Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh a implementace mobilní aplikace pro platformu Android využívající NASA API

Bc. Pavel Šebelík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Pavel Šebelík

Informatika

Název práce

Návrh a implementace mobilní aplikace pro plaformu Android využívající NASA API

Název anglicky

Design and implementation of a mobile application for Android platform utilizing NASA API

Cíle práce

Cílem práce je vytvoření mobilní aplikace pro platformu Android s pomocí programovacího jazyka Kotlin. Aplikace bude mít informativní charakter a bude využívat otevřené aplikační programové rozhraní, tzv. API, poskytované Národním úřadem pro letectví a kosmonautiku (NASA). Uživatel bude moci prostřednictvím aplikace zjišťovat informace o aktuálním dění v oboru astronomie a kosmonautiky.

Metodika

Teoretická část diplomové práce je založena na studiu odborných informačních zdrojů. Na základě zjištěných poznatků budou popsány teoretická východiska nutná pro zpracování aplikace.

V navazující praktické části práce budou definovány požadované funkce aplikace a následně bude proveden její návrh a implementace. Při návrhu budou využity standardní metody a nástroje softwarového inženýrství. Implementace bude provedena v programovacím jazyce Kotlin s využitím IDE Android Studio.

Aplikace bude otestována, budou shrnuty poznatky z jejího vývoje a testování a na jejich základě navrženy případné další možnosti jejího rozvoje.

Doporučený rozsah práce

60-80 stran

Klíčová slova

Android, Kotlin, Java, vývoj mobilní aplikace, NASA, open API

Doporučené zdroje informací

Android Developers. Android Developers [online]. Dostupné z: https://developer.android.com/
Download Android Studio and SDK Tools | Android Developers. Android Developers [online]. Dostupné z: https://developer.android.com/studio/
JEMEROV, Dmitry a Svetlana ISAKOVA. Kotlin in action. Shelter Island, NY: Manning Publications Co., 2017. ISBN 1617293296.
Kotlin Programming Language. Kotlin Programming Language [online]. Dostupné z: https://kotlinlang.org/
LACKO, Ľuboslav. Mistrovství – Android. Přeložil Martin HERODEK. Brno: Computer Press, 2017. Mistrovství. ISBN 9788025148754.
LACKO, Ľuboslav. Vývoj aplikací pro Android. Brno: Computer Press, 2015. ISBN 9788025143476.
NASA Open APIs. NASA Open APIs [online]. Dostupné z: https://api.nasa.gov/

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 3. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 3. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 23. 10. 2019

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh a implementace mobilní aplikace pro platformu Android využívající NASA API" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 27. 11. 2019 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi, Ph.D. za věcné podněty a připomínky ke zpracování teoretické i praktické části práce a za čas, který mi věnoval při konzultacích.

Návrh a implementace mobilní aplikace pro platformu Android využívající NASA API

Abstrakt

Diplomová práce je zaměřena na vývoj mobilní aplikace na platformě Android OS. Práce se skládá z teoretické a praktické části a zhodnocení výsledků. V teoretické části je čtenář seznámen s operačním systémem Android, jeho historií a architekturou. Dále je kladen důraz na představení programovacího jazyka Kotlin a jeho srovnání s jazykem Java. Stručně jsou také představeny použité technologie API a JSON. Praktickou část práce tvoří návrh aplikace definováním požadavků, funkcí a vzhledu. Na základě této specifikace je implementována aplikace v jazyce Kotlin za použití moderních postupů a nástrojů softwarového inženýrství. Na závěr následuje zhodnocení výsledků a shrnutí poznatků, získaných při implementaci řešení.

Klíčová slova: Android, Kotlin, Java, API, JSON, vývoj aplikace

Design and implementation of a mobile application for Android platform utilizing NASA API

Abstract

This thesis is focused on the development of mobile application on the Android OS platform. The thesis consists of theoretical and practical part and evaluation of results. In the theoretical part, the reader is acquainted with the Android operating system, its history and architecture. Furthermore, emphasis is placed on the introduction of the Kotlin programming language and its comparison with the Java language. The used technologies of API and JSON are also briefly introduced. The practical part of the thesis consists of application design by defining requirements, functions and appearance. Based on this specification an application in Kotlin language is implemented using modern procedures and software engineering tools. Finally, there is an evaluation of results and a summary of the knowledge gained during the implementation of the solution.

Keywords: Android, Kotlin, Java, API, JSON, app development

Obsah

Seznam obrázků

Seznam použitých zkratek

AAC – Advanced Audio Coding

API – Application Programming Interface (aplikační programové rozhraní)

ART – Android Runtime

GPS – Global Positioning System (globální polohový systém)

HEIV – High Efficiency Image File Format

HTML – Hypertext Markup Language

IMAP – Internet Message Access Protocol

IDE – Integrated Development Environment (vývojové prostředí)

IPC – Inter Process Communication (meziprocesová komunikace)

ISS – International Space Station (Mezinárodní vesmírná stanice)

JDK – Java Development Kit

JIT – Just In Time

JSON – JavaScript Object Notation

JVM – Java Virtual Machine (virtuální stroj Java)

NASA – National Aeronautics and Space Administration (Národní úřad pro letectví a kosmonautiku)

NFC – Near Field Communication

OS – operační systém

POP3 – Post Office Protocol

QR – Quick Response

SDK – Software Development Kit (sada vývojových nástrojů)

SMTP – Simple Mail Transfer Protocol

UI – User Interface (uživatelské rozhraní)

USB – Universal Serial Bus (universální sériová sběrnice)

VoIP – Voice over Internet Protocol

WVGA – Wide Video Graphics Array

XML – Extensible Markup Language

1 Úvod

V dnešní moderní době patří používání mobilních zařízení ke každodennímu životu většiny z nás. Před odchodem do práce se podíváme na počasí v aplikaci a vyhledáme si dopravní spoj, cestou v hromadné dopravě si přečteme elektronické noviny v zpravodajské aplikaci, nebo poslechneme elektronickou knihu. V práci se při obědě podíváme na nové příspěvky na naší oblíbené sociální síti, nakoupíme si v nákupní aplikaci. Při cvičení v posilovně si měříme chytrým náramkem tep a počet spálených kalorií, večer sledujeme zábavná videa nebo komunikujeme s přáteli. Chytrá mobilní zařízení a aplikace na nich nainstalované nás provází na každém kroku, mnoho lidí si bez svého smartphonu nebo tabletu nedovede představit jediný den. Poptávka po aplikacích je tedy značná a stále roste i díky vyvíjejícím se technologiím a možnostem.

Má osobní zkušenost s mobilními aplikacemi vyplývá hlavně z pracovních zkušeností, jelikož se už několik let věnuji testování aplikací pro mobilní zařízení. Díky tomu tyto aplikace již nevnímám jen jako koncový uživatel, ale přemýšlím o nich i v kontextu uživatelské přívětivosti, designu, funkcionality či implementace. Z tohoto důvodu mě začala přitahovat i problematika samostatného vývoje aplikací, kde bych mohl uplatnit své doposud získané zkušenosti ve svůj prospěch a skloubit je se základními znalostmi o technologiích vývoje a programovacími jazyky.

Dnešní trh mobilních zařízení je rozdělen do 3 hlavních platforem. Těmi jsou OS Android společnosti Google, iOS společnosti Apple a Windows Mobile společnosti Microsoft. Z těchto platforem je dnes zdaleka nejrozšířenější Android, následován iOS a Windows. Hlavní příčinou takového rozdělení je především zásadně nižší pořizovací cena Android zařízení spolu s velkou škálou dostupných aplikací zdarma. Vývojové prostředí Android Studio je bezplatné a dostupné pro instalaci na počítače s operačními systémy Windows, MacOS i Linux. Vyvíjet na této platformě se tedy může naučit prakticky kdokoliv. Jediným výdajem je vstupní vývojářský poplatek 25 dolarů.

Díky těmto faktům a stále chybějícím odborníkům v IT sféře v ČR jsem se rozhodl, že bych si chtěl rozšířit obzory v tomto oboru. Jako téma diplomové práce jsem si tedy zvolil vytvoření aplikace, která bude pro svou funkci využívat moderních technologií a nástrojů vývoje. Charakter této aplikace bude informativní se zaměřením na události v blízkém vesmíru, o které se jako amatér ve svém volném času rád zajímám.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této diplomové práce je vytvořit aplikaci pro mobilní zařízení s operačním systémem Android, a to pomocí nového programovacího jazyka Kotlin.

Mezi dílčí cíle teoretické rešerše patří charakteristika platformy Android, představení jazyka Kotlin a porovnání se stávajícím jazykem pro vývoj na OS Android, kterým je Java. Mezi dílčí cíle praktické části práce patří vytvoření specifikace, návrh vzhledu aplikace a samotná implementace použitím jazyka Kotlin. Výsledná aplikace bude publikována v aplikačním obchodě Google Play.

2.2 Metodika

Teoretická rešeršní část práce se zabývá představením operačního systému pro mobilní zařízení Android. Čtenář bude stručně seznámen s historií systému a jejími verzemi, architekturou operačního systému a anatomií aplikace. V dalším kroku bude představen jazyk Kotlin s jeho výhodami a porovnáním se stávajícím jazykem Java. Konec rešeršní části bude věnován krátkému představení funkce aplikačního programového rozhraní a datového formátu JavaScript Object Notation.

Praktickou část tvoří technická specifikace a vlastní řešení implementace. Specifikace zahrnuje definované user story jednotlivých obrazovek, návrh logického uspořádání prvků metodou zakreslení wireframe a popis chování aplikace. Implementace bude provedena s využitím vývojového prostředí Android Studio. V práci budou znázorněny zásadní části zdrojového kódu spolu s vysvětlením způsobu implementace.

Na závěr budou shrnuty poznatky z vývoje, připomínky vzniklé při testování a naznačeny možnosti dalšího rozšíření aplikace v budoucnosti.

3 Teoretická východiska

3.1 Historie OS Android

Společnost Android Inc. byla založena v roce 2003 ve městě Palo Alto v Kalifornii. Jejími zakladateli byli Andy Rubin, Rich Miner, Nick Sears a Chris White. Původním plánem bylo vytvoření operačního systému pro digitální videokamery a fotoaparáty, díky kterému bude možné např. bezdrátově připojit digitální fotoaparát k počítači a jednoduše ukládat pořízené fotografie na vzdálené úložiště. Avšak vzhledem ke klesajícímu trhu s digitálními fotoaparáty, se nakonec tvůrci systému Android rozhodli přesunout své pole působení a zaměřili se na použití svého OS v mobilních telefonech. (1)

V roce 2005 byla firma Android Inc. odkoupena společností Google za zhruba 50 milionů dolarů, přičemž původní zakladatelé ve firmě zůstali a dále se podíleli na vývoji OS. Bylo rozhodnuto o použití Linux platformy jako základ pro Android OS. Toto rozhodnutí znamenalo možnost v budoucnu poskytovat systém výrobcům mobilních telefonů třetích stran zdarma. (1)(2, s. 66)

Open Handset Alliance je název sdružení technologických a telekomunikačních firem a výrobců mobilních telefonů, které bylo založeno v roce 2007 a dodnes se věnuje vývoji OS Android. V této době také vyšla první verze vývojářského kitu (SDK) pro Android. (2, s. 66)

První oficiální uvolnění operačního systému Android, verze 1.0, na mobilním telefonu proběhlo v září 2008 a to v modelu HTC Dream, nazývaném rovněž T-Mobile G1. Po roce, v září 2009 obdržel telefon aktualizaci na Android 1.1. Tato verze nesla interní označení „Petit Four“ (lze přeložit jako drobné cukroví, zákusek), které výjimečně nenásledovalo konvenci pojmenování Android verzí dle abecedy, jak známe dodnes (před uvolněním první oficiální verze Android 1.0, existovaly verze alpha a beta, které byly pojmenovány „Astro Boy“ a „Blender“). (2, s. 66)(3)(4)

Android 1.0

Jak už bylo v úvodu kapitoly uvedeno, Android 1.0 byla první, oficiálně uvolněná verze OS Android, která byla distribuována pouze na jediném zařízení, HTC Dream. Tato verze již obsahovala množství dobře známých funkcí a aplikací, např. Android Market,

Gmail, Google Calendar, Google Maps nebo YouTube. Mezi funkcionalitami byla například podpora protokolů elektronické pošty (POP3, IMAP, SMTP), online synchronizace zařízení přes Google Sync, ale také odemykání zámku displeje gestem, přístup k notifikacím, více domovských obrazovek či správce úloh, běžících na pozadí aj. (3)

Android 1.1 Petit Four

Jednalo se o spíše o drobnější aktualizaci pro verzi Android 1.0, která nabízela např. možnost připojit přílohu ke zprávám, nebo zobrazení uživatelských recenzí podniků, vyhledávaných v aplikaci Google Maps. (3)

Android 1.5 Cupcake

Verze 1.5 byla uvolněna v dubnu 2009, již na širší spektrum zařízení. Android Cupcake představil podporu virtuálních klávesnic třetích stran, funkci schránky pro kopírování a vkládání obsahu a v neposlední řadě automatické otáčení grafického UI. Tato verze odstartovala trend pojmenovávání jednotlivých verzí Android OS podle oblíbených sladkostí, jejichž první písmena představovala za sebou jdoucí písmena v abecedě a v době jejího vydání, bylo na Android Marketu dostupných již zhruba 3 000 aplikací. (2, s. 66)(3)

Android 1.6 Donut

Jedním z hlavních přínosů Android verze, pojmenované podle koblihy, byla podpora WVGA rozlišení displeje, díky čemu mohl být OS nainstalován na zařízeních s různými poměry stran displeje. Vylepšený vyhledávač nyní poskytoval i vyhledávání v rámci zařízení, uživatel tak mohl vyhledávat jedním slovem jak na internetu, tak ale i např. v aplikacích, kontaktech, nebo hudební knihovně. V aplikačním marketu přibyly nově obrázky z aplikací a uživatelské recenze. Mimo jiné byla přidána podpora komunikačních standardů CDMA, 802.1x, VPN. Android 1.6 byl uvolněn v září roku 2009. (2, s. 66)(3)

Android 2.0 Eclair

V říjnu 2009 byl vydán Android 2.0 s novým designem UI, optimalizací výkonu a podporou standardů HTML5 a Bluetooth 2.1. Spolu s aktualizací 2.1 Eclair, také Google

představil svůj první telefon, Google Nexus One (zařízení vyrobila firma HTC), který byl uvolněn do prodeje v lednu 2010. Google Nexus telefony představovaly v podstatě referenční zařízení pro vývoj aplikací na Android platformu, protože obsahovaly „čistý“ operační systém bez dalších nadstaveb, které do svých zařízení instalovali další výrobci mobilních telefonů. Dalším znakem těchto Google zařízení bylo obdržení systémových aktualizací jako první ve srovnání s telefony dalších výrobců. Později v roce 2016 byla řada Nexus ukončena a dodnes následuje řada Pixel. (2, s. 66)(3)

Android 2.2 Froyo

Android Froyo doručil další optimalizaci výkonu, díky Dalvik JIT kompilátoru, který značně zvýšil rychlost práce systému. Další zlepšení se týkalo JavaScriptu a spočívalo v implementaci V8 JavaScript enginu v rámci internetového prohlížeče. Navíc byla přidána podpora technologie Flash 10. V době vydání verze 2.2, obsahoval Android Market již přes 100 000 aplikací, dostupných ke stáhnutí. Android Froyo byl uvolněn v květnu 2010. (2, s. 67)(3)

Android 2.3 Gingerbread

Novinkou ve verzi Gingerbread, uvedené během prosince 2010, byla nově podpora NFC či například podpora internetových hovorů prostřednictvím technologie VOIP. Vylepšení se dočkala také výdrž baterie, díky lepší správě běžících aplikací. Důraz byl kladen také na multimédia, Android začal podporovat přehrávání videa ve formátu WebM/VP8 a AAC audio. (3)

