Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství


Diplomová práce

# Vývoj mobilní aplikace pro správu vozového parku

Bc. Lukáš Petrbok

© 2018 ČZU v Praze

# ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Lukáš Petrbok

Informatika

Název práce

**Vývoj mobilní aplikace pro správu vozového parku**

Název anglicky

**Development of mobile application for car fleet management**

---

**Cíle práce**

Práce je zaměřena na problematiku vývoje aplikací pro OS Android. Hlavním cílem je navrhnout a implementovat mobilní aplikaci pro tento OS sloužící ke usnadnění správy vozového parku. Dílčím cílem práce je zdokumentovat proces návrhu, vývoje a následné distribuce aplikace.

**Metodika**

Práce sestává ze dvou hlavních částí – přehledu teoretických východisek a praktické části.

Metodika zpracování teoretické části je založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude popsána problematika vývoje aplikací v OS Android.

V praktické části práce provedena analýza a implementace mobilní aplikace sloužící ke správě vozového parku firmy. Při návrhu a implementaci bude využito standardních metod a nástrojů softwarového inženýrství. Výsledná aplikace bude nasazena, otestována a na základě poznatků z nasazení a testů budou formulovány možnosti dalšího případného budoucího rozvoje aplikace.

**Doporučený rozsah práce**

60-80 stran

**Klíčová slova**

Android, mobilní aplikace, Java, SQLite, Realm

---

**Doporučené zdroje informací**

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

FRIESEN, J. Learn Java for Android Development. Berkley: Apress, 2014. ISBN 978-1-4302-6455-2.

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

ZAPATA, B. Android Studio 2 Essentials. Birmingham: Packt, 2016. ISBN 978-1-78646-795-9.

---

**Předběžný termín obhajoby**

2017/18 LS – PEF

**Vedoucí práce**

Ing. Jiří Brožek, Ph.D.

**Garantující pracoviště**

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 1. 2018

**Ing. Martin Pelikán, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 11. 1. 2018

**Ing. Martin Pelikán, Ph.D.**

Děkan

V Praze dne 27. 03. 2018

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Vývoj mobilní aplikace pro správu vozového parku" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 29.3.2018 _____

**Poděkování**

Rád bych touto cestou poděkoval panu Ing. Jiřímu Brokžovi, Ph.D. za velice vstřícný přístup, odborné vedení a přínosné konzultace.

# Vývoj mobilní aplikace pro správu vozového parku

**Souhrn**

Diplomová práce se zabývá tématem vývoje aplikací pro mobilní operační systém Android. Aplikace vzniklá v rámci této práce se nazývá Spravoz a slouží pro správu firemního vozového parku, ale lze využít i pro koncové uživatele. Nabízí evidování tankování i výdajů a následné vyhodnocení v podobě sedmi statistik a pětice grafů. Podkladová data lze filtrovat dle datumu nebo za jednotlivý automobil či celý vozový park. Aplikace taky umožňuje automatické notifikace na končící STK na základě údaje o první registrace vozidla nebo datumu poslední statní technické kontroly.

Teoretická část práce obsahuje základní přehled o operačním systému Android jeho historii a verzích. Dále je zde uvedena architektura systému a koncepce aplikace. Poslední část obsahuje metody, jak lze v operačním systému Android ukládat data.

Praktická část mapuje celý proces vývoje aplikace. První krokem je sepsání požadavků, následuje zpracování analýzy konkurenčního prostředí. Po této analýze následují scénáře užití, kde jsou popsány některé konkrétní průchody aplikací. Pomocí prototypů jsou dále vytvořeny návrhy grafického rozhraní Spravozu. Následně jsou tyto prototypy převedeny do skutečných jednotlivých obrazovek. Na to navazuje architektura objektové databáze, která je demonstrovaná diagramem tříd. Následuje již analýza jednotlivých tříd, kde je popsána jejich funkčnost a jakých knihoven bylo při vývoji využito. Posledním krokem je testování, které bylo provedeno na čtyřech zařízeních, z nichž jedno bylo tablet. Při něm bylo využito testovacích scénářů, které se vždy soustředili na konkrétní funkci aplikace.

**Klíčová slova:**

Android, mobilní aplikace, Java, SQLite, Realm

# Development of mobile application for car fleet management

**Summary:**

This thesis is focusing on topic of development application for mobile operating system Android. Application created during this work call Spravoz and serve for car fleet management for company, but it can be used also for end users. It offers recording fueling and costs and following evaluation as seven statistics and five graphs. Underlying data can be filtered by date or for individual car whose all car fleet. Application allows also automatic notifications MOT test based on data about first vehicle registration or date of last MOT test.

Theoretical part of the thesis contains basic overview about operating system Android his history and versions. The architection system and conception application are also here. Last part contains methods, how can be saved data in operating system Android.

Practical part maps whole process of development application. First step is writing requirements, next is processing of analysis of the competitive environment. After this analysis follow use case, where are described some of specific application passages. User interface design proposals of application Spravoz are created by using prototypes. Next these proposals are transferred to real specific screens. Architecture of object oriented database is demonstrated by class diagram. Next is analyses of individual class where is described their function and what libraries was used during development. Last step is testing which was done on four devices. One of these devices was tablet. Testing was founded on uses test case which concentrated on specific function application. During this testing were used test scenarios which are always concentrated on specific application function.

**Keywords:**

Android, mobile application, Java, SQLite, Realm

# Obsah

# 1. Úvod

Vzhledem k aktuálnímu vývoji v IT světě se stávají chytré telefony a jejich aplikace nejrychleji rostoucí oblastí. Díky velkému rozvoji již v mnohém dokážou nahradit klasický počítač, a to v přenosné velikosti, která nám umožnuje mít smartphone vždy k dispozici. Proto se stal součástí života většiny populace, především tedy mladší generace, která ho každodenně využívá jak pro osobní, tak pro profesní život.

Samotné zařízení by ovšem neplnilo svou funkci, kdyby na něm nebyl nainstalován některý z operačních systémů pro ně určených. Nejvíce používané systémy v dnešní době jsou Android, iOS a Windows Phone. Nejrozšířenějším systémem na poli chytrých telefonů, ale i tabletů, je operační systém Android od společnosti Google. Tento systém implementují největší výrobci mobilních zařízení, jako jsou Samsung, Huawei, LG a mnoho dalších.

Platforma Android je od počátku vyvíjena na linuxovém jádru jako opensource, což znamená, že si každý může prohlédnout zdrojový kód nebo ho upravit k obrazu svému. Pokud se ho výrobce rozhodne použít na svém zařízení, neplatí za jeho využití žádný licenční poplatek. K platformě je výborně zpracována dokumentace pro vytvoření aplikace, a dokonce Google nabízí zdarma i vývojové prostředí, které se nazývá Android Studio. Pro finální distribuci aplikací k uživatelům slouží obchod Google Play, kde lze najít nepřeberné množství aplikací, kdy je většina poskytována zdarma.

Google má se systémem Android velké plány pro rozšíření do dalších chytrých zařízení. Již dnes tuto platformu můžeme nalézt v chytrých hodinkách, a dokonce i v automobilech. Další rychle rozvíjející se oblastí je bezesporu virtuální a rozšířená realita, v níž Google svůj prototyp produktu pro rozšířenou realitu Google Glass představil již v roce 2012.

# 2. Cíl práce a metodika

## 2.1 Cíl práce

Diplomová práce je zaměřená na problematiku vývoje aplikací pro OS Android, konkrétně na vývoj aplikace pro správu vozového parku. Hlavním cílem je navrhnout a implementovat mobilní aplikaci pro tento operační systém sloužící k usnadnění správy vozového parku.

Další dílčí cíle diplomové práce jsou:

- zdokumentovaní procesu návrhu, vývoje a následné distribuce aplikace,
- analyzovat architekturu systému a aplikací pro operační systém Android,
- charakterizovat různé možnosti ukládaní dat.

## 2.2 Metodika

Metodika zpracování teoretické části je založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude popsána problematika vývoje aplikací v OS Android. Hlavními kapitolami této časti budou ty o architektuře systému, skladbě aplikací a charakteristice možností, jak lze v OS Android ukládat data.

V praktické části práce bude provedena analýza a implementace mobilní aplikace sloužící ke správě vozového parku firmy. Aplikace bude umožňovat zobrazit vyhodnocující statistiky i grafy a také automatické připomenutí STK. Při návrhu a implementaci bude využito standardních metod a nástrojů softwarového inženýrství. Proces vývoje se bude skládat ze specifikace požadavků, analýzy konkurenčního prostředí, scénářů užití, návrhu GUI, implementace GUI a realizace aplikačního a databázového kódů včetně závěrečného testování. To bude probíhat na celkem čtyřech zařízení na základě vytvořených testovacích případy. Aplikace bude postavena na objektové databázi Realm. Výsledná aplikace bude nasazena, otestována a na základě poznatků z nasazení a testů budou formulovány možnosti jejího případného budoucího rozvoje.

# 3. Teoretická východiska

## 3.1 OS Android

OS Android je operační systém založený na linuxovém jádře určený pro mobilní zařízení a je dostupný jako open source. Používá se na nejrůznějších zařízeních od chytrých televizí až po systémy v automobilech. Majoritu ovšem tvoří chytré telefony. O vývoj tohoto systému se stará společnost Google pod hlavičkou konsorcia firem Open Handset Alliance.

## 3.2 Historie

Společnost Android Inc. byla založena v roce 2003. Na počátku stáli čtyři lidé: Andy Rubin, Rich Miner, Nick Sears, a Chris White [1]. Prvním záměrem nově vznikající společnosti bylo vyhotovit operační systém pro digitální fotoaparáty. S touto myšlenkou firma získala v listopadu roku 2004 finanční prostředky od investorů. Společnost ovšem náhle zjistila, že pro jejich velké cíle není trh s fotoaparáty dostatečný. O pět měsíců později se proto zakladatelé rozhodli změnit svůj postoj a začali vyvíjet mobilní operační systém Android, který by konkuroval tehdejším systémům na trhu Symbianu a Microsoft Windows Mobile [2].

V červenci nastává velký zlom. Firma Android je koupena společností Google za nejméně 50 milionů dolarů [3]. Klíčoví lidé původní společnosti Rubin, Miner a White se připojili ke Googlu jako součást akvizice. Andy Rubin se stal vedoucím divize, která měla na starost vývoj operačního systému Android [1]. Spekulace o vstupu společnosti Google na trh mobilních zařízení se táhli až do prosince roku 2006. Prototyp, na kterém v té době společnost pracovala, měl fyzickou QWERTY klávesnici a displej bez podpory dotyku. Jednalo se o zařízení, které bylo svými specifikacemi velmi podobné zařízením od společnosti BlackBerry. Zlom nastal v roce 2007, kdy společnost Apple uvedla svůj revoluční dotykový telefon IPhone. Tým vedený Andy Rubinem musel tedy zcela přepracovat svůj koncept [4].

Open Hadset Allince je konsorcium technologických společností Googlu, výrobců zařízení, mobilních operátorů a výrobců chipsetů. Dne 5. listopadu 2007 tato organizace představila cíl vyvinout první skutečně otevřenou a komplexní platformu pro mobilní zařízení [5]. První komerčně dostupným smartphonem se systémem Android byl HTC

Dream , známý též jako T-Mobile G1, vydaný 23. září 2008. Ten nabízel jak dotykové ovládání, tak plnohodnotnou hardwarovou QWERTY klávesnici [6].


Obrázek 1 - T-Mobile G1 zdroj: (http://cdn2.gsmarena.com/vv/pics/t-mobile/t-mobile-g1-black.jpg)

## 3.3 Verze OS Android

Operační systém Android si během deseti let svého života prošel nepřeberným množstvím různých verzí. Každá verze přidala nové funkce, které systém přiblížily k dokonalosti. Od verze 1.5 Google všechny nově vycházející verze, které přinášejí zásadnější změny, označuje kódovými jmény různých sladkostí či cukrovinek. Dále platí pravidlo, že další verze vždy začíná následujícím písmenem v abecedě. Spolu s nově vydaným systémem je většinou k dispozici i další nová verze vývojářského rozhraní (API) pro tvorbu aplikací třetích stran, přičemž je zaručená zpětná kompatibilita [7].

### OS Android 1.0 a 1.1

Tyto verze se objevily pouze na prvním zařízení T-Mobile G1 [6]. Nenesly ale ještě žádné kódové značení. Již prvotně vydané systémy obsahovaly obchod pro aplikace, který sloužil pro stahování a aktualizaci aplikací třetích stran. Byl ovšem ještě pojmenován jako Android Market, až později došlo ke změně názvu na dnešní Google Play. Systémy obsahovaly jednoduchý webový prohlížeč, Gmail, Google Mapy a aplikaci pro fotoaparát, který ale nenabízel možnosti dalšího nastavení např. změny rozlišení či vyvážení bílé. Tyto verze již dokázaly obsluhovat Wifi a bluetooth konektivitu [8].

### OS Android 1.5 (Cupcake)

Verze Cupcake přinesla důležitou funkci v podobě podpory widgetů a složek. Widgety jsou miniaplikace, které se dají umístit na domovskou stránku s možností periodické aktualizace. Mezi další velké změny lze bezesporu zařadit možnost nahrávání videa do formátů 3GP a MPG-4 a podporu automatického otáčení obrazovky pomocí vestavěného gyroskopu v zařízení [7].

### OS Android 1.6 (Donut)

Toto vydání mělo řadu spíše menších vylepšení. Mezi ty významnější lze zařadit podporu hlasového vyhledávání, možnost běhu na displeji s rozlišením WVGA (480x800 pixelů), které v době vydání bylo považováno za velmi vysoké. Uživatelé z business sféry jistě ocenili podporu VPN pro vzdálené připojení do firmy. Google Mapy byly obohaceny o plnohodnotnou navigaci, která plně dokázala využít technologie GPS [7].

### OS Android 2.0/2.1(Eclair)

Verze Eclair přinesla podporu technologie synchronizace emailu Exchange od společnosti Microsoft. Dále modernizované uživatelské rozhraní, či podporu většího množství nastavení u fotoaparátu např. digitální zoom. Na domovské stránce se nově na pozadí mohla objevit živá tapeta [9].

### OS Android 2.2 (Froyo)

Vydání Froyo bylo oznámeno 20. května 2010 spolu s prvním zařízením řady Nexus, které Google vyvinul spolu s HTC. Tyto telefony se vyznačovaly vždy čistým systémem bez jakýchkoliv nadstaveb. Tuto řadu později nahradily zařízení řady Pixel. Froyo významně vylepšilo rychlost celého systému a správu paměti. Jednou z věcí, která měla velký vliv na rychlejší systém, byla implementace technologie JIT [7]. Ta umožňuje, aby běžící aplikace byla přímo v době provádění přeložena do nativního strojového kódu zařízení, čímž dochází k urychlení celkového běhu [9]. Dle informací z testovacích měření přinesl 4 až 5násobné zrychlení spouštění aplikací [10]. Z dalších vylepšení lze jmenovat možnost instalace aplikací na SD kartu a možnost vytvořit Wi-Fi hotspot pro sdílení mobilního internetu do dalších zařízení [7].

### OS Android 2.3 (Gingerbread)

Verze 2.3 vylepšila uživatelské rozhraní spolu s rychlostí systému. Prvně zde byla představena podpora technologie NFC, jejíž prvotní záměr byl umožnit placení telefonem v obchodech. To se nakonec ukázalo jako dosti velký problém a trvalo dlouhou dobu, než se prosadila technologie Android Pay. V České republice v současnosti (listopad 2017) lze platit přes Android Pay pouze pomocí karet vydaných Monetou, mBank nebo J&T bankou. Mezi další vylepšení lze zařadit podporu dalších typů senzorů jako je např. gyroskop či barometr [7].

### OS Android 3.0 (Honeycomb)

Honeycomb byla první a zároveň poslední verze určená pouze pro tablety. Přinesla rozhraní optimalizované pro tablety spolu s novou grafickou úpravou Holographic. Nabídla také jednodušší multitasking s náhledy aplikací. Vylepšila výkon podporou hardwarové akcelerace, vícejádrových procesorů a umožnila připojit USB zařízení, jako např. flash-disk či klávesnici [12].

### OS Android 4.0 (Ice Cream Sandwich)

V listopadu roku 2011 přišla na trh verze 4.0, která odstranila rozdíly mezi verzemi pro telefony a tablety. Objevilo se zde nové grafické rozhraní Holo spolu s fontem Roboto. Prvním zařízením s tímto systémem byl Galaxy Nexus od společnosti Samsung. Google zde implementoval technologii Android Beam, která využívá technologii NFC pro přenos dat mezi zařízeními pomocí dotyku. Statistiky používání dat se staly podrobnějšími, nyní bylo možno sledovat, kolik MB přenesly jednotlivé aplikace [13].

### OS Android 4.1/4.2/4.3 (Jelly Bean)

Jelly Bean přinesl ještě rychlejší grafické rozhraní díky projektu Butter, který udělal animace plynulejší zvýšením snímkové frekvence na šedesát snímků za sekundu. Prvně se zde objevila podpora více účtů, ale pouze pro tablety. Vylepšila se také zamykací obrazovka, která nyní podporovala widgety i pravý swipe pro otevření fotoaparátu. Aplikace náročné na grafiku především hry mohli nově využít API OpenGL ES 3.0 [14].

### OS Android 4.4 (KitKat)

Kitkat byl vydán v říjnu roku 2013 spolu s novým telefonem Nexus 5, na kterém Google spolupracoval s firmou LG. Novinkou byla transparentní lišta, softwarová tlačítka a také celoobrazový režim. V oblasti výkonu byla vylepšena podpora vícejádrových procesorů. Objevil se zde také požadavek na velikost RAM, která nyní musela mít velikost alespoň 340 MB, doporučených ale bylo 512 MB. V systému se objevila podpora bezdrátového tisku a také technologie Mirecast, která umožňovala bezdrátově přenášet obraz i zvuk do jiného zařízení např. do televize [15].

### OS Android 5.0/5.1 (Lollipop)

Android 5.0 jako první použil nový virtuální stroj ART (Android Runtime) ve výchozím nastavení. Ten nahradil předešlý Dalvik. Ve verzi 4.4 se již tato funkcionalita objevila, ale pouze jako experimentální. Zásadním přínosem je možnost kompilovat část kódu již při instalaci. To způsobí významné urychlení spouštění aplikací. ART má i pozitivní vliv na výdrž zařízení, jelikož se nemusí aplikace kompilovat při každém spouštění a tím znovu zatěžovat procesor. Grafické rozhraní se znovu mění tentokrát na Material design, který má za účel vývojářům pomoci s designem a měl by se objevovat skrz celý systém. Dále byl přidán úsporný režim pro delší výdrž na úkor snížení výkonu [16].

### OS Android 6.0 (Marshmallow)

V listopadu 2015 přichází verze 6.0, ve které je implementována technologie Doze, která snižuje spotřebu baterie tím, že dokáže rozpoznat, pokud telefon odložíme. V tento okamžik přivede zařízení do režimu spánku, při kterém jsou všechny nedůležité aplikace uspané [16]. Mezi další vylepšení bezesporu patří podpora vysokého 4K rozlišení, konektoru USB-C nebo čtečky otisků prstů zvyšující zabezpečení zařízení [7].

### OS Android 7.0/7.1 (Marshmallow)

Verze Marshmallow nabízí možnost práce ve více oknech a také plovoucí okno. Nový systém má rychlejší proces aktualizace díky dvěma oddílům paměti, a tak lze upgradovat bez zpozorování uživatele. Android 7 zavadí další změnu při kompilaci aplikací, nyní je ART doplněn o nový JIT, který způsobuje až o 75 % rychlejší instalaci

aplikace a až o 50% menší velikost zkompilované aplikace. Nově také byla implementována podpora API Vulkan 3D pro ještě lepší grafiku aplikaci. Tato technologie je použita především pro hry, ale také pro virtuální realitu, kde pomáhá k navození pocitu skutečného světa [17].

### OS Android 8.0 (Oreo)

V současnosti (září 2017) poslední verze systému Android s kódovým označením Oreo byla vydána 21. srpna 2017 a řeší především velký problém s pomalou aktualizací systémů ze strany výrobců zařízení. Novinka se jmenuje projekt Treble a nabízí modulární architekturu systému tím, že odděluje implementaci dodavatele (specifické úpravy pro zařízení, software nižší úrovně pro obsluhu hw) od platformy Android přes nové rozhraní. Díky tomu odpadá potřeba aktualizovat velké množství kódu [18]. Další novou funkcí je obraz v obraze, kdy se obrazovka aplikace zmenší do malého plovoucího okna. Obraz v obraze je k dispozici pouze v několika aplikacích, ale nechybí ty nejpoužívanější od společnosti Googlu,

jako je Youtube nebo Chrome. Mezi menší vylepšení patří nový design nastavení s více seskupenými položkami nebo možnost rychlého zoomu dvojitým poklepáním, které je užitečné při jednorukém uchopení telefonu [19].


