Univerzita Pardubice
Fakulta elektrotechniky a informatiky

# Geolokační mobilní aplikace interpretující historické fotografie bodů zájmu

František Brýl

Bakalářská práce
2021

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2020/2021

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **František Brýl** |
| Osobní číslo: | **I18107** |
| Studijní program: | **B2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Téma práce: | **Geolokační mobilní aplikace interpretující historické fotografie bodů zájmu** |
| Zadávající katedra: | **Katedra informačních technologií** |

## Zásady pro vypracování

Cílem bakalářské práce bude v teoretické části charakterizovat a popsat existující geolokační technologie (včetně dvoupásmových GNSS – GPS a GALILEO) a analyzovat jejich využitelnost v mobilních aplikacích. Součástí práce bude i přehled lokačních a bezdrátových technologií a srovnání jejich výhod a nevýhod.
Druhým cílem v praktické části bakalářské práce bude navrhnout a realizovat mobilní aplikaci pro operační systém Android. Vlastní aplikace bude využívat databázi, geolokační technologii GPS a GALILEO a fotoaparát mobilního zařízení. Aplikace bude uživateli na základě jeho polohy na které se právě nachází poskytovat z databáze historickou fotografii místa využívající geotagging. Tu zobrazí na mapových podkladech a umožní uživateli historickou fotografii prohlédnout, vyfotit si dané místo pomocí fotoaparátu a následně pořízenou fotografii porovnat s historickou variantou. Poté může získané srovnání obou fotografií uložit či sdílet.

Rozsah pracovní zprávy: **35**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:


- LACKO, Luboslav. *Vývoj aplikací pro Android*. 1. vyd. Brno: Computer Press, 2015. ISBN 9788025143476. VÁVRŮ, Jiří a UJBÁNYAI, Miroslav. *Programujeme pro Android*. Praha: Grada, 2013. ISBN 9788024748634. KARIMI, Hassan A. *Advanced location-based technologies and services*. Boca Raton: CRC Press, 2013. ISBN 9781466518186.
- GENTILE, Camillo, Nayef ALSINDI, Ronald RAULEFS a Carole TEOLIS. *Geolocation techniques principles and applications*. Springer, 2013. ISBN 9781461418368
- ALLEN, Grant a OWENS, Michael. *The definitive guide to SQLite*. 2nd ed. New York: Apress, 2010. The expert's voice in open source. Books for professionals by professionals. ISBN 9781430232254.


Vedoucí bakalářské práce: **Ing. Jiří Kysela, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2020**
Termín odevzdání bakalářské práce: **14. května 2021**

L.S.

**Ing. Zdeněk Němec, Ph.D.** v.r.
děkan

**Ing. Jan Panuš, Ph.D.** v.r.
vedoucí katedry

V Pardubicích dne 26. února 2021

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 8. 5. 2021

František Brýl

# PODĚKOVÁNÍ

Chtěl bych poděkovat své rodině a svým blízkým, kteří mě podporovali po celou dobu mého studia. Dále bych rád poděkoval panu Ing. Jiřímu Kyselovi Ph.D., vedoucímu mé bakalářské práce, za pomoc při vypracovávání této práce. Na závěr bych rád poděkoval všem, kteří mi pomáhali při sběru historických fotografií včetně zhotovení jejich důvěryhodného popisu.

**ANOTACE**

Cílem této bakalářské práce je vytvoření mobilní aplikaci pro operační systém Android. Aplikace bude uživateli na základě jeho polohy, na které se právě nachází, poskytovat z databáze historickou fotografii místa využívající geotagging. Fotografii následně zobrazí na mapových podkladech a umožní uživateli historickou fotografii prohlédnout, vyfotit si dané místo pomocí fotoaparátu a následně pořízenou fotografii porovnat s historickou variantou. Poté může uživatel získané srovnání obou fotografií uložit či sdílet.

**KLÍČOVÁ SLOVA**

Android, geolokace, GPS, mobilní aplikace, fotografování, databáze

**TITLE**

Geolocation mobile application interpreting historical photographs of points of interest

**ANNOTATION**

The aim of this bachelor thesis is to create a mobile application for the Android operating system. Based on user's location, the application will provide the user a historical photo of the place, where user is currently located, from the database using geotagging. Then, the photo is displayed on the map. User is allowed to view the historical photo, take a picture of the place with a camera and compare the taken photo with the historical variant. In the end, the user can save or share the comparison of these two photos.

**KEYWORDS**

Android, geolocation, GPS, mobile application, photography, database

# OBSAH

# SEZNAM ILUSTRACÍ

# SEZNAM ZKRATEK

| | |
|---|---|
| PDF | Portable Document Format |
| API | Application Programming Interface |
| GPS | Global Positioning Systém |
| GSA | European GNSS Agency |
| MEOSAR | Medium Earth Orbiting Search and Rescue |
| GL | Graphics Library |
| IEEE | Institute of Electrical and Electronics Engineers |
| FFD | Full-Function Device |
| RFD | Reduced-Function Device |
| UWD | Ultra Wide Band |
| WiMAX | WorldWide Interoperability for Microwave Acces |
| JSON | JavaScript Object Notation |
| IDE | Integrated Development Environment |
| JDK | Java Development Kit |
| JRE | Java Runtime Environment |
| USB | Universal Serial Bus |
| APK | Android Application Package |
| SDK | Software Development Kit |
| NDK | Native Development Kit |
| XML | Extensible Markup Language |
| MB | Megabyte |
| JPEG | Joint Photographic Experts Group |
| URI | Uniform Resource Identifier |
| RAM | Random Access Memory |

# ÚVOD

Hlavním cílem bakalářské práce je návrh a realizace mobilní aplikace pro operační systém Android. Realizovaná aplikace poskytne uživateli na základě jeho aktuální zeměpisné polohy historickou fotografii místa. Historická fotografie se načte z databáze, ve které jsou umístěny fotografie i s popisem místa v době jejich vzniku. Na displeji mobilního zařízení je v jedné polovině zobrazena historická verze daného místa a ve druhé polovině si uživatel může prohlédnout lokalitu prostřednictvím fotoaparátu. Následně si uživatel může místo vyfotografovat a fotografii uložit i ve verzi s paralelním vyobrazením historické podoby daného prostoru.

Kapitola první se zaobírá teoretickou částí vývoje aplikace. Jsou zde podrobně rozebrány požadavky na aplikaci a geolokační metody k určování polohy. Hlouběji jsou analyzovány především metody GPS a GALILEO a je provedeno jejich srovnání. Dále jsou v této části práce představeny mapové podklady, které aplikace využívá. V závěru kapitoly je uveden seznam lokačních a bezdrátových technologií s jejich popisem a využitím v aplikaci.