Android 3.0 Honeycomb

Verze 3.0 Honeycomb byla uvedena na trh v únoru 2011 a jednalo se o verzi Android OS, určenou výhradně pro tablety. Grafické uživatelské rozhraní bylo optimalizováno pro práci na větším displeji. Vedle vylepšeného UI taktéž přibyla hardwarová akcelerace spolu s podporou vícejádrových procesorů. Jedna z pozdějších aktualizací rovněž doplnila systém o možnost použít USB periferie jako klávesnice a myši. (3)

Android 4.0 Ice Cream Sandwich

V říjnu 2011 byla uvolněna verze 4.0, která především sjednotila předchozí verzi Gingerbread pro telefony a Honeycomb pro tablety, do jednoho OS, určeného pro oba typy zařízení. Od této verze tedy vypadal OS Android na telefonech i tabletech stejně. Ice Cream Sandwich představil funkci odemykání zamknutého telefonu pomocí rozpoznání obličeje uživatele pomocí přední kamery. (3)

Android 4.1 Jelly Bean

Android Jelly Bean dorazil na mobilní zařízení v červenci 2012 a doručil uživatelům další výkonnostní optimalizace pro zaručení ještě plynulejší práce v systému. Poprvé se zde objevily notifikace s akcemi, tzn. uživatel měl možnost na notifikaci reagovat přímo v rámci zobrazené notifikace bez nutnosti nejdříve otevřít požadovanou aplikaci. (3)

Android 4.4 KitKat

Od verze 4.4 mohli uživatelé využívat hlasového příkazu „OK Google“ pro spuštění hlasového asistenta. Mimo jiné byl, zatím jen experimentálně, do systému přidán virtuální stroj Android Runtime jako budoucí náhrada za aktuálně používaný virtuální stroj Dalvik. Dalšími novinkami bylo například chytré vyhledávání neznámých volajících čísel, či bezdrátový tisk. Android KitKat byl oficiálně uvolněn v říjnu 2013. (3)

Android 5.0 Lollipop

Největší změnu a značné vylepšení představuje uvedení nového virtuálního stroje ART do „ostrého“ provozu, nahrazujícího dosavadní stroj Dalvik. Toto představovalo značný vliv na výkon aplikací. Důvodem je způsob, jakým ART pracuje se zdrojovým kódem aplikace v rámci systému. Narozdíl od Dalvik virtuálního stroje, který kompiloval zdrojový kód aplikace při každém spuštění, Android Runtime engine zkompiluje část kódu již při instalaci aplikace a díky tomu je každé spuštění aplikace rychlejší, což má ve výsledku i výhodu nižší spotřeby, a tedy prodloužení celkové výdrže baterie. Za zmínku také stojí nový design grafického rozhraní, tzv. Material Design, který spočívá v umisťování jednotlivých prvků na obrazovce tak, že vzbuzuje zdání třetího rozměru (dalo

by se přirovnat k pokládání barevných papírů různých tvarů postupně na sebe, přičemž se vrstvy různě překrývají). Verze z listopadu 2014 přinesla podporu 64bitových procesorů. (2, s. 68)(3)

Android 6.0 Marshmallow

Oproti předchozí verzi Lollipop je Marshmallow spíše orientovaný na designové úpravy grafického UI s cílem zvýšit dynamičnost a intuitivnost ovládání pro uživatele. Android 6.0 obdržel technologii s názvem Doze. Úkolem této technologie je inteligentní správa běžících procesů na pozadí s ohledem na aktuální využívanost telefonu. Chování této správy ovlivňuje akcelerometr v zařízení, který monitoruje, jak moc je s telefonem manipulováno, případně jak dlouho je již v nehybném stavu. Díky těmto informacím o telefonu Doze např. může omezit počet oznámení či sníží interval synchronizace aj. Marshmallow, který byl uvolněn v říjnu 2015, obohacuje OS Android také o důležité hardware funkce. Jednou z nich je nativní podpora snímání otisku prstu pro autorizační akce uživatele. Další se týká podpory USB typu C. (2, s. 70)(3)

Android 7.0 Nougat

Nový JIT kompilátor přináší další výkonnostní vylepšení, snižuje množství úložiště, potřebného pro aplikace. Zvýšení výkonu se dočkaly také graficky náročné aplikace, obvykle 3D hry, díky použití aplikačního rozhraní pro trojrozměrnou grafiku, Vulkan API. Doze nyní částečně šetří baterii i při manipulaci se zařízením. S cílem zkvalitnění práce v systému byl představen Split screen mód, tedy rozdělení plochy displeje na dvě části, přičemž každá část bude vyplněna jednou aplikací. Uživatelé nyní mohou instalovat aktualizace OS na pozadí, díky čemuž nadále není potřeba čekat na dokončení zařízení před dalším použitím. Díky funkci spořiče dat, lze nově omezit spotřebu využitých dat, aplikacemi, běžícími na pozadí. Nougat k uživatelům dorazil v srpnu 2016. (3)(5)

Android 8.0 Oreo

Signifikantní změnou základů OS Android je Project Treble. Tato inovace spočívá v oddělení kódu hardwarové vrstvy zařízení, kterou tvoří např. procesor a chipset, od kódu samotného operačního systému. Díky tomu lze samotný operační systém aktualizovat

nezávisle na kódu spodní vrstvy hardwaru, která může zůstat nezměněna. V srpnu 2017, kdy aktualizace s názvem Oreo byla vydána, bylo uvedeno nativní C/C++ API pro vysoko kvalitní audio. Mezi méně zásadní změny a novinky patří např. možnost stahování fontů nebo nutnost potvrzení důvěryhodnosti zdroje před instalací aplikace mimo obchod Play. (6)(7)

Android 9 Pie

Android OS, pojmenovaný podle koláče, byl vypuštěn do světa v srpnu 2018. Mezi nové funkce patří např. adaptivní baterie, jejíž princip spočívá v učení uživatelových návyků z hlediska používání zařízení a díky tomu dokáže předpovídat, které aplikace pravděpodobně uživatel spustí v následujících hodinách. Na základě této informace přizpůsobí správu procesů a služeb, díky čemu se sníží spotřeba energie z baterie. I z hlediska bezpečnosti přináší Pie nové funkce, např. šifrování zálohy systému, konzistentní autorizace otiskem prstu napříč aplikacemi nebo omezení přístupu aplikací k mikrofonu, fotoaparátu apod., pokud je aplikace pozastavena nebo běží na pozadí. (1)(8)

Android 10

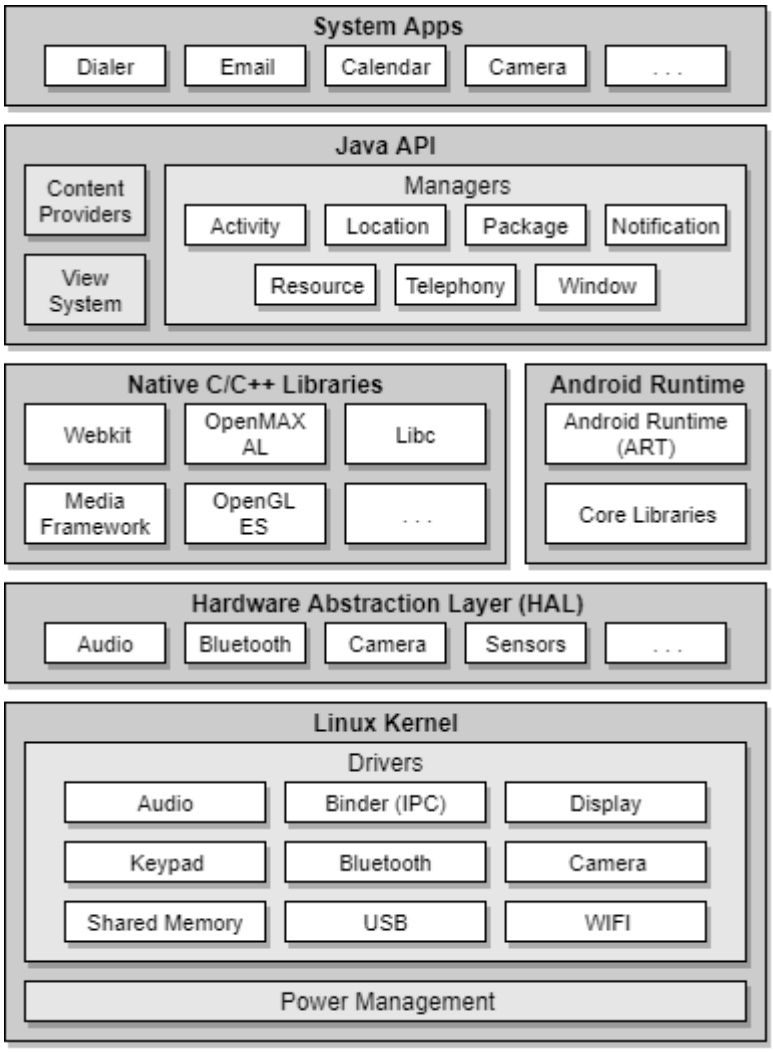
Zatím poslední představená verze OS Android je zároveň první, která již není pojmenována dle žádného dezertu, nýbrž nese pouze číslovku 10. Oficiální vydání systému bylo v září 2019. V této poslední aktualizaci Android nabízí funkci generování titulků k mluvenému slovu např. ve videích či v namluvených zprávách, a to v reálném čase. Zajímavou novinkou jsou chytré odpovědi, které spočívají v zobrazení automaticky navržených odpovědí a dostupných akcí na základě obsahu zprávy, přímo v notifikaci. Uživatel tedy nemusí pro odpověď ani otevřít aplikaci zpráv. Velmi očekávanou funkcionalitou, která je uživatelům již velmi známá z jednotlivých aplikací, je tmavý režim v rámci celého operačního systému. Fotoaparát podporuje novější formát fotografií, HEIF. Android 10 uživatelům nabízí větší moc nad nastavením sdílení lokace a obecně přehled a ovládání povolení pro aplikace. (1)(9)

3.2 Architektura OS Android

Struktura Androidu jako operačního systému lze rozdělit do 6 částí, respektive architektonických vrstev, na Linux Kernel, hardwarová abstraktní vrstva, nativní knihovny,

Android Runtime, Java API Framework a samotné aplikace. Na následujícím obrázku je vyobrazené schéma jednotlivých vrstev a jejich uspořádání a prvky. (2, s. 74)

Obrázek 1 - Schéma architektury OS Android



Zdroj: autor podle (2, s. 74)

3.2.1 Linux Kernel

Absolutním základem a nejnižší vrstvou operačního systému Android je upravené jádro operačního systému Linux, přizpůsobené pro použití na mobilních zařízeních. Účel Linux jádra spočívá v prostředkování interakce mezi hardwarem telefonu a vyššími softwarovými vrstvami. Jeho úlohou v systému je především správa operační paměti, procesů a napájení. Správa napájení zajišťuje, aby potřebné moduly disponovaly stabilním přísunem energie, a naopak některé méně důležité procesy mohou být po dobu nečinnosti

pozastaveny. Dále zabezpečení systému, vstupní a výstupní operace, grafika a dle dostupných prvků v zařízení např. ovladače pro Bluetooth, 3G, LTE, Wi-Fi, digitální fotoaparát, GPS aj. Binder je v této vrstvě meziprocesový ovladač komunikace a volání metod. V rámci procesů jsou spouštěny aplikace a služby odděleně. Obvykle však spolu potřebují některé z nich komunikovat. Sdílení dat mezi jednotlivými procesy umožňuje právě Binder. (2, s. 75)

3.2.2 Hardware Abstraction Layer

Jedná se o vrstvu abstrahující hardware zařízení, tedy jakési rozhraní mezi hardwarem telefonu a vyšších vrstev. Vrstva byla vytvořena s cílem ulehčit práci vývojářům aplikací, díky ní totiž nemusí přesně znát technické vybavení a specifikace každého jednotlivého zařízení, pro které aplikace vytváří (což by dnes vzhledem k velkému množství typů telefonů bylo prakticky nemožné). (2, s. 75)

3.2.3 Nativní knihovny

Nativní knihovny, napsané v jazyce C nebo C++, zprostředkovávají aplikacím kontrolu nad komponentami. Ve své podstatě tyto knihovny doplňují funkcionalitu Linux jádra o prvky, které jsou přizpůsobeny pro mobilní zařízení. Mezi zmiňované komponenty patří např. grafické knihovny OpenMAX a OpenGL, dále Webkit, používaný pro renderování a zobrazování webových stránek v zařízení. (2, s. 76)

3.2.4 Android Runtime – ART

Výstupem vývojového prostředí Android Studio, jsou aplikace ve formátu bytekódu DEX. Tento bytekód vzniká kompilací souborů typu CLASS a JAR, je ovšem kompaktnější. Android Runtime je běhové prostředí, spravované aplikacemi a některými systémovými službami. Spolu s předchůdcem Dalvik, bylo vytvořeno speciálně pro Android a oboje jsou tedy kompatibilní s formáty typu DEX. Android Runtime pomocí tzv. Ahead-Of-Time procesu překládá DEX bytekód aplikace do strojových instrukcí procesoru v telefonu nebo tabletu. Aplikace, vyvinuté pro práci v Dalvik prostředí by měly fungovat na ART, některé Dalvik techniky však není možné použít na Android Runtime prostředí. (2, s. 76)(10)

3.2.5 Java API Framework

Framework, tedy aplikační rámec je soubor znovupoužitelného Java kódu, který slouží jako hlavní podpora pro vývojáře. Skládá se z modulů, jako např. Package Manager, Window Manager, View System, Activity Manager, Notification Manager, Location Manager a další. Prvním jmenovaným modulem je Package Manager, tedy správce balíčků. Úkolem tohoto modulu je držení informací a správa všech nainstalovaných aplikací v systému. Jednotlivé aplikace mezi sebou mohou také navzájem komunikovat v podobě sdílení údajů, či za účelem využití některé služby. Spuštěné aplikace na zařízení se systémem Android obvykle využívají více oken současně. Tato okna, která jsou zobrazována na obrazovce telefonu, obsluhuje modul Window Manager. Grafické prvky uživatelského rozhraní, jako jsou např. tlačítka, zaškrtávací políčka, posuvníky atd., spravuje View System modul. V neposlední řadě, Activity Manager modul pracuje s životním cyklem aplikace. (2, s. 76)

3.2.6 Aplikace

Nejvyšší vrstvu Android architektury představují samotné aplikace, které jsou uživatelem přímo používány. Aplikace jsou reprezentovány na obrazovce dvojicí ikony a názvu a jedná se např. o aplikaci kalendáře, fotoaparátu, hudebního přehrávače a veškeré uživatelem nainstalované programy. (2, s. 77)

3.3 Architektura Android aplikace

3.3.1 Activity (aktivity)

Aktivita je základní komponentou každé Android aplikace. Jedná se o třídu, kterou uživatel uvidí jako první po spuštění aplikace. Aktivit může a obvykle je v aplikacích více a navzájem si mezi sebou odevzdávají informace. Aktivita může disponovat vlastní specifikací grafického rozhraní, které je definováno XML souborem. Prostřednictvím tohoto grafického rozhraní je uživateli umožněno pracovat s aplikací, například zobrazovat data, zadávat data, měnit nastavení aplikace apod. Aktivity se v aplikaci řadí hierarchicky, tzn. že hlavní aktivita je zobrazena ihned po spuštění aplikace, následně je možný přesun uživatele na další dostupné aktivity. (2, s. 83)