Obrázek 2- Vývoj verzí systému zdroj: (Android https://geeksnews.co.uk/wp-content/uploads/2016/07/Android-N.png)

## 3.4 Architektura systému

K vývoji aplikací je dobré znát základní strukturu systému, na kterém následně budou aplikace spouštěny. Android se skládá celkem z 6 vrstev, které jsou vzájemně

propojeny. Níže budou uvedeny jednotlivé vrstvy a pro každou bude uvedena krátká charakteristika.

### 3.4.1 Linux Kernel

Nejnižší vrstvu architektury tvoří upravené jádro Kernel, které je převzato z hojně rozšířeného systému Linux, který je používán především na serverech. Jádro bylo upraveno především redukcí funkcí, jelikož mobilní zařízení nepotřebují většinu modulů, co klasický počítač [7]. Některé funkce, které Google připravil pro Android, se naopak zpětně dostaly do jádra Linuxu. Jedná se primárně o funkci pro správu napájení Warelocks [20]. Hlavním úkolem jádra je zajištění přímé interakce s hardwarem mobilního zařízení, čímž zaručuje plnou abstrakci prostředků pro vyšší vrstvy. Na úrovni jádra jsou také implementovány ovladače, jak pro moduly zajištující konektivitu (bluetooth, Wi-Fi), tak i pro senzory (akcelometr, kompas). Dále jádro obstarává přirazení paměti a procesorového času pro jednotlivé běžící aplikace. Jednou ze součástí jádra je i správa napájení, která primárně zajišťuje, aby součásti zařízení, jež jsou energeticky nejnáročnější jako je procesor nebo displej, se při dlouhé nečinnosti vypínaly.

V roce 2017 patří mezi nejvyužívanější verze Kernelu 3.18 nebo 4.4. Jádro pro mobilní zařízení Android, vždy vychází z větve Linuxu s dlouhodobou podporou označované také zkratkou LTS. Vlastní jádro závisí vždy na konkrétním zařízení [20]. Flash úložiště na zařízeních s Android je rozděleno do několika oddílu, jako je např. /system pro samotný operační systém a /data pro uživatelská data a instalaci aplikací třetích stran. Na rozdíl od klasických distribucí Linuxu pro desktopy nejsou uživatelům Androidu dostupná práva root pro přístup do složky /system a je dovoleno z tohoto oddílu pouze číst [22]. Přistup root lze získat pomocí bezpečnostních nedostatků v systému. Této možnosti vyžívá komunita pro úpravu systému a zlepšení vlastností jejich zařízení. Taková úprava je ale nebezpečná, jelikož může škodlivý software, zasahovat přímo do systémového oddílu [23].

### 3.4.2 HAL vrstva

HAL (Hardware abstract Layer) poskytuje standartní rozhraní, které zprostředkovává přístup k hardwaru zařízení pro vyšší vrstvy Java API. HAL se skládá z několika knihoven modulů, z nichž každý implementuje rozhraní pro konkrétní typ hardwaru, jako je např. kamera nebo bluetooth modulu. Tyto moduly jsou načteny

systémem ve vhodnou dobu. Implementace HAL je většinou provedena ve sdílených knihovních modulech (.so souborech) [24].

### 3.4.3 Runtime Android

U zařízení se systémem Android alespoň ve verzi 5.0 (API 21) každá aplikace běží ve vlastním procesu a instanci ART. ART je koncipováno pro běh více virtuálních procesů vykonáním DEX souborů [24]. Tyto soubory jsou tvořeny bytekodém, který je přímo určený speciálně pro Android. Dochází zde k optimalizaci pro běh na mobilním zařízení, jelikož jsou brány v úvahu omezené možnosti napájení a velikost pamětí. DEX soubory vznikly z klasických CLASS a JAR, které jsou typické pro jazyk Java [7].

### 3.4.4 Knihovny

Nativní C/C++ knihovny běží přímo nad jádrem a zpřístupňují služby jádra běžícím aplikacím a ART. Tyto služby jsou včetně grafické podpory (2D, 3D, SGL, OpenGL), přehrávání videa a audio, strukturovaného datového úložiště (SQLite), libc, prohlížeče webových stránek přes jádro WebKit a SSL pro síťovou komunikaci [26]. Knihovny v této vrstvě nahrazují chybějící funkce z původního Linuxového jádra, o které bylo jádro při optimalizaci redukováno [7]. Pokud se vyvíjí aplikace, která vyžaduje C nebo C ++ kód, může se využít nástroje Android NDK pro přístup k některým z těchto nativních knihoven platformy přímo z kódu [25].

### 3.4.5 Aplikační Framework

Celá sada funkcí OS Android je k dispozici přes API napsané v jazyku Java. Tyto API tvoří základní kámen pro tvorbu aplikací. Skrze implementované rozhraní lze opakovaně přistupovat k jádru a systémovým službám včetně následujících: [25].

**Package Manager** – modul správce balíčků je ve skutečnosti databáze, která ukládá informace o aktuálně nainstalovaných aplikací. Vizuálním zobrazením je seznam aplikací v zařízení. Tento modul lze využít, pokud aplikace chce spolupracovat s jinou aplikací např. sdílet údaje nebo zavolat jinou službu.

**Windows Manager** – organizuje okna, ve kterých se zobrazují aplikace. Ty jsou většinou tvořeny více okny. Pokud má aplikace pouze jedno okno, stále se musí zobrazovat systémová lišta s informacemi o signálu či stavu baterie.

**View System** – je modul, který spravuje UI tedy grafické rozhraní, které tvoří širokou paletu společných prvků, jako jsou ikony, tlačítka, prvky na zobrazení i editování textu a mnohé další.

**Notification Manager** – umožňuje všem aplikacím zobrazit vlastní notifikace v systémové liště.

**Activity Manager** – pomocí tohoto modulu, systém Android spravuje zásobník aktivit, které mohou být v různých stavech (starting, running, paused, stopped, destroyed) [7].

### 3.4.6 Systémové aplikace

Poslední vrstvu tvoří systémové aplikace jako je např. emailový klient, kontakty, SMS zprávy nebo webový prohlížeč Chrome. Systémové aplikace nemají v Androidu výhradní právo sloužit jako výchozí, tudíž lze toto privilegium předat aplikacím třetích stran. Příkladem může být změna výchozího webového prohlížeče za jiný, který více vyhovuje preferencím uživatele.

Systémové aplikace také poskytují cenné služby vývojářům, kteří mohou využívat jejich služeb. Pokud tedy chtějí ze své aplikace odeslat SMS zprávu, stačí zavolat správnou funkci a systém se již postará o spuštění výchozí aplikace pro odesílání zpráv [25].

## 3.5 Koncepce aplikace Android

Aplikace pro operační systém Android se skládá ze čtyř základních částí, které jsou realizované formou tříd. Kromě nich jsou zde i další části, bez kterých by aplikace nebyla kompletní a tudíž spustitelná.

### 3.5.1 Aktivita

Aktivita je hlavní třída, která obvykle odpovídá přímo jednomu uživatelskému rozhraní s odpovídající funkčností. Aktivity jsou zamýšlené jako plně znovupoužitelné a zaměnitelné stavební bloky aplikace a mohou být sdíleny mezi různé aplikace. Jsou vytvářeny jako podtřídy třídy Android Activity a musí být implementovány tak, aby byly zcela nezávislé na jiných činnostech v aplikaci. To způsobuje, že jedna aktivita nemůže přímo volat metody nebo přistupovat k datům instance jiné aktivity. Toho může být dosaženo pouze použitím Intents a Content Providers [27].

### 3.5.2 Služby

Pomocí služeb Android implementuje dlouho trvající operace nebo ty, které se vykonávají na pozadí. Díky tomu nepotřebují žádné uživatelské rozhraní. Služby vývojářům umožnují vykonávat dlouho trvající úlohy paralelně a asynchronně s hlavním vláknem [7].

### 3.5.3 Broadcast Receivers

Broadcast Receivers jsou mechanismy, kterými aplikace získává přístup k Broadcast Intents. Každý Broadcast Receivers musí mít konfigurovaný Intent filter, který definuje, jaké typy událostí ho zajímají. Objekt typu Intent zaštiťuje události, které se odehrávají na zařízení [27]. Díky Broadcast Receivers lze například provést nějakou akci, pokud se změní stav připojení k internetu nebo dojde k zapnutí displeje. V systému Android lze také vytvořit vlastní druh události, ten poté předat frameworku a on následně upozorní všechny Broadcast Receivers, které ve svém Intent filter definovaly, že je daná událost zajímá [28].

### 3.5.4 Content providers

Content providers implementují mechanizmus pro sdílení dat mezi aplikacemi. Každá aplikace může poskytnout ostatním aplikacím přístup ke svým podkladovým datům skrze Content providers včetně možnosti přidávat, odstraňovat a dotazovat požadované informace, pokud má aplikace dané oprávnění. Přistup k datům je prováděn přes Universal Resource Identifier (URI) definovaný v Content Provider. Data mohou být sdílena ve formě souboru nebo přístupu do SQLite databáze [27].

### 3.5.5 Android Manifest

Každá aplikace musí obsahovat soubor AndroidManifest.xml ve svém kořenovém adresáři. Soubor manifest poskytuje základní informace o aplikaci systému Android, který si ho musí nejdříve načíst před tím, než spustí některou část kódu. Manifest především popisuje jednotlivé komponenty aplikace včetně aktivit, služeb, broadcast receivers a contact providers. Soubor mimo jiné definuje seznam oprávnění, kterými aplikace může přistupovat do chráněné části API nebo iterovat s jinými aplikacemi. Mezi další atributy, lze zařadit minimální úroveň API, kterou aplikace potřebuje nebo seznam knihoven, které využívá pro svůj běh [29].

### 3.5.6 Zdrojové soubory

Kromě souboru manifest a souborů Dex, které obsahují zkompilovaný kód, Android balíček aplikace bude také obvykle obsahovat sbírku zdrojových souborů. Tyto soubory obsahují zdroje jako jsou obrázky, písma a barvy, které se objevují v uživatelském rozhraní aplikace. Dále je zde obsažena i XML definice UI. Ve výchozím stavu jsou tyto soubory uloženy v adresáři / res [27].

## 3.6 Aktivita a její životni cyklus

Na rozdíl od aplikací určených pro klasické počítače s operačními systémy Windows nebo Linux se v aplikacích pro Android nenachází v kódu žádný jednoznačný vstupní bod, jako je u klasických aplikací metoda main.

Aplikace Android fungují na odlišném principu, jelikož jsou aplikace tvořeny z více nezávislých komponentů – aktivit a služeb. Operační systém sám řídí, v jakém okamžiku budou instance aktivit vytvořeny, kdy budou odsunuty na pozadí nebo zničeny. To Android určuje pomocí metod životního cyklu. Ty mají přesně určené pořadí, v jakém se budou spouštět a stejně tak definovanou událost, při které dojde k jejímu spuštění.

Každá aplikace s grafickým rozhraním se vždy skládá z několika obrazovek. V systému Android je obrazovka reprezentací určité aktivity. Každá aktivita je potomkem třídy android.app.Activity. Jejím úkolem je zobrazovat uživateli údaje z nižších vrstev v požadované formě. Jedna z aktivit je definována jako hlavní. Toho lze docílit pojmenováním aktivity jako MainActivity, což je nejčastěji používaný způsob, nebo úpravou souboru AndroidManifest.xml. Tato aktivita se poté po startu aplikace zobrazí jako první [7].

### 3.6.1 Zásobník aktivit

Pro každou aplikaci, která běží na Android zařízení, udržuje systém její aktivitu v zásobníku aktivit. Když se aplikace spustí, je první aktivita aplikace umístěna na vrchol zásobníku. Při spuštění druhé aktivity je umístěna do horní časti zásobníku a předchozí aktivita se posouvá dolů. Aktivita na vrcholu se označuje jako aktivní (nebo běžící) aktivita. Když běžící aktivita skončí, vyskočí ze zásobníku a aktivita umístěna bezprostředně pod ní se stává aktuální aktivitou. Dalším příkladem, kdy první aktivita opouští zásobník může být, pokud úloha, za kterou odpovídá, byla dokončena nebo

uživatel stiskl tlačítko zpět, aby se vrátil na předchozí aktivitu. Vizuální znázornění zásobníku aktivit Android je vyobrazeno na obrázku níže [27]:


Obrázek 3 - Zásobník aktivit zdroj: (Vlastní tvorba)

Jak je znázorněno na schématu, nové činnosti jsou při spouštění postaveny na vrchol zásobníku. Běžící aktivita se nachází na vrcholu, dokud není dokončena nebo uživatel nepřepnul na jinou aktivitu. V případě, že hardwarové zdroje budou omezeny, runtime ukončí aktivity počínaje těmi ve spodní části zásobníků. Zásobník aktivit funguje na principu, který se v terminologii programování označuje jako LIFO (LastIn – FirsOut), jelikož první položka, která má být vložena do zásobníku je i první, která zásobník opouští [27].

### 3.6.2 Životní cyklus

Životní cyklus aktivity je v rámci systému Android realizován pomocí metod, které jsou součástí třídy Activity. Tyto metody se poté spouštějí v přesně definovaných situacích v určeném pořadí.

Aktivita se během svého životního cyklu může dostat do těchto tří fází:

- **Aktivita na popředí** – zobrazuje se na displeji zařízení a interaguje s uživatelem, což znamená, že aplikace reaguje na uživatelovi vstupy svými výstupy. Této fázi odpovídají stavy Running nebo Resumed.
- **Pozastavená aktivita** – je stále viditelná a uložená v paměti, ale ztrácí interakci s uživatelem. Příkladem je situace, kdy aplikace zobrazí dialogové okno, a tak se původní aktivita stává pozastavenou. Systém má ovšem právo tyto aktivity odstranit, pokud dojde k extrémnímu nedostatku paměti.
- **Zastavená aktivita** – je úplně překrytá jinou aktivitu. Stále si ale zachovává všechny informace o svém stavu, a tudíž není problém se k ní později vrátit. Pokud není aktivita dlouhodobě zobrazena uživateli, dochází často k odstranění z paměti, pokud je paměť potřeba pro jiné aplikace [7].

Pro navigaci přechodů mezi jednotlivými stavy životního cyklu poskytuje třída aktivity základní balíček šesti zpětných volání onCreate(), onStart(), onResume(), onPause(), onStop(), a onDestroy() [30].

**onCreate()**

Metoda se zavolá při prvotním vytvoření aktivity. Na pozadí se vytvoří rozhraní a konfigurují se proměnné a objekty potřebné k běhu aktivity. V této metodě lze implementovat základní logiku spouštění aplikace, která by měla proběhnout pouze jednou za celou dobu trvání aktivity. Tato metoda přijímá parametr savedInstanceState, který je typu Bundle objekt obsahující dříve uložené stavy aktivity. Pokud aktivita předtím nikdy neexistovala, nabývá proměnná savedInstanceState hodnoty null. Po onCreate() metodě následuje vždy metoda onStart() [30].

**onStart()**

Když aktivita vstoupí do stavu Started, systém volá tuto metodu. Spouští se těsně před tím, než se aktivita stane viditelná pro uživatele. Probíhá zde inicializace ostatního kódu, který neproběhl v metodě onCreate(), udržuje se uživatelské rozhraní nebo lze v této metodě zaregistrovat BroadcastReceiver [31]. Metoda onStart() se ve většině případech dokončí velmi rychle a následně se spouští metoda onResume() [30].

**onResume()**

Po spuštění této metody aktivita přejde do popředí a probíhá interakce s uživatelem. Aplikace zůstává v tomto stavu, dokud nepřijde nějaká událost, ze strany systému nebo ze strany uživatele, která aktivitu odsune. Takovou událostí může být například přijetí telefonního hovoru, přepnutí uživatele do jiné aplikace nebo vypnutí obrazovky zařízení. Pokud dojde k tomuto přerušení aplikace, aktivita se dostává do pozastaveného stavu a systém vyvolá metodu onPause() [30].

**onPause()**

Systém volá metodu onPause() jako první indikaci, že uživatel opouští aktivitu, i když to vždy neznamená, že je činnost zničena. Existuje několik důvodů, proč může aktivita vstoupit do tohoto stavu. Příkladem může být, když událost přeruší spuštění jiné aplikace, jak je popsáno v části onResume(). Tento případ je nečastější především pro situace, kdy uživatel stiskne domovské tlačítko pro návrat do launcheru [32]. V systému Android 7.0 a vyšších nastává vyvolání metody při běhu více aplikací v režimu oken, jelikož pouze jedna aplikace je aktivní a všechny ostatní jsou v pozastaveném stavu. Další situací může být, pokud se objeví nová poloprůhledná aktivita např. dialog, ale původní aktivita je stále viditelná, zůstává tato aktivita ve stavu pozastavena.

Provádění metody onPause() je většinou velmi krátké a není zde zaručen dostatek času k uložení dat. Z tohoto důvodu se nedoporučuje využívat tuto metodu k ukládání dat aplikace nebo uživatelských dat, či provádění databázových transakcí. Tyto operace, které jsou výkonově náročnější, a jejich provedení zabere delší časový úsek, by se měly přesunout do metody onStop(). Dokončení onPause() metody ovšem neznamená, že aktivita opouští pozastavený stav. Aktivita setrvává v tomto stavu, dokud se neobnoví nebo se zcela neznevidtelní. Pokud se aktivita obnoví, systém znovu zavolá metodu onResume(). Jelikož systém uchovává instanci aktivity v paměti, není po obnovení potřeba znovu inicializovat proměnné [30].

**onStop()**

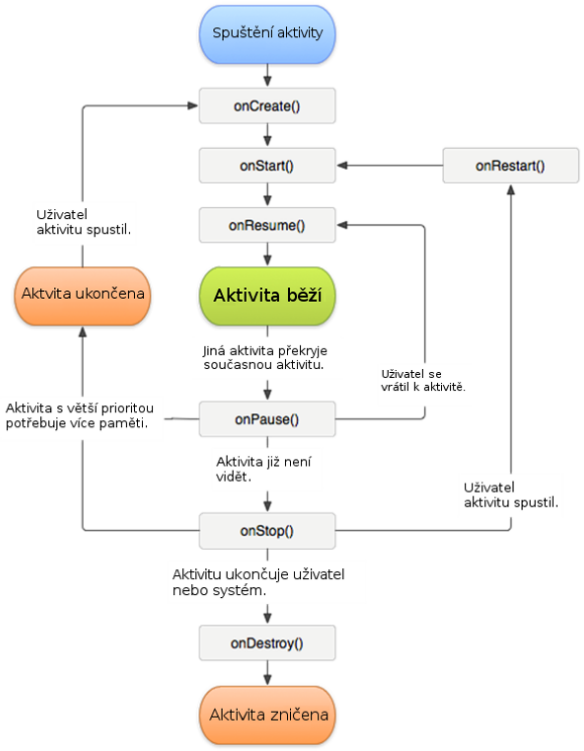
Když se aktivita stane pro uživatele neviditelná, vstoupila do stavu zastavená a systém volá onStop(). V této metodě by mělo dojít k uvolnění téměř všech zdrojů, které nejsou potřebné, pokud uživatel aplikaci nepoužívá. Jako příklad lze uvést registraci

BroadcastReceiver. Aplikace již nezobrazuje své uživatelské rozhraní a nemůže tedy reagovat na události, proto tato registrace ztrácí svůj smysl.

Pokud aplikace vstoupí do stavu zastaveno, aktivita zůstává v paměti, udržuje všechny informace o stavu, ale není připojena ke správci oken. Není tedy potřeba znovu inicializovat součásti aplikace. Systém také udržuje přehled o aktuálním stavu každého view, takže pokud uživatel zadal text do EditText widgetu, zůstane tento obsah zachován a není potřeba jej obnovovat. Ze stavu zastavená se aktivita buď vrací k interakci s uživatelem anebo je ukončena. Pokud se obnovuje, systém vyvolá metodu onRestart() [30].

**onDestroy()**

Tato metoda je volána před zničením aktivity. Systém vyvolá onDestroy(), protože aktivita je ukončena z důvodu zavolání metody finish() nebo Android dočasně zničí proces obsahující tuto aktivitu, aby ušetřil místo v operační paměti. Dalším případem volaní je změna orientace zařízení, kdy se po zavolání onDestroy() okamžitě zavolá onCreate() metoda pro znovu vytvoření aktivity v nové orientaci [30]. Při ukončení aktivity nemusí vždy předcházet volání metody onDestroy() [27].




