Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Aplikace android – teorie a praxe

René Uhrin

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

René Uhrin

Informatika

Název práce

Aplikace android – teorie a praxe

Název anglicky

Android application – theory and practice

Cíle práce

Cílem bakalářské práce je charakterizovat problematiku vývoje aplikací pro operační systém Android. Ten se skládá z následujících dílčích cílů: charakteristika architektury platformy Android a vývojových nástrojů, tvorba aplikace v programovacím jazyce Java, distribuce výsledné aplikace.

Metodika

Metodika je založena na studiu a analýze odborných informačních zdrojů. Na základě těchto zdrojů bude vytvořena jednoduchá aplikace. Na základě syntézy teoretických a praktických poznatků bude formulován závěr bakalářské práce.

Doporučený rozsah práce

30-40 str.

Klíčová slova

Android, mobilní operační systém, Java

Doporučené zdroje informací

ALLEN G. Android 4., Průvodce programováním mobilních aplikací. Computer Press, a.s.. 2011; 369 s. ISBN 978-80-2513-782-6

Android http://developer.android.com/guide/index.htm

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

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

ROGERS, R., LOMBARDO, J. Android Application Development, 1st Edition. Cambridge: O'Reilly Media, Inc. 2009, 336s., ISBN 978-0-596-52147-9.

Windows – http://code.google.com/android/documentation.html

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Čestmír Halbich, CSc.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 21. 10. 2016

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 10. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 08. 03. 2017

Čestné prohlášení

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

V Praze dne 4.3.2017 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Čestmíru Halbichovi, CSc. za rady a cenné poznatky. Dále bych rád poděkovat své manželce za trpělivost a podporu při zpracovávání této práce.

Aplikace android – teorie a praxe

Souhrn

V teoretické části práce, která je zaměřena na historii operačního systému Android od společnosti Google, jsou zmíněny jednotlivé verze systému a shrnutí, v čem byly tyto verze významné. Dále je zde uvedeno krátké funkční porovnání mezi největšími konkurenty v odvětví chytrých telefonů a grafické porovnání prodejnosti za jednotlivá období. V závěru teoretické části je zmíněn životní cyklus aplikace, kde jsou vysvětleny jednotlivé pasáže, kterými aplikace probíhá.

V praktické části je zachycen postup pro návrh mobilní aplikace. Dále jsou popsány základní pojmy, se kterými se lze setkat při vývoji pro tuto platformu, výběr vývojového prostředí a jednotlivé typy architektur. Poté přichází na řadu samotné programování. Na konci praktické části je popsán celý postup pro distribuci aplikace v Google Play a rozebrána optimalizace postavení aplikace ve vyhledávání.

Klíčová slova: Android, mobilní operační systém, Java, Android Studio, Google, programování, vývoj, architektura, distribuce

Android application – theory and practice

Summary

The theoretical part is focused on history of Android operating system by Google, its particular versions and the summary of their significance. There is also short functional comparison with the biggest competitors in smart phone industry and graphical comparison of sales over the past periods. In the conclusion of the theoretical part, life cycle of application and its each passages are explained.

Technique of designing mobile application is captured in the practical part. Fundamental concepts, which can be encountered during development process for this platform, selection of developmental environment and each types of architectures are further described. Then the programming itself is brought into focus. The whole process for distribution of application in Google Play and optimization of application position in search is described in the end of practical part.

Key words: Android, mobile operating system, Java, Android Studio, Google, programming, development, architecture, distribution

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

1 Úvod

První operační systém od firmy Google je datován v roce 2008. Za 9 let existence této platformy prošel mnoha aktualizacemi a technologickými změnami, které drží tento operační systém stále na výsluní. Android slouží jako prostředník mezi uživatelem a daným chytrým zařízením. Tento systém není doménou pouze mobilních telefonů, ale také tabletů a v současné době i chytrých hodinek a televizí. Vzhledem k rostoucí poptávce, si málokdo v dnešní době dokáže představit život bez mobilních zařízení, které zastávají roli pomocníků pro každodenní činnosti. To je také jeden z důvodů, proč se postupně zvyšuje počet programátorů vyvíjejících pro chytré příslušenství.

Programovací jazyky, bez kterých by tato platforma neexistovala, jsou s vývojem aplikací úzce spjaté. Aplikace lze programovat v Javě, C++, C# pro hybridní vyvíjení, ale také v Kotlinu a mnoha dalších jazycích. Nativním jazykem pro firmu Google je Java, ve které je prezentována oficiální dokumentace.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem bakalářské práce je charakterizovat problematiku vývoje aplikací pro operační systém Android. Ten se skládá z následujících dílčích cílů: charakteristika architektury platformy Android a vývojových nástrojů, tvorba aplikace v programovacím jazyce Java, distribuce výsledné aplikace. Dále objasnění konkrétních úkonů, které jsou nezbytné pro to, aby zadaná aplikace odpovídala specifikaci a v dalším průběhu nebyly zapotřebí vetší změny. V praktické části bude předváděna vytvořená aplikace v rámci této bakalářské práce.

2.2 Metodika

Metodika je založena na studiu a analýze odborných informačních zdrojů, na jejichž základě bude vytvořena jednoduchá aplikace. Na základě syntézy teoretických a praktických poznatků bude formulován závěr bakalářské práce.

V teoretické části jsou stručně sepsány jednotlivé verze OS Android a vypíchnuty funkcionality, kterými byla daná verze významná. Dále jsou porovnány dva největší konkurenti ve světě chytrých telefonů, jak z hlediska funkcionality, tak i prodejnosti. Poslední oddíl teoretické části zkoumá životní cyklus celé aplikace.

Praktická část se zabývá vytvořením autorem nadefinované jednoduché aplikace. Projekt je vyvíjen ve vývojovém prostředí Android Studio. Vytváření aplikace je zachyceno od návrhu až po publikování samotné aplikace.

3 OS Android

3.1 Historie OS Android

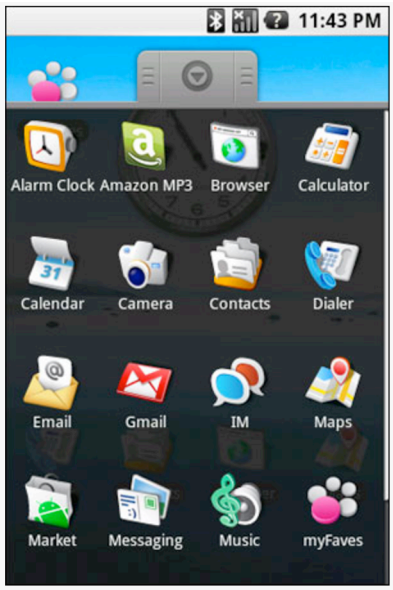
Historie operačního systému Android se datuje od 23. září 2008, kdy byla představena první verze systému. Vlajkovou lodí tohoto operačního systému byl HTC Dream/G1, který byl založen na Linuxu 2.6.25. Jak vypadal tento systém je vidět na obrázku níže. [8]

Obrázek 1 – Android 1.0
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Funkce, kterými se tato verze vyznačovala byly:

- notifikace
- synchronizace s emailovým klientem Gmail
- widgety na domovské stránce
- webový prohlížeč
- podpora fotoaparátu
- synchronizace kalendáře a kontaktů s Googlem
- aplikace Google Talk
- vytáčení hlasem
- podpora Wi-Fi a Bluetooth

Dříve Google vydával nové verze častěji, než je tomu zvykem v dnešní době. Vzhledem k tomu, že to byla první verze, nebyla samozřejmě dokonalá a 9. února roku 2009, tedy zhruba půl roku po vydání první verze, vyšla verze 1.1., zobrazena níže. [8]

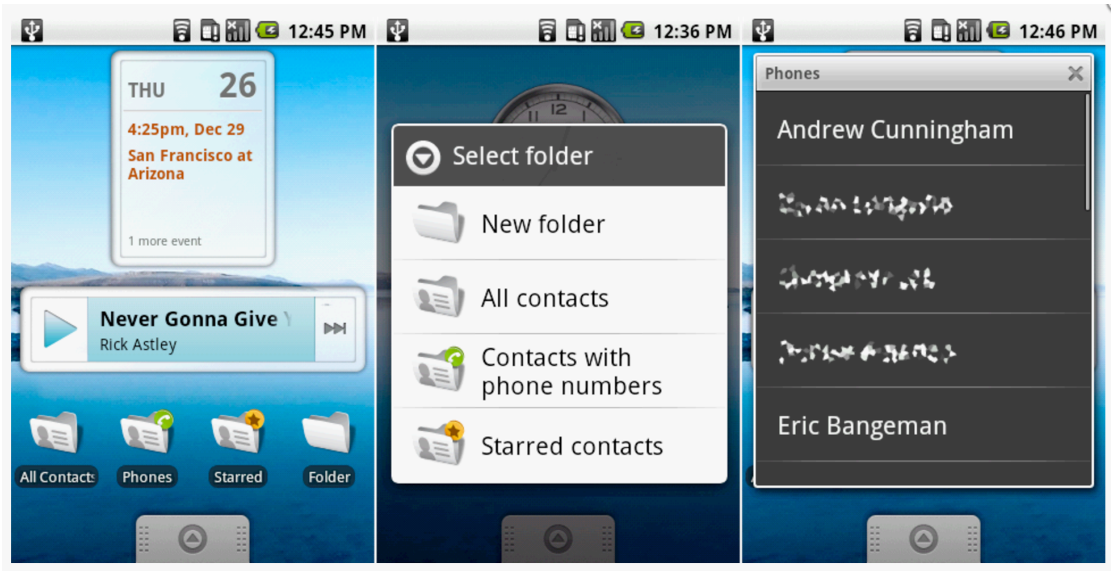


Obrázek 2 - Android 1.1
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

V této verzi také přibyly nové funkce:

- informace v mapách (firmy, podniky)
- změna intervalu vypnutí obrazovky při příchozím hovoru
- možnost ukládání příloh ze zpráv
- zobrazení a skrytí číselníku

Zanedlouho na to vyšel první operační systém Android, který byl pojmenován podle nějaké cukrovinky. Tuto tradici si firma Google zachovala až dodnes. Podle názvu je patrné, že další tradici, která se váže k názvům jednotlivých systémů, je abecední pořadí. Obrázek 3 demonstruje změny v této verzi. [8]