3.3.2 Services (služby)

Služba je aplikační komponenta, která narozdíl od aktivity nemá své vlastní grafické rozhraní. Jejím úkolem je spouštění procesů, které běží na pozadí, nebo pracují v dlouhodobém časovém horizontu. Služby umožňují provádět procesy paralelně s hlavním vláknem, tzn. uživatel pracuje s aktivitou na popředí a současně služba provádí svou práci na pozadí. Příkladem služby běžící na pozadí může být např. hudební přehrávač, navigace, běžící stopky atd. (2, s. 84)

3.3.3 Broadcast receivers

Objekty Broadcast receivers pracují na pozadí tak, že vysílají a poslouchají události na zařízení a na některé z nich reagují. Těmito událostmi jsou objekty Intent (záměr). Broadcast receivers pracují na principu publikování/přihlášení k odběru – služba v aplikaci vytvoří záměr, ten následně vysílá (publikuje) a přijímající služby, které jsou přihlášené k odběru stejného typu záměru, tuto událost zachytí a dále s ní pracuje. (2, s. 84)

3.3.4 Content providers (poskytovatelé obsahu)

Poskytovatel obsahu je služba komponenta aplikace, umožňující sdílení dat dalším aplikacím, nebo jejich uložení. Tímto způsobem spolu aplikace komunikují. (2, s. 84)

3.4 Programovací jazyk Kotlin

3.4.1 Historie

Kotlin je objektový programovací jazyk s typovou kontrolou, který vznikl v roce 2011 ve firmě JetBrains. Tato firma se zabývá tvorbou vývojářského software, mezi něž patří např. známé vývojové prostředí IntelliJ IDEA, na němž je založeno Android Studio IDE. Jazyk byl vytvořen pro práci v prostředí JVM, tedy stejně jako jazyk Java. Díky tomu se Kotlin brzy stal oblíbeným jazykem, protože navzdory upravené syntaxi oproti Javě, poskytuje některé výhody, které nabízí snadnější a příjemnější psaní kódu. Zkompilovaný kód Kotlinu je plně kompatibilní s kompilovaným kódem Javy a z toho důvodu lze bezproblémově využívat například Java knihovny v rámci Kotlin projektu. Zásadním milníkem vývoje tohoto jazyka byl rok 2017, ve kterém společnost Google uznala jazyk

Kotlin jako oficiální programovací nástroj pro platformu Android. Podobnou transformací si prošla také platforma iOS společnosti Apple, která nahradila svůj dosavadní hlavní jazyk Objective-C modernějším Swift. (11)

3.4.2 Výhody a nevýhody jazyka

Jazyk Kotlin plně podporuje JDK verze 6, díky čemu je zajištěna zpětná funkčnost se staršími zařízeními s Android OS. Navíc je jazyk plně podporovaný vývojovým prostředím Android Studio včetně sestavovacího, tzv. build systému Gradle. Podobná skladba bytekódu jako u jazyka Java přináší vyrovnaný výkon aplikací, při použití nově podporovaných lambda výrazů lze však dosáhnout i vyšších rychlostí. Obrovskou výhodou Kotlinu je tzv. interoperabilita s Java kódem, tzn. možnost použít v Kotlin projektu různé knihovny napsané v Javě. Nebýt této interoperability, nový jazyk by pravděpodobně nebyl tak oblíbený a využívaný, protože by bylo potřeba při jeho použití napsat knihovny nové. Další z výhod je podobnost syntaxe se syntaxí Javy, přechod na nový jazyk by zkušenému Java vývojáři tedy nemělo konat velké problémy. Tomu napomáhá i funkce překladu Java kódu do Kotlinu, integrovaná přímo v prostředí Android Studio. Díky multiplatformnímu Kotlin frameworku existuje možnost vytvoření aplikace např. pro platformu Android i iOS, sdílením stejného kódu. Framework pro multiplatformní vývoj lze využít i pro další platformy jako např. JavaScript, Linux, Windows, Mac aj. Mnoho vývojářů rozhodně ocení vestavěnou tzv. „null safety“ funkci. Jde o možnost povolit nebo naopak zakázat možnost vkládání null hodnoty do proměnné. Povolení se dosáhne přidáním znaku otazníku za datový typ dané proměnné. V opačném případě, tzn. pokud proměnnou nepůjde naplnit hodnotou null, kód nebude kompilovatelný a vývojové prostředí uživatele upozorní chybou. Druhým nástrojem zmíněné „null safety“ jsou tzv. „safe calls“ které spočívají v kompilování konkrétního příkazu pod podmínkou, že volaná proměnná nebude mít hodnotu null. Prostřednictvím „null safety“ funkcí jazyka Kotlin lze předejít chybám typu „NullPointerException“, kterými je jazyk Java nechvalně známý. Oproti Javě lze o Kotlinu tvrdit, že samotný kód je snadněji čitelný pro člověka, neobsahuje tolik tzv. „boilerplate kódu“, tj. části kódu, které se v souboru často opakují a samostatně nepřináší žádnou funkcionalitu, ale jsou nezbytné pro zprovoznění požadované funkcionality. (12)(13)(14)(15)

Mezi některé nevýhody jazyka Kotlin lze zařadit zatím menší komunita oproti Javě, která se může projevit při hledání řešení konkrétního problému. Alespoň částečně lze tuto nevýhodu eliminovat konverzí Java kódu na Kotlin kód pomocí vestavěného nástroje v některých vývojových prostředích. Problematický může někdy být přímý přechod na Kotlin z Javy, např. na větších projektech. S tím je úzce spjatý i fakt, že se jedná o nový jazyk a může být nouze o zkušené programátory. (13)

3.4.3 Porovnání s jazykem Java

Deklarace proměnných

V případě Javy se vždy striktně musí zadat datový typ deklarované proměnné. Kotlin datový typ automaticky zjistí na základě hodnoty přiřazené do proměnné. Případně máme možnost datový typ definovat připsáním datového typu za název proměnné, např. „`var myInt: Int = 5`“. Deklarovat proměnnou lze navíc dvěma způsoby, a to klíčovými slovy „`var`“ (variable) nebo „`val`“ (value). Hodnotu proměnné typu variable lze upravovat, hodnotu proměnné value nelze po inicializaci dále měnit.

Kód 1

```java
// Java
String myString = "Hello world";
int myInt = 5;
long myLong = 5L;
double myDouble = 5.5;
float myFloat = 5.5f;
boolean myBoolean = true;
char myChar = 'a';
```

Zdroj: autor

Kód 2

```
// Kotlin
var myString = "Hello world"
var myInt = 5
var myLong = 5L
var myDouble = 5.5
var myFloat = 5.5f
var myBoolean = true
var myChar = 'a'
```

Zdroj: autor

Použití datové třídy

Díky novým „data class“ v Kotlinu odpadá povinnost definovat metody typu „get“ a „set“ jako tomu bylo v Javě. Proměnné třídy se pouze deklarují jako argumenty třídy a poté k nim lze přistupovat z jiné třídy pomocí tečkové notace přímo.

Kód 3

```java
// Java
public class User {
    private String name;
    private String surname;
    private int year;

    public User(String name, String surname , int year) {
        this.name = name;
        this.surname = surname;
        this.year = year;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getSurname() {
        return surname;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public int getYear() {
        return year;
    }
}
```

Zdroj: autor

Kód 4

```kotlin
// Kotlin
data class User(val name: String,
                val surname: String,
                val year: Int)
```

Zdroj: autor

Přiřazení pohledů při spuštění aktivity

Při práci s pohledy (views) je v případě Javy nezbytné pro každý prvek deklarovat instanci třídy konkrétního pohledu a k této instanci následně samotný grafický/ovládací prvek přiřadit: „`Button button = (Button) findViewById(R.id.button);`“ Kotlin dokáže rozpoznat prvky ihned a umožňuje s nimi pracovat ihned.

Kód 5

```java
// Java
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button button = (Button) findViewById(R.id.button);
        final TextView text = (TextView) findViewById(R.id.text);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                text.setText("You've clicked a button");
            }
        });
    }
}
```

Zdroj: (16)

Kód 6

```kotlin
// Kotlin
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?,
                          persistentState: PersistableBundle?) {
        super.onCreate(savedInstanceState, persistentState)
        setContentView(R.layout.activity_main)

        button.setOnClickListener { text.text = "You've clicked a button" }
    }
}
```

Zdroj: (16)

Kolekce

V následujícím příkladu je naznačena práce s kolekcemi, konkrétně vytvoření kolekce studentů. Každý student má definované jméno a známku a poté jsou vygenerovány dva listy, první o velikosti 3 studentů se známkou 0 a druhý o velikosti 2 studentů se

známkou 1. Na první pohled je zřejmý rozdíl v deklaraci a naplnění kolekce, stejně tak v samotném filtrování požadovaných studentů. (16)

Kód 7

```
// Java
ArrayList<Student> students = new ArrayList<Student>() {{
    add(new Student("John", 0));
    add(new Student("Julia", 2));
    add(new Student("Matt", 1));
    add(new Student("Katie", 0));
    add(new Student("Dan", 0));
}};

ArrayList<Student> secondList = new ArrayList<>();

for (Student student: students) {
    boolean isFirstFilled = firstList.size() >= 3;
    boolean isSecondFilled = secondList.size() >= 2;

    if (isFirstFilled && isSecondFilled) break;
    int mark = student.getMark();
    if (mark == 0 && !isFirstFilled) {
        firstList.add(student);
    } else if (mark == 1 && !isSecondFilled) {
    secondList.add(student);
    }
}
```

Zdroj: (16)

Kód 8

```
// Kotlin
var students = listOf(Student("John", 0),
    Student("Julia", 2),
    Student("Matt", 1),
    Student("Katie", 0),
    Student("Dan", 0))

var firstList = students.filter { it.mark == 0 }.take(3)
var secondList = students.filter { it.mark == 1 }.take(2)
```

Zdroj: (16)

Jednoduchá funkce kalkulačky

Jednoduchá počítací metoda s návratovou hodnotou typu double funguje v obou jazycích stejným způsobem. Metoda přijímá argumenty dvou čísel a řetězce, určující typ operace. Řídící strukturou typu přepínač (switch, ekvivalent v Kotlinu je when) je vybrán kód, který se spustí, tzn. switch/when algoritmus vrátí návratovou hodnotu. Přestože je v tomto případě použit stejný způsob výpočtu, zápis v Kotlinu je na první pohled kratší a poněkud čitelnější.

Kód 9

```java
// Java
public static double calculate(double a, String op, double b) throws Exception
{
    switch (op) {
        case "plus":
            return a+b;
        case "minus":
            return a-b;
        case "div":
            return a/b;
        case "times":
            return a*b;
        default:
            throw new Exception();
    }
}
```

Zdroj: (17)

Kód 10

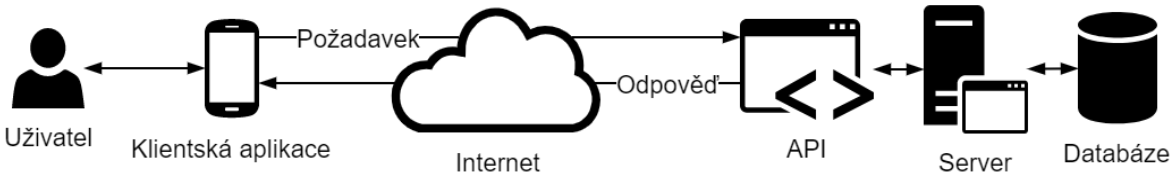
```kotlin
// Kotlin
fun calculate(a: Double, op: String, b: Double): Double = when (op) {
    "plus" -> a + b
    "minus" -> a - b
    "div" -> a / b
    "times" -> a * b
    else -> throw Exception()
}
```

Zdroj: (17)

3.5 API

Aplikační programové rozhraní, zkráceně API, je takové rozhraní, které umožňuje aplikacím na straně klienta přistupovat a získávat data, uložená na vzdáleném serveru či v databázi apod. API představuje soubor metod, který je tedy umístěn mezi serverem a klientskou aplikací, ve kterém jednotlivé metody slouží pro příjem a získávání informací, které jsou přenášeny prostřednictvím HTTP protokolu. Aplikační programové rozhraní se skládá ze dvou částí a to dokumentace, popisující samotné metody a způsob použití a poté samotný program, obsluhující požadavky a vytvářející odpovědi. Obecně řečeno, smysl API spočívá ve zjednodušení implementace software, konzumující data získaná přes internet. Klientská aplikace se při zasílání požadavků identifikuje univerzálním API klíčem. API klíč funguje jako jakýsi účet, kterým je požadavek autorizován a bez kterého by požadavek vrátila negativní odpověď. (18)

Obrázek 2 - Schéma API



Zdroj: autor podle (19)

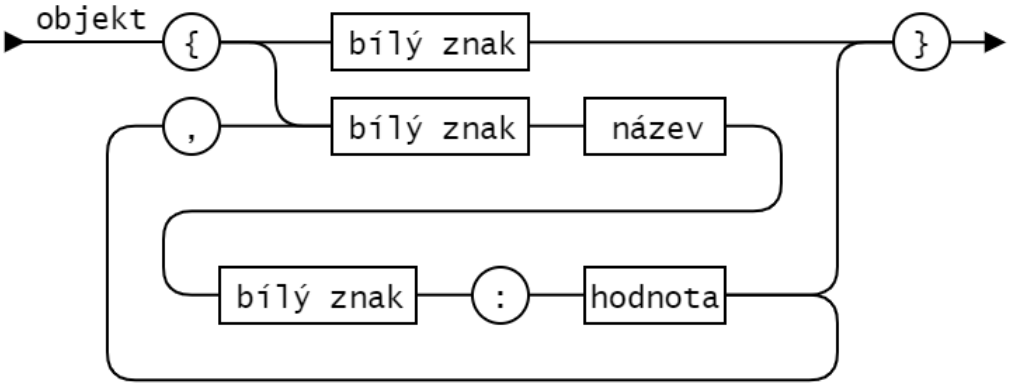
Rozhraní lze rozdělit do tří typů, soukromé, partnerské a veřejné. Mezi soukromé se řadí taková rozhraní, která jsou používaná pouze interně v rámci firmy. Data obsluhovaná v tomto sektoru jsou dostupná pouze uživatelům, patřící do daného subjektu. Partnerská API jsou použita pro přenos dat mezi konkrétními subjekty, např. podniky a jsou vázána určitou smlouvou. Veřejná rozhraní jsou už podle názvu ta rozhraní, která jsou dostupná široké veřejnosti, obvykle zdarma nebo za úplatu. Veřejná API, která jsou zdarma, mohou být omezena počtem obsloužených požadavků za jednotku času, např. 1000 požadavků za hodinu v rámci jednoho API klíče. Počet požadavků se omezuje z důvodu předejití přetížení rozhraní a udržení standardní rychlosti. Cena za placené API klíče se mění v závislosti na počtu obsloužených požadavků. Proto je v zájmu každého vývojáře tyto klíče zabezpečit, aby nebyl vyzrazen neoprávněným osobám. Data mohou být

prostřednictvím API přenášena v různých formátech, mezi nejčastější patří JSON, XML, HTML či v prostém textu. (18)

