost tvorby uceleného tréninku na týden. Jednotlivé tréninkové dny musí být zahrnuty do kalendáře, což znesnadňuje pozdější zpětné hledání tréninku. Dále u definice jednotlivých cviků lze zapsat pouze použitou váhu a počet opakování. Není zde brán vůbec zřetel na potřeby jiných sportovců. Žádná možnost definování intenzity, pauzy, či délky provádění cviku.

Pro vyvíjenou aplikaci by se dalo mnoho vlastností této aplikace použít. Jednotlivé cviky jsou dobře filtrované. Existuje zde vyhledávání. Aplikace obsahuje kopírování jednotlivých tréninkových dnů. Při delším tréninku to ušetří dost času. Jednotlivé tréninky mohou obsahovat poznámky. V závěru aplikace ještě obsahuje statistickou analýzu jednotlivých cviků, opakování či odcvičených týdnů, které jsou znázorněny v grafech.

4.2.5 Souhrn

Při analýze testovaných aplikací bylo zjištěno mnoho nedostatků, které obsahují již existující aplikace. Jednalo se například o zdlouhavé přidávání či editace. Dále to byly nefiltrované cviky či nemožnost vytvoření vlastních cviků či tréninkových plánů. Tyto nedostatky byly při návrhu odstraněny: například při přiřazení cviků k partiím pro lepší filtrování, nebo umožnění lépe specifikovat průběh určitého cviku stanovením intenzity nebo pauzy.

Každá aplikace však obsahovala nějaká ta plus, která byla využita při návrhu anebo mohou být zahrnuta do budoucího zlepšení aplikace. Možným vylepšením by určitě bylo přidání možnosti vyhledávání či přidání obrázků ke cvikům. Záleželo by na uživateli, zda by chtěl vylepšit jednotlivé cviky obrázkem. Měl by možnost přidání vlastního obrázku do

aplikace. Dále by bylo jistě přínosem zahrnutí kalendáře a možnost propojit ho s určeným tréninkem.

4.3 Návrh struktury aplikace

Aplikace se bude sestávat z vnitřních tříd aplikace, které nejsou nedílnou součástí aplikace, ale vzhledem k aplikační logice jsou nezbytné pro její správné fungování. Dále aplikace obsahuje adaptéry, fragmenty a aktivity (včetně připojených layoutů grafického rozhraní).

4.3.1 Třídy

- **Cvik**. Objekt, zařazený do příslušné partie
- **Detail**. Objekt, nesoucí specifikovaný detail cviku v tréninkovém plánu
- **Trénink**. Objekt, obsahující název a popis tréninkové jednotky
- **DatabaseHelper** - Třída dědící z SQLiteOpenHelper. Zajišťuje spojení databáze s aplikací. Poskytuje příslušné metody pro interakci aplikace s databází.

4.3.2 Adaptéry

- **PagerAdapter.** Zajišťuje snadný přechod mezi fragmenty. Adaptér pro správu fragmentů, zpracovává a ukládá stav fragmentů.
- **CvikExpandableListAdapter**. Adaptér pro ovládací prvek ExpandableListView, tedy dvou úrovňový ListView. Obsahuje metody pro získání informací a dat rodiče i potomka.
- **TreninkAdapter a DetailAdapter**. Adaptéry, které získaná data z databáze vystavují z Cursoru na prvek ListView. Předává data z databáze do jednotlivých proměnných, které následně používá pro plnění ovládacích prvků TextView.
- **SectionsPagerAdapter.** Generuje fragmenty (podle dnů). Umožňuje vracet instanci zvoleného fragmentu

4.3.3 Aktivity

- **BaseActivity**. Aktivita, sloužící k nastavení Toolbaru.
- **MainActivity**. Hlavní aktivita, která je zároveň rodičovskou aktivitou pro fragmenty FragmentTabCvik a FragmentTabTrenink
- **CvikEdituj**. Aktivita zodpovědná za editaci a mazání cviků. Stará se o předvyplnění TextBoxu a Spinneru
- **CvikPridej**. Aktivita umožňující přidání nového cviku do databáze. Obsahuje metodu pro plnění Spinneru pro partie.
- **DetailActivity** – Aktivita umožňuje přidání nového detailu tréninku, smazání či odkaz na editaci celého tréninkového plánu. Obsahuje vnořenou statickou třídu PlaceHolderFragment a třídu SectionsPagerAdapter.
- **DetailEdituj**. Aktivita zodpovědná za editaci a mazání detailů tréninku. Stará se o předvyplnění TextBoxů a Spinnerů.
- **DetailPridej**. Aktivita umožňující přidání nového detailu tréninku do databáze. Obsahuje metody pro plnění Spinnerů pro partie a cviky. Při výběru partie filtruje cviky
- **TreninkEdituj**. Aktivita zodpovědná za editaci tréninků. Stará se o předvyplnění TextBoxů.
- **TreninkPridej**. Aktivita umožňující přidání nového tréninku do databáze.

4.3.4 Fragmenty

- **FragmentTabCvik** – Fragment starající se o načítání partií a cviků do ExpandableListView. Dále umožňuje přesměrování na aktivity, určené pro přidávání, editaci či mazání cviků.
- **FragmentTabTrenink** – Fragment starající se o načítání jednotlivých tréninkových plánů do ListView. Umožňuje přesměrování na aktivitu, určenou pro definování nového tréninkového plánu. Při vybrání určitého tréninku zobrazí aktivitu DetailActivity s přesným detailem tréninku.
- **PlaceHolderFragment** – Statická třída, dědící z třídy ListFragment. Vrací novou instanci sebe sama. Pracuje s vygenerovanými fragmenty. Zobrazuje jednotlivé

detaily tréninku. Filtruje zobrazovaná data podle dnů. Stará se o obsluhu tlačítka na přidání nového detailu.

4.4 Návrh uživatelského rozhraní

Při návrhu uživatelského rozhraní je důležité, aby se kladl důraz na jednoduchost a přehlednost. Uživatel si vybírá aplikaci podle toho, pokud se v ní již od začátku snadně pohybuje. Jestliže je aplikace zmatečná, a navádí nebo přesměrovává uživatele jinam, než očekává, je celé UI navrženo chybně. Hlavní části UI aplikace se sestávají z několika aktivit. Ke každé z aktivit je vypracován její návrh, scénář a use case. Scénář popisuje reagování systému na určité události. Psáno z pohledu systému. Use case, neboli případ použití, popisuje počínání uživatele, a jeho očekávanou reakci. Je psán z pohledu uživatele.

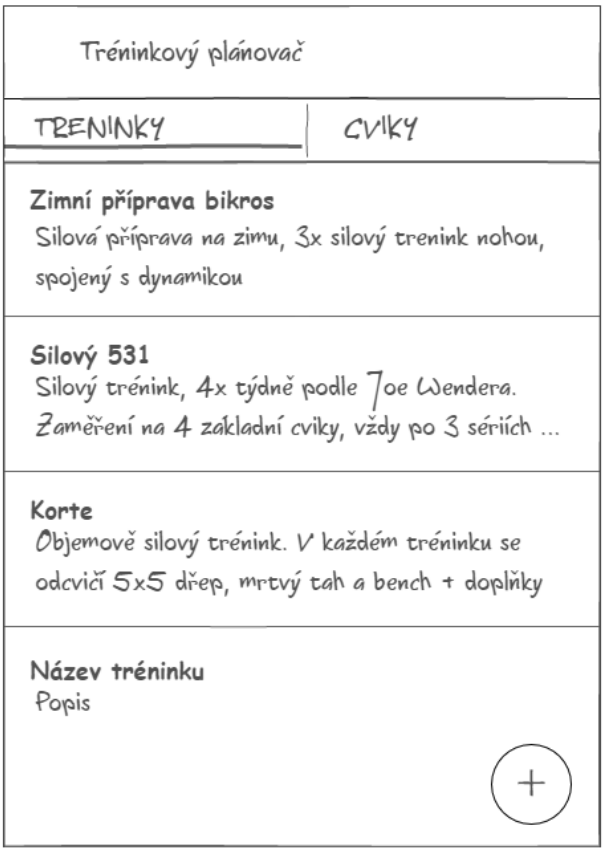
4.4.1 Tréninky

Scénář

Po zapnutí aplikace systém jako první zobrazí hlavní aktivitu s aktivním fragmentem tréninky. V hlavičce displeje je pod názvem aplikace nachází dvojice záložek Tréninky a Cviky. Dále systém zobrazí všechny tréninkové plány. Každý tento plán má vlastní název a popisek, zkrácený na dva řádky. Při vybrání tréninku a následném kliknutí systém uživatele přesměruje na aktivitu s podrobným detailem vybraného tréninku. V pravém spodním rohu se nachází tlačítko +, které zprostředkuje uživateli přístup do další aktivity, která mu umožní založení nového tréninkového plánu.

Use case

Uživatel očekává, že si bude moci vybrat jeden z vypsaných tréninkových plánů či založit plán nový. Dále pak pomocí kliknutí na záložku Cviky či pohybem prstu směrem vlevo očekává přesměrování na fragment Cviky. Pro přidání tréninku uživatel očekává tlačítko, které se bude nacházet v pravém dolním rohu.