Obrázek 4 - Životní cyklus aktivity zdroj: (https://www.itnetwork.cz/images/17568/lifecycle.png)

## 3.7 Intent

Komponenty aplikace pro android se mohou připojit k jiné android aplikaci. Toto spojení je založeno na využití objektů intent.

Intenty jsou asynchronní zprávy, které umožňují komponentám aplikace požadovat funkčnost jiné aplikace. Záměr může obsahovat data, která jsou uložena prostřednictvím instance třídy Bundle. Tato data mohou být použita přijímací komponentou. Příkladem použití může být spuštění externí aktivity pro pořízení snímku.

### 3.7.1 Typy Intentů

Android podporuje dva typy intentů a to explicitní a implicitní. Aplikace může definovat cílovou komponentu přímo v záměru (explicitně) nebo požádat systém Android o vyhodnocení registrovaných komponent na základě údajů o intentu (implicitně) [33].

Explicitní intent lze vytvořit např. na základě tohoto kódu:

```
Intent i = new Intent(context, MainActivity.class);
```

S explicitními intenty systém okamžitě rozpozná, jakou aktivitu chce aplikace vyvolat.

Implicitní intenty na rozdíl od explicitních umožňují definovat pouze záměr, a nikoliv přesný postup, jak ho provést. Toto řešení usnadňuje práci vývojářům a také umožňuje vzájemnou interoperabilitu aplikací v rámci systému, na které si Android silně zakládá.

Příkladem může být, pokud chceme v rámci aplikace zobrazit stránku https://www.czu.cz/cs/. Kdyby systém nabízel pouze explicitní intenty, byl by to dosti komplikovaný problém pro vývojáře, jelikož by to pro ně znamenalo projít všechny možné prohlížeče a spustit až ten, který by byl v rámci zařízení dostupný. Tento postup lze označit za nepohodlný, a především neumožňuje uživateli si vybrat svůj preferovaný prohlížeč.

Když ovšem existují implicitní intenty je vyřešení tohoto problému velmi jednoduché:

```
Uri uri = Uri.parse("https://www.czu.cz/cs/");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
```

```
startActivity(intent);
```

V rámci kódu byl použit konstruktor Intent(String action, Uri uri), který nám umožnuje předat v rámci vytvoření instance tyto dva parametry; název implicitní akce a potom nějaké URI. Na základě předaných informací v intentu systém zjistí, které aktivity mají nastavený takový intent filter, který by vyhovoval předaným parametrům. Pokud existuje jen jedna, spustí ji, pokud není žádná, nastává zaslání výjimky ActivityNotFoundException. Pokud je jich více, zobrazí se uživateli dialog pro výběr z nabízených aplikací [34].

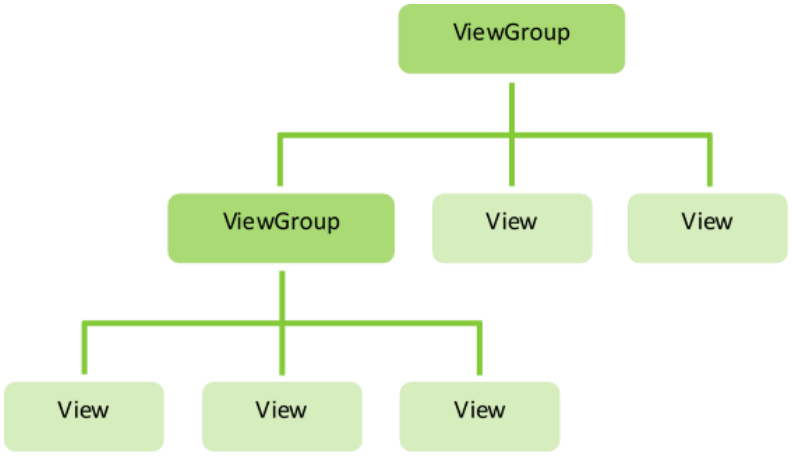
## 3.8 Grafické rozhraní aplikace

Jednou z nejdůležitějších částí každé aplikace je design a vývoj grafického rozhraní. Mnoho nejpoužívanějších aplikací pro Android se stalo populárními hlavně na základě jejich designu, animované grafiky a jejich jednoduchosti a snadného používání. Níže budou uvedeny třídy, které poskytují základnu pro grafické rozhraní aplikace [35].

### 3.8.1 Třídy View a ViewGroup

Všechny prvky uživatelského rozhraní aplikací pro Android jsou postaveny na použití objektů View a ViewGroup. View je objekt, který vykresluje prvky na obrazovce, s nimiž může uživatel interagovat. ViewGroup obsahuje jiné View objekty, aby definoval rozložení rozhraní.

Uživatelské rozhraní pro každou komponentu vyvíjené aplikace je definováno pomocí hierarchie objektů View a ViewGroup, jak je znázorněno na schématu níže. Každá skupina View je neviditelný kontejner, který organizuje podřízené View, kdy potomci View mohou být ovládací prvky nebo jiné widgety. Za jejich pomoci je vykreslena část UI. Tento hierarchický strom může být tak jednoduchý nebo složitý, jak daná situace vyžaduje, ovšem pro výkon aplikace je nejlepší hierarchie jednoduchá.




Obrázek 5 - Hierarchie View zdroj: (Vlastní zpracování)

Při deklarování rozvržení UI lze vytvořit instance třídy View a přímo v kódu začít tvořit strom. Jednodušší a účelnější způsob je definovat rozvržení pomocí souboru XML. Ten nabízí podobu struktury rozvržení, která je lépe čitelná pro lidi podobně jako HTML. Výhodou tohoto řešení je oddělení návrhu uživatelského rozhraní a samotného aplikačního kódu. Tím je umožněno upravit rozložení jednotlivých obrazovek bez nutnosti znovu kompilovat kód.

Název XML elementu odpovídá třídě Android, kterou představuje. Příkladem může být element <TextView>, který vytvoří TextView widget v uživatelském rozhraní a element <LinearLayout> vytvoří LinearLayout skupinu zobrazení [36].

Pro příklad, jednouchý vertikální layout s text view a tlačítkem vypadá takto:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Text v TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:text="Tlačítko1" />
</LinearLayout>
```

Každý objekt View nebo ViewGroup podporuje vlastní variantu atributů XML. Některé atributy jsou specifické pro konkrétní tento objekt např. TextView podporuje textSize atribut. Některé jsou společné pro všechny objekty View, protože jsou zděděny z kořenové třídy View jako je ID atribut.

Každý objekt View může mít ID, které jednoznačně identifikuje View ve stromu. Díky tomuto atributu můžeme odkazovat na příslušnou komponentu v aplikačním kódu Java nebo i v definici rozvržení XML. Když dochází ke kompilaci aplikace, pracuje se s tímto ID jako s integerem, ale obvykle je ID přiřazena hodnota typu String. Syntaxe pro ID je následující:

```
android:id="@+id/tlacitko1"
```

Zavináč na začátku řetězce značí, že by XML parser měl oddělit zbytek řetězce a identifikovat jej jako zdroj ID. Plus oznamuje nástroji appt, který je součástí Android SDK, že se jedná o nový název zdroje. Na základě toho appt přidá referenci do generovaného souboru R.java.

Při kompilaci aplikace je každý soubor XML layout kompilován do View zdroje. Rozvržení je poté načteno v metodě Activity.onCreate() pomocí zavolání metody setContentView(), které se předá reference na zdroj layout ve formátu R.nazev_layoutu. Níže je uveden příklad načtení XML layoutu s názvem main_layout.xml [37].

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

### 3.8.2 Kontejnery

Pomocí kontejnerů je definován způsob organizace kolekcí vybraných widgetů. Z pohledu definice rozložení v XML jsou kontejnery využité jako kořenové elementy, do kterých lze vkládat widgety. Systém Android nabízí velké množství různých kontejnerů, přičemž ty nejpoužívanější budou níže charakterizovány.

#### 3.8.2.1 Linear Layout

Tento kontejner řadí widgety nebo vnořené kontejnery do řádku nebo sloupce jeden pod druhý. Orientace řazení se nastavuje pomocí atributu android:orientation, kdy hodnota horizontal znamená seskupování do řádku a vertical do sloupce. Orientaci lze měnit i v Java kódu zavoláním metody setOrientation() a předaním parametru HORIZENTAL nebo VERTIKAL.

#### 3.8.2.2 Relative Layout

Kontejner RelativeLayout rozmisťuje widgety na obrazovce pomocí vztahů k ostatním widgetům v kontejneru a na základě vztahů k nadřízenému kontejneru. Pozice každého prvku se zde definuje pomocí těchto atributů:

- android_layout_alignParentTop: Zarovnaní horní hrany widgetu s horní hranou kontejneru.
- android_ layout_centerInParent: Zarovnaní widgetu na střed kontejneru.
- android_ layout_alignTop: Zarovnání horní strany widgetu s horní stranou widgetu určitého id (ve tvaru @id/jmeno_widgetu).
- android_ layout_above: Umístění widgetu nad widget určený id.
- android_ layout_toLeftOf: Umístění widgetu nalevo od widgetu určeného id.

Výše určené atributy jsou pouze pro klíčové slovo Top. Samozřejmě tyto atributy existují i pro ostatní zarovnání jako je bottom, above, bellow, left, right.

#### 3.8.2.3 Table Layout

Podobný systém jako jsou tabulky v jazyce HTML, nabízí i Android. V definici kontejneru se určují počty řádků a sloupců mřížky. Kontejner TableLayout ovládá celkové chování kontejneru, ale samotné widgety jsou vkládány do kontejneru TableRow, který reprezentuje jeden řádek tabulky. Mají-li jednotlivé řádky definovaný atribut layout_span, mohou tyto prvky přesahovat do dalších sloupců tabulky.

#### 3.8.2.4 Grid Layout

Kontejner GridLayout rozmisťuje obsah do mřížky tvořené nekonečným množstvím tenkých čar, které rozdělují oblast celého kontejneru do buněk. Jejich počet závisí na nastavení hodnot vlastností rowSpec a columnSpec. Díky tomu lze vytvářet

rozložení, která napodobují jednoduchou tabulku. V případě, kdy aplikace vyžaduje vysokou přesnost, lze vytvořit tisíce i miliony buněk [38].

## 3.9 Vývojová prostředí

Pro tvorbu aplikací pro OS Android lze využít mnoho různých vývojových prostředí, které nabízejí možnost vyvíjet i v jiných jazycích, než je nativní Java. Společnost Google doporučuje prostředí Android Studio, které ovšem vždy nemusí být tou nejlepší volbou. Následující seznam obsahuje šest nejznámějších vývojových prostředí s krátkým popisem [39].

### 3.9.1 Android Studio

Oficiální volba pro vytváření aplikací je Android Studio. Jedná se o IDE vyvíjené společnosti Google a veškerá oficiální dokumentace odkazuje na toto prostředí. Má největší podporu a je založené na platformě JetBrains' IntelliJ IDEA, která byla speciálně upravena na vývoj pro Android [36]. Existuje ve verzích pro všechny tři nejpoužívanější operační systémy Windows, Mac OS X i Linux. Android Studio nabízí mnoho užitečných funkcí jako např. testování kompatibility, propracovaný editor uživatelského rozhraní s možností drag-and-drop nebo přímá podpora vývoje pro Android Wear. Vývojové prostředí Android Studio integruju buildovací nástroj Gradle, který umožňuje automatizovat proces sestavení apk souboru. Velkou výhodou je správa závislostí, kdy si Gradle před procesem sestavení sám stáhne zdrojové kódy knihoven, které jsou při buildu nepostradatelné.

Další součástí studia je emulátor zařízení, který umožňuje vytvořit virtuální prostředí pro testování aplikace. Emulátor lze plně konfigurovat od změny verze API, velikosti RAM, velikosti úložiště až po napojení fotoaparátu virtuálního zařízení na kameru počítače. Nevýhodou emulátorů spočívá, že plynulost běhu je relativně nízká i na výkonném počítači [41].

### 3.9.2 Eclipse

Vytváření aplikací v Eclipse je velmi podobné jako při použití Android Studia a před jeho příchodem bylo toto prostředí hlavní volbou. Nastavení je podobné a postup vývoje je srovnatelný jako v IDE od Googlu. Rozdíl je v tom, že Eclipse nebyl postaven speciálně pro vývoj Androidu a může být používán pro řadu různých platforem a jazyků.

To sebou nese nutnost stahování doplňků včetně nastavování Android SDK, a proto nelze doporučit začátečníkům. Vzhledem k faktu, že se již nejedná o oficiální vývojové prostředí, je obecně doporučovaný přechod na Android Studio, které nabídne více pokročilých funkcí [40].

### 3.9.3 Xamarin

Xamarin je IDE od společnosti Microsoft, které je navržené, tak aby usnadnilo vytváření aplikací, které mají být určeny pro více platforem. Prostřednictvím Xamarinu lze vyvíjet aplikace pro Android a snadno je přesouvat na iOS a Windows, pomocí stejného kódu a rozhraní API. V Xamarinu se vyvíjí pomocí jazyka C#, který se poté kompiluje do nativních aplikací různých platforem. Na začátku dubna 2016 byla oznámena integrace do Micorsoft Visual Studia a to znamená, že od této doby je Xamarin v základní verzi poskytován zdarma a Xamarin runtime se stává open source [42].

### 3.9.4 AIDE

Zkratka AIDE znamená Android IDE. Jedná se o základní vývojové prostředí běžící přímo na platformě Android. To znamená, že ho lze využívat na mobilních zařízeních na cestách. Výhodou je snadné testování aplikací, jelikož není potřeba žádného emulátoru a také vestavěný tutoriál, který ukáže základy ovládání. Bohužel další lekce a pokročilé funkce jsou zpoplatněny formou předplatného. Toto prostředí se bezesporu nehodí na vývoj velkých projektů, ale na malé úpravy nebo vytvoření jednoduché aplikace lze použít. Velkým limitem je především velikost obrazovky mobilního zařízení, proto je určitě lepší použití tabletu nebo externí klávesnice [43].

### 3.9.5 Apache Cordova

Apache Cordova je nástrojem umožňující multiplatformní mobilní vývoj pomocí JavaScriptu. Podporovanými systémy jsou jak tři největší Android, iOS, Windows tak i řada menších. Cordova vznikla ve firmě Nitobi pod názvem PhoneGap, později byla tato společnost odkoupena firmou Adobe. Tato firma následně vydává pod názvem Adobe PhoneGap pouze nadstavbu s plnou podporu. Samotnou technologii převádí pod názvem Cordova pod společnost Apache a ta jej vydává jako open source.

Mobilní aplikace v Cordově se vyvíjí velmi podobně jako webové stránky. Obsah se tvoří pomocí HTML, následně se formátuje v CSS a lze s ním dynamicky manipulovat pomocí JavaScriptu. Na rozdíl od skutečných webových stránek je k dispozici možnost

využít hardware zařízení jako je fotoaparát, akcelometr či GPS lokaci. Při sestavování aplikace pro určitou platformu vytvoří Cordova nativní aplikaci, která obsahuje v podstatě jen prvek typu WebView, který zobrazuje vytvořené stránky aplikace. Proto se často označují tyto aplikace jako hybridní, jelikož výsledkem je normální mobilní aplikace, kterou lze publikovat v příslušném obchodě. Současně jde o aplikaci, kde kód není překládán do instrukcí procesoru, ale za běhu interpretován příslušným prohlížečem. Z toho plyne fakt, že aplikace nemohou využít celého výkonu zařízení [44].

### 3.9.6 Unity

Unity je herní engine a vývojové prostředí pro tvorbu multiplatformních her. Využití najde především při tvorbě her s realistickou fyzikou, 3D grafikou, dynamickými světly atd. Při vývoji poměrně jednoduché hry se tvorba může proměnit pouze na pouhé přetahování elementů. Kromě toho lze pomocí přidaní C# nebo Java kódu změnit způsob, jak se jednotlivé prvky chovají. Vestavěný obchod umožňuje stáhnout/zakoupit 3D modely, skripty, efekty a další. Publikování na Andorid je stejně jednoduché jako propojení se sadou SDK a to výběrem platformy z rozbalovací nabídky. Pro Unity existuje online podpora a velké množství video tutoriálů [39].

## 3.10 Možnosti ukládaní dat v systému Android

Každá netriviální aplikace většinou potřebuje ukládat data. Tato data mohou mít různé podoby; ať už jsou to nastavení uživatele, nastavení aplikace, uživatelská data, obrázky nebo vyrovnávací paměť pro data načtená z internetu. Některé aplikace mohou generovat data, která v konečném důsledku patří uživateli a upřednostňují tak ukládaní dat na veřejné místo, ke kterému může uživatel kdykoliv získat přístup, a to i pomocí jiných aplikací. Opakem je situace, kdy aplikace ukládá data, ale zároveň nechce, aby mohla být přečtena nějakou jinou aplikací. Platforma Android proto nabízí vývojářům více způsobů ukládání dat, přičemž každý způsob má své výhody i nevýhody.

Aplikace pro systém Android mají čtyři základní možnosti ukládání dat:

### 3.10.1 Shared preferences

Slouží pro uložení primitivních dat v párech klíč-hodnota. Ke klíči, jenž musí být typu String, je přiřazena hodnota, která může být jedním z následujících typů: boolean, float, int, long nebo String. Vnitřně platforma Android ukládá shared preferences do

souboru formátu XML do soukromého adresáře. Aplikace mohou využívat více souborů pro uložení shared preferences. Tento způsob uložení se nejčastěji uplatňuje pro uchování předvoleb aplikace.

#### 3.10.1.1 Použití shared preferences

Prvním krokem je vytvoření instance třídy ShraredPrefences; například s názvem předvolby.

```
SharedPreferences predvolby = getPreferences(MODE_PRIVATE);
```

Druhým krokem je uložení požadovaných hodnot do SharedPrefences. Pro příklad jsou zde uvedeny hodnoty jmeno_aplikace:gmail.

```
SharedPreferences.Editor editor = predvolby.edit();
editor.putString(“jmeno_aplikace“, “gmail“);
editor.commit();
```

## 3.11 Vnitřní úložiště

Existuje spousta situací, kdy chceme uchovávat data, ale shared preferences jsou příliš omezující. Příkladem může být uložení objektů Javy, obrázků nebo dat, které si musí zachovávat jejich hierarchii. Metoda uložení ve vnitřní paměti je speciálně určená pro situace, kdy je potřeba ukládat data do souborového systému zařízení, ale vývojář aplikace nechce, aby tato data byla čtena jinou aplikací nebo uživatelem. Data uložená pomocí této metody jsou zcela soukromá a při odinstalaci aplikace jsou odstraněna ze zařízení [45].

### 3.11.1 Použití interního úložiště

Pro zápis do interního úložiště se využívá třídy FileOutputStream. V uvedeném příkladě se bude zapisovat řetězec “Výsledek je 50“ do souboru souborA. Pro uložení řetězce je potřeba využít metody getBytes(), která převede řetězec na proud bitů. Jelikož se jedná o nebezpečnou operaci, která by mohla vést k pádu celé aplikace je potřeba jí ošetřit pomocí bloku try a catch a případně vyhodit výjimku [46].

```
String jmenoSouboru = "souborA";
String text = "Výsledek je 50";
FileOutputStream outputStream;
```

```
try {
  outputStream = openFileOutput(jmenoSouboru, Context.MODE_PRIVATE);
  outputStream.write(text.getBytes());
  outputStream.close();

}

catch (Exception e) {

  e.printStackTrace();
}
```

## 3.12 Externí úložiště

Naopak existují případy, kdy je požadováno, aby uživatel mohl prohlížet data a soubory uložené aplikací. Pokud vyvíjená aplikace bude ukládat data do externího úložiště, musí požádat o oprávnění WRITE_EXTERNAL_STORAGE. Toto oprávnění uděluje přístup pro čtení i zápis. V případě, že aplikace bude pouze číst, a ne zapisovat, lze využít právo READ_EXTERNAL_STORAGE. Od verze Android 4.4 je možno zapsat do soukromé složky externího úložiště bez požadavku na WRITE_EXTERNAL_STORAGE. Tuto složku lze bez problému číst jinými aplikacemi, ale není snímána skenerem medii. Nachází se v adresáři Android/data a je také odstraněna při odinstalaci [45].

Počínaje verzí Nougat (Android 7.0) mohou aplikace uživatele žádat o přístup k určitému adresáři, nikoliv k celému externímu úložiště. Tím je zajištěna vetší bezpečnost, jelikož aplikace poté mohou přistupovat pouze do adresářů, ke kterým dostanou přístup [46].

### 3.12.1 Použití externího úložiště

Princip využití externího úložiště je velmi podobný jako u interního. Rozdíl je v tom, že externí úložiště mohou být vyměnitelnými jednotkami a obsah je možné číst všemi aplikacemi. Vzhledem k tomu, že externí úložiště může být kdykoliv odstraněno anebo sdíleno s počítačem, měla by aplikace zkontrolovat před zápisem, zda je toto médium dostupné.

Kód funkce, která zjišťuje, jestli je možné zapisovat:

```
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

Android má výchozí složky pro různé typy dat například fotky, hudbu, vyzvánění a dokumenty. Při ukládání dat by se měly primárně využívat tyto složky. Pro zjištění konkrétní cesty se využívá třídy Enviroment, například adresa pro obrázky se získá pomocí Enviroment.DIRECTORY_PICTURES.