3.6 JSON

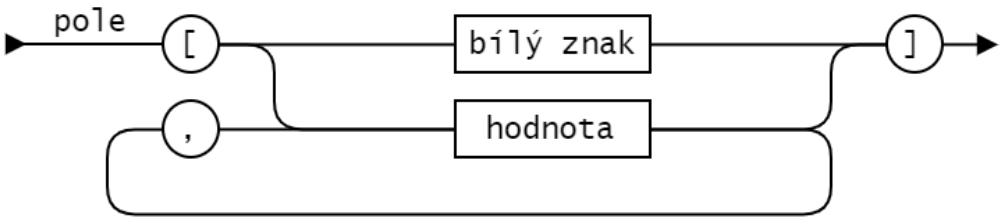
JavaScript Object Notation je datový formát pro uchování dat v textovém zápisu. Elementární data jsou uložena v párech „název : hodnota“, tento styl zápisu je převzatý z jazyka JavaScript. Díky této struktuře je formát snadno čitelný jak pro člověka, tak i lehce zpracovatelný strojem, a proto patří mezi nejpoužívanější a nejoblíbenější formáty dat při vývoji software. JSON zápis je založen na dvou základních strukturách, na kolekcích a listech. Kolekcí je množina různých hodnot, které jsou jako celek ekvivalentní objektu v jazyce Kotlin. Listy jsou soubory za sebou jdoucích prvků stejné struktury, může jít např. o seznam čísel, seznam řetězců apod. Takový list se v Kotlinu definuje jako pole. S pomocí takto navržených struktur je samotné mapování dat z JSON souboru do jednotlivých objektů v aplikaci snadné. Na obrázcích 3 a 4 jsou zakreslena schémata zápisu gramatiky ve formátu JSON pro kolekci (objekt) a list (pole). Bílé znaky představují netisknutelné znaky typu mezera, nový řádek, návrat na začátek řádku a tabulátor. (20)

Obrázek 3 - Schéma zápisu JSON kolekce



Zdroj: autor podle (20)

Obrázek 4 - Schéma zápisu JSON listu



Zdroj: autor podle (20)

4 Vlastní práce

4.1 Technická specifikace

Cílem je vytvořit aplikaci informativního charakteru jejíž tématem bude aktuální dění a události v blízkém vesmíru. Informace prezentované aplikací budou pocházet z veřejně dostupných aplikačních programových rozhraní. Poskytovateli těchto rozhraní budou servery NASA, N2YO a Open Notify. Data přenášená mezi těmito API službami a aplikací budou v textovém formátu JSON. Při vstupu uživatele na konkrétní funkci aplikace vytvoří požadavek a odešle ho prostřednictvím HTTP protokolu na URL adresu vzdáleného rozhraní. Odpovědí bude textový řetězec ve formátu JSON. Aplikace tento JSON soubor rozparsuje do datových tříd, ze kterých se poté informace zobrazí uživateli na obrazovce. Některé z funkcí budou navíc obsahovat odkazy na webové stránky s podrobnějšími informacemi o zkoumaném jevu. Vedlejšími funkcemi v aplikaci bude video přehrávač živého vysílání televize NASA, možnost zaslat zpětnou vazbu, zasílání denních notifikací. Aplikace bude mít dvě jazykové mutace, v českém a anglickém jazyce a výběr mutace bude probíhat automaticky na základě výchozího jazyka v zařízení.

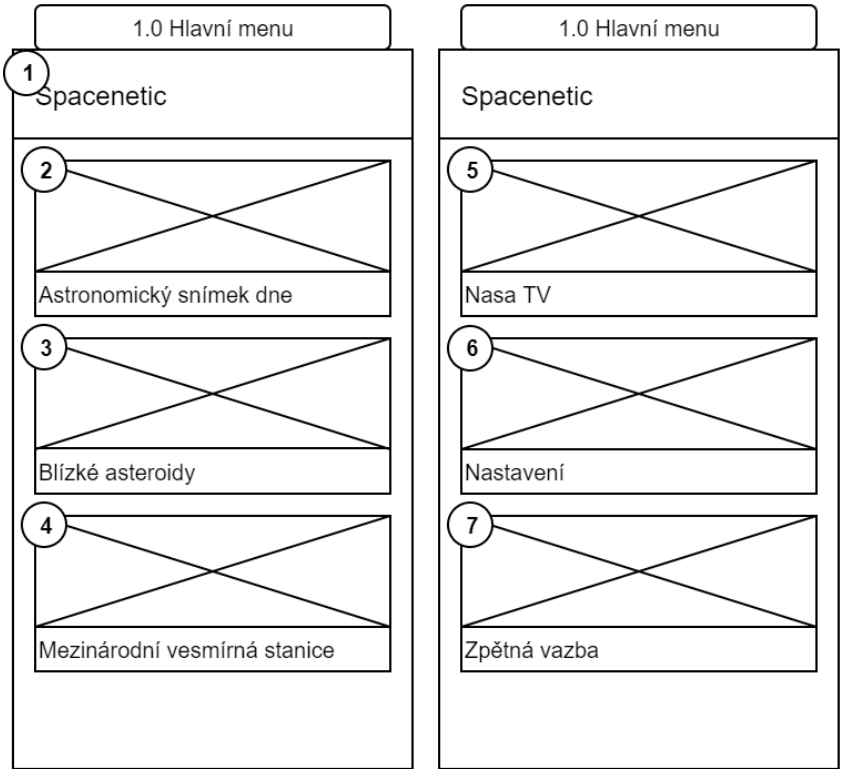
4.1.1 Hlavní menu

User story

- Jako uživatel chci, abych měl z hlavního menu přístup do všech funkcí, které aplikace nabízí.

Wireframe

Obrázek 5 - Wireframe - Hlavní menu



Zdroj: autor

Navigace

ID	Akce
2	přejít na obrazovku 2.0
3	přejít na obrazovku 3.0
4	přejít na obrazovku 4.0
5	přejít na obrazovku přehrávače
6	přejít na obrazovku 5.0
7	spustit výchozí email aplikaci s předvyplněnou adresou

Textace

ID	klíč	CZ	EN
1	@string/app_name	Spacenetic	Spacenetic
2	@string/apod	Astronomický snímek dne	Astronomy Picture of the Day
3	@string/asteroids	Blízké asteroidy	Near asteroids

4	@string/iss	Mezinárodní vesmírná stanice	International Space Station
5	@string/nasa_tv	Nasa TV	Nasa TV
6	@string/settings	Nastavení	Settings
7	@string/feedback	Zpětná vazba	Feedback

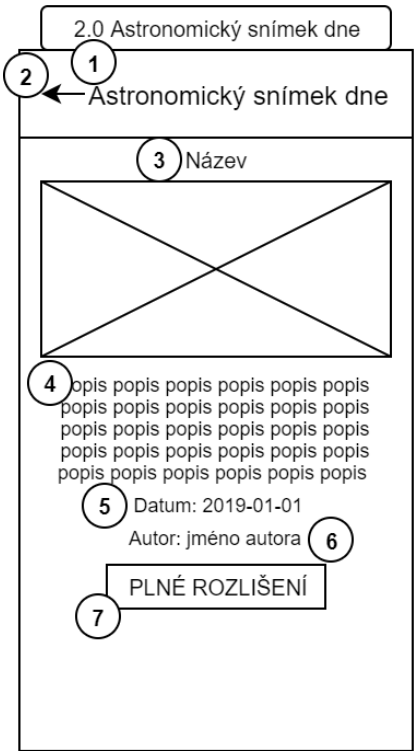
4.1.2 Astronomický snímek dne

User story

- Jako uživatel chci, aby na obrazovce byl název snímku, snímek v nižším rozlišení, popis, datum a jméno autora, pokud je uvedeno.
- Jako uživatel chci mít možnost přejít z této obrazovky na zobrazení snímku v plném rozlišení v okně internetového prohlížeče.
- Jako zadavatel požaduji, aby byla použita data z API služby Astronomy Picture of the Day na adrese api.nasa.gov.

Wireframe

Obrázek 6 - Wireframe - Astronomický snímek dne



Zdroj: autor

Navigace

ID	Akce
2	přejít na předchozí obrazovku (1.0)
7	spustit výchozí webový prohlížeč a přesměrovat uživatele na stránku s obrázkem v plném rozlišení (apod.nasa.gov)

Textace

ID	klíč	CZ	EN
1	@string/apod	Astronomický snímek dne	Astronomy Picture of the Day
5	@string/date	Datum:	Date:
6	@string/copyright	Autor:	Copyright:
7	@string/full_res	Plné rozlišení	Full resolution

Data

ID	API objekt
3	obj / title
4	obj / explanation
5	obj / date
6	obj / copyright

Ukázka odpovědi API

Kód 11


```
{
   "copyright": "Anton Komlev",
   "date": "2019-09-22",
   "explanation": "What do you see when you look into this sky? In the center,
in the dark, do you see a night sky filled with stars? Do you see a sunset to
the left? Do you see the ruins of an abandoned outpost on a hill? (The outpost
is on Askold Island, Russia.)  Do you see a photographer with a headlamp
contemplating surreal surroundings? (The image is a panorama of 38 images taken
last month and compiled into a Little Planet projection.)  Do you see a rugged
path lined with steps? Or do you see the eye of a dragon?",
   "hdurl":
"https://apod.nasa.gov/apod/image/1909/EyeDragonSky_Komlev_2000.jpg",
   "media_type": "image",
   "service_version": "v1",
   "title": "Eye Sky a Dragon",
   "url": "https://apod.nasa.gov/apod/image/1909/EyeDragonSky_Komlev_960.jpg"
}
```


Zdroj: autor podle https://api.nasa.gov/

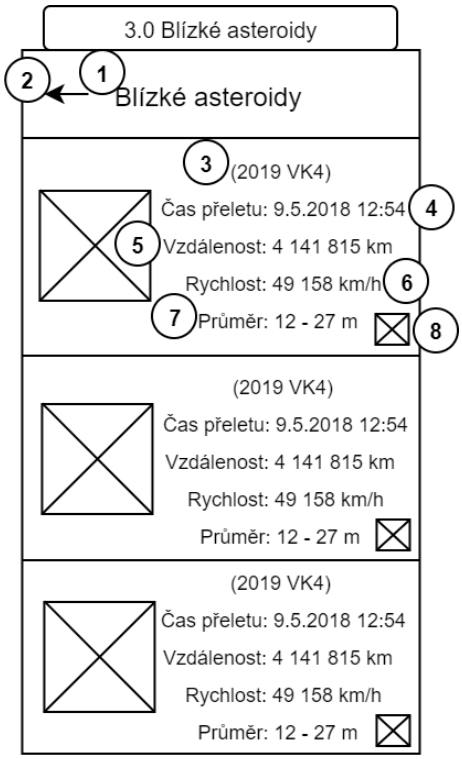
4.1.3 Blízké asteroidy

User story

- Jako uživatel chci, aby na obrazovce byl pro každý záznam v seznamu název asteroidu, čas přeletu kolem Země, vzdálenost od Země, rychlost a odhadovaný průměr asteroidu.
- Jako uživatel chci, aby byla obrázkem naznačena velikost asteroidu.
- Jako uživatel chci mít možnost přejít z této obrazovky na podrobné informace o asteroidu v okně internetového prohlížeče.
- Jako zadavatel požaduji, aby byla použita data z API služby Near Earth Object Web Service na adrese api.nasa.gov.

Wireframe

Obrázek 7 - Wireframe - Blízké asteroidy



Zdroj: autor

Navigace

ID	Akce
2	přejít na předchozí obrazovku (1.0)
8	spustit výchozí webový prohlížeč a přesměrovat uživatele na stránku s informacemi o asteroidu (ssd.jpl.nasa.gov)

Textace

ID	klíč	CZ	EN
1	@string/asteroids	Blízké asteroidy	Near asteroids
4	@string/pass_time	Čas přeletu:	Pass time:
5	@string/approach_distance	Vzdálenost:	Distance:
6	@string/velocity	Rychlost:	Velocity:
7	@string/estimated_diameter	Průměr:	Diameter:

Data

ID	API objekt
3	obj / near_earth_objects / [date] / name
4	obj / near_earth_objects / [date] / close_approach_data / epoch_date_close_approach
5	obj / near_earth_objects / [date] / close_approach_data / miss_distance / kilometers
6	obj / near_earth_objects / [date] / close_approach_data / relative_velocity / kilometers_per_hour
7	obj / near_earth_objects / [date] / estimated_diameter / meters / estimated_diameter_min obj / near_earth_objects / [date] / estimated_diameter / meters / estimated_diameter_max

Ukázka odpovědi API

Kód 12


```
{
   "links": {
      "next": "http://www.neowsapp.com/rest/v1/feed?start_date=2019-11-
15&end_date=2019-11-
15&detailed=false&api_key=J5HWLtcA9ZgTxUu7bAfjL3xcSm0gLeHI2JMk8JMI",
      "prev": "http://www.neowsapp.com/rest/v1/feed?start_date=2019-11-
13&end_date=2019-11-
```

```
13&detailed=false&api_key=J5HWLtcA9ZgTxUu7bAfjL3xcSm0gLeHI2JMk8JMI",
      "self": "http://www.neowsapp.com/rest/v1/feed?start_date=2019-11-
14&end_date=2019-11-
14&detailed=false&api_key=J5HWLtcA9ZgTxUu7bAfjL3xcSm0gLeHI2JMk8JMI"
   },
   "element_count": 1,
   "near_earth_objects": {
      "2019-11-14": [
         {
            "links": {
               "self":
"http://www.neowsapp.com/rest/v1/neo/3892278?api_key=J5HWLtcA9ZgTxUu7bAfjL3xcSm
0gLeHI2JMk8JMI"
            },
            "id": "3892278",
            "neo_reference_id": "3892278",
            "name": "(2019 UN14)",
            "nasa_jpl_url": "http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3892278",
            "absolute_magnitude_h": 20.853,
            "estimated_diameter": {
               "kilometers": {
                  "estimated_diameter_min": 0.1794547576,
                  "estimated_diameter_max": 0.4012730369
               },
               "meters": {
                  "estimated_diameter_min": 179.4547576093,
                  "estimated_diameter_max": 401.2730369001
               },
               "miles": {
                  "estimated_diameter_min": 0.1115079822,
                  "estimated_diameter_max": 0.2493394282
               },
               "feet": {
                  "estimated_diameter_min": 588.7623469549,
                  "estimated_diameter_max": 1316.5126303835
               }
            },
            "is_potentially_hazardous_asteroid": false,
            "close_approach_data": [
               {
                  "close_approach_date": "2019-11-14",
                  "close_approach_date_full": "2019-Nov-14 18:51",
                  "epoch_date_close_approach": 1573757460000,
                  "relative_velocity": {
                     "kilometers_per_second": "16.4122965874",
                     "kilometers_per_hour": "59084.2677145727",
                     "miles_per_hour": "36712.6662363132"
                  },
                  "miss_distance": {
                     "astronomical": "0.4836908831",
                     "lunar": "188.1557535259",
                     "kilometers": "72359125.850178997",
                     "miles": "44961875.8934122786"
                  },
                  "orbiting_body": "Earth"
               }
            ],
```

```
            "is_sentry_object": false
        },
    ]
  }
}
```

Zdroj: autor podle https://api.nasa.gov/

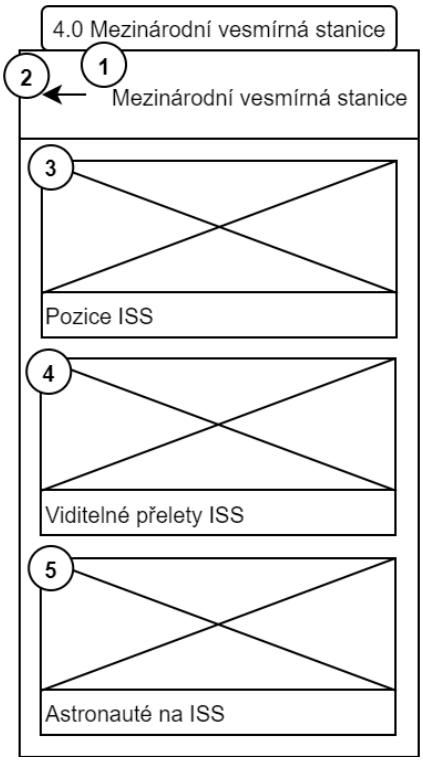
4.1.4 Mezinárodní vesmírná stanice

User story

- Jako uživatel chci, abych měl z obrazovky přístup na obrazovky „Pozice ISS“. „Viditelné přelety ISS“ a „Astronauté na ISS“.