Obrázek 8: Logický design - návrh GUI tréninky (zdroj: vlastní)

4.4.2 Cviky

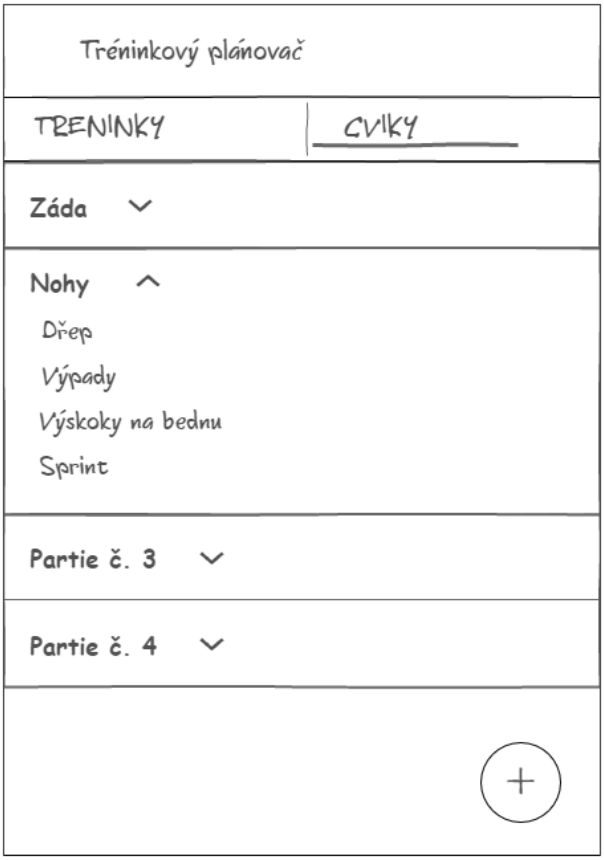
Scénář

Po vybrání záložky Cviky systém zobrazí fragment, který obsahuje seznam tělesných partií, které mají skrytý obsah. Po kliku na určitou partii, se zobrazí seznam všech cviků, náležících pod tuto partii. Při vybrání některého z cviků systém uživatele přesměruje na aktivitu, určenou pro editaci a mazání vybraného cviku. Po opětovném kliknutí na již rozbalenou partii systém zabalí určitý seznam cviků. Stejně jako u záložky tréninky, systém v hlavičce zobrazuje dvojici záložek a v pravém dolním rohu tlačítko pro zprostředkování přístupu do aktivity, umožňující přidání nového cviku.

Use case

Uživatel při kliknutí na partii očekává zobrazení náležitých cviků, přidělených právě k vybrané partii. Dále uživatel při vybrání některého z cviků očekává přesměrování

na jinou aktivitu, která mu zpřístupní možnost editace a smazání tohoto cviku. Pro zabalení seznamu cviků pod vybranou partií uživatel očekává stejný postup jako při rozbalení. Tedy kliknutí na stejnou partii. Uživatel očekává, že pomocí tlačítka + mu bude umožněno přidání nového cviku.



Obrázek 9: Logický design - návrh GUI cviky (zdroj: vlastní)

4.4.3 Detaily tréninku

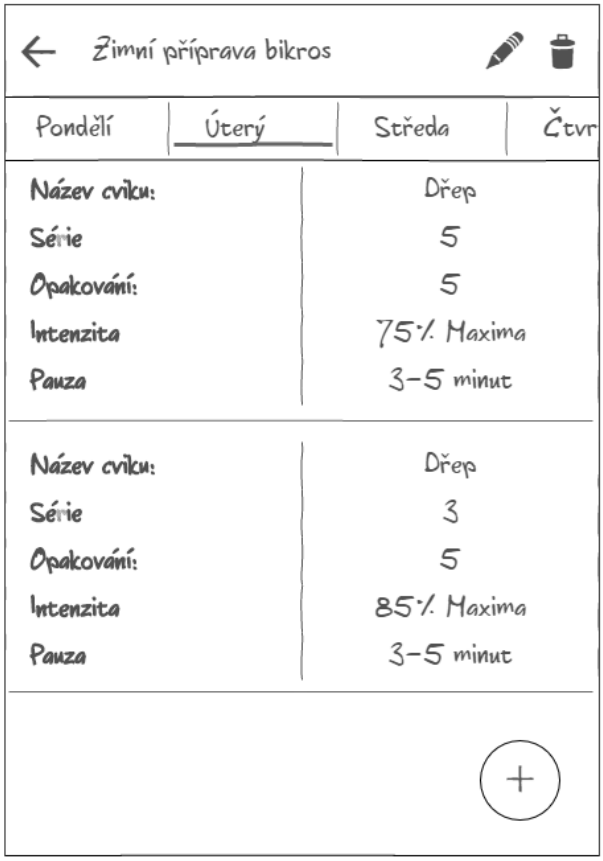
Scénář

Při výběru určitého tréninku systém spustí aktivitu s detaily tréninku. V hlavičce systém zobrazí název tréninku a tlačítka umožňující vrácení se zpět na předešlou aktivitu, editaci či mazání tréninku. Dále systém umožňuje přepínat mezi dny v týdnu. V každém dnu systém zobrazí detailní popis příslušného trénovaného cviku či aktivity. Editaci jednotlivých detailů tréninku systém zprostředkuje při jejich výběru. V pravém dolním

rohu systém zobrazí tlačítko pro zprostředkování přístupu do aktivity, umožňující přidání nového detailu tréninku do příslušného dne.

Use case

U této aktivity uživatel očekává, že mu budou zobrazeny detaily vybraného tréninku, odpovídající příslušnému dni v týdnu. Dále uživatel očekává, že pohybem prstu po obrazovce či kliknutím se bude moci pohybovat mezi jednotlivými dny. Při potřebě editace uživatel po vybrání určitého detailu tréninku očekává přesměrování na aktivitu, určenou k editaci. Pro práci s vybraným tréninkem uživatel očekává navigační tlačítka v hlavičce aktivity. Pro přidání detailu tréninku k určitému dni uživatel očekává tlačítko, které se bude nacházet opět v pravém dolním rohu.



Obrázek 10: Logický design - GUI návrhu detail tréninku (zdroj: vlastní)

4.4.4 Aktivity pro přidávání a editaci

Aktivity pro přidávání a aktivaci se v aplikaci nacházejí každá třikrát. Tedy aktivity pro přidání či aktivaci cviků, tréninků i detaily tréninků. Aktivity určené k editaci mají stejný layout jako aktivity pro přidávání, avšak liší se předem vyplněnými hodnotami editovaného cviku. Zde je popsán návrh aktivity pro přidání detailu tréninku.

Scénář

Aktivitu určenou pro přidání detailu tréninku systém zobrazí při kliknutí na tlačítko plus (+), které se nachází v aktivitě určené pro Detaily tréninku. V hlavičce systém zobrazí pouze tlačítko zpět a titulek popisující prováděnou akci, v tomto případě se jedná o přidávání detailu tréninku. Dále systém v této aktivitě zobrazí Spinner obsahující seznam partií, přičemž při výběru partie systém filtruje seznam cviků, který se zobrazí v druhém Spinneru. Dále systém umožňuje specifikaci dalších parametrů detailu tréninku pomocí textových polí. Pro uložení nového detailu tréninku slouží tlačítko OK, které systému zašle požadavek na uložení.

Use case

Uživatel od této aktivity očekává možnost definování vlastního detailu tréninku a jeho následné uložení. Při tvorbě očekává usnadnění hledání cviku, když po kliku na první Spinner určený pro partie mu bude zobrazen seznam partií, ze kterých si jednu vybere. Díky tomuto kroku se mu zúží výběr možných cviků. Pokud neví, mezi kterou partii jeho cílený cvik patří, očekává, že při nevyplnění partie, se mu v druhém Spinneru zobrazí všechny cviky, které má v aplikaci uložené. U následujících textových polí očekává nápovědu, jaká hodnota patří do kterého pole. Při kliknutí na tlačítko OK očekává uložení detailu tréninku, který si vyplnil. Naopak při kliknutí na tlačítko Cancel očekává návrat do předchozí aktivity bez uložení vyplněných hodnot.

Obrázek 11: Logický design - GUI návrhu přidání detailu (zdroj: vlastní)

4.5 Implementace struktury aplikace

4.5.1 Třídy

V této kapitole jsou stručně popsány třídy, představující reálné objekty. Třídy obsahují privátní vlastnosti, ke kterým přístup zajišťují veřejné metody pro získání a poslání (gettery a settery). Veškeré identifikátory jsou datového typu long, ostatní vlastnosti typu String.

Cvik

Třída obsahuje privátní vlastnosti _id, _cvikNazev, _partieNazev.

Trénink

Třída obsahuje privátní vlastnosti _id, _nazev, _info.

Detail

Třída obsahuje privátní vlastnosti _id, treninkId, _cvikId, _den, _serie. _opakovani, _intenzita, _pauza.

4.5.2 Aktivity

BaseActivity

Aktivita dědící z AppCompatActivity představuje rodičovskou třídu pro ostatní aktivity v aplikaci. Je tvořena za účelem inicializace Toolbaru a jeho základního nastavení. Bez této aktivity by ostatní aktivity musely inicializovat každá svůj Toolbar zvlášť. Aktivita obsahuje abstraktní metodu *getLayoutResource(),* kterou každý potomek musí překrýt.

```
Toolbar toolbar;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(getLayoutResource());
    configureToolbar();
}

protected abstract int getLayoutResource();

private void configureToolbar() {
    toolbar = (Toolbar) findViewById(R.id.toolbar);
    if (toolbar != null) {
        setSupportActionBar(toolbar);
    }
}
```

Zdrojový kód 3: BaseActivity

MainActivity

Tato aktivita představuje rodičovskou aktivitu pro fragmenty FragmentTabCvik a FragmentTabTrenink. Pro tyto dva fragmenty vytváří instance za pomocí PagerAdapteru. Za pomoci třídy TabLayout vytváří dva taby s nastavenými jmény pro grafické rozhraní obou fragmentů. Obsahuje dvě metody (*pridejCvik()* a *pridejTrenink()*). Obě metody za pomoci Intentu volají aktivity určené pro tvorbu nových záznamů.