Adresu soukromé složky je možno získat pomocí funkce GetExternalFilesDir() a veřejnou zavoláním getExternalStoragePublicDirectory(). Samotný kód pro zápis a čtení se nijak neliší od ukládání dat do interního úložiště [45].

## 3.13 SQLite

SQLite je knihovna, která implementuje soběstačný, bezserverový, transakční databázový stroj SQL. Kód pro SQLite je veřejně dostupný a je poskytován zdarma k jakýmkoliv účelům ať už komerčnímu nebo soukromému. Na rozdíl od většiny ostatních SQL databázích SQLite nemá samotný serverový proces. Čte a zapisuje přímo do běžných souborů na disku. Kompletní databáze s více tabulkami, indexy a pohledy je obsažena v jediném souboru.

SQLite implementuje většinu standardů SQL a podporuje skoro celý standart SQL-92, nezaručuje ovšem doménovou integritu. Například zde existuje ALTER TABLE, ale neumožňuje editaci či mazání sloupců. Velkou výhodu SQLite je, že zabírá velmi málo místa (do 1 MB) a většinou není potřeba žádná konfigurace. Díky tomu, že celá databáze je pouze souborem, jde snadno zálohovat i distribuovat. Další nespornou výhodu je do jisté míry také podpora paralelního přístupu, který na rozdíl od podobných řešení SQLite nabízí. Tím je umožněno, aby z jedné databáze četlo najednou více instancí.

Jako hlavní nevýhodu lze určitě zmínit nízký výkon, který se projeví především při zápisu velkého množství dat. SQLite bere každý insert jako vlastní transakci a dle

dokumentace je každá transakce dokončena, až tehdy kdy jsou data bezpečně uložená na disku. Z toho vyplývá, že je zaručena určitá bezpečnost přenosu, ale na druhou stranu je nízký výkon silně omezující faktor. Tuto situaci lze řešit tím, že do jedné transakce zařadíme více insertů pomocí SQL příkazů begin transaction a commit. Tímto krokem je možné dosáhnout výrazného zrychlení vkládání velkého množství dat. Další nevýhodou je uzavření celé databáze při probíhajícím zápisu. Nelze tedy paralelně zapisovat a v tuto dobu není umožněno ani čtení z databáze. SQLite je jeden z nejvíce rozšířených databázových strojů, jelikož se dnes využívá v řadě operačních systémů nebo vestavěných systémech (Google Chrome, Windows 10, Drupal) [47].

### 3.13.1 Použití SQLite

Operační systém Android plně podporuje SQLite databáze. Všechny databáze, které budou vytvořeny, jsou následně přístupné z jakékoliv třídy v rámci aplikace, ale nikoliv mimo aplikaci. Doporučená metoda pro vytvoření nové databáze SQLite je vytvořit podtřídu SQLiteOpenHelper a přepsat onCreate() metodu, ve které by se měl provést SQL příkaz pro vytvoření tabulek v databázi [46]. Například:

```
public class SampleSQLiteDBHelper extends SQLiteOpenHelper {
    private static final int DATABAZE_VERZE = 2;
    public static final String DATABAZE_JMENO = "databaze1";
    public static final String OSOBA_TABULKA_JMENO = "osoba";
    public static final String OSOBA _SLOUPEC_JMENO = "jmeno";
    public static final String OSOBA _SLOUPEC_VEK = "vek";

    public SampleSQLiteDBHelper(Context context) {
        super(context, DATABAZE_ JMENO, null, DATABAZE_VERZE);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL("CREATE TABLE " + OSOBA_TABULKA_JMENO
                " (" + OSOBA _SLOUPEC_JMENO + " INTEGER PRIMARY KEY AU
                TOINCREMENT, " + OSOBA _SLOUPEC_JMENO + " TEXT, " +
                PERSON_COLUMN_AGE + " INT UNSIGNED, " + ")");
    }
```

```
}
```

Pro přidání dat lze využít níže uvedenou metodu. V tomto případě přidáme záznam do tabulky Osoba se jménem Karel Novák a věkem 22. Po uložení bude aplikace uživatele informovat pomocí toastu o úspěšném dokončení [45].

```
private void saveToDB() {
        SQLiteDatabase databaze = new SampleSQLiteDBHelper(this).getWr
        itableDatabase();
        ContentValues values = new ContentValues();
        values.put(SampleSQLiteDBHelper.OSOBA_SLOUPEC_JMENO, "Karel N
        ovák");
        values.put(SampleSQLiteDBHelper.PERSON_COLUMN_AGE, "22";);
        long newRowId = database.insert(SampleSQLiteDBHelper.OSOBA_TAB
        ULKA_JMENO, null, values);

        Toast.makeText(this, "ID nového řádku je " + newRowId, Toast.L
        ENGTH_LONG).show();
}
```

## 3.14 Realm

Realm je objektová databáze, která je od začátku vyvíjena především pro mobilní zařízení. Na rozdíl od ORM, který je pouhou abstrakcí databáze postavené nad SQLite, Realm je úplně nový databázový stroj, který nevyužívá relačního přístupu. Jeho jádro se skládá ze samostatného knihovny C++. V současné době podporuje platformy Android, iOS (Objective-C i Swift), Xamarin a React Native.

Zatímco technologie databázových serverů procházely revolucí od roku 2007 a objevily se mnohá vylepšení, databázové technologie pro mobilní zařízení zůstaly u SQLite a jeho obalech. To byla jedna z klíčových motivací, která vedla zakladatele Realmu k vytvoření nového řešení [48].

### 3.14.1 Výhody Realmu

Níže budou uvedeny oblasti, ve kterých Realm nabízí lepší řešení, než poskytují ostatní řešení.

#### 3.14.1.1 Snadné modelování

Každý vytvářený objekt databáze musí dědit z třídy RealmObject. Realm přijímá všechny primitivní typy proměnných (kromě char) a také String, Date a Byte[]. Podporuje také podtřídy RealmObject a RealmList<? Extends RealmObject> pro modelování vztahů mezi objekty.

Jednotlivé atributy mohou mít libovolnou úroveň přístupu (private, public a protected). Všechna pole jsou ve výchozím nastavení uchovávaná a lze je doplnit poznámkami pro speciální pole např. pro primární klíč @PrimaryKey.

#### 3.14.1.2 Vztahy mezi objekty

Pokud jde o vztahy mezi objekty lze v Realmu využít dvou přístupů:

- První možností je přidat jiný objekt jako atribut. Tento způsob dovoluje realizovat vazby typu 1:1 nebo 1:N.
- Druhým přístupem je přidání RealmList jako atributu. RealmList se chovají jako objekty typu List v Javě. Působí tedy jako kontejner objektů Realm. Touto metodou lze modelovat vazby typu 1:N nebo N:N.

Způsob řešení vztahů mezi objekty je velmi přirozený pro vývojáře Javy na rozdíl od použití SQLite s využitím cizích klíčů.

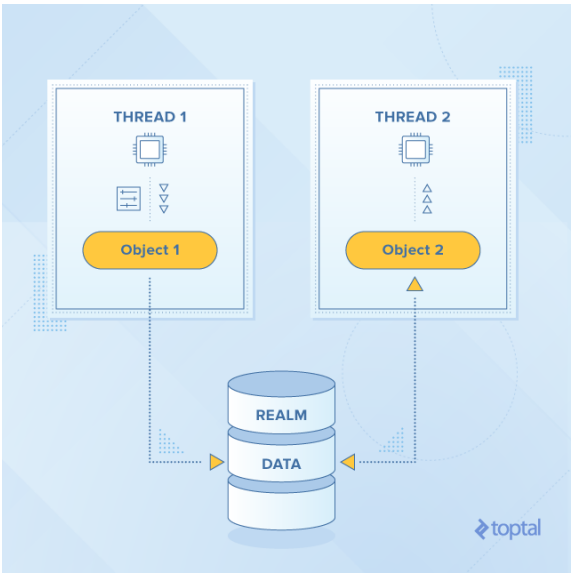
#### 3.14.1.3 Koncept nulové kopie

Realm využívá koncept nulové kopie, což znamená, že data nejsou nikdy zkopírována do paměti. Výsledky dotazů jsou ve skutečnosti pouze ukazatele na skutečná data. Například pokud má model deset polí (sloupce v SQL) a aplikace dotazuje objekty tohoto modelu, ale potřebuje zobrazit pouze tři pole, budou získána pouze tyto tři pole.

V důsledku toho jsou dotazy poměrně rychle zpracovány. To je velká výhoda oproti ORM, které zpravidla načítá všechny sloupce z vybraných řádků SQL. Z tohoto přístupu také vyplývá, že aplikace postavené nad Realmem, spotřebovávají méně paměti. Vzhledem k tomu že mobilní zařízení mají omezené hardwarové zdroje, může mít Realm velký přínos na rychlosti celé aplikace.

Další výhodou konceptu nulové kopie je, že Realm objekty se automaticky aktualizují. Data nejsou nikdy zkopírována do paměti. Pokud má aplikace výsledky z dotazu a jiný podproces aktualizoval data v databázi, původní výsledky již budou odrážet

tyto změny. Výsledky jsou pouze ukazatelem skutečných dat. Pokud aplikace přistupuje k hodnotám z polí, získá vždy ta nejaktuálnější data.




Obrázek 6 - Schéma Realm zdroj: (https://www.toptal.com/android/realm-best-android-database-solution)

### 3.14.2 Omezení

Ani Realm není dokonalý a má několik omezení, se kterými se před tvorbou aplikace musí vývojář seznámit.

- Ačkoliv je možné mít více podprocesů pro současné čtení a zápis do databáze, objekty Realmu nelze přesouvat přes vlákna. Pokud se například načte objekt Realm pomocí AsyncTask doInBackgorun(), který běží v podprocesu, nemůže tuto instanci předat onPostExcute() metodám, protože ty běží na hlavním podprocesu. Tuto situaci je možné vyřešit vytvořením kopie objektu a tu předat nebo předaním ID objektu a znovu načíst objekt pomocí onPostExcecute(). Realm nabízí asynchronní metody pro čtení a zápis.
- Neexistuje žádná podpora primárních klíčů s automatickým přírůstkem, proto musí vývojář sám vyřešit postupné generování klíčů.
- Není možné získat najednou přístup z odlišných procesů. Podle jejich dokumentace se má v dohledné době objevit podpora více procesů.

### 3.14.3 Použití Realmu

Použití Realmu je velmi snadné. Stačí pouze vytvořit podtřídu RealmObejct pro definování modelu a pak již využívat jednoduchý výběr dat z databáze. Jako příklad bude

níže uvedena Realm databáze, která ukládá informace o psech; konkrétně jméno a věk. Následně bude databáze naplněna údajem o psovi Rexovi, kterému je 5 let. Poslední příkaz bude vybírat všechny psi, kteří mají věk menší než 7 [49].

```
public class Pes extends RealmObject {
  public String jmeno;
  public int vek;
}

Pes pes1 = new Pes();
pes1.jmeno = "Rex";
pes1.vek = 5;

Realm realm = Realm.getDefaultInstance();
realm.beginTransaction();
realm.copyToRealm(pes1);
realm.commitTransaction();

RealmResults<Dog> psiMladsi2 = realm.where(Pes.class)
                               .lessThan("vek", 2)
                               .findAll();
```

# 4. Vlastní práce

## 4.1 Specifikace požadavků

Před samotným vývojem aplikace je důležitým prvkem analýza. Jako první se ve většině případů přistupuje ke specifikaci požadavků. Tyto požadavky se dělí na funkční a nefunkční. Funkční požadavky přesně stanovují, jaké funkce musí aplikace poskytovat uživateli. Nefunkční požadavky kladou omezení na design a provedení (např. výkonnost aplikace nebo různé standardy kvality).

### 4.1.1 Funkční požadavky

**Registrace vozidla**

Aplikace nabídne formulář pro registraci vozidla. Bude vyžadováno zadání základních informací o automobilu včetně exspirace STK a průměrné spotřeby.

**Evidence výdaje a tankování**

Správce vozového parku bude mít možnost evidovat každému vozidlu tankování a výdaje. U tankování bude k dispozici záznam hodnot: stav tachometru, cena za litr, celková cena, počet litrů a datum tankování. U výdajů budou pole následující: stav tachometru, cena, datum výdaje a typ výdaje. U typu výdaje bude možnost vybrat z číselníků pro pozdější snadné vyhodnocování. Evidence by měla být přehledná a používat ikony pro snadné rozlišení tankování od ostatních výdajů.

**Statistiky**

V záložce Statistiky budou k dispozici tyto vypočítané parametry: průměrná spotřeba, náklady na 1 kilometr, náklady celkem, celkový objem paliva, počet najetých kilometrů, výdaje na 1 kilometr a výdaje celkem. Tyto výsledky bude možné filtrovat po dnech se snadným výběrem dnů pomocí kalendáře a dále za jednotlivé vozidlo nebo za celý vozový park.

#### Grafy

Aplikace by měla zobrazit minimálně těchto pět grafů: graf celkových cen tankování, graf cen paliva, graf rozložení typů výdajů, graf průměrné spotřeby, graf najetých kilometrů mezi tankováním. Všechny grafy kromě jednoho budou ve formě plošného s vyznačenými body pro větší přehlednost. Výjimkou bude graf rozložení typů výdajů, který bude sloupcový pro lepší znázornění výše jednotlivých výdajů. Uživateli bude nabídnuto možnost stejného filtrování jako ve statistikách.

#### Notifikace

Aplikace umožní uživateli vytvářet notifikaci na STK (Státní technickou kontrolu), která se bude automaticky vypočítávat dle uvedeného stáří vozidla. Upozornění bude probíhat vždy měsíc předem, aby měl uživatel dostatečný čas na zajištění kontroly. V nastavení vozidla bude možnost vypnutí.

### 4.1.2 Nefunkční požadavky

#### Operační systém

Aplikace bude určena pro zařízení s operačním systémem Android. Minimální verzí, na které bude Spravoz fungovat, by měla být verze 5.0 (Lollipop).

#### Použitelnost a stabilita

Používání aplikace by mělo být intuitivní tak, aby se nový i stávající uživatel rychle zorientoval v prostředí aplikace a mohl jí snadno každý den využívat. Aplikace by měla být za každých okolností stabilní.

## 4.2 Analýza konkurenčních aplikací

Podobných aplikací, jako je v této diplomové práci vyvíjená aplikace Spravoz, existuje v Google Play několik. Níže budou představeny ty, které jsou nejvíce používané a dosahují vysokého hodnocení od uživatelů.

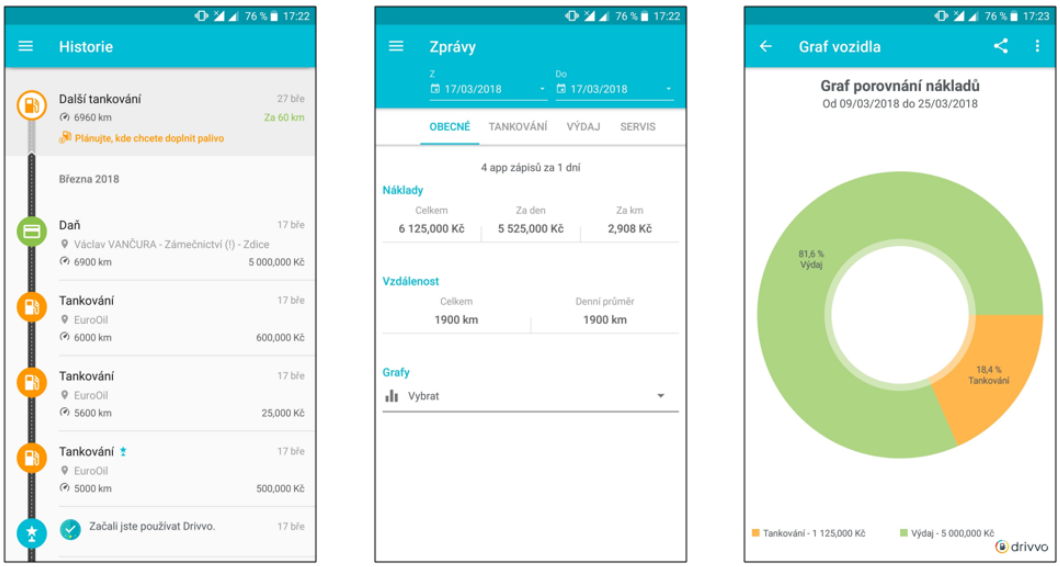
### 4.2.1 Drivvo

Jako první aplikaci lze bezesporu uvézt Drivvo, které přináší moderní design a základní přehled ve formě timeline. Po první spuštění vyběhne obrazovka s registrací. Nechybí možnost se přihlásit pomocí účtu na Facebooku nebo Googlu. Následuje registrace vozidla, kde jsou vyžadovány tyto údaje název automobilu, výrobce, model,

SPZ, rok, kapacita nádrže a jednotky objemu. Další možností je obnova dat z jiných aplikací. Konkrétně se jedná o těchto sedm aplikací Fuelio, aCar, Carango, Fuel Log, My Cars, Car Expenses, Fuel Manager. Drivvo nabízí podrobné statistiky o tankování jako je např. průměrná spotřeba na 100 km, náklady na 1 km, včetně možnosti zobrazit si grafy. Zaznamenávat ovšem lze i výdaje (mytí auto, pokuty, parkovné atd.) a servis. Je možné také nastavit upomínku na budoucí servis či výdaj. Základní verze je poskytována zdarma. Existuje také PRO verze, která za cenu 12 euro za rok nabídne zálohu dat do cloudu, absenci reklam, export do CSV/Excel nebo přednostní technickou podporu.

Aplikace dosahuje v Google Play hodnocení 4,6 a počet stažení se nachází mezi 1 000 000 – 5 000 000.

Nevýhodou aplikace je fakt, že její cílovou skupinou jsou ve většině případů jednotliví uživatelé vozů, jelikož nenabízí žádný komplexní pohled na vozový park. Mezi další nevýhody patří velké množství reklam v neplacené verzi.