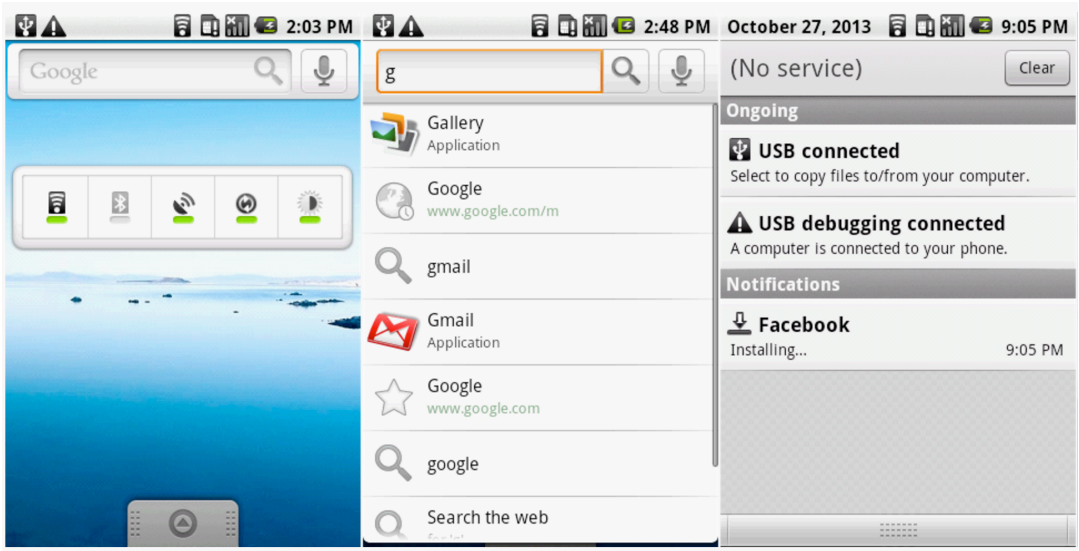


Obrázek 3 - Android 1.5
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Android verze 1.5, pojmenován Cupcake, samozřejmě dostal také nové funkce:

- klávesnice na obrazovce
- kopírování a vkládání v prohlížeči
- nahrávání a přehrávání videa
- animace při přechodu obrazovky

V nové verzi Android 1.6 Donut začali brát na zřetel různé rozměry obrazovek a celkové estetické vyznění systému. Tím, že tato verze měla podporu různých poměrů stran, se otevřeli dveře pro další zařízení. Vlajková loď tohoto systému bylo HTC Magic. Demonstrace změn níže. [8]



Obrázek 4 - Android 1.6
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Tato verze Androidu byla založena na verzi Linuxu 2.6.29. Jejími novými funkcemi byly:

- podpora různých rozlišení obrazovky
- panel pro rychlé vyhledávání
- plná integrace fotoaparátu

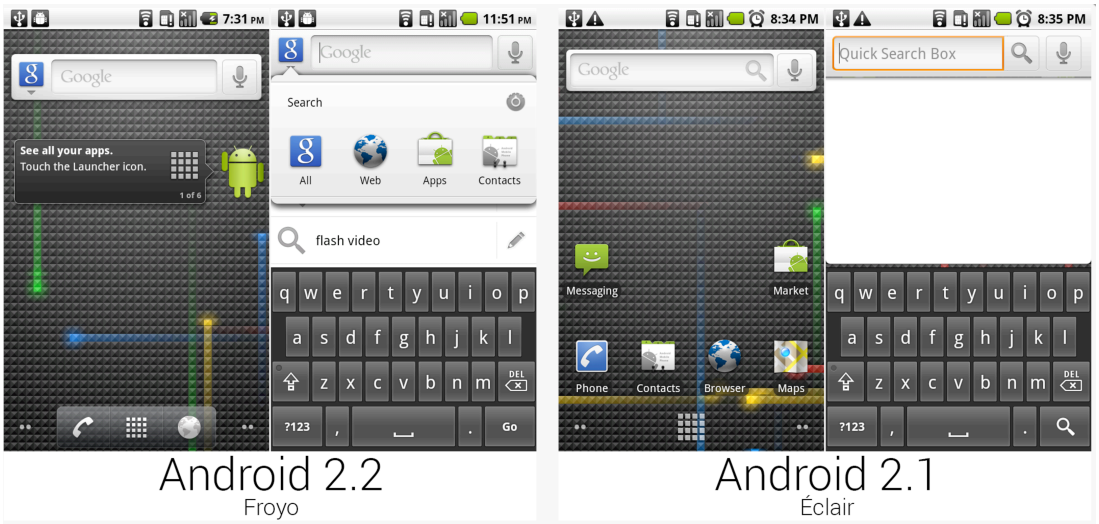
Jedna z přelomových verzí byl Android 2.0/2.1 Éclair. Vyšla pouze necelé dva měsíce po verzi 1.6, ale obsahovala velké změny. Vlajkovou lodí v tomto případě byla Motorola Droid, kterou dodal poprvé v historii Androidu jiný výrobce, než bylo HTC. Na uvedeném obrázku 5 jsou vidět změny v této verzi. [8]

Obrázek 5 - Android 2.0 / Android 2.1
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Mezi nové funkce patřilo:

- podpora více účtů
- nativní podpora MS Exchange
- převod textu na mluvenou řeč
- odemykací obrazovka
- Youtube widget pro rychlé nahrávání

Následující verze Androidu byla nazvána Froyo, jednalo se už o verzi 2.2, která se nesla především v duchu oživení uživatelského rozhraní. Od této verze zahájila firma Google prodej nové řady telefonů s označením Nexus, což bylo jedním z dalších důležitých milníků její historie. Tento model měl v sobě nainstalovaný již čistý systém, podporovaný přímo firmou Google. Z tohoto důvodu dostala tato verze, jakožto první Nexus svého druhu, označení One. Rozdíly mezi verzemi se zdají být nepatrné, ale Android verze 2.2 s sebou nese mnoho důležitých změn v UI. Na obrázku 6 jsou pro snadnější porovnání verze 2.1 a 2.2 zobrazeny vedle sebe. [8]

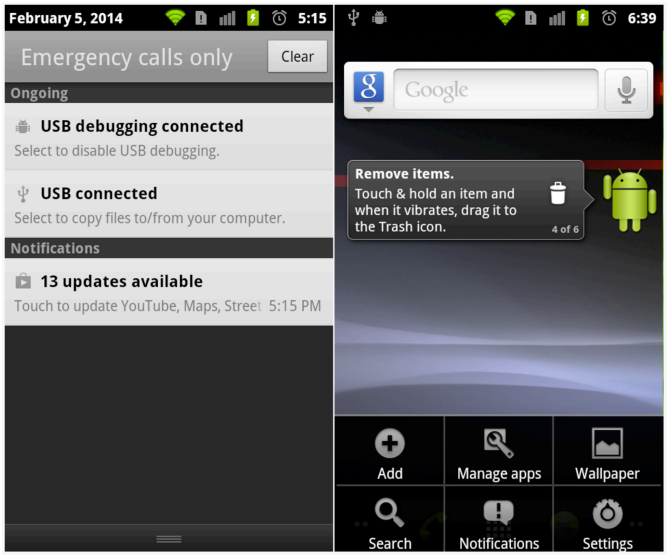


Obrázek 6 - Android 2.1 / Android 2.2
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Funkce, které jsou novinkou u této verze systému:

- podpora USB a Wi-Fi tethering
- vylepšení rychlosti, správy paměti a využívání výkonu
- push notifikace
- možnost zakázat přístup k internetu skrze mobilní data
- možnost Bluetooth handsfree v autech a docích
- podpora instalace aplikací do jiné než interní paměti
- podpora pro Adobe Flash (dnes už tato podpora neexistuje)

Společně s představením druhého zařízení řady Nexus – Nexus S, který vyráběl Samsung, byl představen také Android 2.3 se jménem Gingerbread založený na Linuxu 2.6.35. Ačkoli přinesl relativně malé změny, stal se dobrým základem pro další generace tohoto mobilního operačního systému. Na obrázku 7 je vidět grafický posun u této verze. [8]



Obrázek 7 - Android 2.3
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Mezi nové funkce této verze patřilo:

- vylepšení virtuální klávesnice
- podpora čelního fotoaparátu
- vylepšená správa energie
- aktualizace uživatelského prostředí (zjednodušení, zrychlení)
- podpora obrazovek s vysokým rozlišením (WXGA a vyšší)
- podpora SIP a VoIP
- podpora NFC – bezdrátové komunikační technologie na krátké vzdálenosti
- zvukové efekty
- vylepšená správa napájení
- vylepšení v oblasti zvuku, grafiky a vstupu se zaměřením na mobilní hraní
- podpora senzorů (gyroskop, barometr a podobné)

Android 3.0 Honeycomb byla považována ve vývoji historie operačního systému Android za slepou ulici. Jednalo se o verzi operačního systému určenou pro tablety. Nicméně popularita Androidu 3.0 byla mnohem nižší, než si Google představoval. Uživatelé i nadále používali verzi Androidu 2.x. Později začal Google nabízet tablety rovnou s novou verzí Android 4.0. S tímto novým OS byl představen telefon Motorola Xoom. Jak je ukázáno na obrázku 8, vzhled se u této verze příliš nezměnil. [8]

Obrázek 8 - Android 3.0
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Nové funkce:

- vylepšený multitasking (paralelní běh více aplikací (vláken))
- panel akčních nabídek
- podpora tabletů
- podpora hardwarové akcelerace
- HTTPS byla vylepšena metodou SNI
- seznam posledních spuštěných aplikací
- USB (USB On-The-Go)
- Změna velikosti widgetů na domovské obrazovce
- Možnost přehrát formát FLAC

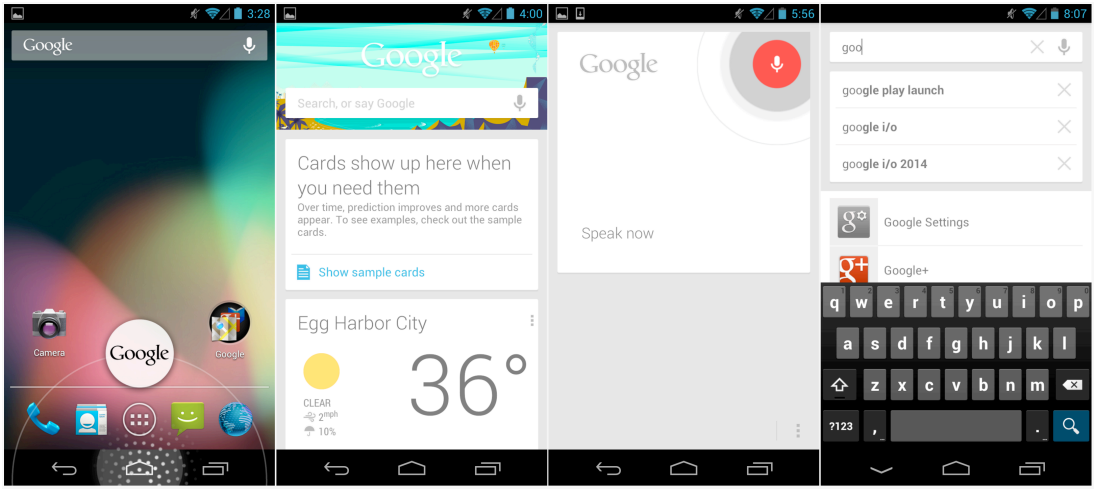
V říjnu 2011 byl, spolu s novým mobilním zařízením nazvaným Galaxy Nexus, představen Android 4.0 Ice Cream Sandwich. Byl založen na Linuxu 3.0.1, který byl po předchozím nepovedeném pokusu již správnou cestou. Tato varianta byla vhodná pro všechny typy zařízení. Podle odborníků byla tato verze jedna z nejdůležitějších kapitol v novodobé historii tohoto operačního systému. Na níže uvedeném obrázku 9 jsou ukázány změny, které s sebou tento operační systém přinesl. [8]