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

```
    setContentView(R.layout.activity_main);

    TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);
    tabLayout.addTab(tabLayout.newTab().setText("Treninky"));
    tabLayout.addTab(tabLayout.newTab().setText("Cviky"));

    final ViewPager viewPager = (ViewPager) findViewById(R.id.pager);
    final PagerAdapter adapter = new PagerAdapter
            (getSupportFragmentManager(), tabLayout.getTabCount());
    viewPager.setAdapter(adapter);
    viewPager.addOnPageChangeListener(new
TabLayout.TabLayoutOnPageChangeListener(tabLayout));
}

@Override
protected int getLayoutResource() { return R.layout.activity_main; }

public void pridejCvik(View view){
    Intent i = new Intent(getApplication(),CvikPridej.class);
    startActivity(i);
}

public void pridejTrenink(View view){
    Intent i = new Intent(getApplication(),TreninkPridej.class);
    startActivity(i);
}
```

Zdrojový kód 4: MainActivity

DetailActivity

Jedná se o aktivitu starající se o správu, výpis i filtrování detailů tréninku. Obsahuje statickou třídu PlaceHolderFragment a třídu SectionsPagerAdapter. Aktivita obsahuje deklarované proměnné, dostupné v rámci celé aktivity.

```
private ViewPager mViewPager;        //instance třídy ViewPager s
nastaveným adaptérem
private SectionsPagerAdapter mSectionsPagerAdapter;     //instance
adaptéru vracející fragment
private DatabaseHelper db;      //instance databáze
private static long longId;        //identifikátor tréninkového plánu
```

Zdrojový kód 5: DetailActivity - hlavička

V metodě *onCreate()* jsou výše zmíněné proměnné inicializovány. Metoda dále povoluje v ActionBaru zpětné tlačítko, získává ID tréninkového plánu a podle něho nastavuje nadpis aktivity. Na závěr se metoda stará o tvorbu instance třídy TabLayout, její inicializaci a přidělení adaptéru pro správu a zobrazování fragmentů.

```
@Override
protected void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    Intent i = getIntent();
    db = new DatabaseHelper(this);
    longId = i.getLongExtra("_id",-1);
    Trenink t = db.getTrenink(longId);
    setTitle(t.get_nazev());

    mSectionsPagerAdapter = new
SectionsPagerAdapter(getSupportFragmentManager());

    mViewPager = (ViewPager) findViewById(R.id.container);
    mViewPager.setAdapter(mSectionsPagerAdapter);

    TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
    tabLayout.setupWithViewPager(mViewPager);
}
```

Zdrojový kód 6: DetailActivity - onCreate()

Metoda *onCreateOptionsMenu()* se stará o přidání nových položek do ActionBaru. Spolusouvisející metoda *onOptionsItemSelected(),* spouštějící se při aktivaci některé z položek přidaných v menu, zajišťuje jejich správnou interakci s dalšími metodami.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_trenink, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();

    if (id == R.id.b_smazat) {
        vymazZaznam(longId);
        return true;
    } else if (id == R.id.b_edit)  {
        Intent i = new Intent(getApplication(),TreninkEdituj.class);
        i.putExtra("_id",longId);
        startActivity(i);
    }
    return super.onOptionsItemSelected(item);
}
```

Zdrojový kód 7: DetailActivity – Menu

Metoda *vymazZaznam()* se stará o smazání celého tréninkového plánu. Při úspěšném či neúspěšném smazání vypíše uživateli hlášku o úspěšnosti operace. Při úspěšném smazání přesměruje uživatele zpět na hlavní aktivitu. To provede metoda *aktivujMainActivity()* pomocí třídy Intent.

```
private void vymazZaznam(){
    if(db.deleteTrenink(this.longId)){
        Toast.makeText(this, "Smazáno", Toast.LENGTH_SHORT).show();
        aktivujMainActivity();
    } else{
        Toast.makeText(this, "Nesmazáno", Toast.LENGTH_SHORT).show();
    }
}

private void aktivujMainActivity(){
    Intent i = new Intent(getApplication(),MainActivity.class);
    startActivity(i);
}
```

Zdrojový kód 8: DetailActivity - vymazZaznam()

SectionPagerAdapter

Vložená třída SectionPagerAdapter, dědící z třídy FragmentPagerAdapter, zajišťuje správnou tvorbu a zobrazení názvů fragmentů. Konstruktor pomocí *super()* volá zděděný konstruktor. Metoda *getItem()* vrací novou instanci fragmentu. Metoda *getCount()* vrací celkový počet fragmentů. V tomto případě podle počtu dní v týdnu, tedy 7. Poslední metoda této třídy *getPageTitle()* podle získávaného parametru o pozici vrací jméno příslušného fragmentu, které bude obsaženo v TabLayoutu.

```
public SectionsPagerAdapter(FragmentManager fm) { super(fm); }

@Override
public Fragment getItem(int position) {
    return PlaceholderFragment.newInstance(position,longId); }

@Override
public int getCount() { return 7; }

@Override
public CharSequence getPageTitle(int position) {
    switch (position) {
        case 0:
            return "Pondělí";
        case 1:
            return "Úterý";
        case 2:
            return "Středa";
        case 3:
            return "Čtvrtek";
        case 4:
            return "Pátek";
        case 5:
            return "Sobota";
        case 6:
            return "Neděle";
    }
    return null; }
```

Zdrojový kód 9: SectionPagerAdapter

PlaceholderFragment

Vložená statická třída PlaceholderFragment, dědící z třídy ListFragment, se stará o správu vybraného fragmentu, jeho spojení s adaptérem a naplnění správnými daty. Třída má v hlavičce deklarované dvě proměnné. První se stará o textové uchování dne vybraného fragmentu, druhá je argumentem s číslem fragmentu.

```
private static final String ARG_SECTION_NUMBER = "section_number";
private String day;
```

Zdrojový kód 10: PlaceholderFragment - hlavička

Třída dále pro vracení instance sebe sama obsahuje metodu *newInstance()*. Tato metoda pomocí třídy Bundle s instancí předává i argument, obsahující číslo fragmentu, podle kterého lze poznat, o jaký den se jedná.

```
public static PlaceholderFragment newInstance(int sectionNumber) {
    PlaceholderFragment fragment = new PlaceholderFragment();
    Bundle args = new Bundle();
    args.putInt(ARG_SECTION_NUMBER, sectionNumber);
    fragment.setArguments(args);
    return fragment;
}
```

Zdrojový kód 11: PlaceholderFragment - newInstance()

Metoda *onCreateView()* vrací View, které reprezentuje zobrazovaný fragment. Prvním krokem této metody je inicializace proměnné rootView, která je propojena layoutem příslušného fragmentu. Dále metoda naplní proměnou day příslušným názvem dnu. Za pomoci databáze a třídy Cursor nastaví DetailAdapter a naplní ho správnými hodnotami. Před vrácení View ještě metoda obsahuje Listener, který odchytává kliknutí na tlačítko +. Při kliknutí na toto tlačítko se zavolá metoda *onClick()*. Tato metoda pomocí třídy Intent předá parametry nové aktivitě pro přidání nového záznamu. Následně ji spustí.

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

    List<String> listDay =
Arrays.asList(getResources().getStringArray(R.array.dny_array));
    this.day = listDay.get(getArguments().getInt(ARG_SECTION_NUMBER));

    Cursor c = db.getDetailyPodleDnu(longId,this.day);
    DetailAdapter adapter = new DetailAdapter(getActivity(),c,0);
    setListAdapter(adapter);

    FloatingActionButton fab = (FloatingActionButton)
rootView.findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Toast.makeText(getActivity(), day,
Toast.LENGTH_SHORT).show();
            Intent i = new Intent(getActivity(),DetailPridej.class);
            i.putExtra("_day",day);
            i.putExtra("_treninkId",longId);
            startActivity(i);
        }
    });
    return rootView;
}
```

Zdrojový kód 12: PlaceholderFragment - onCreateView()

Poslední metoda vyskytující se v třídě PlaceholderFragment se jmenuje *onListItemClick()*. Tato metoda se stará o předání id vybraného detailu a následného spuštění aktivity pro editaci zvoleného detailu tréninku.