Obrázek 7 - Drivvo zdroj: (Vlastní zpracování)

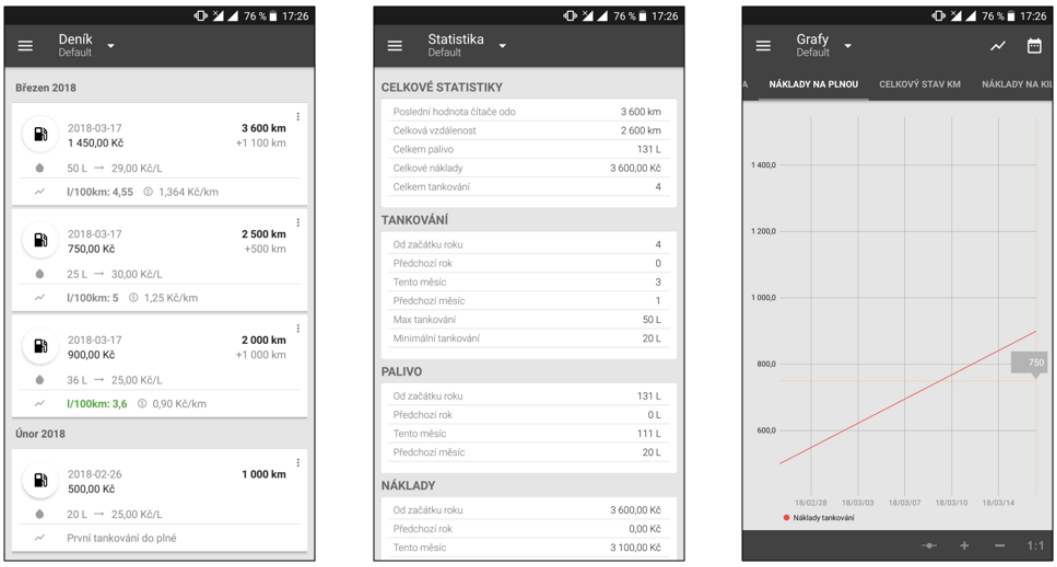
### 4.2.2 Fuelio

Mezi další populární aplikace v tomto segmentu patří Fuelio, které nabízí moderní grafické rozhraní ve stylu material design a podrobné statistiky. Při registraci vozidla je nutné zadat tyto parametry: název, popis, jednotky (délkové, palivové a spotřeby) a druh paliva. Výhodou Fuelia je možnost zaznamenávat i automobil se dvěma nádržemi, což je typické pro vozidla, které jezdí na LPG nebo CNG. Po naplnění aplikace daty o tankování

nabídne Fuelio podrobné statistiky. Lze zde zjistit například počet tankování, celkovou cenu, objem natankovaného paliva, největší a nejmenší natankovaný objem, nejvyšší a nejnižší účet, nejlepší a nejhorší cenu, náklady na kilometr i průměrné hodnoty všech uvedených údajů. Zajímavou hodnotou je také nejvyšší a nejnižší průměrná spotřeba paliva na sto kilometrů. V záložce Grafy nalezneme grafické zobrazení průměrné spotřeby v průběhu času, měsíčních nákladů, vývoje ceny paliva, nákladů na plnou nádrž, celkového stavu kilometrů a nákladů na kilometr. Aplikace nabídne i možnosti připomenutí, a to i u opakující se události. Na úvodním přehledu lze také zobrazit seznam nejbližších čerpacích stanic i s aktuálními ceny. Aplikace nabízí mapu, kde zobrazuje polohu jednotlivých stanic, které uživatel využil. Fuelio je v základní verzi nabízeno zdarma. Verze PRO stojí 39 Kč a nabídne například synchronizaci s Dropboxem nebo Google diskem, widget a další možnosti statistik a grafů.

Aplikace dosahuje v Google Play hodnocení 4,6 a počet stažení se nachází mezi 1 000 000 – 5 000 000.

Nevýhodou aplikace je fakt, že její cílovou skupinou jsou ve většině případů jednotliví uživatelé vozů, jelikož nenabízí žádný komplexní pohled na vozový park. Další nevýhodou jsou méně přehledné grafy a také synchronizace, která v některých případech dle recenzí uživatelů na Google Play nefunguje.



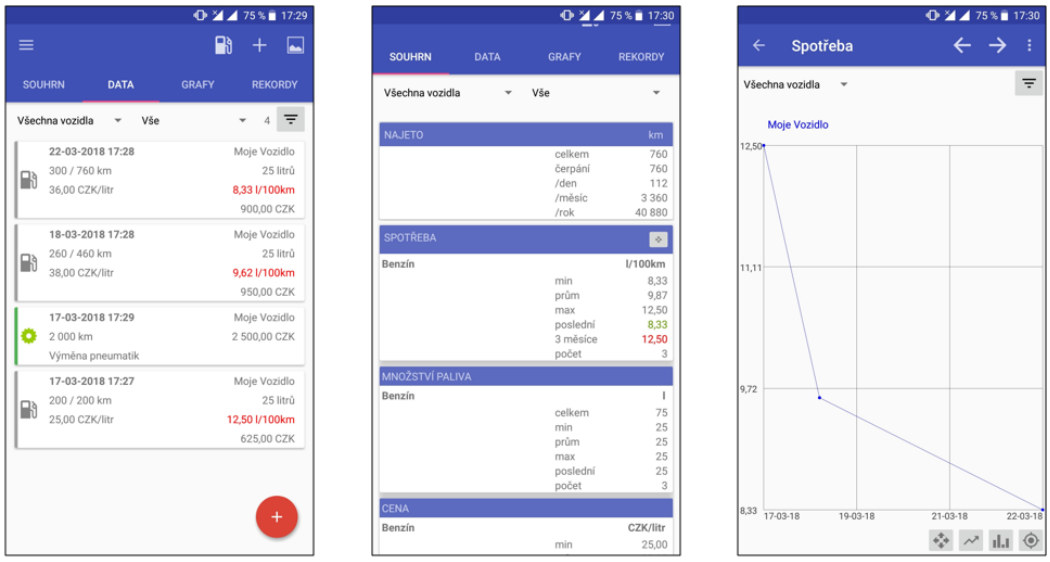

Obrázek 8 - Fuelio zdroj: (Vlastní zpracování)

### 4.2.3 My Cars

Aplikace My Cars se také řadí na Google Play mezi populární nástroje v oblasti sledování výdajů a spotřeby automobilů. Jako v každé aplikaci tohoto druhu je potřeba nejdříve registrovat vozidlo. My Cars umožňuje zadat nespočet údajů o automobilu od základních a povinných jako je název, stav tachometru, palivo až po opravdu detailní informace jako je např. velikost a tlak kol, VIN nebo datum koupě. Ke každému vozidlu lze uložit i fotografii. Aplikace umožňuje sledovat tyto typy událostí: čerpání, účet, údržba, poznámka, cesta a opakující účet. Při vyplňování údajů o tankování se uvádí datum, typ, vzdálenost, stav tachometru, množství, cena za litr a celková cena. Volitelně lze zaznamenat čerpací stanici, oktanové číslo nebo způsob platby. V tabu souhrn se nachází statistiky rozdělené do několika kategorií jako je najeto, spotřeba, cena atd. U většiny z nich jsou vypočítaná maxima, minima a průměry. Další statistiky lze nalézt v tabu rekordy, kde jsou uvedeny nejlepší a nejhorší spotřeby, které vyhovují zadanému filtru. My Cars nabízí i mnoho různých grafů, které nabízejí informace o nákladech např. spotřeba, cena nebo ujetá vzdálenost. Všechny grafy lze filtrovat dle data a počtu vozidel, lze zobrazit všechny nebo jenom námi vybraná. My Cars umožňuje vytvářet upomínky na události, a to jednorázové i opakované. Tyto notifikace se mohou řídit dle datumu nebo stavem tachometru. Aplikace je v základní verzi poskytována zdarma. Placená verze, která stojí 85 Kč, umožňuje zálohu na DropBox, podporu více měn včetně konverzí, podporu Google Print či cestovní deník.

Aplikace dosahuje v Google Play hodnocení 4,5 a počet stažení se nachází mezi 1 000 000 – 5 000 000.

Nevýhodou aplikace MyCars je menší přehlednost, na druhou stranu nabízí opravdu spoustu informací, které lze zaznamenávat a vyhodnocovat. Autor diplomové práce měl na svém zařízení (OnePlus 5 Android 8.0.0) problém s přidáváním fotografie k vozidlu. Pořízení nové nebo výběr již pořízené fotografie se nejevil jako funkční, jelikož nedošlo k načtení obrázku. Také stabilita aplikace nebyla na dobré úrovni, jelikož za dobu testování MyCars se aplikace autorovi práce dvakrát nečekaně ukončila bez upozornění.




Obrázek 9 - My Cars zdroj: (Vlastní zpracování)

### 4.2.4 Vyhodnocení

Cílem analýzy konkurenčních aplikací, bylo zmapovat nástroje, které nabízejí podobnou funkcionalitu jako vyvíjený Spravoz.

Z analýzy vyplynulo, že na Google play se nachází velké množství aplikací na správu vozového parku. Pro srovnání byli vybráni tři konkurenti Drivvo, Fuelio a My Cars. Tyto aplikace dosahují velkého počtu stažení a velmi dobrého hodnocení od uživatelů. Dále bylo zjištěno, že všechny analyzované nástroje používají business model ve formě Premium modelu, kdy je základní verze zdarma a další funkcionalita je dostupná ve verzi placené. Fuelio a My Cars jsou zpoplatněny fixními částkami. Drivvo se platí ve formě ročního předplatného a navíc neplacená verze obsahuje velké množství reklam, což je jedním z nevýhod tohoto řešení.

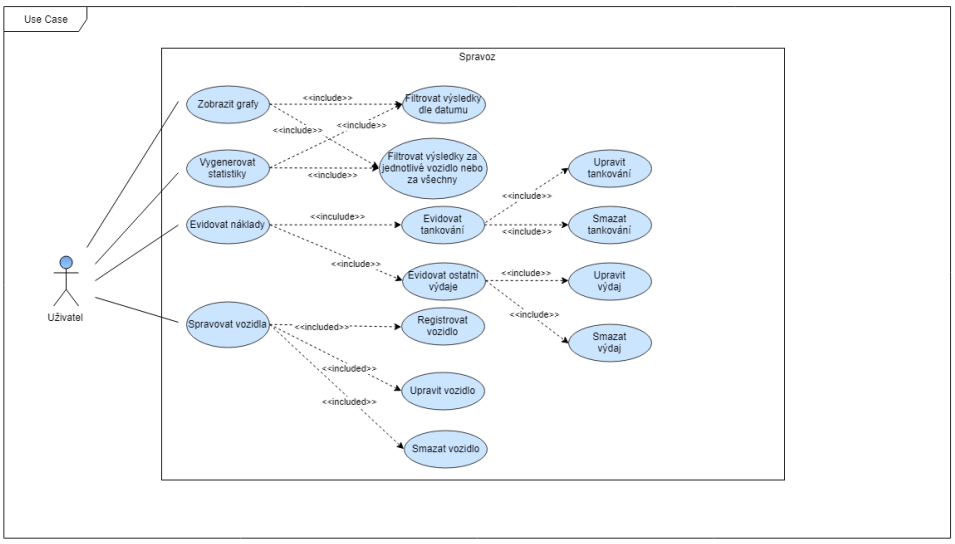
První dvě aplikace nabízejí moderní přehledný design, velkou funkcionalitu, ale jsou určena spíše pro koncové zákazníky. Umožňují sice možnost zaznamenávat více vozidel, ale v rámci statistik a grafů lze vždy vyhodnocovat pouze údaje pro jedno vozidlo. My Cars naproti tomu umožňuje nejen evidenci více vozidel, ale i vyhodnocování statistik a grafů pro všechny vozidla. Aplikace zaznamenává velké množství informací, ale její rozhraní v některých oblastech nepůsobí moc přehledně. Zároveň autor této práce narazil u aplikace na některé problémy s funkcionalitou a stabilitou.

Všechny testované aplikace nabízejí možnost vytvořit notifikace na události, ale žádná z nich nedokáže automaticky upozorňovat na STK, které Spravoz vypočítává na základě první registrace vozidla nebo datumu poslední státní technické kontroly. Zároveň vyvíjená aplikace nabízí přehledné a jednoduché rozhraní, kde lze snadno dohledat důležité údaje pro správu vozového parku.

## 4.3 Use Case diagram

Případy užití jsou níže demonstrovány pomocí diagramu užití. Diagram užití popisuje chování systému z pohledu uživatele a interakci s ním. Dále znázorňuje možné akce, které jsou uživateli v rámci aplikace dostupné.

Aplikace Spravoz má pouze jeden typ uživatele (aktéra), jelikož v rámci své funkčnosti nerozlišuje registrovaného a neregistrovaného uživatele. Ani zde není žádný uživatel, který by vstupoval do aplikace a měl jiné možnosti jako např. správce systému.




Obrázek 10 - Use case diagram zdroj: (Vlastní zpracování)

## 4.4 Scénáře užití

### 4.4.1 Evidence tankování

Scénář popisuje situaci od prvního spuštění aplikace po registraci vozidla a následnou evidenci tankování.

**Hlavní scénář: Evidence tankování; Role: Uživatel**

| | |
|---|---|
| 1. | Po spuštění aplikace se zobrazí formulář pro registraci vozidla. |
| 2. | Uživatel vyplní tyto údaje o vozidle: název, značka, model, datum první registrace případně datum poslední technické kontroly. U pole značka se uživateli po zadání prvního písmene zobrazí našeptávač, ze kterého snadno vybere značku nově evidovaného vozidla. Následně stiskne registrovat.<br><br>**Alternativní:** Uživatel nevyplní údaje korektně, aplikace upozorní dialogem o nesprávném formátu, nebo že daný údaj není vůbec vyplněn. |
| 3. | Aplikace zobrazí prázdný seznam tankování a výdajů vozidla. |
| 4. | Uživatel stiskne tlačítko pro přidání záznamu a ze zobrazeného menu zvolí položku Tankování. |
| 5. | Aplikace zobrazí formulář pro přidání položky Tankování. |
| 6. | Uživatel vyplní tyto údaje o tankování: stav tachometru, cena za litr, celková cena, počet litrů a datum a čas tankování. Po zadaní alespoň dvou hodnot z ceny za litr, celkové ceny a počtu litrů, se uživateli zbylý údaj dopočítá automaticky. Pomocí checkbox uživatel zvolí, zda tankoval plnou nádrž. Následně stiskne tlačítko Uložit.<br><br>**Alternativní:** Uživatel nevyplní údaje korektně, aplikace upozorní dialogem o nesprávném formátu, nebo že daný údaj je v kolizi s předcházejícími údaji o tankování nebo o výdajích. |
| 7. | Aplikace zobrazí seznam tankovaní a výdajů vozidla, kde uživatel uvidí nově přidané tankování. |

### 4.4.2 Zobrazení statistik

Scénář popisuje situaci, kdy uživatel již aplikaci používá, má registrovaná vozidla, zaevidované tankování a chce zobrazit statistiky pro celý vozový park za určitý časový úsek.

**Hlavní scénář: Zobrazení statistik; Role: Uživatel**

| | |
|---|---|
| 1. | Aplikace je spuštěná a uživatel se nachází v některé z jejích částí.<br><br>**Alternativní:** Aplikace není spuštěná. Uživatel aplikaci spustí. |
| 2. | Uživatel rozbalí postranní menu a vybere záložku Statistiky. |
| 3. | Aplikace zobrazí vypočítané statistiky pro zvolené vozidlo za celé časové období. |
| 4. | Uživatel zaškrtne checkbox Všechna vozidla a statistiky se přepočítají pro všechna vozidla. |
| 5. | Uživatel stiskne textové pole Od a aplikace zobrazí kalendář, kde uživatel vybere počáteční datum a potvrdí výběr stisknutím tlačítka OK. |
| 6. | Aplikace zobrazí vypočítané statistiky pro všechna vozidla filtrovaná od datumu, který uživatel vybral. |
| 7. | Uživatel stiskne textové pole Do a aplikace zobrazí kalendář, kde uživatel vybere koncové datum a potvrdí výběr stisknutím tlačítka OK. |
| 8. | Aplikace zobrazí vypočítané statistiky pro všechna vozidla filtrovaná časovým úsekem, který uživatel vybral v předchozích krocích. |

### 4.4.3 Editace vozidla

Scénář popisuje situaci, kdy uživatel již aplikaci používá, má registrovaná vozidla, zaevidované tankování a chce upravit název vozidla.

**Hlavní scénář: Editace vozidla; Role: Uživatel**

| | |
|---|---|
| 1. | Aplikace je spuštěná a uživatel se nachází v některé z jejích částí.<br><br>**Alternativní**: Aplikace není spuštěná. Uživatel aplikaci spustí. |
| 2. | Uživatel rozbalí postranní menu a vybere záložku Vozidla. |
| 3. | Aplikace zobrazí seznam vozidel. Aktuálně vybrané vozidlo se zvýrazní okrajem. Uživatel vyhledá v seznamu požadované vozidlo a kliknutím na něj se objeví menu, kde uživatel zvolí položku Editovat. |
| 4. | Zobrazí se formulář s předvyplněnými daty o editovaném vozidle. Uživatel upraví pole název a stiskne tlačítko Uložit. |
| 5. | Aplikace zobrazí seznam vozidel s již upraveným názvem automobilu. |

### 4.4.4 Evidence výdaje a zobrazení grafů

Scénář popisuje situaci, kdy uživatel již aplikaci používá, má registrovaná vozidla a potřebuje zaevidovat nový výdaj a poté zobrazit grafy pro jedno vozidlo za celé období.

**Hlavní scénář: Evidence výdaje a zobrazení grafů; Role: Uživatel**

| | |
|---|---|
| 1. | Aplikace je spuštěná a uživatel se nachází v některé z jejích částí.<br><br>**Alternativní**: Aplikace není spuštěná. Uživatel aplikaci spustí. |
| 2. | Uživatel se nachází v přehledu.<br><br>**Alternativní**: Uživatel pomocí postranního menu vybere záložku Přehled. |
| 3. | V dolním pravém rohu uživatel stiskne tlačítko pro přidání a z rozbaleného menu vybere položku Výdaj. |
| 4. | Aplikace zobrazí formulář pro nový výdaj. V poli Stav tachometru automaticky doplní |

| | |
|---|---|
| | údaj o stavu tachometru z posledního tankování nebo výdaje. |
| 5. | Uživatel vyplní tyto údaje o tankování: stav tachometru, typ výdaje, cena a datum a čas tankování. Typ výdaje uživatele vybere z rozbalené nabídky po kliknutí na pole. Následně stiskne tlačítko Uložit.<br><br>**Alternativní**: Uživatel nevyplní údaje korektně, aplikace upozorní dialogem o nesprávném formátu, nebo že daný údaj je v kolizi s předcházejícími údaji o tankování nebo o výdajích. |
| 6. | Uživatel rozbalí postranní menu a vybere záložku Grafy. |
| 7. | Aplikace zobrazí grafy o nákladech zvoleného vozidla za celé období. |

## 4.5 Návrh GUI

Před samostatným vývojem aplikací je vhodné nejdříve navrhnout grafické rozhraní aplikace. Autor diplomové práce zvolil pro návrh prostředí online nástroj MockFlow. Ten nabízí rozsáhlou databázi jednotlivých grafických prvků nejen pro operační systém Android. Díky tomu se stává návrh podstatně jednodušší a rychlejší, jelikož odpadá nutnost ručního kreslení každého ovládacího prvku rozhraní. MockFlow lze využít i pro návrh UI pro web (Bootstrap), iOS, Windows nebo Apple Watch. Tento nástroj je pro jeden projekt a jednoho uživatele poskytován zdarma. Pro více projektů a zpřístupnění více funkcí je MockFlow zpoplatněn 14$ na měsíc.

### 4.5.1 Wireframs

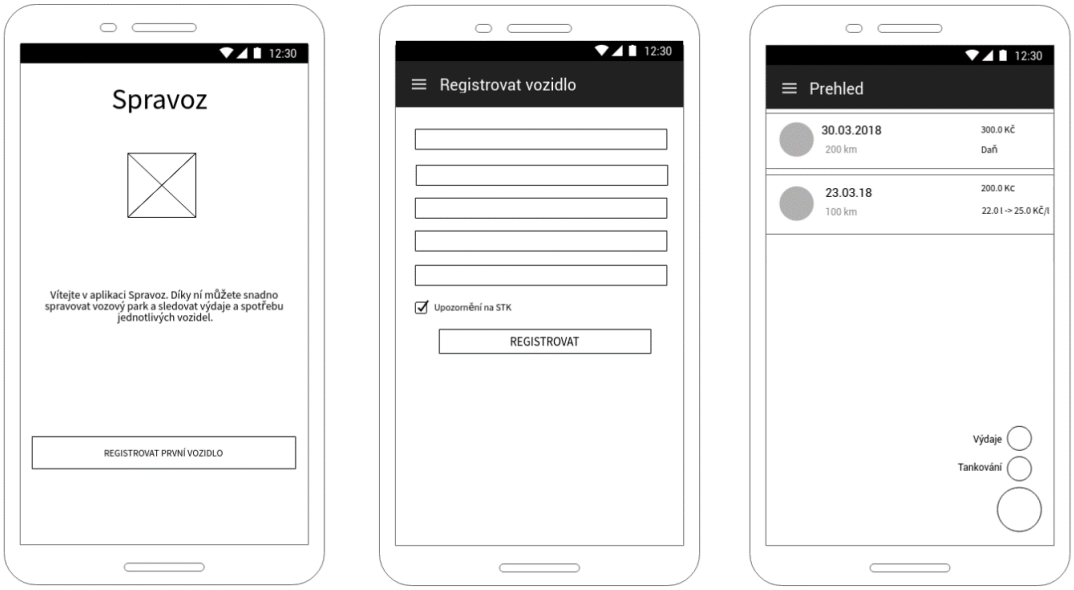
V rámci návrhu UI aplikaci byly zpracovány wireframy všech osmi obrazovek Spravozu a navíc jedna obrazovka, kde je k vidění nákres rozbalovacího menu.