Druhá kapitola se věnuje praktické části vývoje, tedy samotné realizaci aplikace. Nejdříve je prezentována použitá databáze a uložiště, se kterým aplikace komunikuje. Poté je představena vyvíjená aplikace. Rozebírá se zde použité vývojové prostředí, struktura aplikace a ukazují se použité technologie s podrobným výkladem. Každá třída v aplikaci je konkrétně popsána a je objasněno její použití. Kapitola také obsahuje ukázky z uživatelského rozhrání s detailními vysvětlivkami. V neposlední řadě jsou v kapitole zmíněny využívané API v aplikaci. Podrobněji se hovoří především o Mapbox API a Camera2 API. V závěru této části práce je vysvětleno, jak probíhá testování hotové aplikace.

Poslední kapitola pojednává o distribuci aplikace mezi ostatní uživatele. Důraz je kladen převážně na umístění aplikace do obchodu s aplikacemi Google Play.

Jedním z důvodů, proč jsem si vybral toto téma, je můj dlouhodobý zájem o historii a cestování. Rád objevuji nová místa, přičemž si často kladu otázku: „Jak to tady asi vypadalo před 100 lety?“ Věřím, že si tuto otázku nepokládám na cestách pouze já, ale i mnoho dalších lidí. Rozhodl jsem se proto k realizaci mé myšlenky spojení historie se současností ve fotografii prostřednictvím této práce.

# 1 NÁVRH MOBILNÍ APLIKACE

Návrh aplikace se zabývá teoretickým popisem použitých technologií včetně definovaní požadavků na aplikaci. Úvodem jsou rozebrány funkční a nefunkční požadavky na aplikace, které byly definovány před samotnou realizací aplikace. Další části kapitoly se zaměřuje na popis geolokačních metod a mapových podkladů, jež jsou důležitým prvkem realizované aplikace. V závěru se prezentují bezdrátové technologie včetně jejich srovnání a využití.

## 1.1 Požadavky na aplikaci

Hlavním cílem v návrhu aplikace je si stanovit základní požadavky pro pozdější realizaci aplikace. Tyto požadavky určují očekávání od systému vzhledem k požadovaným cílům. V základu se dají požadavky rozdělit na funkční a nefunkční. Funkční požadavky definují nezbytné úkoly a akce, které by měla aplikace bezprostředně vykonávat. Nefunkční požadavky definují omezení implementace a obecné požadavky na architekturu aplikace, uživatelské rozhraní a prostředí aplikace. [1]

### 1.1.1 Funkční požadavky

Aplikace bude umožňovat vykonávat tyto funkcionality:

- pořizování fotografií pomocí fotoaparátu,
- zobrazení aktuální polohy uživatele,
- pohybování v aplikaci pomocí tlačítek,
- zobrazení historické fotografie na mapě pomocí geotaggingu,
- navigování pomocí metody GPS,
- podporování načítacích dialogů,
- ošetřování vyvolaných výjimek při nedostupnosti připojení k síti,
- ošetřovaní vyvolaných výjimek při nedostupnosti služeb sloužících k určování polohy,
- ukládání pořízené fotografie do uložiště zařízení ve více rozlišeních,
- načítání a zobrazování historických míst v uživatelově okolí,
- vybrání stylu fotografování,
- detailní prohlížení historických fotografií daných lokalit.

### 1.1.2 Nefunkční požadavky

V rámci nefunkčích požadavků navrhovaná aplikace:

- Bude naprogramovaná v jazyce Java.
- Bude spustitelná na zařízeních s OS Android 4.4 a vyšší.
- Bude dostupná v českém jazyce.
- Bude získávat data z cloudové databáze.

## 1.2 Geolokační metody aplikace

V této kapitole budou prezentovány vybrané geolokační metody pro navrhovanou aplikaci. V první části se pozornost zaměří na charakteristiku systému GPS včetně popisu jeho vývoje a komponent. Další část je věnována projektu Galileo a na závěr proběhne srovnání obou služeb včetně popisu jejich využitelnosti v mobilních aplikacích.

### 1.2.1 GPS

GPS je satelitní navigační systém pod správou USA. Vznikal v období studené války a byl původně určen pouze k vojenským účelům. V roce 1983 avšak americký prezident Ronald Reagan rozhodl o částečném zpřístupnění systému běžným uživatelům. To mělo za následek rychlé rozšíření GPS do celého světa. Dnes o jeho četném využití v turistice a dopravě nemůže být pochyb. První prototypové družice byly do vesmíru vypuštěny mezi lety 1978 a 1985. V roce 2000 došlo z podnětu tehdejšího amerického prezidenta Billa Clintona k vypnutí služby *Selective Avalability,* díky které měli neautorizovaní uživatelé podstatně horší přesnost systému. Vypnutí této služby mělo za následek zvýšení přesnosti určování polohy ze 100 metrů na 20 metrů. V současnosti disponuje GPS přesností 3 metrů. [2], [3]

Systém GPS tvoří 3 segmenty – kosmický, řídící a uživatelský: [2], [4]

- Segment kosmický původně obsahoval 24 navigačních družic, jejichž dráhy jsou víceméně kruhové. Oběžná doba takové družice je polovina našeho dne. Družice jsou rozmístěny tak, aby bylo po celou dobu nad obzorem minimálně šest družic. Nyní již segment obsahuje nejméně 32 aktivních družic, což má za následek větší přesnost a dostupnost systému.
- Řídící segment má na starosti měření oběžné dráhy, udržování GPS času a monitorování funkčnosti družic a udržování jejich stavu. Tento segment umí také předpovídat dráhu

družice, či dokonce počítat a nahrávat navigační zprávu. V případě nouze je segment nastaven tak, aby uměl přemístit družici na jinou pozici nebo s ní uměl manévrovat.

- Poslední, uživatelský segment se skládá z GPS přijímačů uživatelů. Segment má kvůli bezpečnosti pouze pasivní roli. Protože tyto přijímače nekomunikují přímo s družicemi, je systém schopen obsluhovat neomezený počet uživatelů.

### 1.2.2 GALILEO

Projekt Galileo je výsledkem snahy Evropské unie o sestrojení plnohodnotného navigačního systému. Na rozdíl od systému GPS, byl Galileo od počátku vývoje vytvářen jako systém pro běžné civilní uživatele a byl financován od soukromých investorů. Systém je od roku 2004 pod správou GSA, která od roku 2012 sídlí v Praze. Kvůli postupné ztrátě zájmu od investorů, byla Evropská unie nucena financovat projekt ze svých zdrojů. První navigační družice se začaly vypouštět začátkem roku 2011. Konečná konstelace má obsahovat celkem 30 družic rozmístěných ve třech oběžných rovinách. V současné době obíhá naši Zemi 22 funkčních družic a další satelity mají být vypuštěny do konce roku 2024. [2]

Systém Galileo poskytuje uživatelům čtyři základní typy navigačních služeb: [2], [3]

- Služba OS (Galileo Open Service) je základní služba pro běžné uživatele poskytovaná na jednom kmitočtu E1 nebo na dvou kmitočtech E1 a E5.
- Další služba SOL (Galileo Safety of Life Service) je podobná základní službě OS s rozšířením o varování už ivatele, že přesnost nebo integrita systému není zaručena.
- Služba HAS (High Accuracy Service) je zaměřena na aplikace pro profesionální nebo komerční užití, které vyžadují vyšší výkon než základní služby. Tato služba běží na jiném kmitočtu než předešlé dvě služby, konkrétně na kmitočtu E6.
- Poslední služba PRS (Galileo Public Regulated Service) je určena pouze pro uživatele schválenými vládami států patřící do Evropské unie. Služba nabízí technologie proti rušení signálu a slouží k zabezpečení strategických odvětví.