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Intent i = new Intent(getActivity(),DetailEdituj.class);
    i.putExtra("_detailId",id);
    startActivity(i);
}
```

Zdrojový kód 13: PlaceholderFragment - onListItemClick()

CvikPridej

Třídy končící názvem „Pridej“ se starají o přidávání nových záznamů do databáze. Aplikace dále obsahuje tři podobně fungující třídy. Pro demonstraci implementace byla vybrána třída CvikPridej.

Jedná se o aktivitu starající se o naplnění Spinneru daty a přidání záznamů do databáze. Metoda *onCreate()* inicializuje proměnné a volá metodu *spinnerFilled(),* která se stará o naplnění Spinneru daty.

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

    db = new DatabaseHelper(this);
    cvikET = (EditText) findViewById(R.id.cvikET);
    partieSP = (Spinner) findViewById(R.id.partieSP);

    spinnerFilled();
}
```

Zdrojový kód 14: CvikPridej - onCreate()

Výše zmíněná metoda *spinnerField()* obsahuje proměnnou typu ArrayAdapter, které předává kolekci textových polí (Stringů). U tohoto adaptéru následně nastaví způsob zobrazování jednotlivých položek. Na konci již zbývá pouze proměnné partieSP nastavit příslušný adapter jako zdroj dat.

```
public void spinnerFilled(){
    ArrayAdapter<CharSequence> adapter =
ArrayAdapter.createFromResource(
this,R.array.partie_array,android.R.layout.simple_spinner_item);
    adapter.setDropDownViewResource(
            android.R.layout.simple_spinner_dropdown_item);
    partieSP.setAdapter(adapter);
}
```

Zdrojový kód 15: CvikPridej – spinnerFilled

Pro potvrzení a odeslání třída obsahuje metodu *okCvik()*. Metoda zjišťuje počet znaků v textovém poli. Nový cvik lze přidat pouze při délce delší než 3 znaky, v opačném případě vypíše varovnou hlášku. Při splnění tohoto požadavku následuje další podmínka, která zjišťuje, pokud název zadaného cviku v příslušné partii již neexistuje. Pokud je název cviku unikátní, přidá se do databáze nový záznam. V opačném případě bude vypsána opět varovná hláška.

```
public void okCvik(View view) {
    int delka = cvikET.length();

    if (delka >= 3) {
        Cvik cvik = new Cvik(cvikET.getText().toString(),
partieSP.getSelectedItem().toString());
        if (db.addCvik(cvik) == 0) {
            Toast.makeText(this, "Tento název cviku již existuje",
Toast.LENGTH_SHORT).show();
        } else {
```

```
            aktivujMainActivity();
        }
    } else {
        Toast.makeText(this, "Název cviku musí obsahovat minimálně 3
znaky", Toast.LENGTH_SHORT).show();
    }
}
```

Zdrojový kód 16: CvikPridej - okCvik()

CvikEdituj

Podobně jako v předešlém případě se v aplikaci nacházejí 3 podobné aktivity, tentokrát ale končící názvem „Edituj“. Tyto aktivity se starají o předvyplnění textových polí či Spinnerů a následné přepsání hodnoty v databázi.

Metoda *onCreate()* opět inicializuje proměnné a volá metody pro plnění Spinnerů a předvyplnění textových polí. Dále plní kolekci typu long identifikátory cviků, které jsou již použité v detailech tréninkových plánů.

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

    cvikET = (EditText) findViewById(R.id.cvikET);
    partieSP = (Spinner) findViewById(R.id.partieSP);
    db = new DatabaseHelper(this);

    spinnerFilled();
    fillComponents();
    listIdCviku = db.getDetailyCviku();
}
```

Zdrojový kód 17: CvikEdituj - onCreate()

Důležitou pro tuto aktivitu je metoda *fillComponents(),* která se stará o vyplnění komponent správnými hodnotami. V počátku si metoda za pomoci třídy Intent, kterou předává hlavní aktivita, vytáhne data o editovaném cviku.

```
public void fillComponents() {
    Intent ii = getIntent();
    longID = ii.getLongExtra("_id",-1);
    String cvikNazev = ii.getStringExtra("_cvikNazev");
    String partieNazev = ii.getStringExtra("_partieNazev");
}
```

Zdrojový kód 18: CvikEdituj - fillComponents()

Metoda *okCvik()* se stará o editaci samotných záznamů. Kontroluje, zda název již neexistuje a jestli má alespoň 3 znaky, jinak uživateli zobrazí varovnou hlášku a neumožní mu cvik upravit.

```
public void okCvik(View view) {
    int delka = cvikET.length();

    if(delka >= 3) {
        Cvik c = new Cvik(longID,cvikET.getText().toString(),partieSP
.getSelectedItem().toString());
        if( db.updateCvik(c) == 0) {
            Toast.makeText(this, "Tento název cviku již existuje",
Toast.LENGTH_SHORT).show();
        } else {
            aktivujMainActivity();
        }
    } else {
        Toast.makeText(this, "Název cviku musí obsahovat minimálně 3
znaky", Toast.LENGTH_SHORT).show();
    }
}
```

Zdrojový kód 19: CvikEdituj - okCvik()

Třída dále obsahuje metodu *vymazZaznam()*, která kontroluje, zda mazaný cvik není obsažen v některém z detailů tréninku. Pokud ano, mazání skončí nezdarem a zobrazením chybové hlášky. V opačném případě systém pokračuje dále v metodě, která ještě obsahuje další podmínku, tentokrát již pro mazání. Při úspěšném či neúspěšném smazání je zobrazena hláška informující o výsledku.

```
private void vymazZaznam(){
    int i = 0;
    while (i < listIdCviku.size()) {
        if (listIdCviku.get(i).longValue() == longID) {
            Toast.makeText(this, "Nelze smazat, cvik je používán !",
Toast.LENGTH_LONG).show();
            return;
        } else {
            i++;
        }
    }
    if(db.deleteCvik(longID)){
        Toast.makeText(this, "Smazáno", Toast.LENGTH_SHORT).show();
        aktivujMainActivity();
    } else{
        Toast.makeText(this, "Nesmazáno", Toast.LENGTH_SHORT).show();
    }
}
```

Zdrojový kód 20: CvikEdituj - vymazZaznam()

Dále aktivita obsahuje metody již zmíněné v předešlých kapitolách. Jedná se o *stornoCvik(),* která díky *aktivujMainActivity()* ruší editaci přesměrováním. Výčet metod uzavírá dvojice *onCreateOptionsMenu()* a *onOptionsItemSelecte(),* upravující ActionBar.

4.5.3 Fragmenty

FragmentTabCvik

Jedná se o fragment, který je potomkem hlavní aktivity. Druhým potomkem hlavní aktivity je dále FragmentTabTrenink. Tyto dva fragmenty fungují podobně, avšak fragment určený pro cviky je díky použití BaseExpandableListAdapteru o poznání zajímavější.

Hlavní pracovní náplní popisované třídy FragmentTabCvik je práce s prvkem ExpandableListView, který funguje jako dropdown menu. Hlavní skupinu zde zaujímají partie, které mají své potomky (položky), cviky.

V metodě *onCreateView()* se z počátku inicializují všechny proměnné a na instanci ExpandableListView se nastaví adaptér, který představuje CvikExpandableListAdapter.
Pro inicializaci adaptéru je potřeba, aby mu byly předány instance aktivity a kolekce partií a cviků, které má zobrazovat. Dále se povolí odchytávání kliknutí, které následně zajišťuje listener. Tento listener se stará o předání hodnot cviku a o spuštění aktivity CvikEdituj. Tyto operace se provedou za pomoci třídy Intent. Pokud je vybraný cvik již někde použit, zobrazí se hláška, která na tuto skutečnost upozorňuje.

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

    expListView = (ExpandableListView) v.findViewById(listViewExp);
    this.db = new DatabaseHelper(getActivity());
    prepareListData();
    listIdCviku = db.getDetailyCviku();
    CvikExpandableListAdapter adapter = new
CvikExpandableListAdapter(getActivity(), listPartie, listCviky);
    expListView.setAdapter(adapter);
    expListView.setClickable(true);

    expListView.setOnChildClickListener(new
ExpandableListView.OnChildClickListener() {
        @Override
        public boolean onChildClick(ExpandableListView parent, View v,
int groupPosition, int childPosition, long id) {
            int i = 0;
            cvik = (Cvik)
parent.getExpandableListAdapter().getChild(groupPosition,
childPosition);
            Intent ii = new Intent(getActivity(), CvikEdituj.class);
            ii.putExtra("_id", cvik.get_id());
            ii.putExtra("_cvikNazev", cvik.get_cvikNazev());
            ii.putExtra("_partieNazev", cvik.get_partieNazev());
```

```
            while (i < listIdCviku.size()) {
                if (listIdCviku.get(i).longValue() == id) {
                    Toast.makeText(getActivity(), "Pozor! Změna se
projeví i v jiných tabulkách.", Toast.LENGTH_LONG).show();
                    startActivity(ii);
                    return false;
                } else {
                    i++;
                }
            }
            startActivity(ii);

            return false;
        }
    });
    return v;
}
```

Zdrojový kód 21: FragmentTabCvik - onCreateView()

Metoda *prepareListData()* se stará o naplnění kolekcí příslušnými daty. Nejprve naplní listPartie daty z kolekce Stringů, a poté pomocí for cyklu i HashMapu listCviky. Do kolekce typu HashMapu nejprve určitou partii a k ní příslušnou kolekci cviků, získanou z databáze.

```
private void prepareListData() {
    listPartie =
Arrays.asList(getResources().getStringArray(R.array.partie_array));
    listCviky = new HashMap<>();
    for (int i = 0; i < listPartie.size(); i++) {
        listCviky.put(listPartie.get(i),
db.getCvikyList(listPartie.get(i)));
    }
}
```

Zdrojový kód 22: FragmentTabCvik - prepareListData()

4.5.4 Adaptéry

CvikExpandableListAdapter

Třída představující adaptér pro ExpandableListView, tedy dvouúrovňový ListView. Obsahuje metody pro získání dat a informací ze skupiny i z potomka. Tato třída obsahuje velké množství překrytých metod, díky dědění z třídy BaseExpandableListAdapter a implementace z rozhraní ExpandableListView. Adaptér zajišťuje správné rozdělení dat určených pro potomka i skupinu. Tedy mezi cviky a partie. Jednotlivé metody mohou vracet View, samotný objekt Cvik nebo datové typy jako jsou boolean, int a long.