Obrázek 9 - Android 4.0
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Ve verzi 4.0 se objevily především tyto změny:

- notifikace podporující gesta (odstranění notifikace přetažením do strany)
- zásadní změna celého operačního systému
- souhrn a kontrola využívaných dat
- změna softwarových tlačítek (dostupnost tabletových tlačítek i na mobilních zařízeních)
- možnost spouštět aplikace přímo z odemykací obrazovky
- rozpoznání tváře
- možnost nahrávání videí v rozlišení 1080p
- VPN
- synchronizace oblíbených položek webového prohlížeče Chrome
- Android Beam

Historie operačního systému Android pokračuje verzemi 4.1 až 4.3 Jelly Bean s jádrem Linuxu 3.0.31 – 3.4.0. Tyto verze byly zaměřeny zejména na design a uživatelskou zkušenost (UI a UX). V této sérii aktualizací byl nejvíce diskutovaný Project Butter. Hlavní úlohou bylo zrychlit chod systému a celkově zmenšit veškeré prodlevy. Na níže uvedeném obrázku (obrázek 10) je vidět značný rozdíl od předchozí verze. [8]

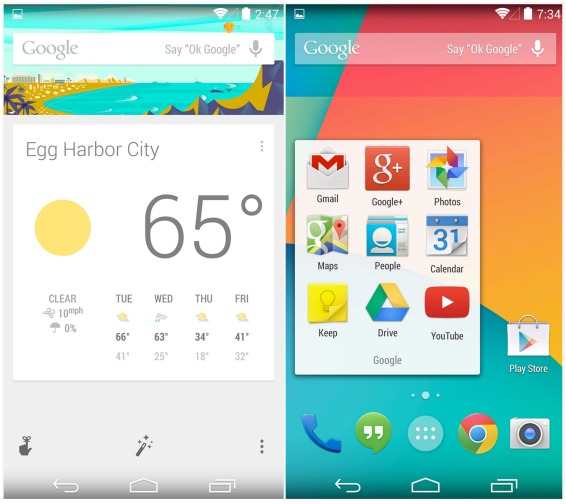


Obrázek 10 - Android 4.1 až 4.3
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Tato verze s sebou přinesla:

- plynulejší prostředí
- upravitelnost notifikace
- vylepšená práce s grafikou
- tablety s malou obrazovku mohou využívat rozšířenou verzi pro telefony
- Android Beam – přenos dat pomocí Bluetooth
- vícekanálový zvuk
- vylepšený fotoaparát
- vícekanálový zvuk
- přizpůsobivé rozhraní podle typu zařízení

Původně se měl operační systém Android verze 4.4 jmenovat Key Lime Pie, ale nakonec se Google spojil s firmou Nestlé, a tak nese tato verze název KitKat. Nejzásadněji se změny dotkly vizuální stránky systému, jak je možno vidět na obrázku 11. Tato verze se vzhledově již výrazně lišila od předchozích verzí. [8]

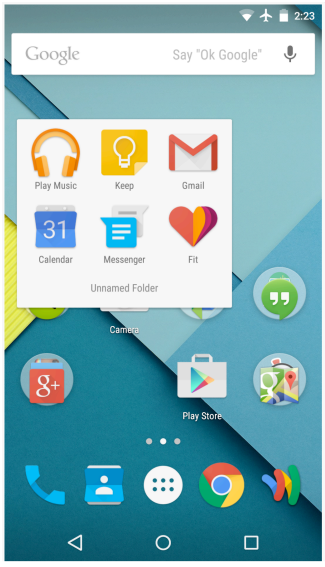


Obrázek 11 - Android 4.4
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Funkce, které byli od této verze očekávány:

- vylepšené uživatelské prostředí
- kompatibilita s Android Wear
- postupné odstoupení od dvoupalcových zařízení
- senzory pro krokoměr
- podpora bezdrátového tisku
- optimalizace výkonu (pro méně výkonná zařízení)
- poskytnutá API pro vývoj klientů zpráv

Android 5.0/5.1 nesla s sebou název Lollipop. Vlajkovou lodí tohoto systému byla Motorola Nexus 6 s jádrem Linuxu 3.4.0, která se nesla ve stylu Material designu. Tato verze se vyznačovala vylepšenou spotřebou baterie. Ilustrační obrázek 12 níže. [8]



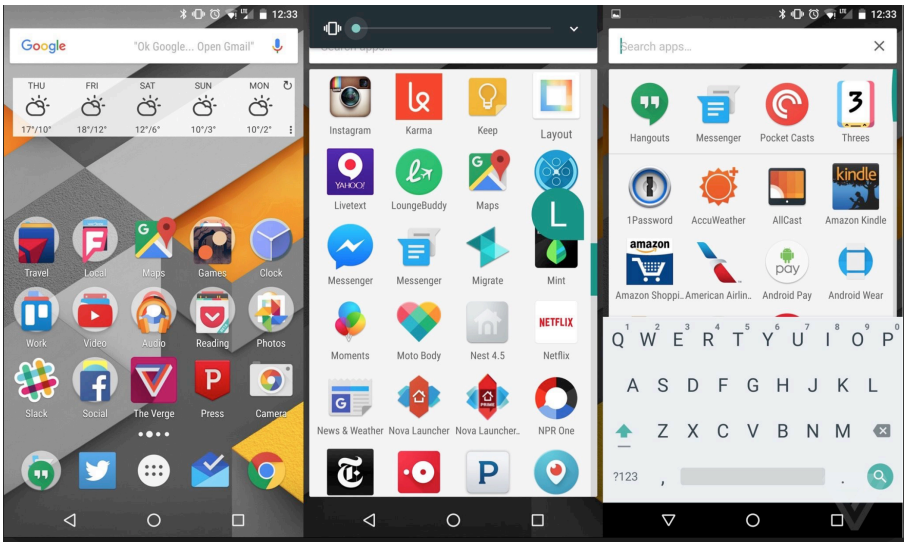
Obrázek 12 - Android 5.0 / 5.1
Zdroj: https://www.svetandroida.cz/historie-androidu-201506

Nové funkce verze 5.0 / 5.1:

- material design
- vylepšená spotřeba baterie – projekt Volta
- vylepšená oznamovací oblast
- odemykací obrazovka nabízí vstup k nastavení oznámení aplikací
- uživatelsky nastavitelné priority pro oznámení aplikací
- podpora 64 bitových procesorů
- OpenGL ES 3.1.
- funkce chytrého zamykání
- seznam naposled spuštěných aplikací si přístroj zapamatuje i po restartování
- Android Widgety nejsou přístupné na odemykací obrazovce

V květnu 2015 byla na konferenci Google I/O oficiálně představena verze Android 6 s označením Marshmallow, kde tento systém reprezentovali poprvé v historii hned dvě vlajkové lodě. Nexus 5x s menší uhlopříčkou a Nexus 6p s větší uhlopříčkou. Tento systém byl spojován hlavně s Doze módem. Tento mód slouží k značnému vylepšení výdrže baterie. Telefon sám pozná, kdy byl položen na stůl a kdy se delší dobu nepoužíval. Poté přepne do Doze módu, kde se automaticky uspí přebytečné procesy. Další změna, v oblasti povolování práv, se především týkala programátorů. Dříve bylo zvykem, že všechna práva

se povolovala při instalaci v Google Play, ale od této verze se všechna práva musela povolovat v aplikaci. Vzhled se oproti předchozí verzi lišil jen nepatrně (obrázek 13). [8]

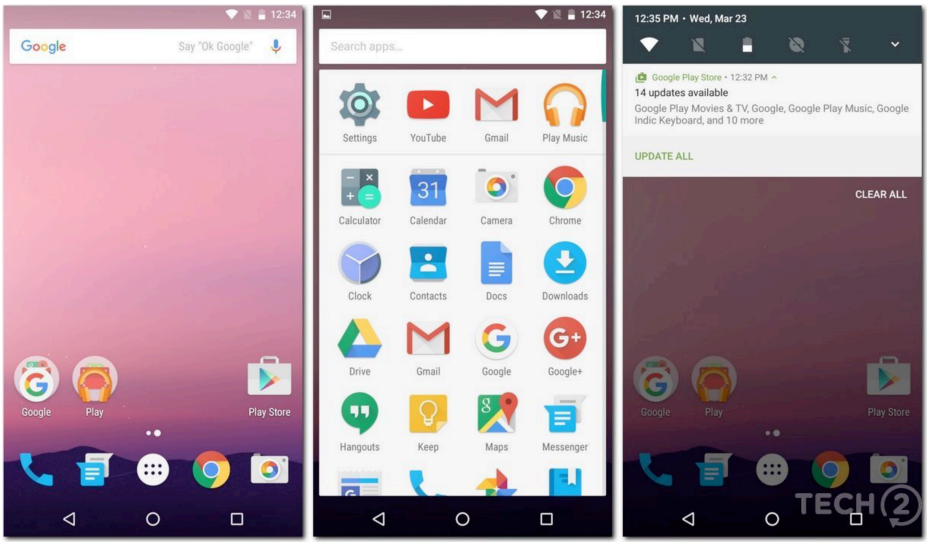


Obrázek 13 - Android 6
Zdroj: https://cdn0.vox-cdn.com/thumbor/2-ljgcWy_JXrGeqPRQb3Lts6lw8=/cdn0.vox-cdn.com/uploads/chorus_asset/file/4164678/marshmallowhome.0.jpg

Mezi nové funkce verze Android 6 se řadily:

- Podpora mobilních plateb skrze nový systém Android Pay
- Správce operační paměti
- aplikace Fotky Google
- nativní podpora snímače otisků prstů
- vylepšené ovládání hlasitosti
- podpora displejů s rozlišením 4K
- výrazně lepší podpora integrace paměťových karet
- přístup k hlasovému vyhledávání z odemykací obrazovky
- podpora konektoru USB 3.1 typu C

Posledním operačním systémem Android v historickém součtu je Android 7 Nougat. Spolu s tímto systémem se poprvé v historii představily telefony přímo od firmy Google, nazvané Google Pixel. Android 7 je přímo optimalizován pro hardware těchto telefonů. Stejný princip používají telefony od firmy Apple. Stejně tak, jak tomu bylo u předchozí verze, vzhled nebyl hlavní doménou této verze (obrázek 14). [8]



Obrázek 14 - Android 7
Zdroj: http://mashupcorner.s3-ap-southeast-1.amazonaws.com/wp-content/uploads/20160707190048/Google-Android-N-Homescreen-App-Drawer.jpg