Všechny navigační družice systému Galileo jsou vybaveny transpondérem MEOSAR, který slouží pro pátrání a záchranu lidí na Zemi. Tato služba umožňuje lokalizovat lidi v nouzi a udržovat s nimi oboustranné spojení. [2]

### 1.2.3 Využitelnost v mobilních aplikacích

Jelikož je systém GPS o dost starší než systém Galileo, téměř všechny chytré mobilní telefony ho podporují. Galileo v současnosti zavádí do svých čipů přes 30 světových společností. Mezi ně se mohou zařadit např. společnosti Samsung či Intel. Systém Galileo najdeme například u všech mobilů iPhone značky Apple od modelu 6S a novější. Dále jej můžeme nalézt u řady chytrých telefonů značek Samsung, Huawei, Nokia či Sony. Nutno podoknout, že se jedná spíše o dražší modely telefonů, tudíž v běžných, levnějších telefonů se systému Galileo zatím nedočkáme. Na stránkách samotných vývojářů lze nalézt oficiální seznam všech mobilních zařízeních, které Galileo podporují. Seznam je neustále pravidelně aktualizován o nové mobilní zařízení. Technologie Galileo je navíc kompatibilní nejen s GPS, ale i dalšími polohovacími systémy jako např. ruský GLONASS nebo čínský BeiDou. Není tedy nutné v mobilním zařízení nic nastavovat či přepínat, nýbrž se používají všechny technologie vždy najednou. [4], [5]

Mezi největší výhody Galilea oproti GPS je přesnost a rychlost, s jakou systém umí zaměřit zařízení na Zemi. Nejpřesnější služby zvládnou vypočítat přesnost polohy s odchylkou pár centimetrů. Navíc lze obzvláště v Evropě získat signál i v oblastech, kde u jiných navigačních systémů nastávají problémy s připojením. [5]

U systému GPS jsou mobilní zařízení, jak již bylo zmíněno, pouze pasivní, tedy zvládnou pouze přijímat signál, ale nedokážou jej zpět odesílat a držet si oboustrané spojení. Samotné připojení je bezplatné, není vázané na internet a mobilní sítě, jak si mnoho lidí může mylně myslet. V mobilních aplikacích je systém GPS využíván v široké sféře odvětví, po navigační aplikace až po aplikace k zjišťování počasí, kde GPS zjistí uživatelovu polohu a ihned nabídne předpověď v jeho oblasti. [2]

Navigační systém Galileo je, co se týká přesnosti určení polohy, preciznější, poněvadž dokáže určit pozici s tolerancí do jednoho metru, na rozdíl od navigačního systému GPS, který zvládne lokalizovat polohu s odchylkou tři metry. Každopádně jak Galileo, tak i GPS patří mezi nejpřesnější technologie, které si lidé na Zemi mohou dovolit používat. [2]

## 1.3 Mapové podklady aplikace

Navrhovaná aplikace bude jako mapové podklady využívat mapy od společnosti Mapbox, která poskytuje své mapy pro vývojáře a designéry prostřednictvím veřejné knihovny Maps SDK. V současnosti společnost podporuje mapy a lokalizační služby pro širokou škálu webových, mobilních, automobilových a herních aplikací. Firma také poskytuje editor Mapbox Studio, což je bezplatný editor stylů map. [6]

Mapová data společnosti jsou získávána a zpracována pomocí mobilních senzorů, leteckých snímků a zpětné vazby řidičů. Dopravní data se používají k aktualizaci provozních informací silničních tras na celém světě každých několik minut, což umožňuje výkonné navigační funkce na mapě, jako je například přesměrování trasy kvůli dopravním nehodám. [7]

Ukládání a poskytování většiny mapových dat je realizováno pomocí formátu tzv. vektorových obrazů (dlaždic). Tento formát je určen pro škálování a ukládání map do mezipaměti aplikace. Vektorové obrazy obsahují metadata a geometrické útvary, které lze vykreslit na mapě. [7]

Sada knihoven Mapbox GL načítá prostorová data z vektorových obrazů a styl map ze stylového dokumentu a pomáhá klientovi vykreslit 2D a 3D mapy s pomocí nástroje OpenGL. Mapbox GL se opírá o dva dokumenty se specifikacemi, které definují standardy umožňující správně instruovat klienty: [7]

- Vector Tile Specification – Veřejně přístupná specifikace, která popisuje, jak jsou geometrické útvary a další vlastnosti v prostorových datech uloženy a zakódovány ve vektorových obrazech.
- Style Specification – Jedná se o specifikací popisující způsob, jakým je potřeba zapsat styl mapy k vykreslení včetně barvy, úrovně krytí a dalších vlastností.

**Mapbox Maps SDK**

Knihovna nabízí interaktivní a upravitelné mapy. Mapy jsou na zařízení zobrazovány v určitých stylech. Tyto styly a jejich správa se nachází v Mapbox API, která bude popsána v kapitole o využívaných API. Vybraný styl je aplikován na vektorové obrazy, které jsou vykreslovány pomocí standardu OpenGL. Sada je k dispozici pro zařízení s operačním systémem iOS a systémem Android verze 4.4 a vyšší. [7]

K využití těchto map ve vývoji aplikací je potřeba se zaregistrovat na oficiálních stránkách společnosti Mapbox. Po vytvoření účtu vývojář získá přístupový token, který bude v aplikaci využívat k práci s produkty Mapbox. [8]

## 1.4 Lokační a bezdrátové technologie

Bezdrátová technologie se stala velmi populární v oblasti sítí a počítačové komunikace. Rychlý růst používání mobilních telefonů způsobil četné používání různých satelitních služeb a později i bezdrátových technologií. U těchto technologií není potřeba instalovat rozvody nebo kabeláž, komunikace je zprostředkována pomocí frekvenčních pásem. Bezdrátové technologie lze rozdělit do čtyř síťových skupin podle jejich použití a dosahu signálu: [9]