Pro příklad demonstrující metody vracející View s názvy partií, tedy skupin, byla vybrána metoda *getGroupView()*. V počátku metody jsou zakázány vlastnosti ViewGroup,

a to Focusable a Clickable. Tuto funkcionalitu později zastává listener. Třída LayoutInflater se postará o naplnění View. Ještě před koncem metody nastaví nadpisy vygenerovaných TextView.

```
@Override
public View getGroupView(int groupPosition, boolean b, View view,
ViewGroup viewGroup) {
    viewGroup.setFocusable(false);
    viewGroup.setClickable(false);
    String headerTitle = (String) getGroup(groupPosition);
    if (view == null) {
        LayoutInflater infalInflater = (LayoutInflater)
this._context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        view = infalInflater.inflate(R.layout.cviky_listrow_group,
null);
    }

    TextView listPartieheader = (TextView)
view.findViewById(R.id.listPartieHeader);
    listPartieheader.setTypeface(null, Typeface.BOLD);
    listPartieheader.setText(headerTitle);

    view.setClickable(false);

    return view;
}
```

Zdrojový kód 23: CvikExpandableListAdapter - getGroupView()

PagerAdapter.

Tento adaptér je používán hlavní aktivitou, pro zobrazování a navigaci mezi jejími fragmenty. Třída, představující tento adaptér, dědí z třídy FragmentStatePagerAdapter. Adaptér zajišťuje správu fragmentů, jejich zpracování a ukládání stavů. Adaptér umožňuje navigaci mezi fragmenty za pomocí třídy TabLayout, která je v hlavní aktivitě používána. Konstruktor třídy inicializuje proměnnou, uchovávající počet tabů. Dále je ve třídě k dispozici metoda *getCount(),* která se stará o vypsání této hodnoty proměnné.

```
int mNumOfTabs;

public PagerAdapter(FragmentManager fm, int NumOfTabs) {
    super(fm);
    this.mNumOfTabs = NumOfTabs;}

@Override
public int getCount() { return mNumOfTabs; }
```

Zdrojový kód 24: PagerAdapter

Metoda *getItem()* zajišťuje vracení instancí jednotlivých fragmentů na základě parametru určující jeho pozici.

```
@Override
public Fragment getItem(int position) {
    switch (position) {
        case 0:
            FragmentTabTrenink tab1 = new FragmentTabTrenink();
            return tab1;
        case 1:
            FragmentTabCvik tab2 = new FragmentTabCvik();
            return tab2;

        default:
            return null;
    }
}
```

Zdrojový kód 25: PagerAdapter - getItem()

TreninkAdapter

Mezi adaptéry zajišťující zobrazení správných dat v určených ovládacích prvcích patří TreninkAdapter a DetailAdapter. Oba fungují na stejných principech, pouze se mění názvy sloupců z databáze, ze kterých tahají data. Cursorem vytažené hodnoty z databáze jsou ukládány do odlišných ovládacích prvků. Pro demonstraci byla vybrána třída TreninkAdapter, dědící z třídy CursorAdapter.

O plnění jednotlivých View daty se opět stará třída LayoutInflater v metodě *newView()*. Metoda vrací instanci třídy View.

```
@Override
public View newView(Context context, Cursor cursor, ViewGroup viewGroup)
{
    LayoutInflater inflater = LayoutInflater.from(context);
    View view = inflater.inflate(R.layout.treninky,viewGroup,false);
    bindView(view,context,cursor);
    return view;
}
```

Zdrojový kód 26: TreninkAdapter - newView()

Další metoda přispívající k fungování adaptéru se nazývá *bindView()*. Tato metoda se stará o správnou inicializaci vytvořených proměnných. Jednotlivé proměnné jsou svázány pomocí Cursoru s příslušnými sloupci v databázi. Jednotlivé proměnné jsou pak zdrojem textu pro ovládací prvky v adaptéru. V tomto případě se jedná o prvky TextView.

```
@Override
public void bindView(View oldView, Context context, Cursor cursor) {

    int idIndex =
cursor.getColumnIndex(DatabaseHelper.COLUMN_TRENINK_ID);
    int nazevIndex =
cursor.getColumnIndex(DatabaseHelper.COLUMN_TRENINK_NAZEV);
    int infoIndex =
```

```
cursor.getColumnIndex(DatabaseHelper.COLUMN_TRENINK_INFO);

    ((TextView)oldView.findViewById(R.id.treninkId))
.setText(cursor.getString(idIndex));
    ((TextView)oldView.findViewById(R.id.treninkNazev))
.setText(cursor.getString(nazevIndex));
    ((TextView)oldView.findViewById(R.id.treninkInfo))
.setText(cursor.getString(infoIndex));
}
```

Zdrojový kód 27: TreninkAdapter - bindView()

Poslední metodou adaptéru je *getView()*. Metoda slouží k získání View, které zobrazuje data na specifické pozici. Dále se na přeskáčku nastavuje barva pozadí pro vracené View. Záleží, zda je na sudé či liché pozici. Výčet barev je definován v hlavičce třídy.

```
private int[] colors = new int[] { 0x30FFFFFF, 0x30cccccc };


@Override
public View getView(int position, View convertView, ViewGroup parent) {
    View view = super.getView(position, convertView, parent);
    int colorPos = position % colors.length;
    view.setBackgroundColor(colors[colorPos]);
    return view;
}
```

Zdrojový kód 28: TreninkAdapter - getView()

4.6 Implementace uživatelského rozhraní

Při implementaci uživatelského rozhraní se primárně vycházelo z návrhů jednotlivých aktivit. Cílem implementace bylo se těmto návrhům co nejvíce přiblížit. Layouty v aplikaci definují vizuální strukturu uživatelského rozhraní pro jednotlivé aktivity. Při tvorbě layoutu se designér či vývojář musí rozhodnout, zda ho bude definovat pomocí xml nebo pomocí Android designeru. Ze zkušenosti autora byly zvoleny oba způsoby tvorby. Pomocí Android designeru v Android Studiu se jednotlivé layouty a ovládací prvky dají snadno nadefinovat, avšak pro jejich přesnou pozici či hodnotu je vhodné použít úpravu pomocí xml. Některé prvky se totiž v Android designeru špatně hledají, nebo se tam dokonce vůbec nenacházejí. Pro návrh funkčního a uživatelsky přívětivého layoutu pomáhá tzv. Google Material design, který slouží jako takový předpis pro zaběhlé a moderní pojetí designu.

V následujících kapitolách budou popsány jednotlivé layouty a prvky použité v aplikaci. K jednotlivým prvkům bude přiložena ukázka kódu aplikace, nebo snímek obrazovky, reprezentující její grafické rozhraní.

4.6.1 Layouty

Layout v systému Android představuje rozvržení aktivity. Jedná se kontejner, který rovná jednotlivé prvky určitým směrem, nebo podle závislostí či kotev k ostatním prvkům. Každý layout i ovládací prvek musí obsahovat hodnoty pro šířku či výšku. Šířka (layout_width) nebo výška (layout_height) mohou obsahovat přesnou velikost určenou číslem. V praxi se však téměř vždy používají níže uvedené hodnoty parametru:

- **match_parent** - Vyplní prvkem celý rodičovský prvek, např. layout.
- **wrap_content** - Udává, že by rozměr měl být takový, aby se prvek vešel do View + odsazení. Při použití dvou prvků na jednom řádku, je rovnoměrně rozprostře.
- **fill_parent** – Od API 8 používat pouze match_parent. Fill_parent je zastaralé.

LinearLayout

Jedná se layout, který rovná jednotlivé prvky vertikálně či horizontálně. Nastavuje se ve vlastnosti orientation. Jedná se základní určení chování layoutu. V ukázce je uveden lineární layout obsahující dvě tlačítka zobrazená vedle sebe. Lze použít při editaci nebo vytváření nových záznamů.

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/storno"
        android:id="@+id/bStorno"
        android:layout_weight="1"
        android:onClick="stornoCvik" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/OK"
        android:id="@+id/bOK"
        android:layout_weight="1"
        android:onClick="okCvik" />
</LinearLayout>
```

Zdrojový kód 29: LinearLayout

RelativeLayout

Tento layout zobrazuje jednotlivé prvky na relativních pozicích, ovlivněných jiným prvkem k layoutu. Jeden prvek se dá například ukotvit pod prvek druhý. Dále je možné v tomto layoutu prvky zarovnat na spod, po stranách, či doprostřed layoutu. V ukázce jsou zobrazeny dva TextView, které jsou ukotveny k sobě. Při pohybu s prvním TextView, určeným pro id, se zároveň bude měnit pozice i druhého.

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:text="@string/id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"
        android:id="@+id/detailId"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cvikid"
        android:id="@+id/cvikId"
        android:layout_alignParentTop="true"
        android:layout_toEndOf="@+id/detailId"/>
</RelativeLayout>
```

Zdrojový kód 30: RelativeLayout

4.6.2 Ovládací prvky

V této kapitole jsou uvedeny všechny ovládací prvky použité v aplikaci. U každého z prvků je jednotlivě popsána jeho funkce a k čemu se využívá. Dále je na screenshotech z aplikace demonstrováno jejich přesné využití v rámci aplikace.