Novinkou tohoto systému jsou rozdělené obrazovky, Doze mode 2.0 a další funkce:

- podpora virtuální reality
- API Vulkan
- rychlé přepínání mezi aplikacemi
- přímá odpověď z notifikací
- změna množství DPI na obrazovce
- rychlejší nastartování systému

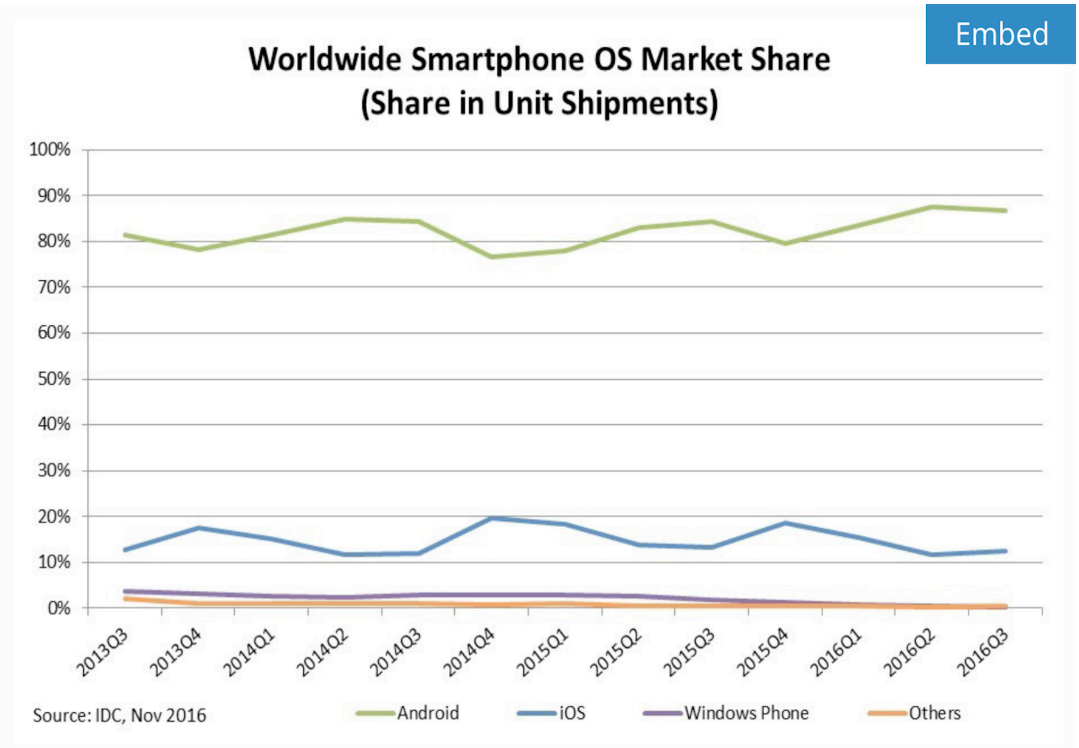
3.2 Funkční porovnání platforem Android a iOS

Obě platformy zastávají jiný názor na to, jakou otevřenost poskytnout uživateli. Pod tímto pojmem si lze představit možnost zásahu do systému a vývoj aplikací. Zatímco platforma Android je zcela otevřená a poskytuje uživateli téměř celý přístup, iOS je naopak téměř uzavřený a umožnuje minimum přístupnosti. Typickým rozdílem v přístupnosti je přenos souborů, vývoj aplikací, jejich testování na zařízeních a možnosti nastavení systému. Rozdílnost lze také vnímat v celkovém vzhledu systému, který je pro mnoho uživatelů rozhodujícím faktorem. Výběr platformy je specifický a je spíše založen na preferencích daného uživatele.

3.3 Porovnání Android vs. iOS v číslech

Poptávka po chytrých telefonech každoročně roste. V dnešní době stále více lidí nahrazuje v každodenních činnostech běžné počítače za menší a kompaktnější telefony či tablety.

Opravdovým konkurentem pro OS Android je v této chvíli pouze firma Apple se svým iOS. Tyto dvě platformy tvoří téměř veškerou poptávku po chytrých telefonech. V průběhu mobilní éry se snažila firma Microsoft prorazit na trh se svými Windows Phones. Ze začátku se jí i poměrně dařilo, jelikož zaujala především ve střední třídě s modelem Lumia. Své postavení si však dlouho neudržela, což znamenalo pro Google s OS Android a Apple s iOS, že na trhu i nadále budou dominovat. Zbylé konkurenční platformy jsou zastoupeny operačními systémy BlackBerry, Symbian a Linux.



Graf 1 – Prodejnost
Zdroj: http://www.idc.com/promo/smartphone-market-share/os

Z výše uvedeného grafu je zřejmá dominance operačního systému Android, jehož podíl na trhu chytrých telefonů se pohybuje mezi 75% a 90%. Tento podíl na trhu je také dá tím, že systém Android je používán u více značek (Samsung, Lenovo, LG a mnoho dalších), zatímco iOS je používán pouze u telefonů od firmy Apple, kterým patří podíl mezi 10% a 20%. Z tabulky uvedené níže vyplívá, že se poslední kvartál roku 2015 a první

tři čtvrtletí roku 2016 nesou v duchu posilování systému Android a vyklízení pozic Windows Phones.

Tabulka 1 - Podíl na trhu
Zdroj: http://www.idc.com/promo/smartphone-market-share/os

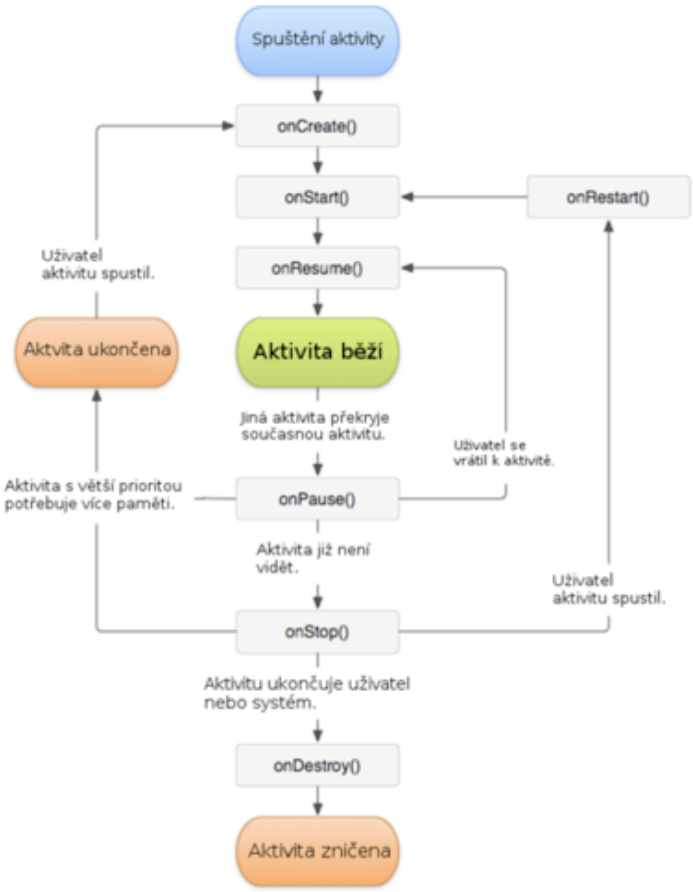
Period	Android	iOS	Windows Phone	Ostatní
2015Q4	79,6%	18,7%	1,2%	0,5%
2016Q1	83,5%	15,4%	0,8%	0,4%
2016Q2	87,6%	11,7%	0,4%	0,3%
2016Q3	86,8%	12,5%	0,3%	0,4%

3.4 Životní cyklus

Android aplikace mají takzvaný životní cyklus. Tento koloběh aplikace si lze představit jako stavy, které říkají, jak se mají chovat aktivity/fragmenty, popřípadě logika dané obrazovky. Oba tyto objekty mají životní cyklus velice podobný, ale v některých detailech se rozchází. Porozumění životnímu cyklu aplikací je velice důležité pro správné chování aplikací v různých režimech. [1][2]

3.4.1 Životní cyklus aktivit

Zapnutím aplikace je okamžitě vyvolána spouštěcí aktivita, která odstartuje životní cyklus. Dokud je aplikace přítomna v systémovém zásobníku, cyklus stále běží. Celý proces od spuštění aktivity, přes běh programu, až po její zničení je znázorněn na obrázku 15, který je umístěn níže.



Obrázek 15 - Životní cyklus aktivity
Zdroj: http://www.itnetwork.cz/images/17568/lifecycle.png

Při spuštění aktivity se Android postará o to, aby se vytvořil objekt a spustil daný proces. Dále zavolá metodu onCreate(). V této metodě se nadefinuje vše potřebné, aby aktivita mohla fungovat. Mezi typické prvky patří: grafické rozhraní, elementární logika apod. Android studiem vygenerovaný kód pro metodu onCreate().[1][2]

```
01.  @Override
02.      protected void onCreate(Bundle savedInstanceState) {
03.          super.onCreate(savedInstanceState);
04.          setContentView(R.layout.activity_main);
05.          }
06.      }
```

Obrázek 16 - onCreate() metoda
Zdroj: vlastní

Aktivita v metodě onCreate() nastaví svůj layout pro vzhled dané obrazovky (řádek 4).

Odkazuje na activity_main, která je uložena v adresáři res/layout. R. je automaticky vygenerována třída s referencemi.

Další metodou v životním cyklu je metoda onStart(). Tato metoda je volána pokud je aktivita nově nainicializovaná nebo pokud byla vyvolána do popředí (příchozí SMS, přesun aplikace do pozadí a znovu spuštěna).

Metoda onResume() je volána ve stejných situacích jako je metoda onStart(), ale navíc je volána ve chvíli, kdy uživatel přeskakuje mezi obrazovkami. Po této metodě aktivita běží.

onPause() je metoda, která se vyvolá okamžitě po tom, co dojde k překrytí aktivity/fragmentu jinou aktivitou/fragmentem.

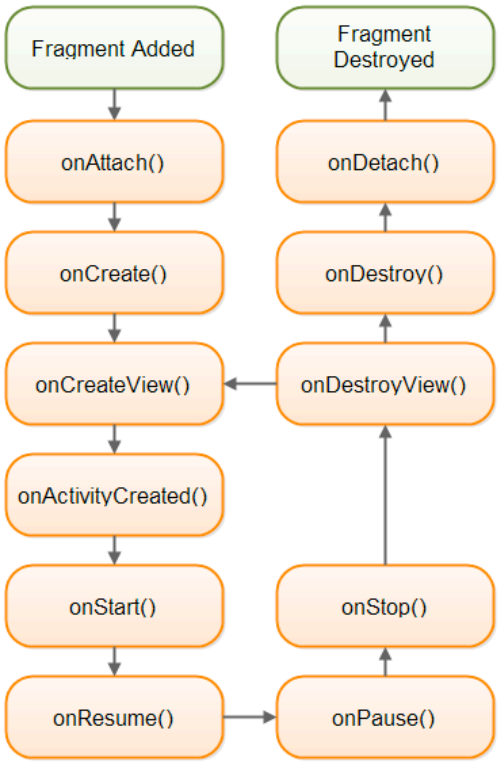
Pokud je aplikace na pozadí v zásobníku, tak je po metodě onPause() hned volána metoda onStop(). Tato metoda zajišťuje, aby aktivita/fragment nebyla zničena, protože jsou na ni stále aktivní reference.