- WPAN – Bezdrátové personální sítě, které jsou založeny na standardu IEEE 802.15. Komunikaci prostřednictvím této sítě lze praktikovat jen na krátkou vzdálenost (v jednotkách několika metrů). Přenos pomocí těchto sítí je energeticky nenáročný, jeho rychlost však není příliš vysoká. Protože tyto sítě nevyžadují k externímu připojení k jiným sítím složitou infrastrukturu, jejich použití je nenákladné a velmi komplexní.
    - Blueetoth – Jedná se o technologii, která je tvořena z množství pikosití. Umožňuje propojit až 8 aktivních zařízení pomocí modelu Master/Slave. První připojení zařízení v pikositi je označováno jako *Master*, ostatní jsou typu *Slave*. Propojení skrze tuto technologii jsou kódována a zabezpečena proti odposlechu a rušení. Klasický rozsah signálu dosahuje asi 10 metrů, za určitých okolností však může mít dosah až na vzdálenost 100 metrů od zařízení.
    - IrDa – Tato technologie představuje soubor infračervených standardů určených ke komunikaci. Slouží vytváří jednosměrné spojení mezi dvěmi zařízeními, například mezi notebookem a tiskárnou. Technologie disponuje nízkou spotřebou, dosahem až 1 metr s rychlostí přenosu v řádech několika jednotek megabitů za sekundu.
    - ZigBee – Slouží k vytvoření bezdrátových sítí s nízkými požadavky na datovou propustnost. Připojené zařízení se dělí na dva druhy – plně funkční zařízení (FFD) a omezené zařízení (RFD), které si můžeme představit jako jednoduchou aplikaci (např. vypínač světla). Oproti tomu FFD pracuje jako směrovač poskytující synchronizaci ostatním zařízením.
    - UWB – Charakteristickou vlastností této technologie je přenos souborů mezi připojenými zařízeními vysokou rychlostí na vzdálenost až několika metrů. Tyto přenosy dosahují rychlosti od 110 Mbit/s do 480 Mbit/s. Dají se použít k propojení multimediálních aplikací či jako náhrada za sběrnici USB.
- WLAN (Wi-Fi sítě) – Jedná se o bezdrátové lokální sítě, které zajišťují přenos dat v rozsahu až 100 metrů. Zakládají si na standardech IEEE 802.11, ve kterém je definována celá řada dalších standardů. Každý standard má své specifické vlastnosti. Například standard IEEE 802.11.b, jenž byl první standardem v této skupině, dokáže podporovat přenosy dat s maximální rychlostí 11 Mbit/s v pásmu 2,4 GHz. Novější standard IEEE 802.11.g na rozdíl od něj disponuje maximální přenosovu rychlostí

54 Mbit/s využívající rozšířené kmitočtové pásmo. Nutno podotknout, že přenosová rychlostí klesá při ztrátě rádiového signálu nebo při připojení více zařízeních.

- WMAN – Bezdrátové metropolitní sítě disponující standardem IEEE 802.16, který bývá často označován jako WiMAX. Tato technologie se zaměřuje na přenos dat pomocí v metropolitních sítích. Technologie WiMAX pokrývá svým signálem výrazně větší území než např. výše zmíněná technologie Wi-Fi. Technologie může běžet ve dvou kmitočtových pásmech, která lze kombinovat za účelem zvýšení přenosové rychlosti. V pásmu s frekvencí od 10 GHz do 66 GHz musí být poskytnuta přímá viditelnost mezi zařízeními, tj. nesmí v cestě signálu stát žádné překážky. Ve druhém pásmu s frekvencí od 2 GHz do 11 GHz lze komunikovat i bez přímé viditelnosti zařízeních. Vzájemnou kombinací těchto dvou pásem lze dosáhnout přenosové rychosti až 70 Mbit/s na vzdálenost až 50 kilometrů.
- WWAN – Tyto rozsáhlé sítě jsou schopny překonat přenosovou vzdálenost i větší než 50 kilometrů používající licencovaná kmitočtová pásma. Tyto technologie se používají k signálovému pokrytí celých měst nebo i států pomocí satelitních a anténních systémů.
    - Satelitní sítě – Technologie využívá satelity, které spolu vzájemně spolupracují. Pomocí těchto satelitů je umožněno pokrýt velkou plochu určitého území. Satelity disponují transpondéry, jejichž součástí jsou antény a vysílače, které přijímají příchozí signál a odesílají ho zpět, ovšem na odlišné frekvenci.
    - Buňkové telefonní sítě – Tento systém tvoří z velké pokryté oblasti menší podčásti (buňky). Ve středu každé buňky je vždy umístěna jedna základnová stanice, která poskytuje propojení koncového zařízení s telefonní ústřednou. Tato technologie je energeticky nenáročná a efektivní. Typy těchto systémů se označují pojmem generace. První generace (1G) sloužila především pro telefonní komunikaci a její přenosová rychlost byla pouhých 2,4 kbit/s. Druhá generace (2G) disponovala rychlostí až 64 kbit/s a umožňovala i přenos SMS zpráv. Třetí generace (3G) nabývala rychlosti přenosu až 2 Mbit/s. Její vylepšená verze (3,5G) dokázala pozvednout tuto rychlost i o několik dalších megabitů. Čtvrtá generace (4G) už dokázala poskytovat přenos dat s rychlostí až 1 Gbit/s a nejnovější pátá generace (5G) přinesla nastavení rychlosti datového přenosu až na teoretických 20 Gbit/s s výrazným snížením odezvy.

# 2 REALIZACE MOBILNÍ APLIKACE

V této kapitole je prezentována realizace navržené aplikace pro operační systém Android. V následujících podkapitolách jsou popsány technologie, které aplikace využívá. Ukázána je zde i struktura aplikace včetně popisu použitého vývojového prostředí. V neposlední řadě kapitola obsahuje i ukázky uživatelského rozhraní a proces, jakým je aplikace testována.

## 2.1 Databáze a uložiště

Databáze v oblasti informačních technologií slouží jako uložiště dat a informací. U mobilních aplikacích obvykle běží na serveru. U některých databází lze ovšem databázový soubor s daty nakonfigurovat přímo do aplikace, bez nutnosti běžícího serveru. V realizované aplikaci se v databázi uchovávají podrobnosti o uložených místech. V případě naimplementování databázového souboru přímo do aplikace nastává problém s obsluhou a velikostí. S přibývajícími daty, zvláště pak s rostoucím počtem uložených historických fotografií v databázi, nabývá i velikost aplikace. Aplikace je postavena na platformě Firebase od společnosti Google, která zdarma poskytuje online uložiště, databázi a mnoho dalších služeb.