#### 4.5.1.1 Obrazovky Úvodní, Registrovat vozidlo a Přehled

Na prvních třech prototypech se nejdříve nachází úvodní obrazovka, která se uživateli zobrazí po prvním spuštění aplikace. Tento návrh reprezentuju aktivitu UvodActivity. Ta slouží pro první seznámení s aplikací. Nachází se zde název aplikace, logo a základní popis, k čemu lze aplikaci využívat.

Pomocí tlačítka Registrovat první vozidlo se uživatel dostane na další obrazovku VozidloActivity. Tato aktivita slouží k registraci vozidla a obsahuje pět textových polí, checkbox a tlačítko. Poslední textbox STK do, se zobrazuje a skrývá na základě údaje o registraci vozidla, zda již vozidlo bylo na státní technické kontrole nebo zda ho čeká první.

Po stisknutí tlačítka Registrovat se aplikace přepne do MainActivity konkrétně do PrehledFragment. Ten obsahuje seznam výdajů a tankování implementovaný pomocí komponenty RecyclerView a jednotlivé položky jsou realizovány za pomoci CardView. V pravém dolním rohu se na této obrazovce nachází FloatingActionButton pro přidání položky, který po kliknutí rozvine menu pro zvolení, zda chce uživatel přidat výdaj či tankování.




Obrázek 11 - Návrh GUI obrazovky: Úvodní, Registrovat vozidlo a Přehled zdroj: (Vlastní zpracování)

#### 4.5.1.2 Obrazovky Nové tankování, Nový výdaj a Menu

Další obrazovka Nové tankování obsahuje formulář pro přidání údajů o tankování. TankovaniActivity se skládá z pěti textových polí, z nichž některé jsou opatřeny ikonami pro lepší přehlednost, aby uživatel snadno rozeznal, jaký údaj dané pole obsahuje. Pod nimi je checkbox pro informaci, zda se jednalo o tankování plné nádrže a tlačítko Uložit, které data zaznamená do databáze a uživatele přepne zpátky na přehled výdajů a tankování PrehledFragment.

Velmi podobný je wirefram Nový výdaj, který se liší pouze v tom, že má o jedno pole méně, jelikož neobsahuje textbox Cena za litr a také neobsahuje checkbox. Tuto

obrazovku reprezentuje aktivita VydajActivity. Stejně funguje i tlačítko Uložit, které zaznamená nový výdaj do databáze a přesune uživatele na PrehledFragment.

Poslední obrazovka z obrázku níže zobrazuje postranní menu, které je dostupné z aktivity MainActivity a ze všech fragmentů, které se mohou v této aktivitě zobrazit. V horní části bude umístěn obrázek vozového parku s ikonou Spravoz a také text, který bude obsahovat název právě aktivního vozidla. Pod touto části se bude nacházet menu o čtyřech položkách, z nichž každá se skládá z ikony a samotného názvu položky. Z každé této položky se uživatel bude moct přepnout do jiného fragmentu v rámci aktivity MainActivity.


Obrázek 12 - Návrh GUI obrazovky: Nové tankování, Nový výdaj a Menu zdroj: (Vlastní zpracování)

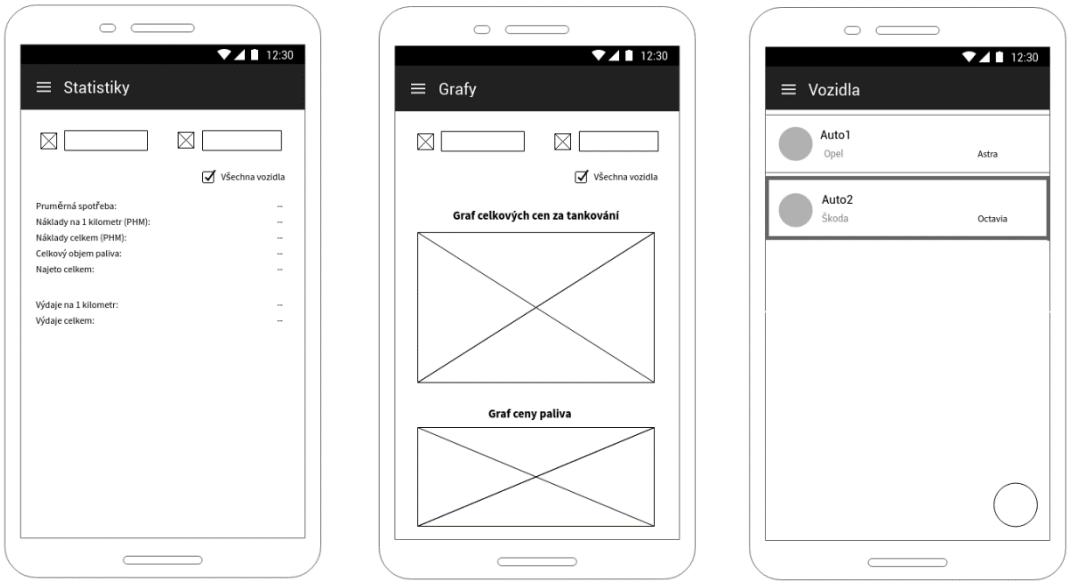
#### 4.5.1.3 Obrazovky Statistiky, Grafy a Vozidla

Wirefram Statistiky obsahuje v horní části dvě textová pole spolu s ikonami. Po jejich stisknutí se uživateli objeví kalendář pro výběr datumu. Pod nimi je checkbox pro volbu všech vozidel. Následují již samostatné statistiky, které jsou reprezentovány ve formátu název statistiky dvojtečka hodnota. Takto je seřazeno pod sebou celkem sedm statistik. Obrazovku statistiky reprezentuje fragment StatistikyFragment.

Na podobném principu je postavena obrazovka Grafy, která zastupuje GrafyFragment. Horní část je navrhnuta naprosto identicky – dvě pole pro datum a checkbox pro všechna vozidla. Pod nimi jsou již samotné grafy, kterých je celkem pět a

jsou ve formátu nadpis s tučným fontem a graf. Na displej zařízení se vejdou dva grafy, poté je potřeba obrazovku posunout.

Poslední wireframe Vozidla, který reprezentuje VozidlaFragment, je implementací podobný obrazovce přehledu výdajů a tankování. Obsahuje seznam registrovaných vozidel implementovaný také pomocí komponenty RecyclerView a jednotlivé položky jsou realizovány za pomocí CardView. Jak je z nákresu patrné, právě zvolenému vozidlu se nastaví tlustý rámeček, aby uživatel neztratil přehled o tom, jaké vozidlo má právě aktivní. V pravém dolním rohu se nachází FloatingActionButton pro přidání vozidla, který tentokrát nemá rozklikávací menu, ale přímo přechází na aktivitu. Není zde totiž více možností jako u tankování a výdajů.




Obrázek 13 - Návrh GUI obrazovky: Statistiky, Grafy a Vozidla zdroj: (Vlastní zpracování)

## 4.6 Implementace

### 4.6.1 Realizace GUI

Níže uvedené screenshoty ukazují již skutečně vytvořené uživatelského rozhraní aplikace Spravoz, které se uživateli zobrazí na jeho zařízení. Pro celou aplikaci byla zvolena jako hlavní barva modrá, která se objevuje ve všech obrazovkách. V rámci pořízení screenshotů byla do aplikace Spravoz vložena ukázková data, aby bylo znázorněno, že aplikace je fakticky funkční a připravena k užití.

#### 4.6.1.1 UvodActivity, VozidloActivity a PrehledFragment

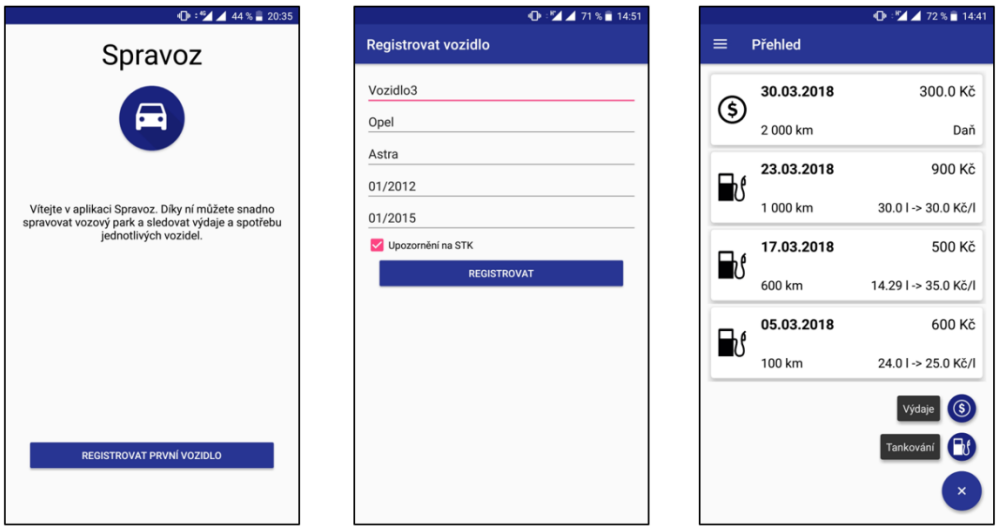
První zobrazenou obrazovkou je stejně jako v rámci kapitoly Návrh uživatelského rozhraní UvodActivity. Ta je tvořena RelativeLayoutem, do něhož je vnořen ScrollView. Ten obsahuje LinearLayout, ve kterém jsou již jednotlivé prvky GUI. ScrollView musel být využit z důvodu použití aplikace na zařízeních s malou úhlopříčkou displeje, jelikož by při horizontálním rozložení nebyly některé prvky layoutu viditelné. Do LinearLayoutu je vnořen TextView s nápisem Spravoz, dále ImageView, které obsahuje ikonu aplikace. Pod ním je umístěn další TextView, který obsahuje krátký popis aplikace. Jako poslední se zde nachází tlačítko AppCombatButton s nápisem Registrovat první vozidlo, které přepne uživatele na VozidloActivity. Toto tlačítko má jako barvu pozadí zvolenou colorPrimary, což je parametr aplikace, který je uložený v xml souboru colors.xml ve složce values. V případě aplikace Spravoz byly zvoleny barvy colorPrimary. V šestnáctkovém zápise mají hodnotu #283593, ColorPrimaryDark #1a237 a ColorAccent #FF4081. Všechny tyto barvy jsou zvoleny z materiální kolekce a jsou použity skrz celou aplikaci. Veškeré prvky rozhraní v této obrazovce jsou pomocí parametru *android:layout_gravity="center_horizontal*" vyrovnány na střed.

Obrazovka VozidloActivity je postavena na stejném principu jako UvodActivity. V RelativeLayoutu je vnořen ScrollView a v něm LinearLayout, který obsahuje EditText pro název vozidla, dále AutoCompleteView pro značku vozidla. Tento prvek funguje na principu, kdy se po napsání prvních pár písmen objeví menu, ve kterém se dají vybrat záznamy z předem definovaného pole, pokud se tato písmena shodují. V případě Spravozu obsahuje toto pole 46 nejvyskytovanějších značek automobilů. Pod tímto AutoCompleteView existují další prvky typu EditText pro parametry: model, registraci, STK do. Poslední EditText STK do se zobrazuje a skrývá na základě údaje o registraci vozidla – zda již vozidlo bylo na státní technické kontrole nebo zda ho čeká první. V případě screenshotu níže je zadáno datum registrace vozidla leden 2012, tudíž již musel být automobil na STK, jelikož doba po kterou STK platí je 4 roky od první registrace vozidla, dále pak 2 roky. Jako poslední se zde nachází CheckBox, který určuje, jestli si uživatel přeje být informován o končící STK u vozidla. Barva zaškrtnutí je zvolena výchozí růžová ColorAccent.

Poslední je PrehledFragment, který je zobrazován v rámci MainActivity. Ten ve své XML definici fragment_prehled.xml obsahuje pouze RecycleView, jenž zobrazuje

jednotlivé položky tankování a výdajů. Ty jsou reprezentovány pomocí CardView, které jsou navrženy v rámci XML card_view_row. Toto rozvržení obsahuje LinearLayout, ve kterém je vnořen CardView widget. V něm je vytvořen RelativeLayout, který již zahrnuje jednotlivé prvky každé položky RecyclerView. Mezi ně patří TextView pro datum, stav tachometru, cenu za tankování nebo výdaj. Poslední TextView je rozdílné podle toho, zda se jedná o tankování nebo výdaj. V případě výdaje je to typ výdaje a u tankování text ve formátu počet litrů -> cena za litr.

Z navržení aktivity si ještě přenáší dolní tlačítko pro menu FloatingActionMenu, které je implementováno za pomoci importu knihovny com.github.clans.fab.FloatingActionMenu. Jako barva tlačítka byla znovu zvolena základní colorPrimary, aby bylo dodržen jednoduchý a stejnorodý design. Po stisknutí tohoto tlačítka se objeví dva FloatingActionButton – jeden pro tankování druhý pro výdaje, které pocházejí ze stejné knihovny. Zde autor diplomové práce narazil na problém s ikonami, jelikož tato tlačítka jsou ve velikosti mini, tudíž jejich ikona musí mít rozměry 16 x 16 px. Autor vyrobil pro tento případ dvě ikony s takovýmto rozlišením a průhledným pozadím ve formátu PNG. Po spuštění aplikace na testovacím zařízení ale bylo zjištěno, že ikony jsou rozmazané, což je dáno takto nízkým rozlišením. Proto muselo být využito vektorových ikon. Ty jsou totiž definovány pomocí matematických funkcí a budou se kvalitně vykreslovat i v takto malých rozměrech. Aby autor práce nemusel vytvářet ikony ručně, jelikož jejich tvorba je poměrně složitá, bylo využito online databáze zdarma dostupných vektorových ikon flaticon.com. Zde byly k nalezení ikony jak pro výdaj, tak pro tankování. Jako barva těchto ikon byla použita colorPrimaryDark pro mírné odlišení od většího tlačítka pro přidání.




Obrázek 14 - Realizace GUI UvodActivity, VozidloActivity a PrehledFragment zdroj: (Vlastní zpracování)

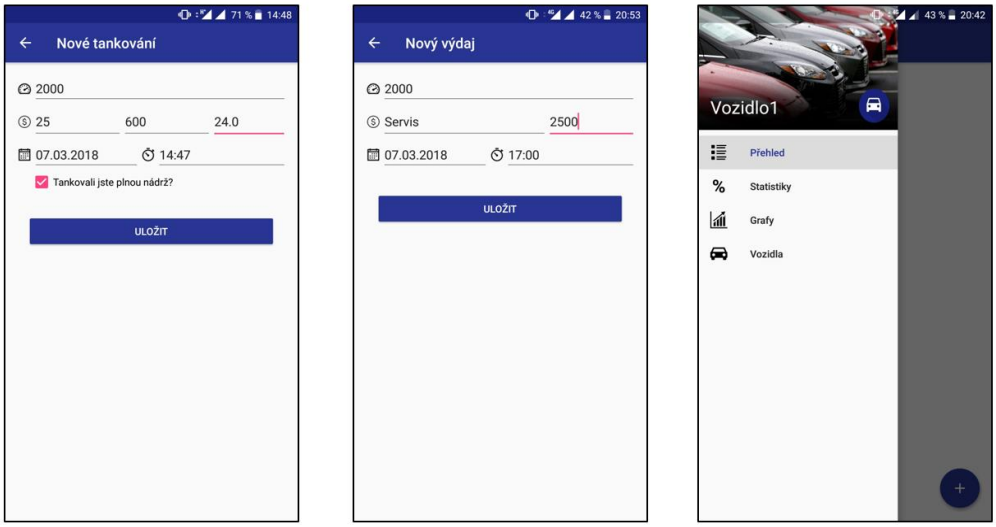
#### 4.6.1.2 TankovaniActivity,VydajActivity a MenuDrawer

První ze třech níže zobrazených obrazovek TankovaniActivity je tvořeno hlavním LinearLayoutem s vertikálním řazením prvků. Do něho je vnořeno LinearLayout s horizontálním řazením, které zahrnuje ImageView s ikonou tachometru a EditText pro zadání stavu tachometru. Dále je do hlavního LinearLayoutu vnořen další LinearLayout s horizontálním řazením, který obsahuje ImageView s ikonou dolaru a prvky EditText pro hodnoty cena za litr, celková cena a počet litrů. V dalším horizontálním LinearLayoutu je umístěn ImageView s ikonou kalendáře, dále EditText, který po klinutí na něj otevře grafický kalendář pro snadný výběr datumu. Po straně se nachází další ImageView s ikonou stopek a vedle něj EditText, který po kliknutí na něj otevře rozhraní pro snadný výběr hodiny a minuty. Pod nimi se nachází CheckBox, který má opět jako barvu zaškrtnutí zvolenou výchozí ColorAccent. Jako poslední se na obrazovce nachází tlačítko AppCombatButton Registrovat, které uloží záznam do databáze a přepne uživatele do PrehledFragmentu.

Na stejném principu rozvržení je realizováno GUI VydajActivity, ovšem i zde se nachází pár rozdílů. V prostředním horizontálním LinearLayout je o jedno pole méně a prvním z nich není cena za litr ale typ výdaje. Ten je implementován jako AutoCompleteTextView, ale za pomocí parametru *android:focusable="false"* je zakázáno přímého zápisu do tohoto pole. Místo toho je zde nastaven onClickListener, který po

kliknutí na AutoCompleteTextView otevře menu vytvořené z ArrayAdapteru. Tento adaptér je tvořen polem řetězců, které obsahuje jednotlivé typy výdajů.

Nahoře na postranním menu je umístěn obrázek v podobě fotografie zaparkovaných vozidel, doplněn o ikonu aplikace Spravoz. Spodní okraj byl pomocí černého stínu upraven tak, aby byl bílý text s názvem vozidla dostatečné čitelný. Pod tímto obrázkem se již nachází samotné menu, které ve formátu ikona a název položky obsahuje: Přehled, Statistiky, Grafy a Vozidla.




Obrázek 15 - Realizace GUI TankovaniActivity, VydajActivity a MenuDrawer zdroj: (Vlastní zpracování)

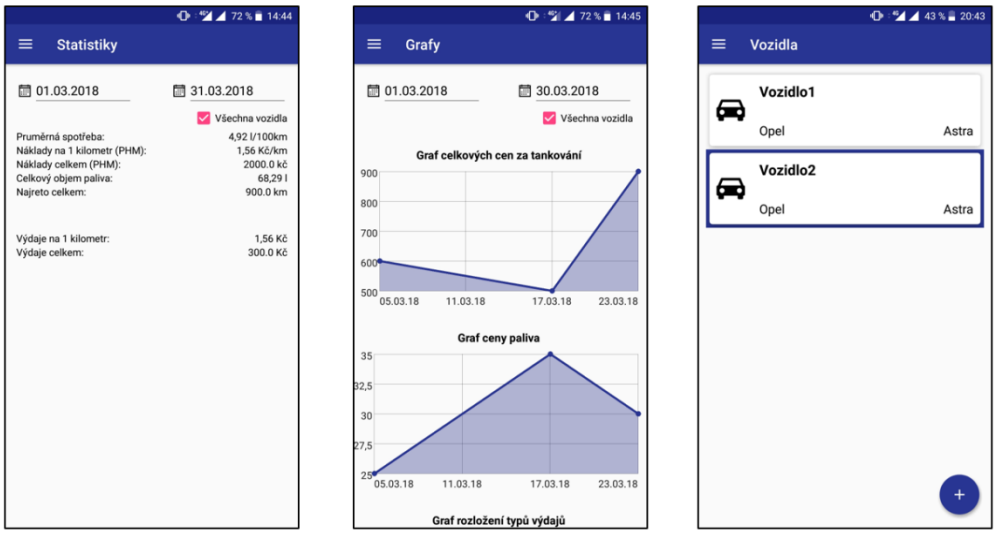
#### 4.6.1.3 StatistikyFragment, GrafyFragment a VozidlaFragment

Layout fragmentu StatistikyFragment je tvořen RelativeLayoutem, v něm je vnořen ScrollView, který obsahuje další RelativeLayout. Ten zahrnuje ImageView s ikonou kalendáře a vedle této ikony je EditText, po jehož rozkliknutí se zobrazí grafický kalendář pro výběr počátečního datumu. Stejná situace je i u EditTextu pro výběr koncového datumu. Pod ním je CheckBox, který určuje, zda jsou statistiky za celý vozový park nebo jenom za vybrané vozidlo. Následují již samostatné statistiky ve formě TextView, které jsou navzájem relativně pozicovány do tabulky o dvou sloupcích.