Každá aplikace má přidělené místo od systému, se kterým může pracovat. Proto si aplikace musí hlídat, jaké objekty jsou stále aktivní. Pokud je velikost místa překročena, aplikace už nadále nemůže pracovat a tudíž je systémem vypnuta. Z tohoto důvodu je v systému Android poslední metodou v tomto životním cyklu metoda onDestroy(). Pokud je zavolána tato metoda, tak je celá obrazovka zničena a alokovaná paměť je opět k dispozici dalším akcím.

Při programování není potřeba používat vždy všechny výše zmíněné metody. Jediná metoda, která musí být vždy volána je onCreate(). Ostatní metody jsou sice volány (systém Android musí postupovat celým životním cyklem), ale pokud tyto metody nejsou přepsány (takovou metodu poznáme podle slova @Override nad metodou), tak na ně nemusí být brán zřetel. [1][2][4]

3.4.2 Životní cyklus fragmentů

Na první pohled je zřejmé, že je v životním cyklu fragmentů v aplikaci zastoupeno větším množství metod než v životním cyklu aktivit. Tyto fragmenty jsou ukládány do zásobníku, o který se stará FragmentManager. Fragmenty dodávají aplikacím flexibilitu a znovupoužitelnost uživatelského rozhraní. Rozdělení metod životního cyklu fragmentů je zobrazeno níže na obrázku 17.



Obrázek 17 - Životní cyklus fragmentu
Zdroj: http://cdn.journaldev.com/wp-content/uploads/2015/10/android-fragment-lifecycle.png

Většina těchto metod se chová tak, jak bylo popsáno u životního cyklu aktivit. Rozdílné chování tvoří metody, které jsou v životním cyklu fragmentu navíc.

V tomto cyklu je lépe vyjádřeno, kdy se má inicializovat vše týkající se grafiky. Pro tuto činnost slouží metoda onCreateView().

Metoda onDestroyView() je volána v případě, kdy není na objekt vázána žádná reference. To ovšem neznamená, že metoda onDestroyView() bývá volána hned před metodou onDestroy(), protože fragment může sice být zničen, ale přiléhlá aktivita může být stále aktivní. Tyto metody jsou naprosto nezávislé. [6]

4 Android aplikace

4.1 Téma aplikace

Aplikace funguje jako prostředník mezi serverem a uživatelem. Při startu aplikace se uživateli ukáží příspěvky, které jsou k dispozici na portálu o vaření. Při procházení příspěvků se automaticky načítá následující obsah. Po kliknutí na text daného tématu může uživatel očekávat detail příspěvku.

Všechny načtená data se ukládají do cache, tudíž pokud si data uživatel předem načte, může aplikaci použít i v případě, že není k dispozici připojení k internetu. Pokud je zařízení bez internetového připojení a uživatel chce načíst data, která nejsou k dispozici, aplikace upozorní uživatele na absenci internetového připojení a zviditelní nabídku s opakováním akce. Aplikace má také upravený vzhled pro podporu tabletů. Dále aplikace podporuje Android zařízení od verze IceCreamSandwich (verze 4.0) a výše.

4.2 Vytvoření vzhledu a UX/UI aplikace

Aby bylo možné začít programovat, je velice důležité stanovit si, jak bude daná aplikace vypadat a specifikovat její chování.

4.2.1 UML diagram

UML (Unified Modeling Language) je grafický jazyk, který se používá při vývoji informačních systémů. Při návrhu aplikací se tento jazyk stal nedílnou součástí vývoje, a proto je důležité, aby se v něm programátoři uměli orientovat.

Celý jazyk UML je založený na třech elementech, které jsou reprezentovány grafickými značkami. Mezi základní elementy patří předměty, relace a diagramy.

Předměty jsou elementy zpracovávaného modelu, které jsou členěny do několika navzájem rozdílných podkategorií.

Dalším elementem je relace, která zajišťuje spojení mezi různými předměty (relace patří mezi základní element UML).

Posledním elementem je diagram, který zachycuje aspekty modelového systému.

4.2.2 Drátový model

Drátový model je termín popisující funkční prvky aplikace (neboli jak bude aplikace vypadat) a jednotlivé rozmístění prvků po obrazovce. Důležité je si uvědomit, že se nejedná o grafický návrh (ten je zhotoven na základě předložení hotového drátového modelu). Tento model se skládá pouze z daných obrysů a textů, kde je doporučeno používat pouze barvu černou a bílou (např. odkazy jsou jedna z výjimek, kde se může použít navíc speciální barva). Tento drátový model se v praxi téměř nevyskytuje, protože bývá často zavádějící.

Existuje více druhů drátových modelů. Nejzákladnější z nich je tzv. textový drátový model, ve kterém je vše popsáno v textu (není podobný schématu, jedná se spíše o souvislý text).

Další v řadě drátových modelů je blokový. Nejdůležitějším prvkem je rozmístění bloků. Bloky mají obvykle pouze stručný popis svého obsahu. Jak už je z názvu patrné, podrobný drátový model obsahuje detailní blokové schéma a rozvržení jednotlivých stránek. Dané bloky by měly obsahovat texty, které se plánují použít v aplikaci (potom je následný proces výroby aplikace mnohem snadnější).

Poslední typ je proklikávací drátový model. Tento typ je velice podobný předchozímu modelu a používá se u rozsáhlejších projektů typu B2B, B2C atp. Výhodou tohoto modelu je jeho interakčnost, díky které nedochází k nedorozumění, jelikož je jasně daná funkcionalita.

Nejznámější programy pro tvorbu drátových modelů jsou UXPin, Fluid UI, Balsamiq Mockups a Axure.

Vzhledem k rozsáhlosti projektu je zvolen blokový drátový model, který je vypracovaný v papírové podobě.

4.2.3 UX aplikace

Díky IT technologiím se do veřejného podvědomí dostává pojem UX neboli uživatelská zkušenost. Tento princip použitelnosti se však používá v každodenním životě i mimo mobilní telefony. Cílem UX je udělat takový produkt, jehož účel je zřejmý a ovladatelnost intuitivní. Tyto principy z reálného světa se postupně přenesly do světa virtuálního.

Pro projekt v bakalářské práci jsou použity konvence od společnosti Google.

4.2.4 UI aplikace

Grafické uživatelské rozhraní je kombinace technologie a prostředků, které umožnuje uživateli komunikovat s aplikací. U mobilních aplikací se jedná hlavně o grafické nebo textové prvky a jejich jednotlivé umístění. Ovládací prvky umožňují uživateli práci s aplikací, která získává od uživatele vstupní data. Tyto vstupní data aplikace vyhodnotí a prezentuje je zpátky uživateli. Uživatelské rozhraní ovlivňuje vnímání celé aplikace. První pohled a pocit z aplikace dále určuje, jakým způsobem budou uživatelé s aplikací pracovat a co mohou od aplikace očekávat.

Hlavním poznávacím znamením u moderního grafického rozhraní je oddělitelnost obsahu, který je výhodou při větších úpravách aplikace, aktualizacích nebo při tvorbě nových grafických variant.

V praxi existují určitá doporučení, která by se měla při tvorbě uživatelského prostředí dodržovat. Jedním z hlavních pravidel je viditelnost stavu, přičemž systém musí vždy vykazovat uživateli, že proces probíhá. Dalším příkladem je pravidlo konzistence v zobrazování prvků.

V bakalářském projektu je využíván Material design, který se defaultně používá pro nativní aplikace platformy Android.

4.3 Architektura aplikace

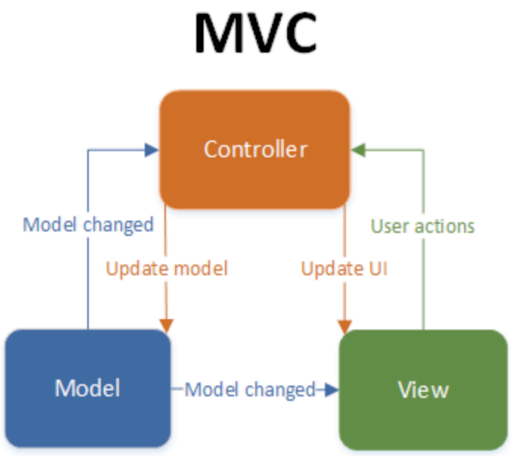
Přestože je zadání aplikace dané již před samotným tvořením aplikace, neznamená to ještě, že se specifikace nemůže kdykoli změnit nebo rozšířit. Snahou programátorů je vždy vytvořit takovou architekturu aplikace, díky které by byla při budoucí změně práce co nejméně náročná. V dnešní době jsou u platformy Android nejvíce používané tyto návrhové vzory: MVC (model-view-controller), MVP (model-view-presenter) a MVVM (model-view-view model).

Hlavními výhodami těchto modelů jsou: rozdělená logika jednotlivých vrstev, mnohem snazší testovatelnost (unit testy) a v neposlední řadě snadná rozšiřitelnost. Samozřejmě má každý z těchto návrhových vzorů svá pozitiva i negativa.

4.3.1 Návrhový vzor MVC

MVC je první z této řady, které bylo používáno. Tento návrhový vzor je velmi často používaný u platformy iOS nebo u webových aplikací. Vrstvou, kde vše začíná, je view. View obsahuje prvky, které vidí uživatel na displeji telefonu. Další vrstvou je controller, kde se řeší např. uživatelské akce. Poslední vrstvou je model, kde se očekává serverová komunikace nebo komunikace s databází.