ListView

Prvek, používaný na zobrazení rolovaného seznamu položek. Jedná se o kontejner, obsahující další prvky. V aplikačním kódu je dále s listem pracováno. Je mu přidělen adaptér, nebo také listener, který se zaktivuje při kliknutí na některou z položek listu.

ExpandableListView

Jedná se o pokročilejší verzi ListView. Tento prvek je rozdělen na skupiny a potomky. Pro správnou funkčnost je potřeba k layoutu, který obsahuje ovládací prvek

ExpandableListView, ještě přidat další dva layouty představující listy právě pro skupiny a potomky.

ScrollView

Prvek, umožňující svisle rolovat obrazovku při přesažení plochy displeje. Je použit u aktivity pro přidávání a editaci tréninků. Aktivita totiž obsahuje prvek EditText, který se dynamicky zvětšuje podle délky textu. ScrollView zabraňuje nečitelnosti.

TextView

Prvek, umožňující zobrazování textu - jedná se tedy o jednosměrnou komunikaci aplikace s uživatelem. Používán například pro zobrazení textové proměnné. Prvek TextView je obsažen v listech zobrazujících jednotlivé záznamy. Základním nastavujícím parametrem tohoto prvku je text.

EditText

Prvek patřící do Text Fields. Jedná se tedy o textové pole, kam má uživatel přístup zapisovat. Je hojně využíván při tvorbě nových záznamů či u editace. U aktivit sloužících pro přidávání záznamů je mu nastaven parametr hint (tip), který radí podobu textu, jakou má dané textové pole obsahovat.

Spinner

Prvek sloužící k výběru více položek. Je přizpůsobený pro jejich výběr na malé ploše dotykového displeje. V aplikaci slouží k výběru partií a cviků.

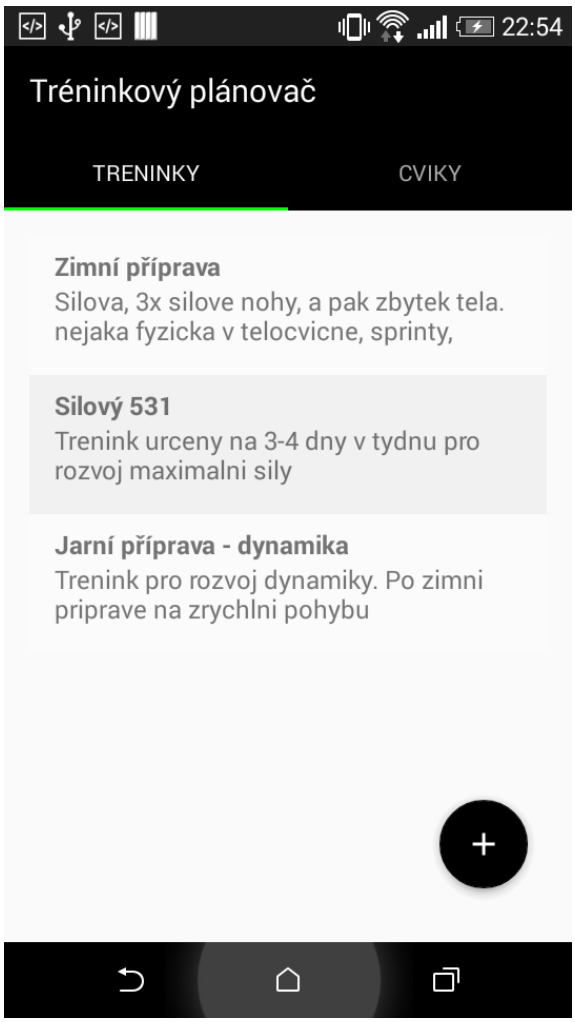
Button

Prvek, který představuje standartní tlačítko. Při jeho aktivaci dotykem je volána určitá metoda.

4.6.2.1 Screenshoty aplikace

Tréninky

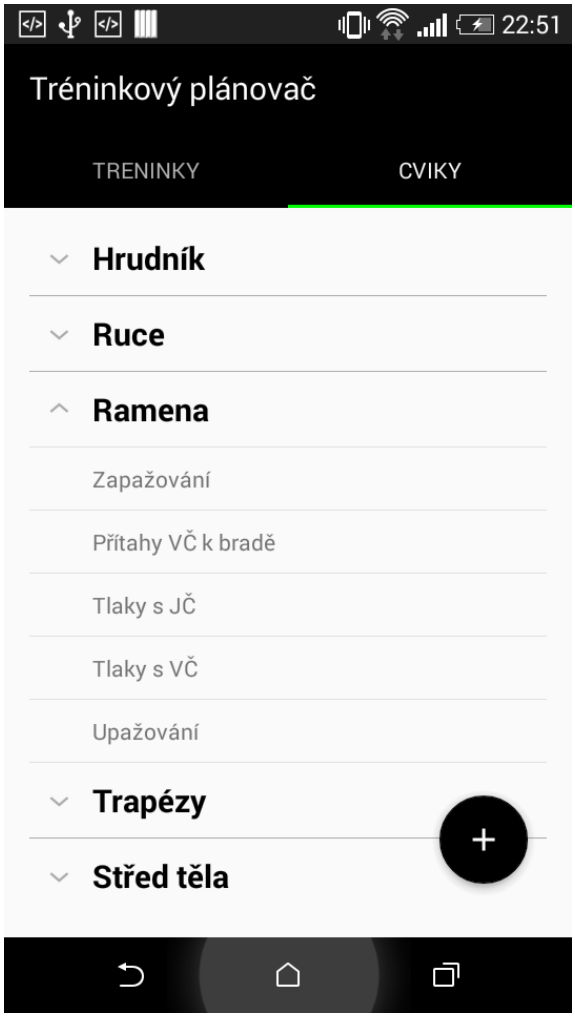
Jedná se o fragment, obsahující prvky FloatingActionButton a ListView, obsahující dva prvky TextView. Fragmenty pro Tréninky i Cviky však náleží pod hlavní aktivitu. Ta dále obsahuje prvky Toolbar pro zobrazení hlavičky aplikace, TabLayout pro navigaci mezi fragmenty a ViewPager, pro definování místa pro obsah jednotlivých fragmentů.



Obrázek 12: Tréninky screenshot (zdroj: vlastní)

Cviky

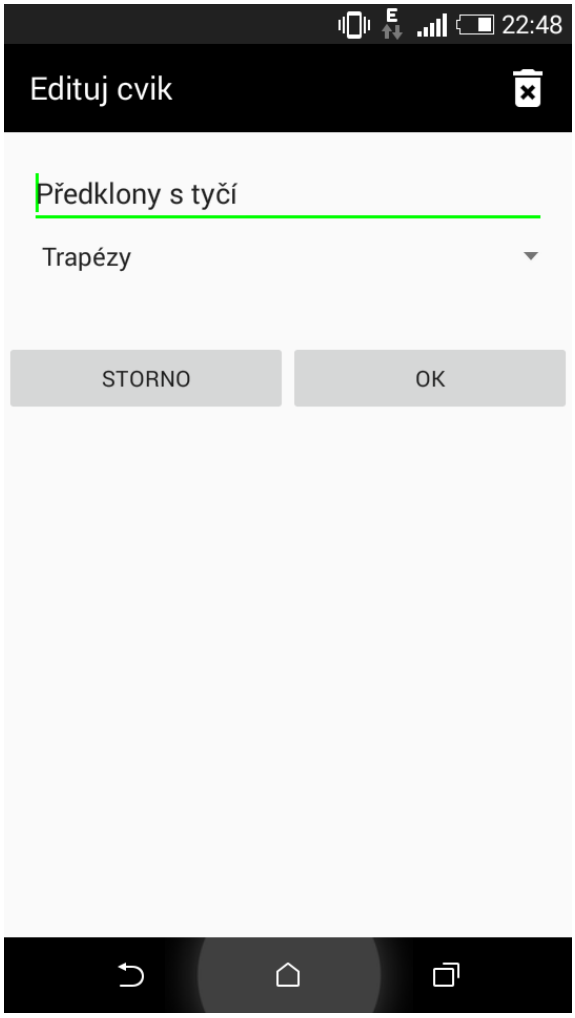
Fragment, obsahující ovládací prvky ExpandableListView a FloatingActionButton. Prvek ExpandableListView ke svému fungování potřebuje dva další layouty, které obsahuji prvky TextViews.



Obrázek 13: Cviky screenshot (zdroj: vlastní)

Edituj Cvik

Aktivita obsahuje prvky EditText pro zadání názvu cviku, Spinner pro výběr partie a Buttony pro uložení úpravy nebo naopak její vyrušení. Hlavička menu obsahuje item s nastavenou ikonou koše, sloužící k odstranění cviku.



Obrázek 14: Edituj cvik (zdroj: vlastní)

Detail tréninkového plánu

V hlavičce obsahuje Toolbar s definovaným vlastním layoutem. Obsahuje dva itemy a v aplikačním kódu má povolené tlačítko zpět. Dále obsahuje prvek TabLayout pro navigaci mezi vygenerovanými fragmenty a ViewPager pro definování obsahové plochy aktivity. Každý z fragmentů obsahuje prvek FloatingActionButton a ListView, obsahující mnoho dalších prvků typu TextView.