GrafyFragment je založen na layoutu ScrollView, do kterého se vnořuje TableLayout s nastaveným jedním sloupcem, který odpovídá nadpisu a grafu. Do tohoto layoutu je ještě vložen RelativeLayout, který má úplně stejnou podobu jako ve fragmentu Statistiky a slouží k filtrování výsledků grafů. Nadpis grafu je nastaven jako tučný a

velikost písma na 16dp. Pro zobrazení grafů bylo využito knihovny GraphView, která umožňuje snadné vykreslení a formátování různých typů grafů. Pro grafy celkových cen za tankování, cen za palivo, průměrné spotřeby a najetých kilometrů byl zvolen plošný graf s vyznačenými body. Pro graf rozložení typů výdajů byl zvolen formát sloupcového. Barva čáry a sloupců u plošného grafu odpovídá colorPrimary. Barva pozadí byla zvolena o několik odstínů světlejší, konkrétně #2E1A237E.

VozidlaFragment obsahuje ve své XML definici RelativeLayout, ve kterém je stejně jako v případě PrehledFragmentu umístěn prvek typu RecycleView, v němž jsou zobrazována jednotlivá vozidla. Ta jsou reprezentována pomocí CardViews, které jsou navrženy v rámci XML card_view_row. V případě vozidel ale nejsou všechny TextView naplněny textem. Hlavní TextView obsahuje název vozidla, v dalším je značka. Třetí pole není vyplněno a poslední obsahuje model. V layoutu je také umístěn FloatingActionButton pro přidání nového vozidla, který je stylizován stejně jako v PrehledFragmentu za použití colorPrimary, ovšem zde již nedochází k zobrazení svou dalších menších tlačítek.

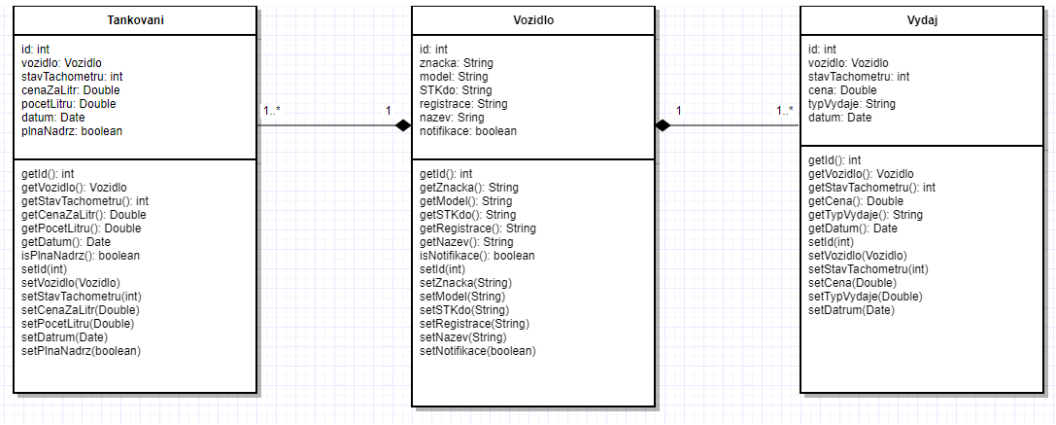



Obrázek 16 - Realizace GUI StatistikyFragment, GrafyFragment a VozidlaFragment zdroj: (Vlastní zpracování)

### 4.6.2 Databáze

Databáze aplikace Spravoz je postavena na objektové databázi Realm. Ta slouží k ukládání informací o jednotlivých tankováních, výdajích a vozidlech. Model databáze se skládá celkem ze třech tříd, které všechny dědí od třídy RealmObject. Z diagramu tříd níže lze vidět, že všechny třídy obsahují většinou atributy základních typů proměnných (int,

Double, Date) až na vozidlo, které je instancí třídy Vozidlo. Dále každá z namodelovaných tříd obsahuje getry a setry pro snadné získání a nastavení konkrétních atributů. Diagram byl vytvořen pomocí online nástroje Gliffy, který nabízí uživatelsky přívětivé prostředí a nabízí i možnost vytvořit i více dalších diagramů nežli class diagram. Nástroj je na 14 dní poskytován v trial verzi zdarma.




Obrázek 17 - Class diagram objektové databáze zdroj: (Vlastní zpracování)

### 4.6.2.1 Použití databáze Realm v aplikaci

#### 4.6.2.1.1 Založení nové instance objektu Výdaj

Nejdříve se zavolá metoda beginTransaction() u instance objektu Realm, která vytvoří transakci a další příkazy již budou zahrnuty v této konkrétní transakci. Poté se vytvoří instance vozidlo, která je definována jako výběr z databáze, kde parametr id se rovná proměnné id, kterou VydajActivity získala z Intentu poslaného předchozí aktivitou. Následuje vytvoření objektu s názvem vydaj z Realm třídy Vydaj. Dále se nastaví tomuto objektu atribut Id jako dalsiID, který obsahuje maximální ID z databáze objektu Vydaj zvětšené o jedničku. Atributu vozidlo se předá instance vozidlo, která se již vytvořila. Následujícím atributům se předají hodnoty z formuláře, vždy přetypované z řetězce do typu proměnné konkrétního atributu.

```
realm.beginTransaction();
Vozidlo vozidlo = realm.where(Vozidlo.class).equalTo("id", id).findFirst(
);
Vydaj vydaj = realm.createObject(Vydaj.class);
vydaj.setId(dalsiD);
vydaj.setVozidlo(vozidlo);
```

```
vydaj.setCena(Double.parseDouble(ediTextCena.getText().toString()));
vydaj.setStavTachometru(Integer.parseInt(editTextStavTachometru.getText()
.toString()));
vydaj.setTypVydaje(autoCompleteTextViewtypVydaje.getText().toString());
vydaj.setDatum(datum);
realm.commitTransaction();
```

#### 4.6.2.1.2 Výběr z databáze

Dotazování do Realmu je velmi jednoduché a snadno lze vybrat data na základě více podmínek. Příklad níže vybírá sadu tankovaní, které jsou mezi datem od zadaným přes grafický kalendář a datem do. Dále že id vozdidla odpovídá proměnné id, kterou StatistikyFragment převzal od volající aktivity. Všechny tankování jsou nakonec seřazeny dle atributu stav Tachometru.

```
RealmResults resultsTankovani = realm.where(Tankovani.class).between("dat
um", getDateWithOutTime(kalenadarOd.getTime()), getDateLastTime(kalenadar
Do.getTime())).equalTo("vozidlo.id", id).findAllSorted("stavTachometru");
```

Autor práce ovšem narazil na problém, kdy potřeboval seskupit data dle nějakého atributu, jako to lze v klasické SQL databázi pomocí příkazu GROUP BY. Realm žádnou náhradu tohoto příkazu nenabízí a musí se ručně doprogramovat pomocí cyklu. Na příkladu níže se požaduje vypočítat celkovou cenu pro každý typ výdaje v databázi. Nejdříve se do vysledkyVydaje uloží všechny výdaje seřazené dle data. Následně vznikne cyklus, který postupně prochází každý výdaj. V něm se vytvoří nová sada výdajů a ta vybere pouze ten, který má stejný typ výdaje jako právě procházený výdaj. Pokud tato sada není prázdná, spočítá se pro tento typ výdaje suma všech cen. Následně se přidá do pole dp3 souřadnice i+1 a spočítaná suma. Do popisku se přidá řetězec s typem výdaje, proměnná i se zvětší o jedna a cyklus prochází dále. Po dokončení cyklu se v polích objeví součty za všechny výdaje. Tento postup je tedy o dost složitější nežli pomocí GROUP BY v SQL.

```
int i = 0;
RealmResults vysledkyVydaje = realm.where(Vydaj.class).findAll().sort("da
tum");
for (Vydaj vydaj1 : vysledkyVydaje) {
RealmResults<Vydaj> rmVydaj = vysledkyVydaje.where().equalTo("typVydaje",
vydaj1.getTypVydaje()).findAll().sort("datum");
```

```
        if (rmVydaj != null) {
                double sum1 = rmVydaj.sum("cena").doubleValue();
                dp3[i] = new DataPoint(i + 1, sum1);
                popisky[i] = vydaj1.getTypVydaje();
                i++;
        }
}
```

#### 4.6.2.1.3 Smazání instance třídy Vozidlo

Pro smazání instance se znovu musí vytvořit transakce pomocí beginTransaction() a požadovanou operaci vložit do ní. Zaprvé je nutné vytvořit sadu objektů, která bude určena ke smazání. V případu níže je to vozidlo, které odpovídá hodnotě ID, která reprezentuje TextView příslušnou položku CardView v RecycleView. Následně se toto vozidlo smaže příkazem deleteAllFromReal(). Následovat musí commitTransaction() pro potvrzení transakce.

```
realm.beginTransaction();
RealmResults<Vozidlo> vozidlo = realm.where(Vozidlo.class).equalTo("id",
Integer.valueOf(ID.getText().toString())).findAll();
vozidlo.deleteAllFromRealm();
realm.commitTransaction();
```

## 4.7 Analýza tříd a funkcí

Níže budou vypsány všechny třídy aplikace Spravoz, které reprezentují aktivitu nebo fragment. Dále bude vysvětlena jejich funkčnost a jejich role v celém projektu.

### 4.7.1 UvodActivity

Tato třida ve svém kódu obsahuje pouze metodu registrovatPrvniVozidlo, která je vázána na tlačítko Registrovat první vozidlo pomocí parametru android:onClick. Metoda má za úkol pouze spustit aktivitu VozidloActivity.

### 4.7.2 VozidloActivity

V třídě VozidloActivity nejdříve dochází k získání id vozidla, pokud je aktivita volána pro úpravu nikoliv zakládání. Následně se pro textViewZnacky nastaví adapterZnacky, který obsahuje pole řetězců se značkami vozidel. Ve VozidloActivity je také nastaven addTextChangedListener na pole editTextSTkdo a editTextPrvniRegistrace,

který automaticky doplňuje lomítko po dvou znacích. U editTextPrvniRegistrace tento listener ještě skrývá a zobrazuje pole editTextSTkdo na základě zadaného roku registrace. Pokud je rozdíl mezi aktuálním rokem a rokem registrace větší než čtyři, pole se zobrazí. Následuje volání knihovny awesomeValidation, která zajišťuje automatické validace podle nastavených regulárních výrazů. Poté je zde podmínka, zda vozidloId je větší než nula, pokud ano nastaví se nadpis Úprava vozidla a do formuláře se nahrají data o konkrétním vozidle z databáze. V metodě registrovat, která odpovídá stisknutí tlačítka Registrovat, probíhá validace přes awesomeValidation, dále update dat v databázi, pokud je vozidloId větší než nula. Dále zde dochází k nastavení notifikací, kdy se využívá tříd Notification a především AlarmRecieved, kde je implementována hlavní logika vytváření notifikací. Interval opakování vytvořených notifikací je vyhodnocován v rámci VozidloActivity dle zadaných hodnot z polí editTextPrvniRegistrace a editTextSTKdo.

### 4.7.3 MainActivity

V MainActivity se zaprvé získá boolean prvniBeh z SharedPreferences a na základě toho se při spuštění aplikace spustí MainActivity nebo v případě prvního běhu UvodActvity. Následně se vytvoří aktivitě postranní menu a jako text v horním obrázku se nastaví název vozidla získaný z databáze. Poté dochází k nadefinování akcí na FloatingActionButton, aby docházelo při kliknutí na tlačítko ke spuštění správné aktivity. Na základě parametru fragmentID se vybere správný fragment zobrazení, výchozí je PrehledFragment, ale pokud dochází ke spouštění z VozidlaFragment aplikace se vrátí na tento fragment. Po dokončení onCreate() metody se nastaví parametru prvniBeh hodnota false, aby při příštím spuštění již došlo k nastartování MainActivity.

### 4.7.4 PrehledFragment

PrehledFragment má především za úkol naplnit RecyclerView položkami s tankováním a výdajemi. V metodě onCreateView() dochází k inicializování RecyclerView a jeho naplnění mAdapterem, kterému se nastaví ArrayList z metody getDataSet(). Do něho se přidají postupně instance třídy DataObject, které obsahují jednotlivé informace o tankováních a výdajích z databáze. Těchto pět řetězců je poté vidět ve finálním CardView a poslední šestý řetězec obsahuje hodnotu „tankovani“ nebo „vydaj“, aby se následně dle tohoto řetězce mohla vybrat správná ikona. Z důvodu toho, aby položky nebyly náhodně zobrazeny, dochází k řazení podle datumu za využití třídy

Comparator a její metody compare. Samotnou obsluhu prvku RecycleView spravuje třída MyRecycleViewAdapter, která obsahuje statickou třídu DataObjectHodler. Ta vytváří jednotlivé CardView a implementuje na nich metodu onClick(). Podle toho jestli se jedná o vozidlo nebo tankování či výdaj vytvoří po kliknutí menu. Rozdíl v menu pro vozidlo je ten, že obsahuje kromě položek smazat a editovat ještě položku vybrat. Ta funguje na principu, při kterém si třída převezme z CardView id vozidla a s tímto parametrem spustí MainActivity. Pokud uživatel klikne na položku editovat, stejným způsobem se předá id, akorát dojde ke spuštění VozidloActivity. V případě mazání dochází nejdříve k odstranění z DataSetu, následně dojde k překreslení RecyclerView a až poté ke smazání záznamu z databáze Realm.

### 4.7.5 TankovaniActivity

V TankovaniActivity se jako první provede dotaz do databáze na maximální atribut Stav tachometru do výdajů i tankování a následně se vybere maximum z těchto dvou hodnot. Tento parametr se nastaví poli editTextStavTachometru, aby si uživatel nemusel pamatovat a psát znovu celý stav tachometru, který jako poslední zaznamenal. Následně se opět využívá knihovna AwesomeValidation, která tentokrát testuje pouze na podmínku, jestli není ediText prázdný. Pokud je proměnná tankovaniId větší než nula, což znamená, že byla aktivita spuštěna pro editaci, nastaví se hodnoty z databáze. Pro automatické dopočítaní parametru na základě dvou zadaných hodnot je zde kód s využitím onFocusChangeListener, který pokud jsou dvě hodnoty neprázdné, dopočítá cenu za litr, celkovou cenu nebo počet litrů. Pro obsluhu grafického kalendáře a hodin jsou zde metody, které se starají o zobrazení i nastavení aktuálního datumu a následné načtení hodnot. Na stisknutí tlačítka uložit je vázaná metoda uloz, která nejprve zkontroluje, jestli není nový záznam v kolizi s předchozím nebo následujícím tankováním a pokud ano zobrazí Toast, který uživatele informuje, s jakým záznamem je v rozporu. Následuje již samostatné uložení nebo upravení hodnot v databázi a následné spuštění MainActivity.

### 4.7.6 VydajActivity

VydajActivity je postavena velmi podobným způsobem jako TankovaniActivity. Rozdíl je u pole typ výdaje, které má rozvíjející seznam, ze kterého uživatel vybere požadovaný typ výdaje. Princip realizace tohoto způsobu výběru dat byl již popsán v kapitole GUI.

### 4.7.7 StatistikyFragment

Ve StatistikyFragment opět dochází k převzetí id pomocí služby intent, aby třída věděla, pro jaký automobil statistiky vypočítat. Následuje onCheckedChangeListener nastavený na checBoxVsechnyVozidla, který na základě zaškrtnutí nebo odškrtnutí checkboxu přepočítává statistiky. Toto přepočítávání závisí na podmínkách, zda jsou pole editTextDatumOd nebo editTextDatumDo prázdné či naplněné, jelikož určují datumové omezení v rámci výběru dat o tankováních a výdajích. Tento přepočet je nastaven i v rámci listeneru na editTexty, které po vybrání přes grafický kalendář obsahují datumy. Když totiž uživatel změní filtr datumu, očekává, že dojde i k změně statistik. StatistikyFragment také obsahují řadu pomocných metod, jako jsou datumBezCasu nebo posledniCasDatumu, které buď nastavují datumu čas 00:00:00, nebo 23:59:00. Níže je uvedena metoda spocitejPrumernouSpotrebu, která obsahuje hlavní algoritmus pro vypočtení nejen průměrné spotřeby, ale také nákladů na jeden kilometr. Metoda nejdříve převezme výsledky tankování, jež obsahují údaje o tankování upravené na základě filtračních údajů z formuláře. Na začátku metody dochází k vytvoření pomocných proměnných. Následuje podmínka, zda je počet prvků proměnné RealmResults vysledky větší než dvě, jelikož pro jeden prvek nelze statistiky spočítat. Pokud je podmínka splněna zahájí se cyklus, který prochází jednotlivá tankování. Pokud má atribut plnaNadrz u procházeného tankování hodnotu true, pokračuje algoritmus dále, jelikož průměrná spotřeba lze spočítat pouze v případě, že se jedná o tankování plné nádrže. V ostatních případech to nelze zjistit, protože není možnost určit, kolik již bylo v nádrži paliva. Následuje podmínka na proměnnou prvniTankovani, která pokud má hodnotu 0 (což znamená, že algoritmus narazil na první tankování) pouze zaznamená stav tachometru tankování do proměnné stavTachometru. Pokud je proměnná prvniTankovani rovna jedné, potom se přičtou do pomocných proměnných jednotlivé atributy z databáze a rozdilTachometru se vypočítá jako stav tachometru aktuálního tankování mínus stav tachometru předešlého tankování. Když podmínka na plnou nádrž není splněna, uloží se pouze hodnoty počet litrů a celková cena. Po dokončení cyklu se vypočítá průměrná spotřeba jako podíl proměnných spotrebovanoCelkem a rozdilTachometruCelkem poděleného stem. Náklady na jeden kilometr jako podíl cenaCelkem a rozdilTachometruCelkem. Stejným způsobem se vypočítají i náklady na jeden kilometr z výdajů.

```
public void spocitejPrumernouSpotrebu( RealmResults<Tankovani> vysledky){

    int stavTachometru=0;
    int prvniTakovani=0;
    double spotrebovano=0;
    int rozdilTachometru=0;
    double spotrebovanoCelkem = 0;
    int rozdilTachometruCelkem = 0;
    double cena = 0;
    double cenaCelkem = 0;

    if(vysledky.size() >= 2) {
        for (Tankovani tankovani1 : vysledky) {
            if(tankovani1.isPlnaNadrz()){
                if(prvniTakovani == 0) {
                    stavTachometru = tankovani1.getStavTachometru();
                }
                if(prvniTakovani == 1){
                    spotrebovano += tankovani1.getPocetLitru();
                    cena += tankovani1.getCelkovaCena();
                    rozdilTachometru = tankovani1.getStavTachometru()-sta
                    vTachometru;
                    spotrebovanoCelkem += spotrebovano;
                    cenaCelkem += cena;
                    rozdilTachometruCelkem += rozdilTachometru;
                    stavTachometru = tankovani1.getStavTachometru();
                    spotrebovano = 0;
                    cena = 0;
                }
                prvniTakovani = 1;
            }
            else {
                spotrebovano += tankovani1.getPocetLitru();
                cena += tankovani1.getCelkovaCena();
            }
```

```
        }

    }
    prumernaSpotreba = spotrebovanoCelkem/rozdilTachometruCelkem*100;
    nakladyNaJedenKilometr = cenaCelkem/rozdilTachometruCelkem;
    nakladyNaJedenKilometrVydaje = nakladyVydaje/rozdilTachometruCelkem;
}
```

Dále jsou ve StatistikyFragment metody pro vypsání jednotlivých statistik do TextView ve správném formátu.

### 4.7.8 GrafyFragment

Ve fragmentu, který má za úkol vykreslovat grafy, se také nejdříve získává id vozidla, aby fragment věděl, pro jaký automobil grafy vykreslit. Následuje jako ve StatistikyFragment onCheckedListener na tlačítko checkBoxVsechnyVozidla, který na základě filtru datumů překresluje grafy. Pro samotný výpočet a nastavení dat grafů jsou vytvořeny metody nastavGrafy1A2(), nastavGraf3() a nastavGrafy4A5(). Grafy jsou realizovány pomocí knihovny GraphView, která nabízí možnost snadného vytvoření grafů. Princip použití je založen na třídě DataPoint, pomocí které vytváříme body a vždy se jí jako parametry předávají body x a y. Následně tento bod přidáme do pole bodů a toto pole předáme konstruktorem v případě spojnicového grafu třídě LineGraphSeries. Tuto sérii poté metodu add přidáme konkrétnímu grafu. Jelikož autor práce nenašel jiný způsob, jak celkově smazat všechny data o grafech včetně nastavení os, došlo k vytvoření vlastní třídy. Ta je pojmenována VlastniGraphView a dědí od třídy GraphView. Nově vzniklá třída implementuje metodu init, která zajišťuje kompletní smazání. V rámci GrafyFragment je poté implementována v metodě vymazGrafy(). Dále také tato třida přepisuje metody addSeries, kdy na základě typu předávané série, nastaví vhodný formát čar, případně sloupců a os.