Výhodou a zároveň nevýhodou u tohoto návrhového vzoru je skutečnost, že všechny vrstvy mohou komunikovat navzájem. Výhodou je, že lze předávat data napřímo do logiky vykreslování. Zatímco nevýhodou tohoto systému je, že view mohou měnit obě zbývající vrsty, což může vést k nekonzistentnímu vykreslování. [11]



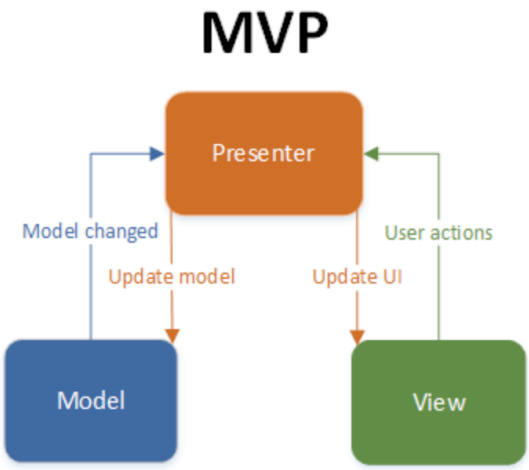
Obrázek 18 - Návrhový vzor MVC
Zdroj: http://www.techyourchance.com/wp-content/uploads/2015/06/MVC_MVP.png

4.3.2 Návrhový vzor MVP

Dalším zmíněným návrhovým vzorem je MVP. Tento model je často zaměňován s modelem MVC, primárně z důvodu podobné logiky. Hlavním poznávacím znamením a výhodou tohoto modelu je, že pouze presenter může komunikovat s modelovou vrstvou a view vrstvou. Stejně jako u MVC je i u tohoto návrhového vzoru view oddělené místo, kde jsou všechny prvky dané obrazovky, které vidí uživatel. Model slouží jako komunikační vrstva se serverem nebo databází.

Největší rozdíl spočívá v presenteru, který nastavuje přenos dat z modelové vrstvy do view vrstvy, posílá uživatelský požadavek a také nastavuje zobrazení (např. načítání).

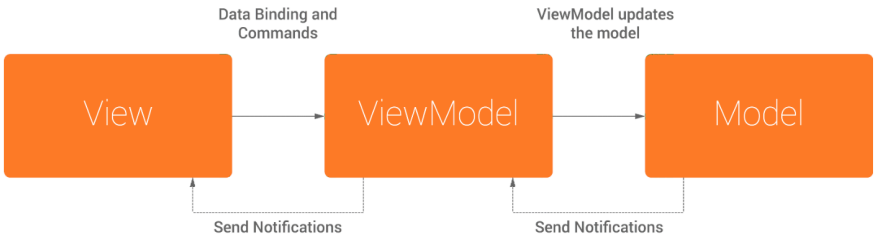
Celá komunikace tedy probíhá jen jedním směrem a nemůže se tedy stát, že by se aplikace chovala nekonzistentně. Nevýhodou je nutnost komunikovat s další vrstvou, přestože to není vždy potřeba. [11]



Obrázek 19 - Návrhový vzor MVP
Zdroj: http://www.techyourchance.com/wp-content/uploads/2015/06/MVC_MVP.png

4.3.3 Návrhový vzor MVVM

Posledním návrhovým vzorem, který je v této souvislosti spojován s předchozími návrhovými vzory je MVVM. View a model mají stejnou funkci jako u MVC a MVP. ViewModel je část, která řeší akce uživatele, chyby vytvořené uživatelem a další. Výhodou tohoto návrhového vzoru je skutečnost, že je nejvhodnější pro použití v situaci, kdy se recyklují části obrazovek. Ty je možné dále skládat do nových obrazovek. Nevýhodou je, že se tento návrhový vzor spíše hodí pro složitější obrazovky. Při jednodušší obrazovce se stává kód komplikovanější, než nabízí např. MVP. [12]



Obrázek 20 - Návrhový vzor MVVM
Zdroj: https://cdn-images-1.medium.com/max/800/1*VLhXURHL9rGlxNYe9ydqVg.png

Při výběru návrhového vzoru hraje vždy hlavní roli specifikace dané aplikace. Při rozhodování, jakým způsobem danou architekturu programovat, vždy rozhodují detaily. Pro naprogramování aplikace v této bakalářské práci byl vybrán model MVP.

4.4 Vývoj aplikace

Při samotném programování aplikace se programátoři drží obecně známých konvencí, aby byl jejich kód snadněji čitelný i pro jiné programátory. Snahou programátora má být rozdělení jednotlivých částí logiky do takzvaných tříd, poté eliminovat dílčí části a nakonec vytvořit metody.

4.4.1 Výběr vývojového prostředí

Nativní aplikace se dají vyvíjet ve vícero programovacích prostředích. První programovací prostředí, které se používalo pro vývoj aplikací byl program NetBeans sponzorovaný společností Oracle. Pro nepříliš velkou podporu od společnosti Google, kvůli které měli programátoři k dispozici pouze základní spektrum komponent a zbylé komponenty si museli dodělat sami, přestalo být vývojové prostředí používáno.

Dalším vývojovým prostředím bylo Eclipse for Android developers od společnosti IBM. Toto vývojové prostředí se v některých starších projektech používá dodnes.

Zatím jediné vývojové prostředí, které je přímo podporované od společnosti Google a nejvíce používané programátory je Android Studio, které je založené na IntelliJ IDEA. Oficiálně představen byl 16.května 2013 na konferenci Google I/O v Kalifornii. Od června téhož roku je pro uživatele zdarma k dispozici. Pro projekt zpracovaný v této bakalářské práci je použito poslední zmíněné prostředí. [10]

4.4.2 Seznámení se ze základními pojmy v projektu

Nově vytvořený projekt obsahuje několik automaticky vygenerovaných složek a souborů. Pro programátora je nejdůležitější složka se jménem src. V dalších podadresářích této složky se dále nacházejí tzv. třídy (aktivity, fragmenty apod.) (logika aplikace), xml soubory (pro UI část), AndroidManifest, Gradle soubor a proguard-rules.pro.

V adresářích aplikace Android je více druhů souborů typu XML. Převážně se používají pro deklaraci grafických prvků, ale také pro vytvoření souboru AndroidManifest (uveden níže), jazykové mutace, soubory pro ukládání hodnot proměnných a mnoho dalšího.

Každá Android aplikace má právě jeden soubor AndroidManifest. Tento soubor uvádí systému Android základní informace o aplikaci, které musí mít před spuštěním.

Gradle je automatizační nástroj, psán v jazyce DSL (Domain Specific Language), který je založen na jazyku Groovy. Výhody tohoto nástroje je např. skutečnost, že se nemusí fyzicky stahovat externí knihovny do projektu (po přidání speciálního znaku může knihovny aktualizovat), dále pomáhá při kompilaci, spouští unit a integrační testy se speciálním nastavením, dokáže projekt spouštět ve více režimech (slouží pro rozdělení verzí) a další.

Soubor proguard-rules.pro je používán od Androidu verze 2.3. Vzhledem k tomu, že je jazyk Java velmi jednoduše dekompilovatelný jazyk, hlavní funkcí tohoto souboru je obfuskace kódu. Dalšími funkcemi jsou optimalizace a zmenšení kódu či odstranění nepotřebných metod a proměnných.

Aplikace musí mít v projektu minimálně jednu aktivitu. Automaticky vygenerovanou aktivitou, kterou celé programování začíná, je s typickým názvem MainActivity.class. Tuto aktivitu je možné dědit v dnešní době nejpoužívanějšími předky (dříve existovali i jiné, která se již nepoužívají) typu: AppCompatActivity, FragmentActivity nebo pouze základní Activity. Každý z těchto předků s sebou nese určité výhody, proto je dobré vědět, jaký typ a kdy použít.

První z řady je Activity, která je předkem obou následujících typů. Activity zaručuje funkčnost obrazovek a je základní komponentou pro vykreslení grafického návrhu. Systém spouštění aktivit je založen na principu LIFO (poslední dovnitř první ven). Každá nově vytvořená aktivita musí být deklarována v souboru AndroidManifest.

AppCompatActivity je předek, který zaručuje podporu pro starší verze systému, díky které je možné používat komponenty z nejnovějšího API.

Aplikace má většinou pouze jednu aktivitu, ostatní obrazovky se skládají z tzv. fragmentů (další aktivity se typicky používají pro přihlašování do aplikace nebo pro nastavení). Někdy je zapotřebí více než jen fragment, z tohoto důvodu je k dispozici FragmentActivity. Tento typ má výhody (ale i nevýhody) obou těchto typů.

Před samotným programováním je vhodné zhodnotit, jaký typ předka použít, jelikož není vždy nutné např. mít podporu pro starší verze systému.

Jak bylo již zmíněno výše, další důležitou složkou při vývoji pro tuto platformu jsou fragmenty. K dispozici jsou od API verze 11, tedy od verze 3.0. Vzhledem k tomu, že

jsou fragmenty úzce spjaty s aktivitou, na které se fragment/fragmenty zobrazují, ovlivňuje životní cyklus aktivity také životní cyklus fragmentu. Jakmile aktivita přejde např. do stavu onPause(), tak i veškeré fragmenty přejdou do tohoto stavu. Fragmenty dávají větší možnosti ve variabilitě vytvoření vzhledu (může se kombinovat více fragmentů v jedné aktivitě, např. rozdělení obrazovky, která je typická na tabletových zařízeních). Každá aktivita nebo fragment má svůj layout, který reprezentuje vzhled obrazovky. Je to soubor typu XML, který definuje jednotlivé prvky. Každý prvek musí obsahovat informaci o velikosti, mít vlastní identifikátor a případné další specifikace (text, barvu pozadí, vycentrování apod.). Informace o velikosti lze definovat několika způsoby. Nejznámější způsob je nastavení výšky nebo šířky na tzv. match_parent, který objekt zvětší na velikost, kterou má jeho rodič. Dalším způsobem je použití tzv. wrap_contentu, při němž se objekt zvětší na takovou velikost, aby pokryl vše, co je uvnitř. Těmto objektům lze také nastavit specifickou velikost v měrných jednotkách DP (dots per inch). Tyto jednotky jsou přepočítávány z pixelů. [1][2][3][5]

4.4.3 Programování aplikace

Tato aplikace se skládá z jedné aktivity a dvou fragmentů. Aktivita řeší společnou logiku životního cyklu pro fragmenty, elementární logiku a zároveň slouží jako obal pro zobrazení fragmentů.

```
01. public class MainActivity extends AppCompatActivity {
02.
03.     String TAG = MainActivity.class.getSimpleName();
04.
05.     @Override
06.     protected void onCreate(Bundle savedInstanceState) {
07.         super.onCreate(savedInstanceState);
08.         setContentView(R.layout.activity_main);
09.
10.         // Check whether the activity is using the layout version with
11.         // the fragment_container FrameLayout. If so, we must add the first fragment
12.         if (findViewById(R.id.activity_main) != null) {
13.
14.             // However, if we're being restored from a previous state,
15.             // then we don't need to do anything and should return or else
16.             // we could end up with overlapping fragments.
17.             if (savedInstanceState != null) {
18.                 return;
19.             }
20.
21.             // Create an instance of ExampleFragment
22.             ListFragment firstFragment = new ListFragment();
23.
24.             // Add the fragment to the 'fragment_container' FrameLayout
25.             getSupportFragmentManager().beginTransaction()
26.                     .add(R.id.activity_main, firstFragment).commit();
27.         }
28.     }
29. }
```

Obrázek 21 - MainActivity.class
Zdroj: vlastní

Pro spuštění fragmentu slouží SupportFragmentManager, který zpracovává požadavky a popřípadě je ukládá do fragmentového zásobníku, jak je zřejmé z obrázku 21. SupportFragmentManager požaduje konkrétní identifikátor obalu a následně fragment k zobrazení. Poté musí být potvrzena změna (řádek 25 a 26).