**Firebase**

Platforma Firebase vznikla v roce 2011 společností Envolve. V roce 2014 ji odkoupil Google, který ji vlastní dodnes. Podstatou Firebase je automatické synchronizování provedených změn v uložišti či databázi s připojenými zařízeními. Hlavním cílem Firebase je podporovat vývoj mobilních a webových aplikací. Mezi základní pilíře platformy patří cloudová NoSQL databáze, statický hosting, cloudové uložiště a autentizační nástroje k připojení do databáze. Firebase databázi lze rozdělit na dva typy – Realtime Database a Cloud Firestore. Realizovaná aplikace využívá databázi typu Realtime Database a cloudové uložiště. [10], [11]

- Realtime Database – Jedná se o typ databáze, která nabízí ukládání a synchronizaci dat v rámci cloudové NoSQL databáze. Uložená data se automaticky synchronizují napříč všemi klienty v reálném čase. Navíc, načtená data z databáze se uchovávají lokálně a budou klientům dále k dispozici i v případě, že zařízení ztratí připojení k síti. Díky tomu může aplikace zobrazovat jednotlivá historická místa z databáze i bez stálého připojení k internetu. Data se ukládají ve formátu JSON a synchronizují se s každým novým připojeným zařízením. [12]

V databázi, kterou využívá realizovaná aplikace, jsou jednotlivá místa uložena v JSON formátu. Každé místo obsahuje jeho název, polohu, obrázek a popis. Poloha místa se skládá ze dvou údajů – zeměpisné délky (longitude) a zeměpisné šířky (latitude). Jak zeměpisná délka, tak zeměpisná šířka jsou ve formátu JSON uložené jako číselný typ. Historický obrázek místa se zde uchovává jako textový řetězec, který obsahuje odkaz na adresu obrázku uloženého v cloudovém uložišti.

- Cloudové uložiště – Představuje úložný prostor, primárně určený pro mediální obsah, například fotografie či videa. V uložišti lze vytvářet složky či libovolně řadit uložený obsah dle daných kritérii a metadat. Každý uložený soubor v uložišti obsahuje kromě metadat i informace o umístění souboru. Umístění obsahuje jak odkaz na daný soubor ve Firebase uložišti, tak i přístupový klíč, přes který lze získat instanci souboru v databázi nebo přímo v aplikaci. Aplikace využívá uložiště pro ukládání historických fotografií. [13]

**JSON**

JSON (JavaScript Object Notation) je odlehčený textový formát primárně sloužící pro výměnu dat. Lze jej snadno číst i zapisovat. Navíc je lehce analyzovatelný i generovaný počítačem. Jeho struktura vznikla z podmnožiny programovacího jazyka JavaScript. Jako vstup lze použít jakoukoli strukturu, přitom na výstupu se vyskytuje vždy řetězec. Jelikož je formát JSON zcela nezávislý na ostatních jazycích, představuje téměř dokonalý jazyk pro výměnu dat a práci s nimi. [14]

JSON se skládá z následujících struktur: [14]

- Hodnota – Může nabývat několika datových typů. Pod hodnotou si lze představit například číslo, textový řetězec, objekt, pole či hodnoty typu null, true a false. V případě textového řetezce je nutné jej ohraničit uvozovkami. Všechny tyto struktury lze vnořovat.

- Objekt – Reprezentuje neuspořádanou množinu dvojic skládající se z názvu a hodnoty. Začátek objektu je vždy označen levou složenou závorkou („{“) a končí pravou složenou závorkou („}“). Uprostřed objektu se nachází samotná dvojice, popř. seznam dvojic. Ve dvojici je první uváděn název zapsaný v uvozovkách, za kterým následuje dvojtečka a hodnota. V případě výskytu další dvojice je za hodnotou uváděna čárka a teprve poté následuje příslušná dvojice. Jednoduchý objekt v JSON lze zapsat např. takto:

{“place” : ”Pardubice” , “latitude” : 42.123456, “longitude” : 15.654321 }

- Pole – představuje seřazenou kolekci hodnot či objektů. Začátek pole se značí levou hranatou závorkou („[“) a ukončuje se pravou hranatou závorkou („]“). Uvnitř pole se nachází příslušné hodnoty. Jako oddělovač se používá čárka. Syntaxi pole lze ukázat na následujícím příkladu:

  [Pardubice , {“latitude” : 42.123456, “longitude” : 15.654321 }]

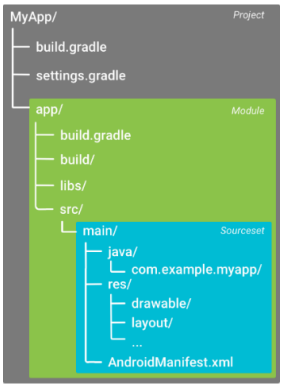
## 2.2 Android aplikace

Tato kapitola se zaměřuje na popis samotné aplikace. V úvodu je popsáno vývojové prostředí, ve kterém byla aplikace vyvíjena, a nástroje potřebné pro samotný vývoj mobilních aplikací v Javě. V následujících podkapitolách je vysvětlena struktura aplikace a projektu, uživatelské rozhraní včetně pohybu v realizované aplikaci, funkcionalita jednotlivých tříd a princip používaných technologií v aplikaci.

### 2.2.1 Vývojové prostředí

Realizace aplikace byla zprostředkována pomocí integrovaného vývojového prostředí (IDE) Android Studio od společnosti Google. Android Studio je založené na prostředí IntelliJ IDEA od firmy JetBrains. Ke spuštení je nutné mít nainstalovanou Javu a sadu JDK od společnosti Oracle obsahující JRE, který je zapotřebí ke spuštění Java aplikací.

K podpoře vývoje aplikací v operačním systému Android se v Android studiu používá buildovací systém založený na nástroji Gradle. Každý projekt obsahuje jeden či více modulů se soubory zdrojového kódu a zdrojovými soubory. Mezi takové moduly lze zařadit moduly knihoven, aplikací a moduly Google App Engine. Všechny soubory určené k sestavení projektu jsou uloženy ve složce Gradle Scripts. Každý modul aplikace obsahuje základní složky manifests, java a res. [15]




**Obrázek 1 Základní struktura projektu v Android Studiu Zdroj: [16]**

**Gradle**

Nástroj Gradle slouží k automatizaci sestavování programu při vývoji softwaru. Stará se o primárně o proces kompilace. Také převádí zdrojový kód aplikace do APK souborů k dalšímu testování, spouštění či distribuci. Každá konfigurace sestavení může obsahovat vlastní sadu kódu a prostředků, přičemž opětovně používá části společné pro všechny verze aplikace. [16]

Android plugin a Gradle běží nezávisle na Android Studiu. To znamená, že lze vytvářet android aplikace nejen z Android Studia, ale i z příkazového řádku na počítači, kde nemusí být nainstalováno Android Studio, přičemž výstup sestavení je stejný. [16]