Wireframe

Obrázek 8 - Wireframe - Mezinárodní vesmírná stanice



Zdroj: autor

Navigace

ID	Akce
2	přejít na předchozí obrazovku (1.0)
3	přejít na obrazovku 4.1
4	přejít na obrazovku 4.2

5	přejít na obrazovku 4.3

Textace

ID	klíč	CZ	EN
1	@string/iss	Mezinárodní vesmírná stanice	International Space Station
3	@string/iss_position	Pozice ISS	ISS position
4	@string/iss_passes	Viditelné přelety ISS	Visible ISS passes
5	@string/iss_astronauts	Astronauté na ISS	Astronauts on ISS

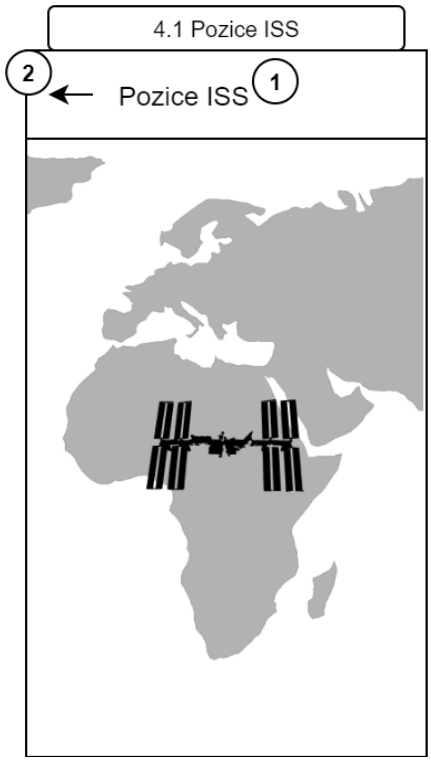
4.1.5 Pozice ISS

User story

- Jako uživatel chci, aby se na obrazovce zobrazovala aktuální pozice vesmírné stanice, která se bude aktualizovat jednou za vteřinu.
- Jako zadavatel požaduji, aby byla použita data z API na adrese api.open-notify.org.

Wireframe

Obrázek 9 - Wireframe - Pozice ISS



Zdroj: autor

Navigace

ID	Akce
2	přejít na předchozí obrazovku (4.0)

Textace

ID	klíč	CZ	EN
1	@string/iss_position	Pozice ISS	ISS position

Ukázka odpovědi API

Kód 13

```
{
   "iss_position": {
      "longitude": "-67.4149",
      "latitude": "-12.9809"
   },
   "timestamp": 1573732954,
   "message": "success"
}
```

Zdroj: autor podle http://open-notify.org/Open-Notify-API/ISS-Location-Now/

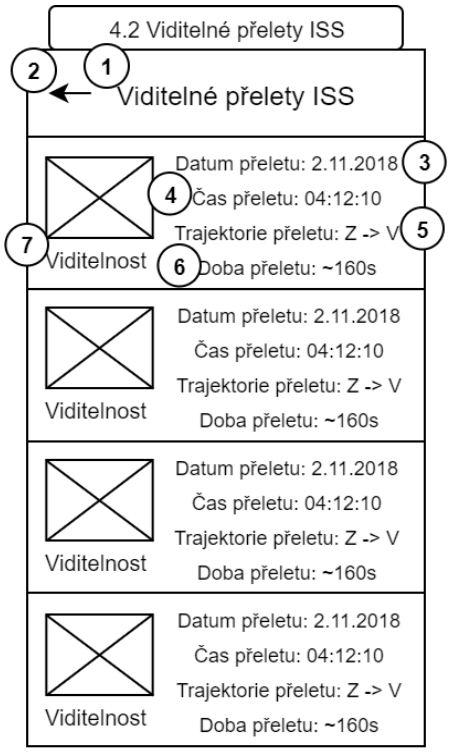
4.1.6 Viditelné přelety ISS

User story

- Jako uživatel chci, aby na obrazovce byl pro každý záznam v seznamu datum a čas přeletu, směr dráhy přeletu a jeho doba.
- Jako uživatel chci, aby úroveň viditelnosti byla prezentována obrázkem.
- Jako uživatel chci, aby pro výběr správných informací byla použita lokace získaná z GPS.
- Jako zadavatel požaduji, aby byla použita data z API na adrese n2yo.com.

Wireframe

Obrázek 10 - Wireframe Viditelné přelety ISS



Zdroj: autor

Navigace

ID	Akce
2	přejít na předchozí obrazovku (4.0)

Textace

ID	klíč	CZ	EN
1	@string/iss_passes	Viditelné přelety ISS	Visible ISS passes
3	@string/pass_date	Datum přeletu:	Pass date:
4	@string/pass_time	Čas přeletu:	Pass time:
5	@string/pass_direction	Trajektorie přeletu:	Pass trajectory:
6	@string/pass_duration	Doba přeletu:	Pass duration:
7	@string/visibility	Viditelnost	Visibility

Data

ID	API objekt
3	obj / passes / maxUTC
4	obj / passes / max UTC
5	obj / passes / startAzCompass obj / passes / endAzCompass
6	obj / passes / duration

Ukázka odpovědi API

Kód 14


```
{
   "info": {
      "satid": 25544,
      "satname": "SPACE STATION",
      "transactionscount": 0,
      "passescount": 3
   },
   "passes": [
      {
         "startAz": 212.11,
         "startAzCompass": "SW",
         "startEl": 0.24,
         "startUTC": 1574443595,
         "maxAz": 143.04,
         "maxAzCompass": "SE",
         "maxEl": 23.04,
         "maxUTC": 1574443895,
         "endAz": 75,
         "endAzCompass": "E",
         "endEl": 17.19,
         "endUTC": 1574444195,
         "mag": -0.2,
         "duration": 210
      }
   ]
}
```


Zdroj: autor podle https://www.n2yo.com/api/

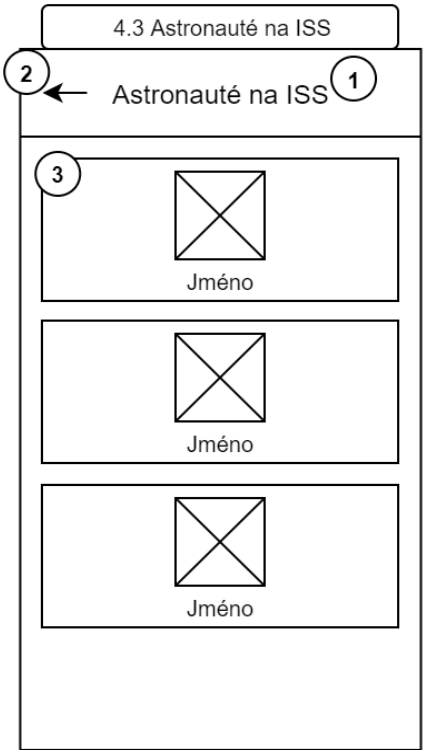
4.1.7 Astronauté na ISS

User story

- Jako uživatel chci, aby na obrazovce byla jména všech astronautů na vesmírné stanici.
- Jako uživatel chci mít možnost přejít z této obrazovky na podrobné informace o astronautovi v okně internetového prohlížeče.
- Jako zadavatel požaduji, aby byla použita data z API na adrese api.open-notify.org.

Wireframe

Obrázek 11 – Wireframe - Astronauté na ISS



Zdroj: autor

Navigace

ID	Akce
2	přejít na předchozí obrazovku (4.0)
3	spustit výchozí webový prohlížeč a přesměrovat uživatele na stránku s informacemi o astronautovi (en.wikipedia.org)

Textace

ID	klíč	CZ	EN
1	@string/iss_astronauts	Astronauté na ISS	Astronauts on ISS

Data

ID	API objekt
3	obj / people / name

Ukázka odpovědi API

Kód 15


```
{
   "people": [
      {
         "name": "Christina Koch",
         "craft": "ISS"
      },
      {
         "name": "Alexander Skvortsov",
         "craft": "ISS"
      },
      {
         "name": "Luca Parmitano",
         "craft": "ISS"
      }
   ],
   "number": 3,
   "message": "success"
}
```


Zdroj: autor podle http://open-notify.org/Open-Notify-API/People-In-Space/

4.1.8 Nasa TV

User story

- Jako uživatel chci, aby aplikace obsahovala vestavěný přehrávač online vysílání společnosti NASA.
- Jako zadavatel požaduji, aby byl použit živý přenos z adresy youtube.com.

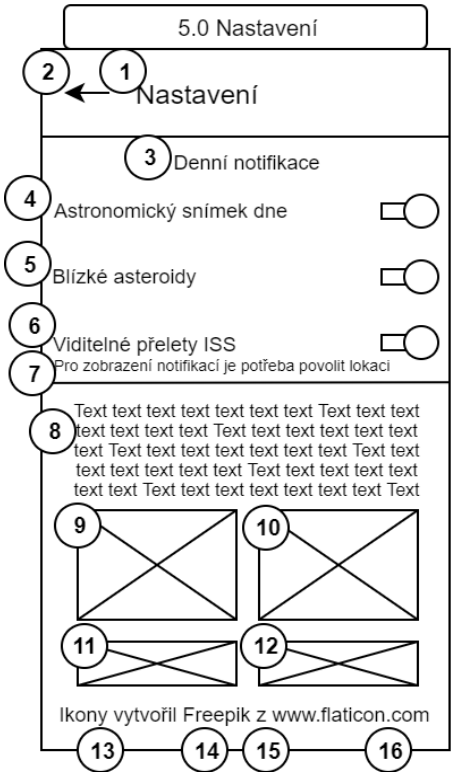
4.1.9 Nastavení

User story

- Jako uživatel chci mít možnost zapínání a vypínání zasílání pravidelných notifikací.
- Jako uživatel chci, aby bylo na obrazovce stručné představení samotné aplikace spolu s logem ČZU.
- Jako uživatel chci, abych se mohl prokliknout na klíčové weby tohoto projektu jako je stránka ČZU a poskytovatelů API.

Wireframe

Obrázek 12 - Wireframe - Nastavení



Zdroj: autor

Navigace

ID	Akce
2	přejít na předchozí obrazovku (1.0)
9	otevřít adresu www.czu.cz ve výchozím internetovém prohlížeči
10	otevřít adresu api.nasa.gov ve výchozím internetovém prohlížeči

11	otevřít adresu www.n2yo.com/api ve výchozím internetovém prohlížeči
12	otevřít adresu api.open-notify.org ve výchozím internetovém prohlížeči
14	otevřít adresu www.freepik.com ve výchozím internetovém prohlížeči
16	otevřít adresu www.flaticon.com ve výchozím internetovém prohlížeči

Textace

ID	klíč	CZ	EN
1	@string/settings	Nastavení	Settings
3	@string/ daily_notifications	Denní notifikace	Daily notifications
4	@string/apod	Astronomický snímek dne	Astronomy Picture of the Day
5	@string/asteroids	Blízké asteroidy	Near asteroids
6	@string/iss_passes	Viditelné přelety ISS	Visible ISS passes
7	@string/ permission_needed	Pro zobrazení notifikací je potřeba povolit lokaci	Location must be enabled to display notifications
8	@string/project	Tento projekt vznikl jako praktická část diplomové práce na Provozně ekonomické fakultě na České zemědělské univerzitě v Praze. Informace v aplikaci pochází z aplikačních programových rozhraní (API), poskytovány servery NASA, N2YO a Open Notify. Tyto servery můžete navštívit kliknutím na některý z obrázků níže.	This project was created as practical part of diploma thesis at Faculty of Economics and Management at Czech University of Life Sciences Prague. Informations in application comes from application program interfaces (API), provided by servers NASA, N2YO and Open Notify. You can visit these servers by tapping on any of the pictures below.
13	@string/set_icons	Ikony vytvořil	Icons made by
14	@string/freepik	Freepik	Freepik
15	@string/set_from	z	from
16	@string/flaticon	www.flaticon.com	www.flaticon.com

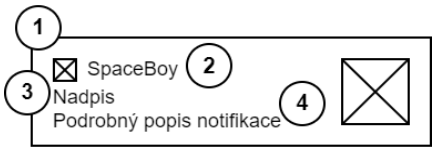
4.1.10 Notifikace

User story

- Jako uživatel chci, aby mi aplikace denně zasílala notifikace s odkazem na obrazovky „Astronomický snímek dne“, „Blízké asteroidy“ a „Přelety ISS“ a to v 9, 12 a 17 hodin v odpovídajícím pořadí

Wireframe

Obrázek 13 - Wireframe - Notifikace



Zdroj: autor

Navigace

ID	Akce
1	otevřít aplikaci a přejít na obrazovku 2.0 / 3.0 / 4.2 dle typu notifikace

Textace

ID	klíč	CZ	EN
2	@string/app_name	Spacenetic	Spacenetic
3a	@string/apod	Astronomický snímek dne	Astronomy Picture of the Day
3b	@string/asteroids	Blízké asteroidy	Near asteroids
3c	@string/iss_passes	Viditelné přelety ISS	Visible ISS passes
4a	@string/ notif_apod	Dobré ráno! Máme pro Vás nový snímek z vesmíru	Good morning! We have a new picture from the space for you
4b	@string/ notif_asteroid	Přejděte do aplikace a podívejte se, které asteroidy dnes budou prolétat kolem Země	Open app and ook which asteroids are passing close to Earth today
4c	@string/ notif_iss	Blíží se čas pozorování přeletů Mezinárodní vesmírné stanice. Zjistěte, kdy bude možné sledovat ISS právě z Vaší lokace	International Space Station observation time is coming! Find out when you can watch ISS passing above your location

4.2 Implementace

Před samotným popisem postupu zpracování aplikace zde souhrnně uvedu použité frameworky a použité obrázky v aplikaci.

Prvním nástrojem je Java knihovna OkHttp, která v Kotlin projektu funguje perfektně díky interoperabilitě obou jazyků. Jedná se o HTTP klienta, kterého používám pro komunikaci s aplikačními rozhraními. Knihovna použitá v projektu je verze 3, novější verze 4 je již naprogramována v Kotlinu. Další použitou knihovnou je Gson. Opět se jedná o Java knihovnu a její funkce spočívá v konverzi Java objektů do zápisu formátu JSON, nebo naopak v parsování JSON řetězce na Java objekt. V tomto projektu je používán pro parsování API odpovědí do datových tříd. I když ne tolik důležitou tak velmi užitečnou je knihovna Picasso, která je použita pro stažení obrázků z internetu a/nebo jejich zobrazení na různých místech v aplikaci. Posledním nástrojem je YouTube Android Player API služba, což je rozhraní, díky kterému jsem mohl do projektu integrovat přehrávač platformy YouTube. V aplikaci se používá pro přehrávání živého přenosu televize NASA. (21)(22)(23)(24)

Obrázky, zobrazující se staticky v aplikaci pochází ze stránek, publikujících materiály pod otevřenou licencí Creative Commons 0, konkrétně se jedná o pixabay.com a unsplash.com. Použité ikony v aplikaci pochází ze stránky flaticon.com a podmínkou pro jejich použití je zmínění autora v produktu, pro který byly ikony použity. Tato povinnost je splněna uvedením autora a stránky na obrazovce nastavení.

4.2.1 MainActivity.kt

Z hlediska implementace patří tato obrazovka mezi ty jednodušší. V této aktivitě jsem pouze nastavil metody setOnClickListener pro proklik na další obrazovky. Za zmínku zde stojí kód, který slouží pro prvotní nastavení alarmů, které na pozadí spouští notifikace. SharedPreferences je rozhraní, které slouží pro ukládání primitivních dat do úložiště telefonu, tzn. při uložení proměnné a restartování aplikace lze pomocí tohoto rozhraní znovu uloženou proměnnou načíst. Zde jsem si vytvořil instanci typu SharedPreferences a v podmínce načítám proměnnou „firstRun“ z této instance. Pokud načítaná proměnná dosud nebyla do paměti zapsána, její výchozí hodnota je true, jedná se tedy o první spuštění a podmínka se provede. Metodou setAlarm třídy AlrManager je nastaven alarm

a proměnná „firstRun“ je zapsána do paměti jako false. Tento kód se již nespustí (v rámci této instalace).