Projekt je podporován i pro starší verze systému, z tohoto důvodu dědí MainActivity předka AppCompatActivity.

Komunikační tunel mezi jednotlivými fragmenty a presenterem má zajistit interface IMainView. [9][7]

```
01. public interface IMainView {
02.
03.     String TAG = IMainView.class.getSimpleName();
04.
05.     void showProgress(boolean show);
06.     void onGeneralError(String error);
07.     void onNotifyArray(Information basePojo);
08.     void onAppStart(ArrayList<?> data);
09.
10. }
```

Obrázek 22 - Hlavní metody Presenteru
Zdroj: vlastní

Vzhledem k vybranému návrhovému vzoru fragmenty slouží pouze pro ukazování aktuálního stavu uživateli a k zobrazení dat. K tomu slouží metody uvedené výše na obrázku 22.

Zmíněný interface má implementovaný fragment, který zobrazuje list témat, jak je uvedeno v obrázku níže.

```
01.  public class ListFragment extends Fragment implements IMainView
```

Obrázek 23 - Hlavička ListFragment.class
Zdroj: vlastní

Presenter musí být k dispozici od raného vytvoření fragmentu.

```
01.  @Override
02.      public View onCreateView(LayoutInflater inflater, ViewGroup container,
03.                               Bundle savedInstanceState) {
04.          // Inflate the layout for this fragment
05.          View view = inflater.inflate(R.layout.fragment_list, container, false);
06.          getPresenter().onViewAttached(this);
07.          return view;
08.      }
09.
10.      private MainPresenter getPresenter() {
11.          return mPresenter;
12.      }
13.
14.      @Override
15.      public void onDestroyView() {
16.          super.onDestroyView();
17.
18.          getPresenter().onViewDetached();
19.          if (getActivity().isFinishing()) {//activity finishing, release reference to presenter
20.              mPresenter = null;
21.          }
22.      }
```

Obrázek 24 – Presenter
Zdroj: vlastní

V metodě onCreateView() je hned po inicializaci vzhledu (řádek 5) na obrázku 24, přidělena informace o fragmentu presenteru (řádek 6). Pokud je aplikace z nejrůznějších důvodů přerušena, musí se smazat reference na tento fragment. K tomuto účelu slouží systémová metoda onDestroyView(), ve které je tato reference na řádku 18 zrušena. [5][6]

Po implementaci musí mít třída k dispozici všechny dané interface metody. V těle těchto metod je zároveň vše, co má fragment funkčně vykonávat.

```
@Override
    public void showProgress(boolean show) {
        if (show) {
            mPbLoading.setVisibility(View.VISIBLE);
        } else {
            mPbLoading.setVisibility(View.GONE);
        }
    }

    @Override
    public void onGeneralError(String error) {
        CustomToast.getLongToast(getContext(), error);
        mRlErrorWrapper.setVisibility(View.VISIBLE);
    }

    @Override
    public void onNotifyArray(Information information) {
        mRlErrorWrapper.setVisibility(View.GONE);

        for (int i = 0; i < information.getItems().size(); i++) {
            mItemsArrayList.add(information.getItems().get(i));
        }

        mRvAdapter.notifyDataSetChanged();
    }

    @Override
    public void onAppStart(ArrayList<?> data) {

        ArrayList<Information> dataInformation = new ArrayList<>();

        ArrayList<Information> informationRealmResult = (ArrayList<Information>) data;
        dataInformation.addAll(informationRealmResult.subList(0, informationRealmResult.size()));

        for (int i = 0; i < dataInformation.size(); i++) {
            for (int j = 0; j < dataInformation.get(i).getItems().size(); j++) {
                mItemsArrayList.add(dataInformation.get(i).getItems().get(j));
            }
        }

        if (mItemsArrayList.size() == 0) {
            mPresenter.getInformation(UtilsPreferences.getPage());
        } else {
            mRvAdapter.notifyDataSetChanged();
        }
    }
```

Obrázek 25 - Řídící metody presenteru
Zdroj: vlastní

Metoda showProgress() s argumentem show, který slouží pro detekci stavu načítání je zobrazena na obrázku 25. K zobrazení konkrétní chybové informace slouží onGeneralError(). K zobrazení dat ze serveru slouží onNotifyArray(). Při opětovném spuštění nesmí aplikace získávat data duplikovaně. O tento problém se stará metoda onAppStart(). Všechny tyto metody, tudíž i jednotlivé stavy, ovládá presenter.

Modelová vrstva komunikuje se serverem pomocí protokolu REST API. Tento komunikátor je vytvořený pomocí singletonu, který ovládá veškerou komunikaci uvnitř aplikace. [6]

```
01. public class RetrofitHelper {
02.
03.     private static RetrofitHelper ourInstance;
04.
05.     public static RetrofitHelper getInstance() {
06.         if (ourInstance == null) {
07.             ourInstance = new RetrofitHelper();
08.         }
09.         return ourInstance;
10.     }
11.
12.     private RetrofitHelper() {
13.
14.         OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
15.         OkHttpClient client = httpClient.build();
16.
17.         retrofit = new Retrofit.Builder()
18.                 .baseUrl(BASE_URL)
19.                 .addConverterFactory(GsonConverterFactory.create())
20.                 .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
21.                 .client(client)
22.                 .build();
23.         mUrlPath = retrofit.create(UrlPath.class);
24.     }
25.
26.     public UrlPath getUrlPath() {
27.         return mUrlPath;
28.     }
29.
30.     public Retrofit getRetrofit() {
31.         return retrofit;
32.     }
33.
34. }
```

Obrázek 26 - Restový komunikátor
Zdroj: vlastní

Singleton, zobrazený na obrázku 26, je návrhový vzor, který má vytvořenou jednu instanci pro celý běh programu s globální proměnou.

Dále se musí specifikovat body, které má aplikace volat, aby dostala požadovaný výsledek, což je možné vypozorovat z obrázku 27. [6]

```
01. public interface UrlPath {
02.
03.     @GET("questions?")
04.     Observable<Information> getData(@Query("filter") String filter,
05.                                     @Query("fromdate") String fromdate,
06.                                     @Query("todate") String todate,
07.                                     @Query("order") String order,
08.                                     @Query("sort") String sort,
09.                                     @Query("site") String site,
10.                                     @Query("pagesize") String pagesize,
11.                                     @Query("page") String page);
12. }
```

Obrázek 27 - Volání na server
Zdroj: vlastní

Značky Query slouží pro specifikaci dat, které aplikace požaduje po serveru. Základní metody pro volání REST API jsou get, post, put a delete. Vzhledem k tomu, že aplikace chce zdroje získat, tak používá první zmíněnou metodu. Klient komunikuje pouze s modelovou vrstvou, která v sobě zahrnuje veškerou správu získávání a ukládání dat.

```
01. public class ApiHandler {
02.
03.     public interface ResultCallback<T> {
04.         void onSuccess(T response);
05.         void onAppStart(ArrayList<?> data);
06.         void onError(String reason);
07.     }
08.
09.     public void getInformationDB(ResultCallback<Information> callback) {
10.         realm = Realm.getDefaultInstance();
11.         RealmResults<Information> moviesDB = realm.where(Information.class).findAll();
12.         ArrayList<Information> list = new ArrayList<>();
13.         list.addAll(moviesDB.subList(0, moviesDB.size()));
14.         callback.onAppStart(list);
15.     }
16.
17.     public void getInformation(int page, final ResultCallback<Information> callback) {
18.         final String FILTER = "withbody";
19.         final String FROM_DATE = "1459468800";
20.         final String TO_DATE = "1461974400";
21.         final String ORDER = "desc";
22.         final String SORT = "creation";
23.         final String SITE = "cooking";
24.         final String PAGE_SIZE = "5";
25.
26.         final UrlPath service = RetrofitHelper.getInstance().getUrlPath();
27.         realm = Realm.getDefaultInstance();
28.
29.         service.getData(FILTER, FROM_DATE, TO_DATE, ORDER, SORT, SITE, PAGE_SIZE, String.valueOf(page))
30.                 .subscribeOn(Schedulers.newThread())
31.                 .observeOn(AndroidSchedulers.mainThread())
32.                 .subscribe(new Observer<Information>() {
33.
34.                     @Override
35.                     public void onCompleted() {}
36.
37.                     @Override
38.                     public void onError(Throwable e) {
39.                         callback.onError(e.getMessage());
40.                     }
41.
42.                     @Override
43.                     public void onNext(Information information) {
44.                         realm.beginTransaction();
45.                         realm.copyToRealm(information);
46.                         realm.commitTransaction();
47.                         callback.onSuccess(information);
48.                     }
49.                 });
50.     }
51. }
```

Obrázek 28 - ApiHandler.class
Zdroj: vlastní

Pokud získání dat ze serveru proběhne úspěšně a vrátí se odpověď, nahrají se data do databáze. Poté jsou data využity k zobrazení uživateli. Tato vrstva implementuje i databázi, která je orientovaná objektově, viz obrázek 28 výše. [6]

```
01. public class Information extends RealmObject {
02.
03.     private RealmList<Items> items;
04.
05.     public RealmList<Items> getItems() {
06.         return items;
07.     }
08.
09.     public void setItems(RealmList<Items> items) {
10.         this.items = items;
11.     }
12. }
```

Obrázek 29 - POJO objekt
Zdroj: vlastní

Systém Android nabízí nativní relační SQLite databázi, která pro svoji implementaci nebo správu stávajícího schématu potřebuje jednotlivou implementaci veškerých tabulek a vazeb. Objektově orientovaná databáze pro svoji správu potřebuje pouze třídu s atributy. Pro programátora znamená každá ušetřená chvíle psaní kódu navíc rychleji napsanou aplikaci.