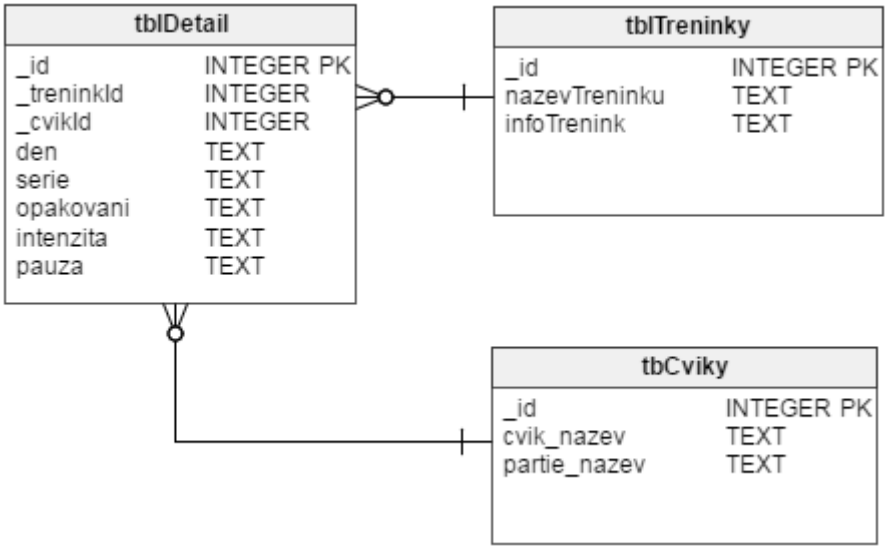
Obrázek 15: Detail tréninkového plánu screenshot (zdroj: vlastní)

4.7 Návrh a implementace databáze

Jako úložiště pro data aplikace byla vybrána databáze SQLite. Jedná se o relační databázi. Velkou výhodou této databáze je absence potřeby instalovat databázové enginy, servery či upravovat konfigurační soubory. Databáze je již součástí operačního systému Android a zastává funkci serveru i klienta současně. SQLite využívá architekturu klient-server. Aplikace k databázovému serveru přistupuje přes rozhraní.

4.7.1 Návrh struktury SQLite

Při návrhu struktury databáze bylo zapotřebí navrhnout jednotlivé tabulky, jejich sloupce a zvolit správné datové typy. Relační databáze aplikace obsahuje 3 vzájemně propojené tabulky. Každá tabulka má vlastní primární klíč. Tabulka tblDetail dále obsahuje cizí klíče provázané s primárními klíči zbylých dvou tabulek. Všechny klíče jsou datového typu INTEGER. Zbylé sloupce jsou typu TEXT. Je to dáno tím, že v sloupcích, například pro počet opakování, je tímto možné zadat rozsah opakování. Všechny sloupce jsou NOTNULL a jednotlivé primární klíče mají nastaveny AUTOINCREMENT pro automatické číslování, které též zabrání duplicitám.



Obrázek 16: Relační databázový model (zdroj: vlastní)

4.7.2 Implementace databázové struktury

Aplikace obsahuje pouze jednu databázovou třídu nazvanou DatabaseHelper, dědící z třídy SQLiteOpenHelper. Zdědila metody *onCreate()* a *onUpgrade()*. Metoda *onCreate()* se volá, když databázové objekty ještě nejsou vytvořeny. Dále je volána metodou *onUpgrade(),* která se volá vždy při aktualizaci databáze při změně její struktury. Proto je důležité při každé změně ve struktuře navýšit hodnotu proměnné DATABASE_VERSION, která je předávána konstruktoru. Jednotlivé názvy tabulek a sloupců jsou inicializovány ve statických final proměnných. Tyto hodnoty se nedají programově změnit. SQL příkazy pro tvorbu tabulek jsou rovněž inicializovány ve statických final proměnných.

Metoda *onCreate()* vytváří nové tabulky a stará se o jejich naplnění defaultními hodnotami. Toho docílí voláním dalších připravených metod. Metoda *onUpgrade()* smaže jednotlivé tabulky a následně zavolá *onCreate(),* pro jejich znovuvytvoření.

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREATE_TABLE_CVIK);
    db.execSQL(CREATE_TABLE_TRENINK);
    db.execSQL(CREATE_TABLE_DETAIL);

    onUpgradeNaplnCviky(db);
    onUpgradeNaplnTrenink(db);
    onUpgradeNaplnDetail(db);
}

@Override
public void onUpgrade(SQLiteDatabase db, int i, int i1) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME_CVIK);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME_TRENINK);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME_DETAIL);

    onCreate(db);
}
```

Zdrojový kód 31: Metody SQLiteOpenHelper

Jednotlivé metody pro práci s tabulkami používají třídu Cursor. Třída Cursor umožňuje přístup k vráceným výsledkům z databázových dotazů. Implementace pro metody zajišťující přidání, úpravu, mazání, získání všech nebo jen filtrovaných záznamů bude demonstrována na metodách pro tblTreninky. Na začátku každé metody je potřeba, při tvorbě instance třídy SQLiteDatabes, definovat práva pro práci s daty, tj. zda má instance právo do databáze zapisovat i číst, nebo pouze číst.

Metoda *addTrenink()* přidává nový záznam o tréninku do databáze, který získá ve vstupním parametru. Před samotným uložením musí projít podmínkou, že již trénink se stejným názvem neexistuje. Pokud nenastala shoda, je vytvořena nová instance třídy ContentValues, do které jsou následně přidány hodnoty sloupců. Po této operaci za pomoci příkazu INSERT bude přidán nový záznam. Pokud bude přidán nový záznam, operace vrátí do proměnné jedničku. Při neúspěchu vrátí číslo nula. Ta samá hodnota je na výstupu metody.

```
public long addTrenink(Trenink t) {
    SQLiteDatabase db = this.getWritableDatabase();
    boolean shoda = false;

    Cursor cursor = this.getTreninky();
    while (cursor.moveToNext()) {
        if (t.get_nazev().equals(cursor.getString(1)))
            shoda = true;
    }

    long id = 0;
    if (shoda == false) {
        ContentValues val = new ContentValues();
        val.put(COLUMN_TRENINK_NAZEV, t.get_nazev());
        val.put(COLUMN_TRENINK_INFO, t.get_info());

        id = db.insert(TABLE_NAME_TRENINK, null, val);
        db.close();
        return id;
    }
    return id;
}
```

Zdrojový kód 32: SQLite - addTrenink()

Metoda *updateTrenink()* upravuje hodnotu v databázi. Obsah metody je téměř stejný jako u *addTrenink()*. Liší se pouze v příkazu INSERT, který je zde nahrazen příkazem UPDATE. Příkazu UPDATE je předáván identifikátor upravovaného tréninku + jeho nové hodnoty.

```
id = db.update(TABLE_NAME_TRENINK, val, COLUMN_TRENINK_ID + " = ?",
new String[]{String.valueOf(t.get_id())});
```

Zdrojový kód 33: SQLite - update

Metoda *deleteTrenink()* podle vstupního parametru maže záznam z databáze. Vstupním parametrem je identifikátor daného tréninku. Při mazání tréninku se dále musí smazat všechny hodnoty z tabulky tblDetail. Příkaz DELETE opět vrací čísla 1 či 0, podle úspěšnosti operace.

```
public boolean deleteTrenink(long id) {
    SQLiteDatabase db = this.getWritableDatabase();
    String[] selectionArgs = {String.valueOf(id)};

    db.delete(TABLE_NAME_DETAIL, COLUMN_DETAIL_TRENINK_ID + "= ?",
selectionArgs);

    int deletedCount = db.delete(TABLE_NAME_TRENINK, COLUMN_TRENINK_ID +
"= ?", selectionArgs);
    db.close();
    return deletedCount > 0;
}
```

Zdrojový kód 34: SQLite - deleteTrenink()

Metoda *getTreninky()* se stará o získání všech záznamů z tabulky tblTreninky. Metoda vrací Cursor, do kterého jsou jednotlivé záznamy uloženy pomocí dotazu.

```
public Cursor getTreninky() {
    SQLiteDatabase db = this.getReadableDatabase();
    return db.query(TABLE_NAME_TRENINK, columns_trenink, null, null,
null, null, null, null);
}
```

Zdrojový kód 35: SQLite – getTreninky

Poslední metodou určenou pro tabulku tblTreninky je *getTrenink()*. Tato metoda na základě vstupního parametru (id) a dotazu nad databází (rawQuery), vrací třídu Trénink. Vytváří novou instanci vracené třídy, které pak Cursor předává jednotlivé hodnoty. Tato metoda je převážně používána při editaci.