Vytváření vlastních konfigurací sestavení vyžaduje provedení změn v konfiguračních souborech sestavení nebo v *build.gradle* souboru. Tyto soubory používají Domain Specific Language (DSL) pomocí nástroje Groovy, což je dynamický jazyk pro Java Virtual Machine (JVM). Skript sestavení Groovy většinou vypadá jako konfigurace obsahující nastavení vlastností a konfiguraci závislostí projektu. Tato konfigurace je založena na jazykových konstrukcích nástroje Groovy. [16], [17]

Pokaždé když se v projektu provedou změny v konfiguračních souborech sestavení, Android Studio vyžaduje synchronizaci souborů projektu. Tato synchronizace je nutná k importování změn konfigurace sestavení a kontroly případných chyb sestavení. [16]

Gradle také obsahuje tyto dva soubory umístěné v kořenovém adresáři projektu, které lze použít k nastavení samotné sady nástrojů Gradle build: [16]

- *gradle.properties* – Slouží ke konfiguraci nastavení Gradle pro celý projekt.
- *local.properties* – Konfiguruje vlastnosti lokálního prostředí. Lze zde nastavit například alternativní cestu k SDK nebo NDK adresáři.

### 2.2.2 Struktura aplikace

Tato podkapitola se věnuje pohledu do kostry celého projektu a popisuje jeho nedílné součásti. Vývojářské prostředí Android Studio nabízí práci na aplikaci v několika různých pohledech. Mezi základní a nejpodstatnější pohledy lze zařadit pohled projektový a pohled Android. Popsán je zde i soubor *AndroidManifest.xml*, jenž je důležitý pro správný chod aplikace. Obsahuje manifest aplikace a musí být obsažen v každém projektu. [18]

**AndroidManifest.xml**

Ve složce manifests lze nalézt soubor *AndroidManifest.xml*, který zahrnuje aplikační manifest. Jedná se o hlavní konfigurační soubor aplikace. Tento soubor nese základní informace

o aplikaci pro sestavovací nástroje a obchod Google Play. Soubor je v projektu také povinen deklarovat název balíčku aplikace sloužící jako unikátní identifikátor, nejnižší možnou kompatibilní verzi Android API a seznam oprávnění ke službám aplikace. [18], [20]

**Projektový pohled**

Android Studio v projektovém pohledu na aplikaci zobrazuje strukturu souborů projektu tak, jak je skutečně uložena na disku počítače, včetně skrytých souborů v pohledu Android. Nejdůležitější soubory a adresáři, které lze najít v adresáři s názvem modulu, jsou: [18], [20]

- *build/* – Zahrnuje výstupy sestavení.
- *libs*/ – Obsahuje dodatečné knihovny.
- *src/* – Shromažďuje všechny zdrojové soubory a soubory, obsahující kód. Tyto soubory lze najít rozřazené v následujících podadresářích:
    - *androidTest/* – Slouží jako adresář pro testovací soubory.
    - *main/* – Obsahuje hlavní soubory aplikace, se kterými programátor pracuje (např. třídy projektu a zdrojové prostředky sdílené všemi variantami sestavení). V tomto adresáři lze nalézt řadu dalších podadresářů, které dále rozdělují dané soubory dle jejich účelu. Mezi ty nejdůležitější patří adresář *java/*, který zahrnuje Java soubory, a adresář *res/*, ve kterém lze najít prostředky aplikace, jako jsou obrázky a XML soubory definující uživatelské rozhraní. V poslední řadě se zde nachází soubor *AndroidManifest.xml*, který popisuje povahu aplikace a její součásti nezbytné pro správný běh aplikace.
- *test*/ – Zahrnuje kód pro lokální testy.
- *build.gradle* soubor – Definuje konfiguraci sestavení specifické pro daný modul.

V projektovém pohledu existuje ještě další soubor se jménem *build.gradle*. Ten se stará o konfiguraci sestavení, která platí pro všechny moduly v projektu. [20]

**Pohled Android**

Android Studio ve výchozím nastavení zobrazuje soubory projektu v pohledu Android. Toto zobrazení nereprezentuje skutečnou hierarchii souborů na disku počítače, nýbrž je uspořádáno podle modulů a typů souborů. To má za následek skrytí určitých souborů a adresářů, které se běžně nepoužívají, a jednodušší pohyb uživatele ve zdrojových souborech. V rámci každého modulu aplikace se soubory zobrazují v následujících skupinách: [20]

- **manifests –** Obsahuje soubor *AndroidManifest.xml*.
- **java –** Zahrnuje soubory zdrojového kódu Java, které jsou rozdělené dle názvů balíčků. Obsažen je zde i testovací JUnit soubor.
- **res –** Shromažďuje všechny projektové zdrojové soubory bez kódu. Spadají sem XML layouty, definované textové řetězce, obrázky a ikony. Tyto soubory jsou rozdělené do podadresářů dle jejich funkce.

Ve srovnání s projektovým zobrazením se pohled Android liší v následujících bodech: [20]

- Pohled Android zobrazuje všechny konfigurační soubory související se sestavením projektu na jednom místě ve skupině Gradle Scripts.
- Pokud existuje více souborů manifestu, které jsou určené pro různé druhy sestavení, jsou manifest soubory v pohledu Android umístěny v jedné skupině na úrovni modulu.
- Všechny zdrojové soubory jsou v pohledu Android uspořádané v jedné skupině, nikoli v samostatných složkách podle typů souborů. Například všechny druhy jedné ikony v projektu jsou umístěny hned vedle sebe.

### 2.2.3 Uživatelské rozhraní aplikace

Tato kapitola pojednává o vzhledu a uživatelském rozhraní aplikace. V první části podkapitoly je rozebrán pojem aktivita a její využití v rámci aplikace. Dále jsou probrány adresáře a jejich obsahy – obrázky, ikony a další grafické soubory. V závěru jsou prezentovány soubory, které určují rozmístění vzhledových prvků grafického uživatelského prostředí včetně ukázek vzhledu jednotlivých obrazovek v aplikaci.