Kód 16

```
val prefs = getSharedPreferences("SharedPreferences", Context.MODE_PRIVATE)
if (prefs.getBoolean("firstRun", true)) {

    AlrManager().setAlarm(this)
    prefs.edit().putBoolean("firstRun", false).apply()
}
```

Zdroj: autor

4.2.2 ApodActivity.kt

V této aktivitě jsem využil knihovny OkHttp pro odeslání požadavku a Gson pro zpracování JSON odpovědi. Tu jsem následně uložil do datové třídy AstronomyPicture. V kódu za klíčovým slovem runOnUIThread se skrývá zobrazení informací v grafickém rozhraní, v této ukázce jsem tuto část ukryl vzhledem k nedůležitosti.

Kód 17

```
private fun fetchJson() {
    val url = "https://api.nasa.gov/planetary/apod?api_key=$apiKey"
    val request = Request.Builder().url(url).build()
    val client = OkHttpClient()

    client.newCall(request).enqueue(object: Callback {
        override fun onResponse(call: Call, response: Response) {
            val body = response.body()?.string()
            val gson = GsonBuilder().create()
            val apod = gson.fromJson(body, AstronomyPicture::class.java)

            runOnUiThread {
                ...
            }
        }

        override fun onFailure(call: Call, e: IOException) {
            println("Failed to execute request")
        }
    })
}
data class AstronomyPicture(val copyright: String,
                            val date: String,
                            val explanation: String,
                            val media_type: String,
                            val hdurl: String,
                            val title: String,
                            val url: String)
```

Zdroj: autor

4.2.3 AsteroidsActivity.kt

Pro získání dat na obrazovce se záznamy o blízkých přeletech asteroidů jsem využil stejný způsob, tedy kombinaci knihoven OkHttp a Gson. Navíc je nyní použita komponenta RecyclerView, která slouží jako seznam s dynamickým počtem položek. Vzhled položky je definován dodatečným layout souborem a při spuštění aktivity je tento layout jedné položky vykreslen tolikrát, kolika záznamy je RecyclerView naplněn. Naplnění seznamu spočívá ve vytvoření třídy RecyclerView.Adapter, která musí obsahovat metody getItemCount(), onCreateViewHolder a onBindViewHolder(). První metoda getItemCount() definuje počet požadovaných záznamů v seznamu, onCreateViewHolder() následně vygeneruje seznam prostřednictvím metody LayoutInflater.inflate() s přiloženým XML layoutem pro záznam. Posledním krokem je provedení metody onBindViewHolder(), do které se vloží kód pro obsloužení jednotlivého záznamu. Protože aplikace obsahuje dvě jazykové mutace, českou a anglickou, zobrazené výsledky jsem naformátoval tak aby odpovídaly aktuální lokalizaci aplikace. Zvláštní by se mohlo zdát použití číselného formátu Locale.FRANCE, důvodem je absence české lokalizace a tento formát je jeden z těch, které jsou shodné se zápisem čísel v češtině.

Kód 18

```
class AsteroidAdapter(val asteroids: AsteroidsFeed):
androidx.recyclerview.widget.RecyclerView.Adapter<AsteroidViewHolder>() {
    ...
    override fun getItemCount(): Int {
        return asteroids.element_count
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
    AsteroidViewHolder {
        val layoutInflater = LayoutInflater.from(parent.context)
        val cellForRow = layoutInflater.inflate(R.layout.asteroid_row, parent,
        false)
        return AsteroidViewHolder(cellForRow)
    }

    override fun onBindViewHolder(holder: AsteroidViewHolder, position: Int) {

        ...

        val sdfEn = SimpleDateFormat("MMM dd, yyyy HH:mm", Locale.US)
        val sdfCz = SimpleDateFormat("dd.MM.yyyy HH:mm", Locale.FRANCE)
        val numEn = NumberFormat.getInstance(Locale.US)
        val numCz = NumberFormat.getInstance(Locale.FRANCE)

        val distance: String
```

```
        val velocity: String
        val meanDia = ((passes.estimated_diameter.meters.estimated_diameter_min
        + passes.estimated_diameter.meters.estimated_diameter_max) / 2).toInt()

        if(Locale.getDefault().displayLanguage == "čeština") {
            holder.view.text_date.text = sdfCz.format(
            passes.close_approach_data[0].epoch_date_close_approach)

            distance = numCz.format((passes.close_approach_data[0]
            .miss_distance.kilometers.toDouble()).toInt()) + " km"

            velocity = numCz.format(((passes.close_approach_data[0]
            .relative_velocity.kilometers_per_hour).toDouble()).toInt())
            + " ${instance.getString(R.string.km_p_h)}"
        }
        else {
            holder.view.text_date.text = sdfEn.format(passes
            .close_approach_data[0].epoch_date_close_approach)

            distance = numEn.format((passes.close_approach_data[0]
            .miss_distance.kilometers.toDouble()).toInt()) + " km"

            velocity = numEn.format(((passes.close_approach_data[0]
            .relative_velocity.kilometers_per_hour).toDouble()).toInt())
            + " ${instance.getString(R.string.km_p_h)}"
        }
...
    }
}
```

Zdroj: autor

Pro ukázku také přikládám zápis data tříd pro tuto funkci, vzhledem k obsáhlosti JSON odpovědi, které toto API poskytuje.

Kód 19

```
data class AsteroidsFeed(val element_count: Int,
                         val near_earth_objects: Map<String, Array<Asteroids>>)
data class Asteroids(val name: String,
                     val nasa_jpl_url: String,
                     val estimated_diameter: EstimatedDiameter,
                     val close_approach_data: Array<CloseApproachData>)
data class EstimatedDiameter(val meters: Meters)
data class Meters(val estimated_diameter_min: Double,
                  val estimated_diameter_max: Double)
data class CloseApproachData(val epoch_date_close_approach: Long,
                             val relative_velocity: RelativeVelocity,
                             val miss_distance: MissDistance)
data class RelativeVelocity(val kilometers_per_hour: String)
data class MissDistance(val kilometers: String)
```

Zdroj: autor

4.2.4 IssActivity.kt

Tato třída slouží zaprvé jako menu druhého řádu pro obrazovky týkající se vesmírné stanice, zadruhé obsahuje logiku získání povolení pro lokaci, která je požadována v jedné z funkcí. Funkce checkPermission() typu boolean vrací příznak povolené lokace v aplikaci a je volána vždy jako první po kliknutí na položku menu (cardView_IssPasses.setOnClickListener). Pokud návratová hodnota je false, potom je spuštěn požadavek na povolení lokace metodou requestPermissions(), po které následuje provedení metod startLocationPermissionRequest() a onRequestPermissionResult(). Výsledkem je povolení lokace uživatelem a spuštění požadované aktivity (IssPassesActivity), ve které je lokace použita, nebo zobrazení chybové hlášky komponentou Snackbar.

Kód 20

```
cardView_IssPasses.setOnClickListener {
    ...
        if (!checkPermissions()) {
            requestPermissions()
        }
        else {
            startActivity(Intent(this, IssPassesActivity::class.java))
        }
    ...
}
...
private fun checkPermissions() = ActivityCompat.checkSelfPermission(
    this, ACCESS_COARSE_LOCATION) == PERMISSION_GRANTED

private fun startLocationPermissionRequest() {
    ActivityCompat.requestPermissions(this, arrayOf(ACCESS_COARSE_LOCATION),
    REQUEST_PERMISSIONS_REQUEST_CODE)
}

private fun requestPermissions() {
    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
    ACCESS_COARSE_LOCATION)) {
        showSnackbar(R.string.permission_rationale, android.R.string.ok,
        View.OnClickListener {
            startLocationPermissionRequest()
        })
    }
    else {
        startLocationPermissionRequest()
    }
}

override fun onRequestPermissionsResult(
    requestCode: Int,
```

```
    permissions: Array<out String>,
    grantResults: IntArray) {
        if (requestCode == REQUEST_PERMISSIONS_REQUEST_CODE) {
            when {
                (grantResults[0] == PackageManager.PERMISSION_GRANTED) ->
                startActivity(Intent(this, IssPassesActivity::class.java))


                else -> {
                    showSnackbar(R.string.permission_denied_explanation,
                    R.string.settings,
                    View.OnClickListener {
                        val intent = Intent().apply {
                            action = Settings.
                                     ACTION_APPLICATION_DETAILS_SETTINGS
                            data = Uri.fromParts("package", APPLICATION_ID,
                                   null)
                            flags = Intent.FLAG_ACTIVITY_NEW_TASK
                        }
                        startActivity(intent)
                    })
            }
        }
    }
}
```

Zdroj: autor

4.2.5 IssPositionActivity.kt

Aktivita obrazovky Pozice ISS sestává z komponenty Google Maps. Jako první jsem si vytvořil na zobrazené mapě ukazatel metodou mMap.addMarker() a tomu nastavil úvodní souřadnice. Aby se ukazatel polohy stanice na mapě obnovoval každou vteřinu, použil jsem postDelayed() metodu z třídy Handler, která se provádí pravidelně dle nastavené hodnoty proměnné delay v milisekundách. Do této opakující se části kódu jsem vložil zaslání požadavku a naparsování odpovědi stejným způsobem jako v předchozích příkladech, a navíc k tomu jsem znovu překreslil ukazatel na mapě s použitím souřadnic, obdržených z API.

Kód 21

```
override fun onMapReady(googleMap: GoogleMap) {
    mMap = googleMap
    val handler = Handler()
    val delay = 1000L
    var iss = mMap.addMarker(MarkerOptions().
              position(LatLng(0.0, 0.0)).visible(false))
    var lat = 0.0
    var lon = 0.0
    val url = "http://api.open-notify.org/iss-now.json"
```

```
    val request = Request.Builder().url(url).build()
    val client = OkHttpClient()

    handler.postDelayed(object: Runnable {
        override fun run() {

            client.newCall(request).enqueue(object: Callback {
                override fun onResponse(call: Call, response: Response) {
                    ...
                }
                override fun onFailure(call: Call, e: IOException) {
                    ...
                }
            })

            iss.remove()
            iss = mMap.addMarker(MarkerOptions()
                      .position(LatLng(lat, lon))
                      .icon(BitmapDescriptorFactory
                      .fromResource(R.drawable.map_marker)))
            mMap.moveCamera(CameraUpdateFactory.newLatLng(LatLng(lat, lon)))
            handler.postDelayed(this, delay)
        }
    }, delay)
}
```

Zdroj: autor

4.2.6 IssPassesActivity.kt

Tato obrazovka je, co se implementace týče, velmi podobná s obrazovkou s přelety asteroidů. Navíc je tu získání lokace ze zařízení a použití souřadnic jako argument v těle požadavku. Podmínka na základě úspěchu či neúspěchu zobrazí uživateli příslušnou informaci pomocí Toast komponenty. Následně je zavolána metoda createRequest() se souřadnicemi jako argumenty, která se postará o zbývající práci.

Kód 22

```
private fun getLastLocation(){
    fusedLocationClient.lastLocation.addOnCompleteListener(this) { task ->
        if (task.isSuccessful && task.result != null) {
            Toast.makeText(this, R.string.location_detected,
            Toast.LENGTH_LONG).show()
        }
        else {
            Toast.makeText(this, R.string.no_location_detected,
            Toast.LENGTH_LONG).show()
        }
        createRequest(task.isSuccessful,
                      task.result!!.latitude,
                      task.result!!.longitude,
                      task.result!!.altitude)
    }
```

```
}
...
private fun createRequest(locationSuccess: Boolean, lat: Double, lon: Double,
alt: Double) {
    val url: String

    if (locationSuccess) {
        if (alt == 0.0) {
            url = "http://www.n2yo.com/rest/v1/satellite/visualpasses/
                  25544/$lat/$lon/400.0/10/10/?apiKey=$apiKey"
        }
        else {
            url = "http://www.n2yo.com/rest/v1/satellite/visualpasses/
                  25544/$lat/$lon/$alt/10/10/?apiKey=$apiKey"
        }
    }
    else {
        url = "http://www.n2yo.com/rest/v1/satellite/visualpasses/
              25544/50.08/14.43/400.0/10/10/?apiKey=$apiKey"
    }
```

Zdroj: autor

4.2.7 IssAstronautsActivity.kt

Aktivita funguje na stejném principu jako AsteroidsActivity a IssPassesActivity, navíc zde funguje proklik na položku seznamu RecyclerView. Na každou položku v seznamu je nastavena akce na klik, která uživatele přesměruje do výchozího internetového prohlížeče, ve kterém se otevře URL. Adresa je v aplikaci spojena z konstanty wikiUrl a jména astronauta, které obdržím z API. Uživateli se tedy zobrazí stránka s astronautem na webu Wikipedie.

Kód 23

```
const val wikiUrl = "https://en.wikipedia.org/wiki/"

...
class AstronautViewHolder(val view: View, var astronaut: People? = null):
androidx.recyclerview.widget.RecyclerView.ViewHolder(view) {

    init {
        view.cardView_issAstronaut.setOnClickListener {
            val intent = Intent(Intent.ACTION_VIEW,
                        Uri.parse(wikiUrl + astronaut!!.name))

            view.context.startActivity(intent)
        }
    }
}
```

Zdroj: autor

4.2.8 NasaTvActivity.kt

Implementace integrovaného prohlížeče je docíleno již předpřipravenými metodami třídy YouTubeBaseActivity(), kterou jsem rozšířil o API klíč ke službě a kód videa.

Kód 24

```
class NasaTvActivity : YouTubeBaseActivity() {

    private val apiKey = "..."
    private val videoCode = "21X5lGlDOfg"

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        player_nasaTv.initialize(apiKey,
        object: YouTubePlayer.OnInitializedListener {
            override fun onInitializationSuccess(p0: YouTubePlayer
            .Provider?, p1: YouTubePlayer?, p2: Boolean) {
                p1?.loadVideo(videoCode)
            }

            override fun onInitializationFailure(p0: YouTubePlayer
            .Provider?, p1: YouTubeInitializationResult?) {
                Toast.makeText(applicationContext,
                R.string.video_cannot_be_played,
                Toast.LENGTH_SHORT).show()
            }
        })
    }
}
```

Zdroj: autor

4.2.9 SettingsActivity.kt

Obrazovka nastavení slouží hlavně pro zapnutí či vypnutí denních notifikací. Při vstupu na obrazovku načtu ze SharedPreferences boolean hodnoty, představující aktuální stav přepínačů, následně na každý z přepínačů aplikuji metodu setOnCheckedChangeListener, která je aktivována se změnou stavu přepínače. Při změně nastavení přepínače je uložena nová hodnota do SharedPreferences a následně se zavolá setAlarmSingle() s argumenty, na základě kterých je alarm ukončen nebo zaregistrován. V ukázce je zobrazena implementace jednoho z přepínačů.

Kód 25

```
val prefs = getSharedPreferences("SharedPreferences", Context.MODE_PRIVATE)
...
switch_notifApod.isChecked = prefs.getBoolean("notifApod", true)
...
```

```
switch_notifApod.setOnCheckedChangeListener {_, isChecked ->
    if (isChecked) {
        prefs.edit().putBoolean("notifApod", true).apply()
        AlrManager().setAlarmSingle(this, "notifApod", 1)
    }
    else {
        prefs.edit().putBoolean("notifApod", false).apply()
        AlrManager().setAlarmSingle(this, "notifApod", 1)
    }
}
```

Zdroj: autor

4.2.10 AlrManager.kt

Třída AlrManager (takto pojmenován slouží pro vytváření a rušení alarmů, tedy takových služeb Androidu, které běží nezávisle na aplikaci na pozadí OS. Proto jsem zvolil Android třídu AlarmManager jako nástroj pro plánování notifikací aplikace.