### 4.7.9 VozidlaFragment

VozidlaFragment také převezme parametr id, aby věděl, které vozidlo je aktuálně zvoleno a má být opatřeno modrý rámečkem. Následuje zneviditelnění floatingActionMenu z fragmentu PrehledFragment a definice vlastního floatingActionButton pro přidání vozidla, které po stisknutí ihned otevře VozidloActivity. V metodě getDataSet() dochází k naplnění RecycleView záznamy s vozidly pomocí cyklu,

který prochází všechna vozidla. Z nich se vyberou pouze ty nejhlavnější atributy Název, Značka a Model, které se nastaví jako první, druhý a třetí parametr DataObjectu. Čtvrtý řetězec zůstane prázdný, pátý obsahuje id a jako šestý je přidělen řetězec „vozidlo“, podle kterého dojde k vykreslení správné ikony tedy vozidla.

## 4.8 Testování

Testování je důležitým bodem celého procesu vývoje, kdy dochází k ověření správné funkčnosti. Aplikaci Spravoz autor diplomové práce otestoval celkem na čtyřech zařízeních, ze kterých jedno zařízení byl tablet. Níže jsou vypsána všechna zařízení spolu s nainstalovanou verzí operačního systému Android.

**Seznam zařízení:**

- OnePlus 5 (Android 8.0)
- LG Nexus 4 (Android 7.1.1) (na tomto zařízení probíhal vývoj)
- Samsung Galaxy Prime Core (Android 5.1)
- Samsung Galaxy Tab A 7.0 (Android 5.1)

Na všech zařízeních bylo otestováno, zda dochází ke správnému vykreslování grafického rozhraní, jestli některý prvek není překrýván jiným a veškeré texty jsou jasně čitelné. Důležitým prvkem testování byla také zkouška, jak se aplikace chová při vertikální a horizontální poloze. Například při testování úvodní obrazovky docházelo při přetočení obrazovky do horizontální polohy k překrývání textView a uživatel neměl možnost si přečíst úvodní text. Řešením bylo nahrazení relativního layotu lineárním, který umožnil po přetočení korektní vykreslení prvků.

Samotné testování probíhalo formou testovacích případů, pro které se často využívá anglického výrazu test case. Tyto testovací případy byly vytvořeny tak, aby prověřili funkčnost, jak jednotlivých komponent, tak aplikace jako celku.

Každý test case se ve většině případů skládá:

- ID – jednoznačný identifikátor
- Účel – krátké vysvětlení k čemu test case slouží
- Podmínky – seznam potřebných dat a jiných vlivů potřebných k vykonání testovacího případu

- Specifikace kroků a vstupů – soupis všech kroků a dat pro úspěšné splnění testovacího případu
- Očekáváné výsledky – vyhodnocení výsledku testu

Autor diplomové práce zvolil pro zaznamenávání testů hojně využívanou aplikaci z kancelářské sady Microsoft Office konkrétně Microsoft Excel. Níže je uveden jeden z testovacích případů, který ověřuje správnost funkce automatického doplnění lomítka.

| | |
|---|---|
| **ID:** | 4 |
| **Název:** | Automatické doplění lomítka u První registrace vozidla |
| **Účel:** | Ověření automatického doplnění lomítka u registrace vozidla |
| **Podmínky:** | Prvně spuštěná aplikace |
| **Kroky:** | 1. Na uvodní obrazovce klikněte na tlačítko Registrovat první vozidlo a nyní jste na obrazovce Registrovat vozidlo<br>2. Zadejte do pole První registrace vozidla hodnotu 02 |
| **Očekávaný výsledek:** | Pole První registrace vozidla automaticky přidá lomítko a nová hodnota bude 02/ |
| **Provedení testu:** | OK |
| **Poznámky:** | |

Obrázek 18 - Test case automatické doplnění lomítka zdroj: (Vlastní zpracování)

Tímto způsobem došlo k ověření funkčnosti celé aplikace a nalezené chyby při testování byly opraveny.

# 5. Zhodnocení výsledku a budoucího vývoje

V rámci diplomové práce byla vyvinuta aplikace Spravoz, která slouží primárně pro firmy, aby mohly sledovat výdaje a tankování vozového parku. Aplikaci ale může využít i koncový uživatel na kontrolu nákladů vlastních vozidel.

Spravoz umožňuje snadnou správu výdajů a tankování včetně podrobných statistik a grafů, které jsou dostupné, jak za jedno vozidlo, tak za celý vozový park. Uživateli také umožnuje tyto data filtrovat dle datumu, a proto lze snadno zjistit informace o výdajích například za poslední měsíc či rok. Dále umožňuje automatickou notifikaci na blížící se prohlídku statní technické kontroly na základě parametrů první registrace vozidla nebo datumu poslední STK.

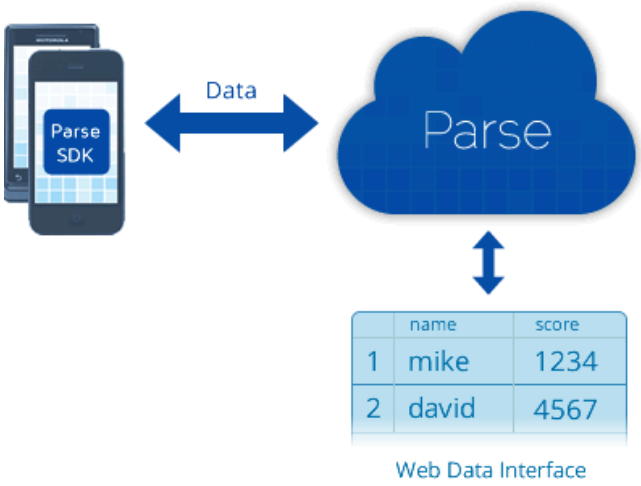
V Google Play se nachází mnoho podobných aplikací, ale ty jsou primárně určeny pro koncového uživatele, a ne pro firemní prostředí, jelikož neumí vyhodnocovat více vozidel ve statistikách a grafech. Pouze My Cars umožňuje zobrazit statistiky za více vozidel, ale její prostředí není pro uživatele příliš přívětivé a nenabízí automatické připomínání STK na základě dvou parametrů, ale musí se složitěji nastavit v menu My Cars.

Grafický design aplikace je stvořen za pomoci material designu, který společnost Google poslední verze mohutně prosazuje. Díky němu je pro uživatele snadné se v aplikace zorientovat a efektivně jí využívat. Pro seznam tankování a výdajů aplikace využívá CardViews, které nabízí možnost přehledného zobrazení včetně použití ikony pro snadné rozeznání výdaje od tankování. Celá aplikace je stylizována do modré barvy, která se vyskytuje ve dvou odstínech. Díky tomu Spravoz působí celistvým dojmem.

V rámci firemního prostředí je myšleno, že do aplikace Spravoz bude data zaznamenávat správce vozového parku, kterému budou jednotliví řidiči odesílat data o svých tankováních a výdajích. Toto řešení by mohlo být nahrazeno online databází, kdy by se touto změnou přeměnila celá architektura aplikace. Každý řidič by měl vlastní účet a po přihlášení do aplikace by zaznamenával data o výdajích na svém vozidle. Správce vozového parku by měl speciální druh účtu, kde by mohl vidět veškerá vozidla společnosti.

K tomuto účelu by mohla být využita služba Parse, která je stvořena přesně pro tento účel použití. Níže je uvedeno schéma, jak tento nástroj pracuje. V samotné aplikaci

musí nejdříve dojít k implementaci knihovny, poté se služba inicializuje a nastaví se parametry jako je adresa serveru a název aplikace. Samotná manipulace s daty probíhá přes třídu ParseObject a pro vkládání záznamů se využívá metoda put.




Obrázek 19 - Architektura Parse zdroj: (http://www.citygridmedia.com/developer/wp-content/uploads/2012/02/parse-data-storage.png)

Již ve specifikaci aplikace ovšem autor diplomové práce určil, že aplikace bude využívat pouze offline databázi, a to konkrétně Realm s objektovým přístupem. Tudíž implementace Parse nebo jiné podobné služby je možná v budoucím rozvoji.

## 5.1 Budoucí vývoj

V této části diplomové práce autor nastiňuje oblasti, které by se daly v rámci aplikaci vylepšit anebo úplně přidat.

- možnost nastavit více typu notifikací a jejich definice na základě ujetých kilometrů nebo datumu
- podpora vozidel s více nádržemi, které jsou typické pro automobily s CNG a LPG
- možnost ukládání dat a exportu CSV
- více statistik a grafů
- filtrování vypočtených údajů dle více vozidel určených uživatelem
- zobrazení cen bez DPH
- předělání aplikace na online databázi např. Parse zmiňované v minulé kapitole
- možnost zaznamenání čerpací stanice při tankování
- podrobnější členění servisu

- podpora i jiných typů vozidel než pouze osobní automobily (např. nákladní, motocykly)
- mapa s údaji, kde uživatel čerpal palivo
- automatický tracking na základě GPS
- možnost přidat fotografii k záznamu o tankování a výdajích např. s účtenkou
- podpora ukládání údajů o stavu dezénu pneumatik a oleje
- upozornění pro uživatele na zvýšenou průměrnou spotřebu oproti běžnému provozu vozidla

# 6. Závěr

Diplomová práce byla tematicky zaměřena na problematiku vývoje aplikace pro operační systém Android. Tato oblast v posledních deseti letech prošla obrovským pokrokem a stále se jedná o velmi aktuální téma.

Cílem práce bylo vytvořit aplikaci pro správu firemního vozového parku. Na Google Play existuje mnoho takových aplikací, ale většina je zaměřena pro koncové uživatele a jejich vozy. Vyhodnocovat data pro více vozidel zvládá aplikace My Cars, která má ovšem poněkud nepřehledné grafické rozhraní a při testování neprojevila stoprocentně stabilní chod. Navíc vyvíjená aplikace Spravoz umožňuje automatické připomínání STK na základě vyplnění údajů o první registraci vozidla či datumu poslední státní technické kontroly. Tuto funkci My Cars dokáže také, ale musí se vytvořit v rámci nastavení upozornění.

Teoretická část práce obsahuje základní přehled o operačním systému Android jeho historii a verzích. Dále je zde uvedena architektura systému a koncepce aplikace, kde je předvedeno, z jakých částí se aplikace skládá. Poslední část obsahuje metody, jak lze ukládat data v tomto operačním systému. Jsou zde uvedeny možnosti ukládání od jednoduchých Shared preferences až po objektovou databázi Realm, která je posléze využita v aplikaci Spravoz.

Praktická část mapuje celý proces vývoje aplikace. Jako první se autor věnuje sepsáním požadavků, pokračuje zpracováním analýzy konkurenčního prostředí. Po této analýze následují scénáře užití, kde jsou popsány některé konkrétní průchody aplikací. Pomocí prototypů jsou dále vytvořeny návrhy grafického rozhraní Spravozu. Posléze jsou tyto prototypy převedeny do skutečných jednotlivých obrazovek a je zde uvedeno, za použití, jakých prvků bylo dosaženo cíle. Na to navazuje architektura objektové databáze, která je demonstrována diagramem tříd. Autor práce zde také vypsal konkrétní využití databáze Realm v aplikaci a uvedl její výhody i nevýhody. Následuje analýza jednotlivých tříd, kde je popsána jejich funkčnost a jakých knihoven bylo při vývoji využito. Posledním krokem je testování, které bylo provedeno na čtyřech zařízeních, z nichž jedno bylo tablet. Při něm bylo využito testovacích scénářů, které se vždy soustředily na konkrétní funkci aplikace.

Výsledná aplikace Spravoz dle autora práce naplnila stanovené cíle a umožňuje evidenci tankování i výdajů. Následně aplikace poskytne jejich vyhodnocení v podobě statistik a grafů. V rámci oblasti statistik Spravoz nabízí sedm hlavních, které lze filtrovat dle datumu buď za celý vozový park, nebo pouze za zvolené vozidlo. Stejný způsob filtrů je implementován i u grafů. Tam Spravoz dokáže zobrazit pětici grafů, z nichž čtyři jsou plošné a jeden sloupcový. Ten zobrazuje rozložení typů výdajů na vozidlo. Tyto grafy jsou navrženy tak, aby byly snadno čitelné pro uživatele. Nabízejí také grafický design, který zapadá do konceptu celé aplikace.

# 7. Seznam použitých zdrojů


1. Google Buys Android for Its Mobile Arsenal [cit. 2017-08-03]. Dostupné z: https://web.archive.org/web/20110205190729/http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm
2. Android founder: We aimed to make a camera OS [cit. 2017-08-03]. Dostupné z: http://www.pcworld.com/article/2034723/android-founder-we-aimed-to-make-a-camera-os.html
3. A Murky Road Ahead for Android, Despite Market Dominance [cit. 2017-08-03]. Dostupné z: https://www.nytimes.com/2015/05/28/technology/personaltech/a-murky-road-ahead-for-android-despite-market-dominance.html
4. This was the original 'Google Phone' presented in 2006 [cit. 2017-08-03]. Dostupné z: https://www.theverge.com/2012/4/25/2974676/this-was-the-original-google-phone-presented-in-2006
5. Industry Leaders Announce Open Platform for Mobile Devices [cit. 2017-08-03]. Dostupné z: http://www.openhandsetalliance.com/press_110507.html
6. T-Mobile officially announces the G1 Android phone [cit. 2017-08-03]. Dostupné z: https://techcrunch.com/2008/09/23/t-mobile-officially-announces-the-g1-android-phone/
7. Lacko, Luboslav. Vývoj aplikací pro Android. Computer press, 2015. 472 s. ISBN 978-80-251-4347-6
8. Android 2.0, Release 1 [cit. 2017-08-03]. Dostupné z: https://web.archive.org/web/20091030044736/http://developer.android.com/sdk/android-2.0.html
9. A Brief History of Just-In-Time [cit. 2017-08-03]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.3985&rep=rep1&type=pdf
10. Android A to Z: What is the JIT? [cit. 2017-08-03]. Dostupné z: https://www.androidcentral.com/android-z-what-jit
11. Platby mobilem android pay [cit. 2017-08-03]. Dostupné z: https://www.svetandroida.cz/platby-mobilem-android-pay-201703/

12. Android 3.0 Platform Highlights [cit. 2017-08-03]. Dostupné z: https://web.archive.org/web/20110216200154/http://developer.android.com/sdk/android-3.0-highlights.html
13. Android 4.0 Ice Cream Sandwich now official [cit. 2017-08-03]. Dostupné z: https://www.engadget.com/2011/10/18/android-4-0-ice-cream-sandwich-now-official/
14. Jelly Bean [cit. 2017-08-03]. Dostupné z: https://developer.android.com/about/versions/jelly-bean.html#media
15. Android 5.0 Lollipop review [cit. 2017-08-03]. Dostupné z: http://www.techradar.com/reviews/pc-mac/software/operating-systems/android-5-0-lollipop-1271651/review
16. Android M a funkce Doze [cit. 2017-08-03]. Dostupné z: https://www.svetandroida.cz/android-m-doze-201506/
17. Android 7.0 for Developers [cit. 2017-08-03]. Dostupné z: https://developer.android.com/about/versions/nougat/android-7.0.html
18. Treble [cit. 2017-08-03]. Dostupné z: https://source.android.com/devices/architecture/treble
19. The 5 Android Oreo features you'll actually want to use [cit. 2017-08-03]. Dostupné z: https://www.cnet.com/how-to/the-5-android-oreo-features-youll-actually-want-to-use/
20. Garrett's LinuxCon Talk Emphasizes Lessons Learned from Android/Kernel Saga [cit. 2017-08-07]. Dostupné z: https://www.linux.com/news/garretts-linuxcon-talk-emphasizes-lessons-learned-androidkernel-saga
21. Android execs get technical talking updates, Project Treble, Linux, and more [cit. 2017-08-07]. Dostupné z: https://arstechnica.com/gadgets/2017/05/ars-talks-android-googlers-chat-about-project-treble-os-updates-and-linux/
22. Android Partitions Explained [cit. 2017-08-07]. Dostupné z: http://www.addictivetips.com/mobile/android-partitions-explained-boot-system-recovery-data-cache-misc/
23. Everything You Need to Know About Rooting Your Android Phone [cit. 2017-08-07]. Dostupné z:

http://lifehacker.com/5789397/the-always-up-to-date-guide-to-rooting-any-android-phone
24. Hardware Abstraction Layer (HAL) [cit. 2017-08-07]. Dostupné z: https://source.android.com/devices/architecture/hal
25. Platform Architecture [cit. 2017-08-07]. Dostupné z: https://developer.android.com/guide/platform/index.html#art
26. Lucas Jordan, Pieter Greyling. Practical Android Projects. Apress, 2011. 424 s. ISBN 978-14-302-3243-8
27. Neil Smyth. Android Studio Development Essentials - Android 6 Edition. CreateSpace Independent Publishing Platform. 2015. 710 s. ISBN 978-15-197-2208-9
28. Vyvíjíme pro Android: Notifikace, broadcast receivery a Internet [cit. 2017-08-31]. Dostupné z: https://www.zdrojak.cz/clanky/vyvijime-pro-android-notifikace-broadcast-receivery-a-internet/
29. App Manifest [cit. 2017-09-04]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-intro.html
30. The Activity Lifecycle [cit. 2017-09-15]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle.html
31. Grant Allen. Beginning Android. Apress, 2015. 411 s. ISBN 978-1-4302-4686-2
32. Activity Lifecycle With Example In Android [cit. 2017-09-27]. Dostupné z: http://abhiandroid.com/programming/activity-life-cycle
33. Android Intents – Tutorial [cit. 2017-09-27]. Dostupné z: http://www.vogella.com/tutorials/AndroidIntent/article.html
34. Intents and Intent Filters [cit. 2017-09-27]. Dostupné z: https://developer.android.com/guide/components/intents-filters.html
35. Wallace Jackson. Android Apps for Absolute Beginners. Apress, 2011. 404 s. ISBN 978-1-4302-4788-3
36. UI Overview [cit. 2017-09-27]. Dostupné z: https://developer.android.com/guide/topics/ui/overview.html
37. Layouts [cit. 2017-09-28]. Dostupné z: https://developer.android.com/guide/topics/ui/declaring-layout.html

38. Grant Allen. Android 4 - Průvodce programováním mobilních aplikací. Computer Press, 2013. 656 s. ISBN 978-8-0251-3782-6
39. Top Android IDEs for Developers [cit. 2017-10-09]. Dostupné z: https://www.developer.com/ws/android/development-tools/top-android-ides-for-developers.html
40. 10 completely different IDEs and methods for making Android apps [cit. 2017-10-09]. Dostupné z: http://www.androidauthority.com/10-completely-different-ides-and-methods-for-making-android-apps-701584/
41. Android Studio: An IDE built for Android [cit. 2017-10-09]. Dostupné z: https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html
42. Xamarin je zdarma [cit. 2017-10-13]. Dostupné z: https://www.zive.cz/clanky/xamarin-je-zdarma/sc-3-a-182184/default.aspx
43. How to start building Android apps on your Android phone using AIDE [cit. 2017-10-13]. Dostupné z: https://www.androidauthority.com/building-android-apps-with-aide-671149/
44. Multiplatformní mobilní aplikace Apache Cordova [cit. 2017-10-13]. Dostupné z: https://blogs.technet.microsoft.com/technetczsk/2016/03/18/multiplatformni-mobilni-aplikace-apache-cordova/
45. How to store data locally in an Android app [cit. 2017-11-15]. Dostupné z: http://www.androidauthority.com/how-to-store-data-locally-in-android-app-717190/
46. Storage Options [cit. 2017-11-15]. Dostupné z: https://developer.android.com/guide/topics/data/data-storage.html
47. Seznámení s SQLite [cit. 2017-11-16]. Dostupné z: https://blog.root.cz/maertienuv-obcasny-blog/seznameni-s-sqlite/
48. Realm Database [cit. 2017-11-17]. Dostupné z: https://realm.io/products/realm-database/
49. Realm Is the Best Android Database Solution [cit. 2017-11-18]. Dostupné z: https://www.toptal.com/android/realm-best-android-database-solution

# 8. Seznam obrázků