**Aktivita (Activity)**

Aktivita je základní komponenta aplikace, která umožňuje uživateli ovládat aplikaci skrze grafické uživatelské rozhraní. Lze konstatovat, že každá aktivita prezentuje obrazovku s uživatelským rozhraním. V aktivitě uživatel získává informace od aplikace a komunikuje s ní. Návrh aktivity by měl obsahovat realizaci konkrétní úlohy, kterou je potřeba vykonat, například zadání údajů či výběr položky ze seznamu. Všechny aktivity musí být definované v souboru *AndroidManifest.xml*, kde musí být také určena jedna hlavní aktivita, která je spuštěna při startu aplikace. [18], [19]

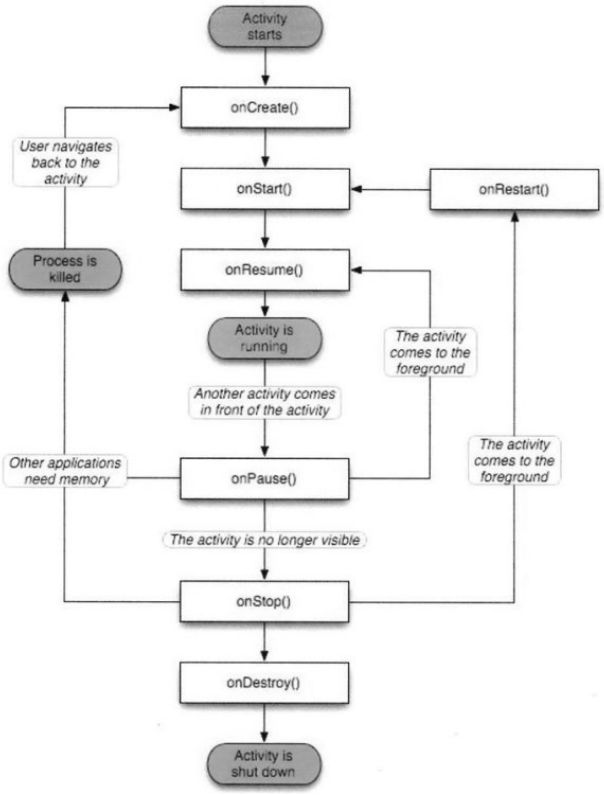
Aktivity si mohou mezi sebou předávat data, různě se mezi sebou přepínat pomocí instance třídy *Intent*. Pokaždé při spuštění další aktivity se současná aktivita zastaví a její činnost je

uchována v zásobníku. To znamená, že každá nově spuštěná aktivita je vložena na vrchol zásobníku a stává se běžící aktivitou. [18]

Každá aktivita má svůj životní cyklus. Tento cyklus se skládá ze tří hlavních fází: [19]

- Aktivita na vrcholu zásobníku – Představuje aktivitu, se kterou provádí uživatel interakci a zobrazuje se na obrazovce zařízení. Může nabývat stavu „running“ či „resumed“.
- Pozastavená aktivita – Jedná se o aktivitu, se kterou uživatel již neinteraguje. Může být stále viditelná, nejčastěji je pouze překrytá dialogovým oknem. Aktivita je stále uložená v paměti a drží svůj stav a informace.
- Zastavená aktivita – Aktivita v tomto stavu stále dokáže držet svoje informace a stav, ale je kompletně překryta jinou aktivitou. V případě nedostatku paměti v aplikaci budou nejdříve odejmuty zastavené aktivity a až po nich aktivity pozastavené.

Aktivita může nabývat několika stavů. Pokud potřebuje programátor vykonat akce v určité fázi životního cyklu, je zapotřebí přepsat stavovou metodu, která se váže na konkrétní fázi cyklu. Stavy jsou popsány na následujícím obrázku, kde obdélníky představují dané metody, které je možno přepsat a detailněji jsou popsány níže. [19]




**Obrázek 2 Životní cyklus aktivity Zdroj: [19]**

- *onCreate()* – Jedná se o první metodu, která se aktivuje po spuštění aktivity. Metoda obvykle obstarává nastavení uživatelského rozhraní aktivity a nastavení proměnných. V parametru se předá objekt typu *Bundle*, který obsahuje informace o existujícím předešlém stavu. Po zavolání této metody ještě aktivita není v popředí, tudíž neinteraguje s uživatelem. Aktivita se nachází ve fázi zastavení a její přechod do popředí je realizován v dalších metodách. [19]
- *onStart()* – Pomocí této metody aktivita přechází ze stavu zastavení na vrchol zásobníku. Tato metoda se volá při znovuspuštění aktivity ze stavu zastavení. [19]
- *onResume()* – Při úspěšném nastavení grafických prvků, kdy je aktivita připravena přejít do popředí, se zavolá metoda *onResume()*. Po zavolání této metody už je aktivita plně schopna komunikovat s uživatelem. [19]
- *onPause()* – Tato metoda se zavolá v případě obnovení jiné aktivity, kdy je aktivita vystřídána na vrcholu zásobníku. Dochází zde k zastavení např. animací a zisku údajů z různých systémových senzorů. [19]
- *onStop()* – Metoda volající se v případech překrytí aktivity jinou aktivitou. Pokud aktivita potřebuje být opět spuštěna, zavolá se metoda *onRestart()*. [19]
- *onDestroy()* – Jedná se o metodu použitou při ukončování životního cyklu aktivity. To může nastat při nedostatku operační paměti nebo při zavolání metody *finish()*. [19]

**Složky k uchovávání grafických prvků aplikace**

V této části se pojednává o složkách, které obsahují soubory určující vzhled jednotlivých aktivit prostřednictvím tzv. layoutů. Složky se nachází v adresáři /*res*.

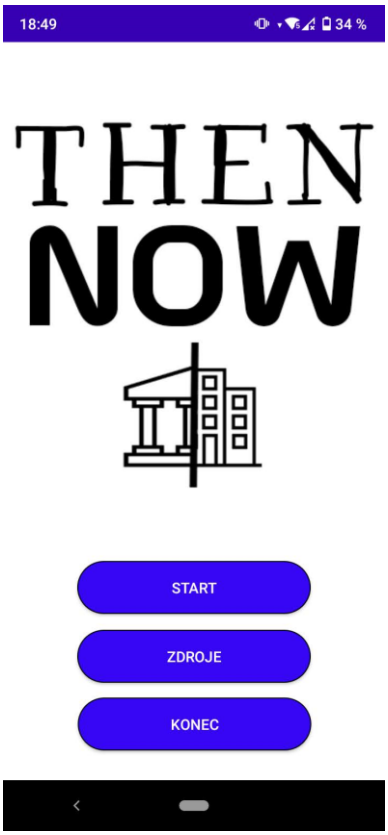
- *drawable* – Ve složce *drawable* jsou obsaženy veškeré importované obrázky určené pro vzhled aplikace. Obrázky lze definovat až v šesti možných rozlišeních, rozdělených do jednotlivých podsložek složky *drawable* podle typu rozlišení. Složka také obsahuje XML soubory animací, pozadí dialogů a tlačítek. [20]
- *mipmap* – Nachází se zde ikony v jednotlivých rozlišeních použité v aplikaci. Všechny ikony použité v aplikaci jsou součástí platformy Android Studio pod licencí Apache License Version 2.0. [21]
- *values* – Obsahuje programátorem definované barvy, styly a textové řetězce. [20]

- *layout* – Do této složky patří soubory, které udávají rozmístění prvků na uživatelově obrazovce. Jedná se o soubory typu XML. Pomocí intuitivního návrháře v Android Studiu, lze jednoduše vytvářet vzhled aplikace bez potřeby psaní složitého programátorského kódu. Rozložení, která jsou orientována na šířku, je potřeba uložit do složky *layout-land*. Podle orientace zařízení si systém poté vybere vhodný soubor, který použije při nastavování rozložení daných prvků. [19]