Smysl funkce setAlarm() spočívá v přípravě pro nastavení všech alarmů v aplikaci (ty jsou 3, každý pro jeden typ notifikace) pod podmínkou, že je jejich zasílání povoleno. Opět jsem použil třídu SharedPreferences, se kterou pracuji při ukládání a načítání uživatelského nastavení notifikací. Při kladném výsledku podmínky volám metodu startAlarm(), v opačném případě cancelAlarm(). Argumentem v obou voláních je záměr, číslo typu notifikace a u startAlarm() navíc konstanta, uvádějící hodinu, ve kterou se notifikace pustí. Tento kód se spustí při prvním spuštění aplikace a po restartu zařízení.

Metoda setAlarmSingle() je podobná metodě první, avšak s tím rozdílem, že připraví jen jeden alarm. Díky argumentům, které dostane, zvolí odpovídající alarm, který buď registruje, nebo zruší. Funkce je volána při změně nastavení v aktivitě Settings.

Registraci alarmu v systému obsluhuje startAlarm(). Nejdříve přidám do existujícího záměru dodatečnou informaci o typu požadované notifikaci pomocí putExtra(). Poté vytvářím instanci třídy Calendar, do které nastavím čas spuštění notifikace. S těmito daty zavolám setInexactRepeating z vytvořeného objektu AlarmManager.

Tyto alarmy fungují na principu Broadcast Receivers. V použitém záměru je od začátku odkaz na třídu, která se spustí v naplánovaném čase. V tu chvíli alarm začne „vysílat“ a toto vysílání zachytí zaregistrovaný receiver, v tomto případě třída NotificationReceiver a ta už obsahuje logiku vygenerování notifikace.

Poslední metodou je cancelAlarm(), která zruší konkrétní alarm běžící na pozadí.

Kód 26

```
fun setAlarm(cont: Context) {

    context = cont
    alarmMgr = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager
    val alarmIntent = Intent(context, NotificationReceiver::class.java)

    val prefs = context.getSharedPreferences("SharedPreferences", MODE_PRIVATE)

    if (prefs.getBoolean("notifApod", true)) {
        startAlarm(alarmIntent, 1, hourApod)}
    else {cancelAlarm(alarmIntent, 1)}

    if (prefs.getBoolean("notifAsteroids", true)) {
        startAlarm(alarmIntent, 2, hourAsteroids)}
    else {cancelAlarm(alarmIntent, 2)}

    if (prefs.getBoolean("notifIss", true)) {
        startAlarm(alarmIntent, 3, hourIss)}
    else {cancelAlarm(alarmIntent, 3)}
}

fun setAlarmSingle(cont: Context, key: String, type: Int) {

    context = cont
    alarmMgr = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager
    val alarmIntent = Intent(context, NotificationReceiver::class.java)

    val prefs = context.getSharedPreferences("SharedPreferences", MODE_PRIVATE)

    if (prefs.getBoolean(key, true)) {
        when (type) {
            1 -> startAlarm(alarmIntent, type, hourApod)
            2 -> startAlarm(alarmIntent, type, hourAsteroids)
            3 -> startAlarm(alarmIntent, type, hourIss)
        }
    }
    else {cancelAlarm(alarmIntent, type)}
}

private fun startAlarm(alarmIntent: Intent, type: Int, hour: Int) {

    alarmIntent.putExtra("type", type)

    val pendingIntent = PendingIntent.getBroadcast(context, type, alarmIntent,
    PendingIntent.FLAG_UPDATE_CURRENT)

    val calendar = Calendar.getInstance().apply {
        timeInMillis = System.currentTimeMillis()
        set(Calendar.HOUR_OF_DAY, hour)}

    if (Calendar.getInstance().after(calendar)) {
        calendar.add(Calendar.DAY_OF_MONTH, 1)
    }

    alarmMgr!!.setInexactRepeating(
```

```
        AlarmManager.RTC_WAKEUP,
        calendar.timeInMillis,
        AlarmManager.INTERVAL_DAY,
        pendingIntent)
}

private fun cancelAlarm(alarmIntent: Intent, type: Int) {

    val pendingIntent = PendingIntent.getBroadcast(context, type, alarmIntent,
    PendingIntent.FLAG_UPDATE_CURRENT)

    alarmMgr?.cancel(pendingIntent)
}
```

Zdroj: autor

4.2.11 NotificationReceiver.kt

Třída rozšiřuje třídu BroadcastReceiver a její hlavní a jediná metoda onReceive() se spouští při zachycení vysílání třídy AlrManager. Jakmile je spuštěna, třída si převezme informace o typu požadované notifikace pomocí getIntExtra() a tuto informaci použije pro zvolení vhodných textů a ikony pro notifikaci. Poté následuje proces zobrazení notifikace dle standardního postupu. Níže je naznačen způsob vytvoření notifikace uvnitř onReceive().

Kód 27

```
notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager

val type = intent.getIntExtra("type", 0)
...
val pendingIntent = PendingIntent.getActivity(context, 0,
    Intent(context, Class.forName(className)),
    PendingIntent.FLAG_UPDATE_CURRENT)

notificationChannel = NotificationChannel(channelId, description,
NotificationManager.IMPORTANCE_HIGH)
notificationManager.createNotificationChannel(notificationChannel)

builder = NotificationCompat.Builder(context, channelId)
    .setContentTitle(contentTitle)
    .setContentText(contentText)
    .setSmallIcon(R.drawable.ic_stat_earth)
    .setLargeIcon(largeIcon!!)
    .setColor(Color.argb(100, 11, 61, 145))
    .setContentIntent(pendingIntent)
    .setAutoCancel(true)
    .setStyle(BigTextStyle().bigText(contentText))

notificationManager.notify(type, builder.build())
```

Zdroj: autor

4.2.12 BootReceiver.kt

Alarmy, které běží na pozadí jako služby zůstávají zachované při vypnutí aplikace. Pokud je ale zařízení restartováno, služby jsou přerušeny. Z toho důvodu jsem vytvořil třídu typu BroadcastReceiver, která čeká na restart systému. Toho jsem docílil přidáním oprávnění RECEIVE_BOOT_COMPLETED a zaregistrováním receiveru do souboru manifest.xml. Následné opětovné spuštění alarmů je uvnitř metody onReceive() volané při zachycení broadcastu po rebootu zařízení.

Kód 28

```
override fun onReceive(context: Context, intent: Intent) {
    if (intent.action == "android.intent.action.BOOT_COMPLETED") {

        AlrManager().setAlarm(context)
    }
}
```

Zdroj: autor

4.2.13 ConnectivityCheck.kt

Samostatná třída ConnectivityCheck obsahuje pouze jednu metodu, kterou je isConnected() typu boolean, s návratovou hodnotou true, pokud je zařízení připojeno k síti.

Kód 29

```
fun isConnected(context: Context): Boolean {
    val connectivityManager = context.getSystemService(
    Context.CONNECTIVITY_SERVICE) as ConnectivityManager
    val networkInfo = connectivityManager.activeNetworkInfo
    return networkInfo != null && networkInfo.isConnected
}
```

Zdroj: autor

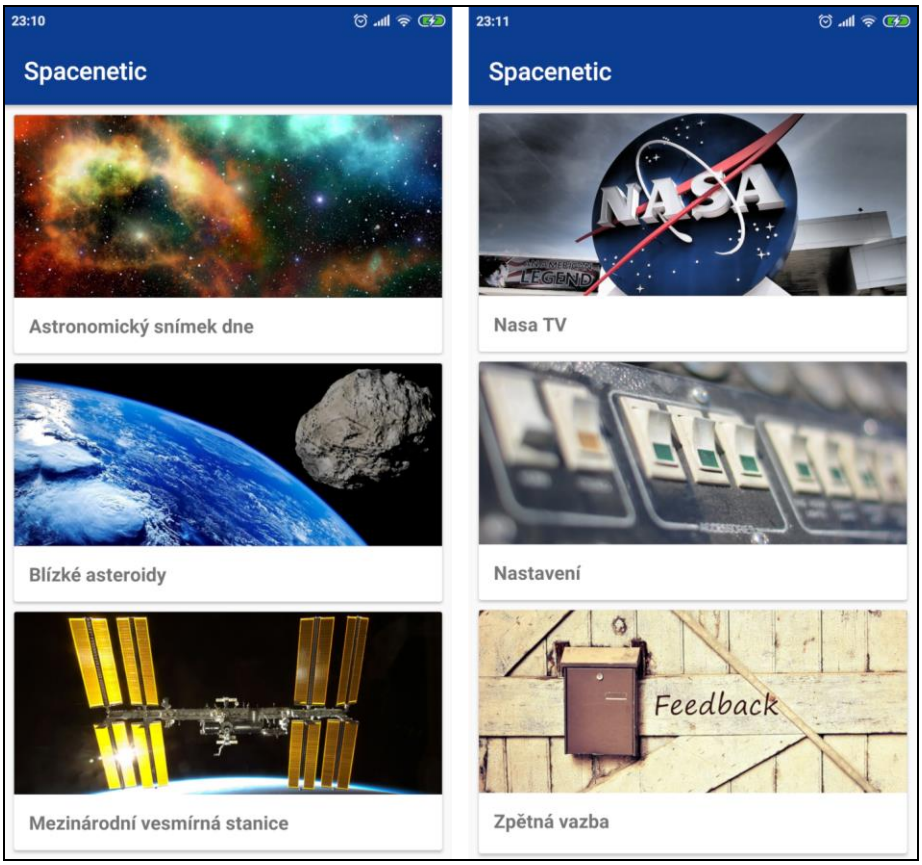
4.3 Výsledný vzhled aplikace

4.3.1 Hlavní menu

Na obrázku 14 je výsledný vzhled obrazovky, která byla navrhnuta pomocí wireframe na obrázku 5, v kapitole 4.1.1.

Hlavní menu je zobrazeno po spuštění aplikace. Uživatel se odtud naviguje do všech částí aplikace. Při otevření položky Zpětná vazba je spuštěna výchozí nainstalovaná aplikace emailového klienta v zařízení s předvyplněnou cílovou adresou.

Obrázek 14 - Výsledný vzhled - Hlavní menu



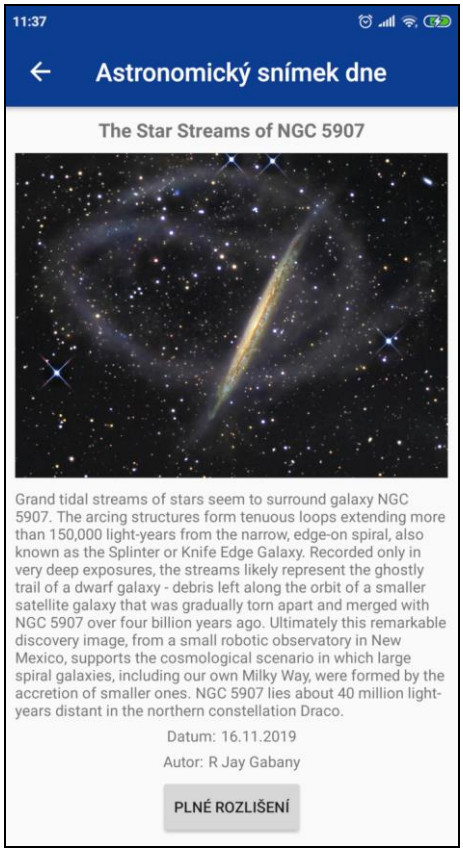
Zdroj: autor

4.3.2 Astronomický snímek dne

Na obrázku 15 je výsledný vzhled obrazovky, která byla navrhnuta pomocí wireframe na obrázku 6, v kapitole 4.1.2.

Každý den se v aplikaci zobrazí nový snímek, může se jednat o fotografii nebo video. Uživatel má možnost si fotografii zobrazit v plném rozlišení po stisku tlačítka, pro přehrání videa je přesměrován do aplikace YouTube.

Obrázek 15 - Výsledný vzhled - Astronomický snímek dne



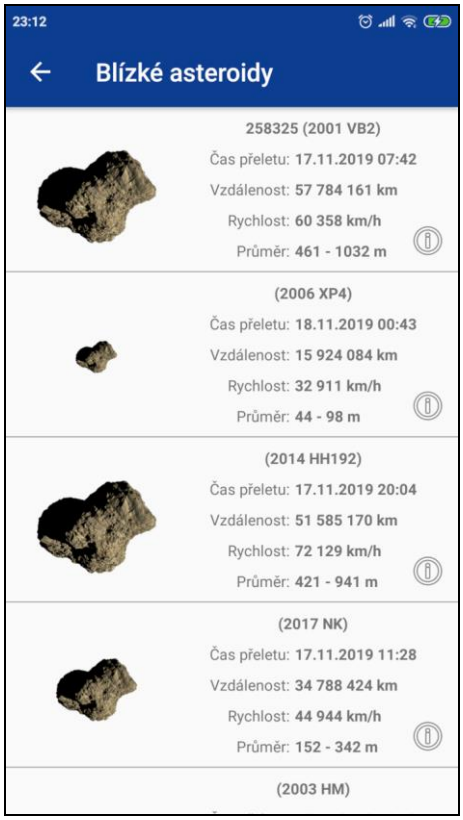
Zdroj: autor

4.3.3 Blízké asteroidy

Na obrázku 16 je výsledný vzhled obrazovky, která byla navrhnuta pomocí wireframe na obrázku 7, v kapitole 4.1.3.

Obrazovka nabízí seznam všech vesmírných těles, které se svou trajektorií letu přiblíží Zemi. Kliknutím na ikonku informace je uživatel přesměrován na webovou stránku s podrobnými informacemi o asteroidu na webu NASA Jet Propulsion Laboratory.

Obrázek 16 - Výsledný vzhled - Blízké asteroidy



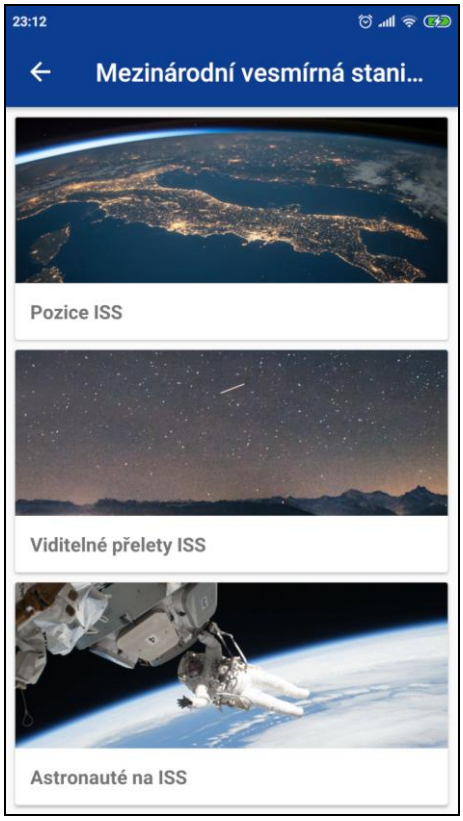
Zdroj: autor

4.3.4 Mezinárodní vesmírná stanice

Na obrázku 17 je výsledný vzhled obrazovky, která byla navrhnuta pomocí wireframe na obrázku 8, v kapitole 4.1.4.

Podmenu, na které se uživatel dostane přes položku z hlavního menu. Při prvním přístupu na funkci Viditelné přelety ISS je uživatel dotázán na povolení lokace, pokud již tak neučinil.

Obrázek 17 - Výsledný vzhled - Mezinárodní vesmírná stanice



Zdroj: autor

4.3.5 Pozice ISS

Na obrázku 18 je výsledný vzhled obrazovky, která byla navrhnuta pomocí wireframe na obrázku 9, v kapitole 4.1.5.