Pro detail tématu, na který se uživatel dostane kliknutím, je přechod na další vytvořený fragment. [6][7]

```
01. public class DetailFragment extends Fragment {
02.
03.     private String mTitle;
04.     private String mBody;
05.
06.     private TextView mTvTitle;
07.     private TextView mTvInformation;
08.
09.     private boolean mIsDualPane = true;
10.
11.     public static DetailFragment newInstance(String title, String body, boolean isDualPane) {
12.         DetailFragment fragment = new DetailFragment();
13.         Bundle args = new Bundle();
14.         args.putString(ConstantsString.ARG_TITLE_DETAIL_FRAGMENT, title);
15.         args.putString(ConstantsString.ARG_BODY_DETAIL_FRAGMENT, body);
16.         args.putBoolean(ConstantsString.ARG_DUAL_PANE_DETAIL_FRAGMENT, isDualPane);
17.         fragment.setArguments(args);
18.         return fragment;
19.     }
20.
21.     @Override
22.     public void onCreate(Bundle savedInstanceState) {
23.         super.onCreate(savedInstanceState);
24.         if (getArguments() != null) {
25.             mTitle = getArguments().getString(ConstantsString.ARG_TITLE_DETAIL_FRAGMENT);
26.             mBody = getArguments().getString(ConstantsString.ARG_BODY_DETAIL_FRAGMENT);
27.             mIsDualPane = getArguments().getBoolean(ConstantsString.ARG_DUAL_PANE_DETAIL_FRAGMENT);
28.         }
29.     }
30.
31.     @Override
32.     public View onCreateView(LayoutInflater inflater, ViewGroup container,
33.                              Bundle savedInstanceState) {
34.         // Inflate the layout for this fragment
35.         View view = inflater.inflate(R.layout.fragment_detail, container, false);
36.
37.         mTvTitle = (TextView) view.findViewById(R.id.textViewTitleFragmentDetail);
38.         mTvInformation = (TextView) view.findViewById(R.id.textViewInformationFragmentDetail);
39.
40.         if (mTitle != null && mBody != null) {
41.             mTvTitle.setText(Html.fromHtml(mTitle));
42.             mTvInformation.setText(Html.fromHtml(mBody));
43.         }
44.
45.         return view;
46.     }
47.
48.     public void setInformation(Items item) {
49.         mTvTitle.setText(Html.fromHtml(item.getTitle()));
50.         mTvInformation.setText(Html.fromHtml(item.getBody()));
51.     }
52. }
```

Obrázek 30 - DetailFragment.class
Zdroj: vlastní

Tento fragment není propojený s presenterem ani modelovou vrstvou, protože má informace, které potřebuje k zobrazení, uložené v balíčku (bundle), viz obrázek 30. Vzhledem k tomu, že text obsahuje i HTML znaky, je použita pro zobrazení metoda HTML.fromHtml (řádek 50). [5][8]

4.5 Distribuce na Google Play

Pro distribuci aplikací se u platformy Android používá Google Play, dříve nazýván (do roku 2012) Android Market. Zatímco tento obchod v předchozích letech nabízel pouze aplikace, později se rozrostl o nabídku elektronických knih, filmů a hudby. V Google Play můžou propagovat své aplikace pouze ti, kteří mají vytvořený svůj vlastní Google účet a také účet pro Google developery. [13]

4.5.1 Funkce developerského účtu

Mezi původní funkce patří herní služba Google Play a Google Wallet. Snahou herní služby je propojení her mezi sebou a motivace hráčů k lepším výsledkům díky vzájemným porovnávacím žebříčkům. Z pohledu majitele aplikace slouží tato služba ke sledování průměrných tržeb za určité období, retenci, monitorování postupu hráčů a k získání demografickým údajů. Google Wallet uchovává získané peníze z monetizace aplikací.

Developerský účet se stále vyvíjí a společnost Google se snaží přicházet s novými doplňky, které by usnadňovaly vývoj a lokalizaci aplikací nebo UI testování. Mezi novější funkce patří typy k optimalizaci, experimenty a Firebase Test Lab. Tipy k optimalizaci slouží převážně pro větší rozšířenost aplikace po Google Play. Další funkce se zabývá experimentováním s textem a grafikou, kde se zadá experimentální položka a procentuální počet lidí, kterým se tato položka zobrazí. Hlavním úkolem Firebase Test Lab je zjišťování problémů na fyzických zařízeních v širokém spektru a kontrola bezpečnostních chyb před samotným vydáním. [13]

4.5.2 Vložení a publikování aplikace v developerském účtu

Celý proces nahrávání nové aplikace začíná na úvodní obrazovce developerského účtu, kde musí uživatel nejprve zvolit výchozí jazyk aplikace a název, který může obsahovat maximálně 30 znaků. Poté se otevře nová nabídka, kde je možné zvolit konkrétní nastavení pro danou aplikaci.

Nejdůležitější se stává záložka s označením Záznam v obchodě. Zde se nastavují informace, které vidí uživatelé při stáhnutí aplikace. Z tohoto důvodu je velmi důležité dát si na těchto záznamech záležet. Mezi první záznamy na doplnění patří krátký a dlouhý popis. Zatímco krátký popis musí vystihnout jen to nejpodstatnější (maximálně 80 znaků), úplný popis by měl obsahovat informace o tom, co všechno aplikace poskytuje (až 4000 znaků).

Dalším důležitým faktorem jsou grafické prvky aplikace. Všechny aplikace musí mít k dispozici grafické podklady pro ikony, hlavní grafiku a propagační grafiku. Pokud je však aplikace vyvíjena pro mobilní telefon a pro tablety, nemusí se dodávat grafické podklady pro Android TV nebo Android Wear. Dále se vybírá typ aplikace a kategorie, která se zvolí na základě původní rozvahy z plánování aplikace. Pro zvýšení atraktivity

aplikace se používá Google hodnocení, které může snadno poskytnout poměrně relevantní hodnocení obsahu.

Cena a distribuce bývá často pokládaná otázka u mobilních aplikací. Pokud je aplikace zvolena jako placená, její cena je možné kdykoliv měnit či ji přepnout na bezplatnou verzi. Jakmile je však aplikace publikována jako bezplatná, nelze ji již změnit na placenou.

Posledním krokem je nahrání aplikace do Google Play. Před nasazením aplikace do produkční verze je doporučeno otestovat danou verzi na reálných uživatelích. K tomuto účelu jsou k dispozici kanály pro alfa a beta testování, kde si lze navolit konkrétní testery.

Jakmile je vše připravené, posledním krokem je publikování aplikace. V řádu několika hodin je aplikace již k dispozici na Google Play. [13]

4.5.3 ASO – optimalizace postavení aplikace ve vyhledávání

ASO (App Store Optimization) je proces zlepšování postavení aplikace ve vyhledávání, čímž se zvyšuje její šance ke stažení větším počtem uživatelů. Obdobou ASO procesu je proces SEO u webových stránek.

Existuje více faktorů, které může majitel ovlivnit pro zlepšení viditelnosti aplikace v Google Play. Další faktory však nejsou již majitelem přímo ovlivnitelné. Tyto faktory se nazývají On-page a Off-page optimalizace.

4.5.4 On-page optimalizace

Klíčová slova jsou nedílnou součástí textového popisu aplikace a jedním z hlavních faktorů při vyhodnocování algoritmu. Hlavní kritéria, která by se měla zohlednit při výběru klíčových slov jsou náročnost, objem vyhledávání a relevantnost.

Název aplikace je stejně důležitý jako titulek na webových stránkách. Bývá vhodné použít slova, která jasně popisují aplikaci a zároveň by měla být co nejkratší (často se používají i klíčová slova).

Dalším důležitým aspektem je popis aplikace, kde musí být rozepsány hlavní funkce aplikace a její výhody.

První věcí, kterou vidí potenciální uživatel je grafika poskytnutá Google Play. Podklady pro grafiku by měli být spjaté s aplikací.

Produktové video lze přidávat k aplikaci pouze v Google Play obchodě. Video by mělo být maximálně dvě minuty dlouhé a obsahovat pouze to nejzajímavější o aplikaci.

4.5.5 Off-page optimalizace

Pokud je vše nastaveno efektivně a aplikace funguje podle popisu, uživatelé hodnotí aplikaci kladně, čímž se zvyšuje počet stáhnutí. Tyto faktory nejsou přímo ovlivnitelné majitelem aplikace, ale jsou velice hodnotné pro algoritmus, který rozhoduje o viditelnosti aplikace.

5 Výsledek práce

Začátek této práce se věnuje popisu funkčních rozdílů mezi jednotlivými verzemi systému, což je pro každého programátora vyvíjejícího pro platformu Android klíčové. Dále je vysvětlen životní cyklus jednotlivých komponent aplikace a jejich začlenění ve vývoji, které je důležité pro správné fungování aplikace.

Postupné kroky této bakalářské práce vedly k vytvoření aplikace, která odpovídá stanoveným požadavkům. Aplikaci lze po nainstalování libovolně používat. Pro efektivnější sledování uživatelských potřeb lze aplikaci pravidelně pozorovat v prostředí Google Play.

6 Závěr

V teoretické části je popsána a graficky ilustrována historie jednotlivých verzí operačního systému Android. Jsou zde také zachyceny funkce, kterými se tyto verze vyznačovaly. Dále se práce věnuje stručnému porovnání konkurentů z pohledu funkcionality a prodejnosti v daných obdobích. Porovnání se zejména týká platformy iOS od firmy Apple. Poslední oddíl teoretické části se týká životního cyklu aplikace, kde je rozebrána a vysvětlena každá část tohoto cyklu.

V praktické části jsou nejprve definovány funkce aplikace a vytvořen její vzhled. Poté je rozebrána architektura, na jejíž základě je naprogramována vlastní aplikace. Na konci praktické části je popsána optimalizace pozice ve vyhledávání a distribuce v Google Play. Aplikace byla vytvořena ve vývojovém prostředí Android Studio, za využití programovacího objektového jazyka Java a značkovacího jazyka XML. V práci jsou použité bloky kódů pro lepší orientaci a snazší pochopení dané problematiky vloženy ve formě obrázků.

7 Seznam použitých zdrojů

7.1.1 Tištěné

1. ALLEN G. Android 4., Průvodce programováním mobilních aplikací. Computer Press, a.s.. 2011; 369 s. ISBN 978-80-2513-782-6
2. MEIER R. Professional Android Application Development, 2. vydání. Indianapolis: Wrox, 2010, 576 str., ISBN 0470565527.
3. MURPHY, Mark L.. Android 2, Průvodce programováním mobilních aplikací. Computer Press, a.s.. 2011; 369 s. ISBN: 978-80-251-3194-7
4. ROGERS, R., LOMBARDO, J. Android Application Development, 1st Edition. Cambridge: O`Reilly Media, Inc. 2009, 336s., ISBN 978-0-596-52147-9.
5. SCHILDT, Herbert. Java 7: výukový kurz. 1. vyd. Brno: Computer Press, 2012. ISBN 978-80-251-3748-2.

7.1.2 Online

6. Android [28.11.2016] http://developer.android.com/guide/index.html
7. Fórum pro programátory [20.12.2016] http://stackoverflow.com/
8. Historie OS Android [2.12.2016] https://www.android.com/history/
9. Oficiální dokumentace pro Android [7.1.2017] http://code.google.com/android/documentation.html
10. Oficiální informace o vývojovém prostředí [15.1.2017] https://developer.android.com/studio/intro/index.html
11. Popis návrhového vzoru MVP a MVC [10.12.2016] http://www.techyourchance.com/mvp-mvc-android-1/
12. Popis návrhového vzoru MVVM [10.12.2016] https://labs.ribot.co.uk/approaching-android-with-mvvm-8ceec02d5442 - .km78nmg9d
13. Stránka pro publikování vlastní aplikace [20.1.2017] https://play.google.com/apps/publish/

8 Přílohy

Aplikace se zdrojovými kódy je dostupná v přiloženém zip souboru.