```
public Trenink getTrenink(long id) {
    SQLiteDatabase db = this.getReadableDatabase();
    String sql = "SELECT * FROM " + TABLE_NAME_TRENINK + " WHERE " +
COLUMN_TRENINK_ID + " = '" + id + "'";
    Cursor cursor = db.rawQuery(sql, null);
    Trenink t = new Trenink(0,"","");

    if (cursor.moveToFirst()) {
        do {
            long idecko = cursor.getLong(0);
            String nazev = cursor.getString(1);
            String info = cursor.getString(2);
            t.set_id(idecko);
            t.set_nazev(nazev);
            t.set_info(info);
        } while (cursor.moveToNext());
    }
    return t;
}
```

Zdrojový kód 36: SQLite - getTrenink()

5 Výsledky a diskuse

Aplikace byla při vývoji průběžně testována autorem, zda neobsahuje nejasné postupy při práci. Toto testování se zaměřovalo převážně na funkčnost jednotlivých částí. Průběžné testování zamezilo možnosti ubírat se špatným směrem. Neznalost způsobu vyvíjení pro operační systém Android byla zpočátku překážkou. Při implementaci byla nejdříve vytvořena aplikační část (backend) a po získání těch správných výstupů do programu přišla na řadu tvorba uživatelského rozhraní. Tento postup nebyl ideální, jelikož zpočátku nebyly použity fragmenty. Díky této chybě byla valná část předělávána. Naštěstí tato změna měla pouze minimální vliv na metody obsažené v databázové třídě.

Po dokončení byla aplikace poskytnuta testovacím subjektům. Jednalo se o cyklistu v zimní přípravě, kulturistu a florbalistu. Jelikož se jednalo o testery, kteří aplikaci neznali, nekladly se jim žádné meze v testování. Všichni ocenili volnost poskytnutou v tvorbě jednotlivých tréninků i vlastních cviků.

Kulturista aplikaci velice ocenil, jelikož často mění tréninkové plány, a je rád, když mu ty staré někde zůstanou uloženy. Dále ocenil možnost přidání poznámky u tréninkových plánů, jelikož si do nich může napsat například svalové přírůstky nebo silové zlepšení. Jednotlivé detaily mu svým rozsahem stačily. Políčko intenzita využil pro uvedení váhy na čince. Aplikaci hodlá nadále využívat.

Cyklista v přípravě ocenil zanesení nových cviků, které se ve výčtu již defaultně přidaných nevyskytovaly. Jednalo se o různé variace sprintů a vzpěračské cviky. U sprintů mu při vytváření detailů tréninku chyběla položka, kde by zanesl vzdálenost anebo poznámku, zda se jednalo o sprint do kopce, po rovině, nebo z kopce. Tyto položky se při případném rozšíření aplikace jistě přidají. Jinak mu funkčnost aplikace vyhovovala. Popsal jí jako jednoduchou, jelikož se v ní velice rychle začal orientovat.

Florbalista se v aplikaci také rychle zorientoval. Jeho trénink se sestával převážně z kondiční přípravy v posilovně, běhání a samotného tréninku techniky a hry. Pro přípravu v posilovně by aplikaci používal, avšak co se týče běhu a samotného tréninku techniky hry dá raději přednost osvědčenému papíru.

Po otestování aplikace byly autorovi sděleny poznatky jednotlivých testerů. Cyklista upozornil na drobné nedostatky a dal podnět k novému rozšíření aplikace. Aplikace se žádnému z testerů náhodně nevypnula ani nepřestala pracovat, což je pozitivní poznatek.

Přestože se povedlo vytvořit plně funkční aplikaci, tak ze zkušeností získaných během návrhu a implementace lze říci, že jistě existují další možnosti jejího rozšíření. Jedná se například o funkcionality objevené v analyzovaných aplikacích, avšak při momentální tvorbě nebyly považovány za klíčové. Jedním z dalších rozšíření by bylo ukládání jednotlivých tréninků na cloudové úložiště, ke kterému by měl uživatel přístup přes uživatelské jméno a heslo. Dalším rozšířením může být zdynamičtění. Jednalo by se o možnost tvorby vlastní struktury tréninkového plánu. Sportovec by si tudíž mohl na základě svých požadavků měnit typy a počty parametrů jednotlivých tréninků. To by však zásadně změnilo strukturu aplikace, ovšem aplikaci by to posunulo o úroveň výše. Neposledním rozšířením by jistě mohl být i poznatek jednoho z testerů, který by uvítal u detailů tréninku možnost specifikace vzdálenosti anebo poznámky.

6 Závěr

V teoretické části této práce byla zpočátku popsána platforma Android a její historie, jednotlivé verze a architektura systému. Bylo popsáno vývojové prostředí Android Studia a programovací jazyk Java, za jejichž pomoci byla aplikace vyvíjena. V dalších kapitolách byly popsány jednotlivé komponenty, které jsou nezbytné pro tvorbu aplikace. V závěru teoretické části byl demonstrován postup při publikaci nové aplikace.Na začátku praktické části byly vybrány aplikace s podobnou tématikou, které byly porovnány se zadáním a cílem této práce. Po porovnání byla analyzována slabá místa aplikací, která byla následně při návrhu aplikace odstraněna. Porovnáním již existujících aplikací podle jejich funkcionality, byl splněn dílčí cíl práce.

Praktická část se dále soustředila na návrh a implementaci samotné aplikace. Návrh vycházel z poznatků kapitoly, zaměřené na porovnání aplikací s obdobnou problematikou. Při implementaci a návrhu struktury aplikace byly popsány jednotlivé třídy. Pro lepší demonstraci byly přidány i části kódu, převzaté přímo z aplikace. Pro návrh uživatelského rozhraní byly využity znalosti z předmětů Interakční design a Interakce člověk a počítač. K jednotlivým grafickým návrhům jsou sepsány scénáře systému a use casy uživatelů. Implementace uživatelského rozhraní zahrnovala popis použitých layoutů a ovládacích prvků. Téměř posledním bodem v práci byl návrh a implementace databáze, kterou aplikace využívá. Popsáním postupu tvorby aplikace, od jeho návrhu až po samotnou implementaci, byl splněn hlavní cíl práce. Na závěr byly sepsány poznatky z testování, které proběhlo úspěšně.

Smyslem této práce bylo zanalyzovat aplikace určené k plánování či vedení tréninkových plánů. Jednotlivá slabá místa měla být odstraněna v nově navržené a vyvinuté aplikaci. Navrhnout a naprogramovat aplikaci, která je schopna pokrýt požadavky pro plánování jakéhokoliv sportu, se úspěšně povedlo. Výsledkem této práce je funkční a stabilní aplikace. Tímto byl splněn cíl bakalářské práce.

Aplikaci je možné mnoha způsoby rozšířit či vylepšit. Před samotnou publikací by jistě chtělo zapracovat více na designu samotné aplikace a vylepšit některé její funkce, které již byly v minulé kapitole popsány.

7 Seznam použitých zdrojů

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

2. Historie Androidu v kostce. *Svět Androida* [online]. [cit. 2017-02-17]. Dostupné z: https://www.svetandroida.cz/historie-androidu-201506

3. Android, the world's most popular mobile platform. *Android Developers* [online]. [cit. 2017-02-17]. Dostupné z: https://developer.android.com/about/index.html

4. Jak vypadá Android uvnitř. *Android Market* [online]. [cit. 2017-02-17]. Dostupné z: http://androidmarket.cz/android/jak-vypada-android-uvnitr-aneb-co-je-rom-kernel-bootloader-a-dalsi/

5. Android Studio – nové vývojové prostředí. *Zdroják.cz* [online]. [cit. 2017-02-17]. Dostupné z: https://www.zdrojak.cz/clanky/android-studio-nove-vyvojove-prostredi/

6. The Activity Lifecycle. *Android Developers* [online]. [cit. 2017-02-17]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle.html

7. Vyvíjíme pro Android: Dialogy a activity. *Zdroják.cz* [online]. [cit. 2017-02-17]. Dostupné z: https://www.zdrojak.cz/clanky/vyvijime-pro-android-dialogy-a-activity/

8. Životní cyklus a nový projekt. *ITnetwork.cz* [online]. [cit. 2017-02-17]. Dostupné z: http://www.itnetwork.cz/java/android/tutorial-programovani-pro-android-v-jave-zivotni-cyklus-a-novy-projekt

9. Fragments. *Android Developers* [online]. [cit. 2017-02-17]. Dostupné z: https://developer.android.com/guide/components/fragments.html

10. ALLEN, Grant. *Android 4: průvodce programováním mobilních aplikací*. Brno: Computer Press, 2013. ISBN 978-80-251-3782-6.

11. Intent. *Android Developers* [online]. [cit. 2017-02-17]. Dostupné z: https://developer.android.com/reference/android/content/Intent.html

12. Vyvíjíme pro Android: Intenty, intent filtry a permissions. *Zdroják.cz* [online]. [cit. 2017-02-17]. Dostupné z: https://www.zdrojak.cz/clanky/vyvijime-pro-android-intenty-intent-filtry-a-permissions/

13. Vyvíjíme pro Android: Bližší pohled na pohledy. *Zdroják.cz* [online]. [cit. 2017-02-17]. Dostupné z: https://www.zdrojak.cz/clanky/vyvijime-pro-android-blizsi-pohled-na-pohledy-2-dil/

14. View. *Android Developers* [online]. [cit. 2017-02-17]. Dostupné z: https://developer.android.com/reference/android/view/View.html

15. ViewGroup. *Android Developers* [online]. [cit. 2017-02-17]. Dostupné z: https://developer.android.com/reference/android/view/ViewGroup.html

16. App Manifest. *Android Developers* [online]. [cit. 2017-02-17]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-intro.html

17. NinjaMock [online]. [cit. 2017-02-26]. Dostupné z: https://ninjamock.com/

18. Dashboards. *Android Developers* [online]. [cit. 2017-03-01]. Dostupné z: https://developer.android.com/about/dashboards/index.html#Screens

8 Přílohy

Vzhledem k rozsáhlosti je zdrojový kód aplikace dostupný v přiloženém zip souboru a na CD, včetně apk souboru, který umožňuje nainstalování aplikace na zařízení s operačním systémem Android (minimální verze 4.4 KitKat).