Na obrazovce se zobrazuje aktuální poloha vesmírné stanice vůči Zemi, která je aktualizována každou vteřinu.

Obrázek 18 - Výsledný vzhled - Pozice ISS

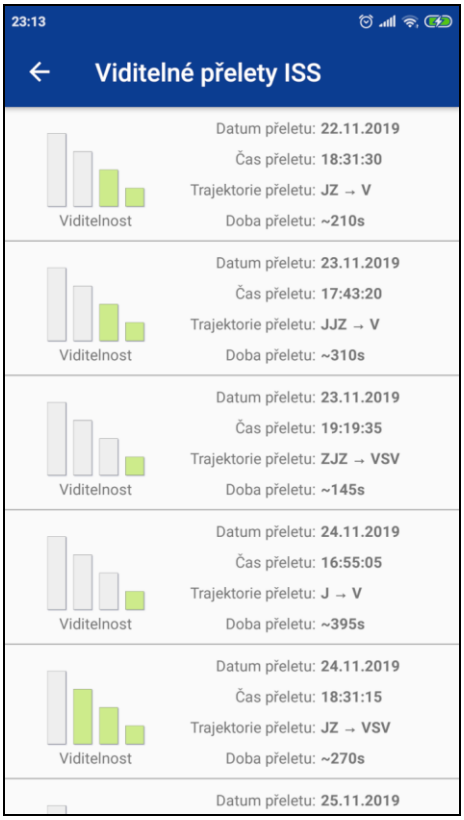
Zdroj: autor

4.3.6 Viditelné přelety ISS

Na obrázku 19 je výsledný vzhled obrazovky, která byla navrhnuta pomocí wireframe na obrázku 10, v kapitole 4.1.6.

Při vstupu na obrazovku je použita poloha zařízení, na základně které jsou zobrazeny výsledky sledovatelných přeletů stanice. Možnost sledování je podmíněno jasnou oblohou.

Obrázek 19 - Výsledný vzhled - Viditelné přelety ISS



Zdroj: autor

4.3.7 Astronauté na ISS

Na obrázku 20 je výsledný vzhled obrazovky, která byla navrhnuta pomocí wireframe na obrázku 11, v kapitole 4.1.7.

Jednotlivé položky v seznamu slouží zároveň jako odkazy. Po kliknutí na některé jméno je uživatel přesměrován do internetového prohlížeče na stránku konkrétní osoby na webu Wikipedie.

Obrázek 20 - Výsledný vzhled - Astronauté na ISS

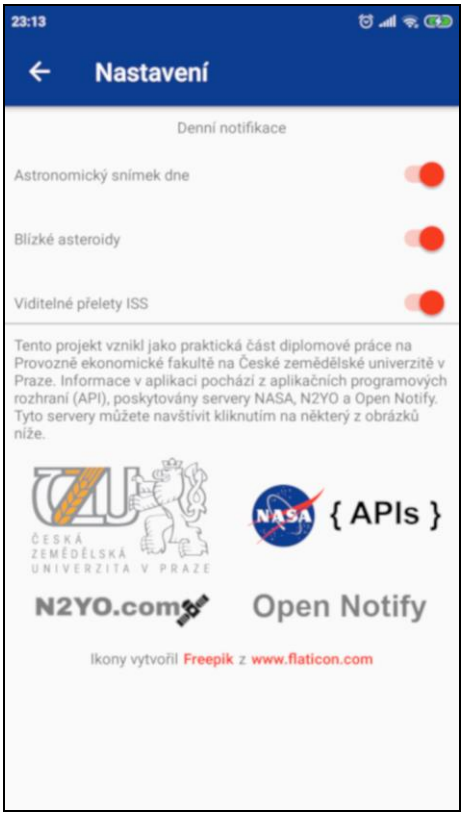
Zdroj: autor

4.3.8 Nastavení

Na obrázku 21 je výsledný vzhled obrazovky, která byla navrhnuta pomocí wireframe na obrázku 12, v kapitole 4.1.9.

Na obrazovce má uživatel možnost vypnutí či zapnutí notifikací, a to i pro konkrétní typ. Dále má možnost přejít na webové stránky univerzity či poskytovatelů použitých zdrojů a to kliknutím na obrázek webu.

Obrázek 21 - Výsledný vzhled - Nastavení



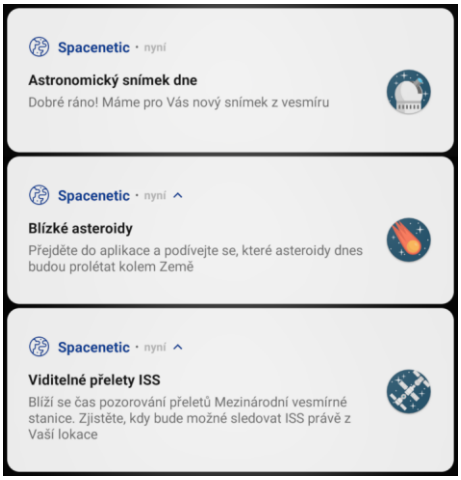
Zdroj: autor

4.3.9 Notifikace

Na obrázku 22 je výsledný vzhled notifikací, které byly navrhnuty pomocí wireframe na obrázku 13 v kapitole 4.1.10.

Spuštění notifikací je naplánováno na dopoledne, poledne a odpoledne, záleží na typu. Při stisknutí notifikace v notifikační liště je spuštěna aplikace a uživatel přesměrován na konkrétní obrazovku.

Obrázek 22 - Výsledný vzhled - Notifikace



Zdroj: autor

5 Výsledky a diskuse

5.1 Komplikace při vývoji

Při implementaci aplikace jsem se setkal s různými komplikacemi. Jedna z častějších souvisela s novostí jazyka Kotlin. Ta se projevovala hlavně při vymýšlení vlastního řešení, kdy jsem v některých případech měl problémy se správnou implementací, která by přesně splňovala chování aplikace, které jsem si stanovil. Zde je nejvíce pocítit rozdíl mezi jazyky Kotlin a Java, Java jakožto dlouholetý předchůdce nového jazyka disponuje daleko širším spektrem již řešených problémů. Mnoho takových problému jsem například řešil prvotním nastudováním příslušné logiky v jazyku Java a poté navrhl a sestrojil vlastní řešení v Kotlinu. Další faktor, se kterým jsem se musel potýkat, vyplývá přímo ze způsobu zpracování řešení, konkrétně používání různých komponent Android frameworku. Tyto komponenty se v čase stále rozvíjí, platformní knihovny a třídy jsou aktualizovány a s tím se mění i požadovaný způsob práce s nimi. Několikrát jsem se tedy v projektu setkal se situací, kdy jsem se k již vyřešenému dílčímu problému musel znovu vracet a řešení přepracovat. Svou roli sehrála také má vlastní zkušenost s novým jazykem, před započetím práce na projektu byly mé praktické zkušenosti zakotvené hlavně v jazyce Java, o Kotlinu jsem měl pouze omezený teoretický přehled.

5.2 Testování

Zpětná vazba od uživatelů byla ve většině případů kladná. V některých případech jsem obdržel náměty na vylepšení či přidání některých funkcí. Například funkce, zobrazující blízké přelety asteroidů by mohla být doplněna vyhledávačem těles v databázi NASA, díky které by uživatel měl možnost si vyhledat informace o asteroidu, který není právě na obrazovce zobrazen. Další podněty se týkaly možnosti změny času notifikace uživatelem na obrazovce nastavení, ukládání astronomických snímků v minulosti či označení konkrétních přeletů ISS a naplánování notifikace pro tuto událost a další.

5.3 Další rozšíření

Aplikaci bych se rozhodně chtěl věnovat nadále, jelikož možnosti rozšíření jsou v tomto projektu velmi otevřené. Rozhodně bych rád implementoval některé zajímavé

nápady dosavadních uživatelů aplikace. Ze své iniciativy bych rád také prozkoumal možnosti integrace sociálních sítí, možnosti personalizace obsahu, využití widgetů na plochu, či například rozšířenou realitu. Programové rozhraní společnosti NASA, které jsem v tomto projektu využil, nabízí mnoho dalších zajímavých zdrojů informací, již bych rád využil. Mezi ně patří například pořizování aktuálních fotografií z pojízdných sond na povrchu planety Mars. Jako příležitosti ke zlepšení stávající aplikace vnímám optimalizaci kódu, zabezpečení API klíčů apod.

5.4 Publikování aplikace v aplikačním obchodu

Finální aplikace byla odeslána, schválena a publikována v aplikačním obchodu Google Play pod názvem Spacenetic a je dostupná zdarma ke stažení na adrese https://play.google.com/store/apps/details?id=cz.czu.pef.spaceboy.

Obrázek 23 - QR odkaz na aplikaci v obchodě

Zdroj: autor podle http://goqr.me/

6 Závěr

Cílem práce bylo představení platformy Android a nového programovacího jazyka Kotlin, dále návrh a vývoj aplikace pro chytrá mobilní zařízení.

V rešeršní teoretické části jsem čtenáře stručně seznámil s historií operačního systému Android. Popsal jsem architekturu operačního systému a její součásti. V následující kapitole jsem charakterizoval komponenty, které tvoří funkcionalitu aplikací. V kapitole, věnované jazyku Kotlin, jsem opět nejdříve shrnul krátkou historii jazyka. Dále jsem charakterizoval výhody a nevýhody, které jsou dle mého názoru těmi nejvíce relevantními. Nadto jsem demonstroval rozdíly ve složitosti syntaxe mezi jazyky Java a Kotlin pomocí ukázek implementace stejné logiky v obou těchto jazycích. Obecně jsem se při poukazování na hlavní přednosti či nedostatky snažil zdůraznit právě ty, které se mi jevily jako nejvíce zajímavé a užitečné pro čtenáře, který by měl zájem se jazyk začít učit. Konec teoretické části jsem věnoval popisu technologií API a JSON, které byly taktéž využity při zpracování praktické části práce.

V praktické části jsem nejprve obecně vysvětlil funkci navrhované aplikace, následně rozložil návrh do jednotlivých obrazovek. Ke každé obrazovce jsem definoval user stories, navrhl logické uspořádání prvků pomocí drátových modelů. Ovládacím prvkům jsem definoval funkci a textovým prvkům přiřadil texty k zobrazení. K obrazovkám, jejichž účel spočívá v prezentaci informací, jsem připojil ukázku JSON souboru, ze kterého aplikace zpracovává požadovaná data. Způsob implementace jsem demonstroval přiložením úseků zdrojového kódu, které považuji za podstatné a stručně vysvětlil princip jejich funkce. Pro porovnání s navrženým designem ve specifikaci jsem nakonec zahrnul otisky obrazovek aplikace v konečné podobě.

Ve výsledcích jsem zhodnotil proces implementace a zmínil některé komplikace, se kterými jsem se setkal. Následně jsem vyzdvihl podněty, které vzešly z testování prvními uživateli a zvážil další možnosti vývoje aplikace v budoucnu.

Tato práce by mohla pomoci těm, kteří se zajímají o programování a chtěli by se případně začít věnovat vývoji na této platformě. Pro zjištění základních informací o platformě poslouží rešeršní část, popis jazykového nástroje a jeho ukázky nastíní jeho praktické použití. Pro mě bylo zpracování práce velmi přínosné, vzhledem k informacím, získaných při nastudování problematiky a zkušenostem, nabitých během praktické části. Na tyto zkušenosti mohu dále navazovat a zdokonalovat své znalosti v oboru.

7 Seznam použitých zdrojů

1. The history of Android OS: its name, origin and more. *Android Authority* [online]. [cit. 2019-10-19]. Dostupné z: https://www.androidauthority.com/history-android-os-name-789433/
2. LACKO, Ľuboslav, 2017. *Mistrovství - Android*. Přeložil Martin HERODEK. Brno: Computer Press. Mistrovství. ISBN 9788025148754.
3. Android's evolution through the years. *GSMArena* [online]. [cit. 2019-10-20]. Dostupné z: https://www.gsmarena.com/google_android_through_the_years-news-31123.php
4. Android 1.0. *Android Wiki | FANDOM* [online]. [cit. 2019-10-21]. Dostupné z: https://android.fandom.com/wiki/Android_1.0
5. Android - Nougat. *Android* [online]. [cit. 2019-10-22]. Dostupné z: https://www.android.com/intl/en_uk/versions/nougat-7-0/
6. Android - 8.0 Oreo. *Android* [online]. [cit. 2019-10-22]. Dostupné z: https://www.android.com/versions/oreo-8-0/
7. What is Project Treble? The Android upgrade fix explained. *Computerworld* [online]. [cit. 2019-10-23]. Dostupné z: https://www.computerworld.com/article/3306443/what-is-project-treble-android-upgrade-fix-explained.html
8. Android 9 Pie. *Android* [online]. [cit. 2019-10-22]. Dostupné z: https://www.android.com/versions/pie-9-0/
9. Android 10 | Android. *Android* [online]. [cit. 2019-10-22]. Dostupné z: https://www.android.com/android-10/
10. Android Runtime (ART) and Dalvik. *Android Open Source Project* [online]. [cit. 2019-10-28]. Dostupné z: https://source.android.com/devices/tech/dalvik
11. Lekce 1 - Úvod do jazyka Kotlin, platformy a IntelliJ. *ITnetwork* [online]. [cit. 2019-11-16]. Dostupné z: https://www.itnetwork.cz/kotlin/zaklady/uvod-do-jazyka-kotlin-platformy-a-intellij
12. Using Kotlin for Android Development. *Kotlin Programming Language* [online]. [cit. 2019-11-16]. Dostupné z: https://kotlinlang.org/docs/reference/android-overview.html

13. Kotlin vs. Java: Which One You Should Choose for Your Next Android App. *Netguru* [online]. [cit. 2019-11-16]. Dostupné z: https://www.netguru.com/blog/kotlin-java-which-one-you-should-choose-for-your-next-android-app
14. Multiplatform Programming. *Kotlin Programming Language* [online]. [cit. 2019-11-16]. Dostupné z: https://kotlinlang.org/docs/reference/multiplatform.html
15. Kotlin - 10 Důvodů, proč ho mám rád. *eMan* [online]. [cit. 2019-11-16]. Dostupné z: https://www.eman.cz/blog/kotlin-10-duvodu-proc-mam-rad/
16. Kotlin VS Java: Basic Syntax Differences. *Yalantis* [online]. [cit. 2019-11-17]. Dostupné z: https://yalantis.com/blog/kotlin-vs-java-syntax/
17. Toppling the Giant: Kotlin vs. Java. *Scott Logic / Altogether Smarter* [online]. [cit. 2019-11-17]. Dostupné z: https://blog.scottlogic.com/2019/04/29/kotlin-vs-java.html
18. What is API: Definition, Types, Specifications, Documentation. *Altexsoft* [online]. [cit. 2019-11-17]. Dostupné z: https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/
19. What exactly IS an API? *Medium* [online]. [cit. 2019-11-17]. Dostupné z: https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f
20. Introducing JSON. *JSON* [online]. [cit. 2019-11-17]. Dostupné z: http://www.json.org/
21. OkHttp. *Square Open Source* [online]. [cit. 2019-11-19]. Dostupné z: https://square.github.io/okhttp/
22. Google/Gson: A Java serialization/deserialization library to convert Java Objects into JSON and back. *GitHub* [online]. [cit. 2019-11-19]. Dostupné z: https://github.com/google/gson
23. Picasso. *Square Open Source* [online]. [cit. 2019-11-19]. Dostupné z: https://square.github.io/picasso/
24. YouTube Android Player API. *Google Developers* [online]. [cit. 2019-11-19]. Dostupné z: https://developers.google.com/youtube/android/player

8 Přílohy

8.1 Příloha A: CD se zdrojovým kódem aplikace