**Layouty použité v aktivitách**

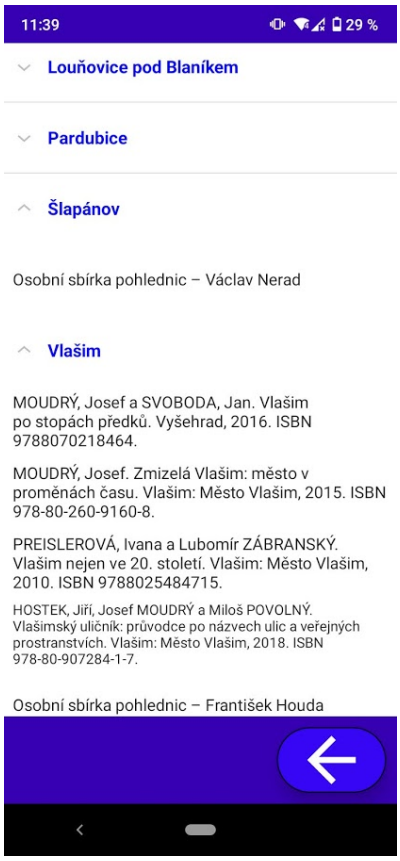
Tato poslední část podkapitoly představuje samotný vzhled jednotlivých aktivit prostřednictvím layoutů, které tvoří rozmístění prvků grafického prostředí. S každým layoutem je současně prezentována i jeho ukázka na obrazovce zařízení uživatele. V této části je popsán pouze vzhled prvků, jednotlivé aktivity včetně vysvětlení funkcionalit budou detailněji popsány v kapitole věnované přehledu a popisu tříd.

- *activity_main_menu.xml* – Hlavní menu aplikace použité ve startovací aktivitě *MainActivity*. Layout obsahuje logo aplikace a tři tlačítka určené k přesměrování do dalších aktivit aplikace.



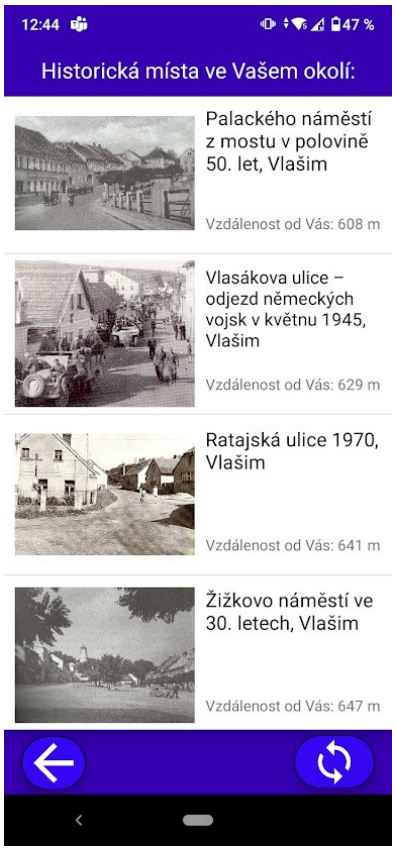

**Obrázek 3 Hlavní menu – MainActivity Zdroj: Vlastní**

- *activity_sources.xml* – Rozložení, které slouží k zobrazení zdrojů použitých historických fotografií v aktivitě *SourcesActivity*. Rozbalovací menu je realizováno pomocí komponenty *ExpandableListView*. Zdroje jsou rozděleny podle měst, ze kterého fotografie pochází. Po rozkliknutí daného města se zobrazí seznam zdrojů, ze kterých byly historické fotografie použity.




**Obrázek 4 Zdroje fotografií – SourcesActivity Zdroj: Vlastní**

- *activity_places_list.xml* – Toto rozložení využívá aktivita *PlacesListActivity*. V rozložení je použita komponenta *ListView*, na které se zobrazují načtená historická místa. Kromě dvou viditelných tlačítek určených k návratu do *MainActivity* a znovunačtení míst dle polohy existuje v rozložení i další tlačítko, které se zobrazí při dolistování na konec seznamu míst. Toto tlačítko nabízí načtení dalších deseti historických míst do komponenty *ListView*. Po kliknutí na místo v seznamu se spustí aktivita *PlaceDetailActivity*.



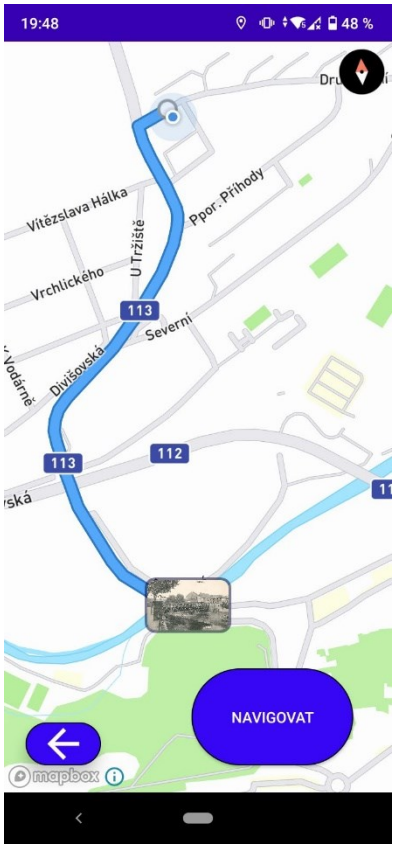

**Obrázek 5 Seznam míst – PlacesListActivity Zdroj: Vlastní**

- *activity_place_detail.xml* – Pomocí tohoto rozložení se v aktivitě *PlaceDetailActivity* zobrazí obrázek vybraného historického místa včetně jeho popisu. Popis je realizován pomocí komponenty *ScrollView*, pomocí které lze popis s větší délkou posouvat. Obrázek je umístěn do komponenty *ImageView* a při kliknutí na něj se spustí aktivita *PlacePhotoDetailActivity*. Layout obsahuje v dolní liště tři tlačítka, sloužící k návratu do *PlacesListActivity*, přechodu do *NavigationActivity*, nebo do jednoho z fotografovacích režimů.


**Obrázek 6 Detail místa – PlaceDetailActivity Zdroj: Vlastní**

- *activity_full_detail_of_historic_photo.xml* – Tento layout je použit v aktivitě *PlacePhotoDetailActivity* a slouží k zobrazení obrázku historického místa v plné velikosti. Zobrazení je realizováno pomocí komponenty *SubsamplingScaleImageView*, která umožňuje uživateli zvětšování obrázku pomocí pohybu prstů po obrazovce zařízení. Při krátkém kliknutí na obrázek dochází k ukončení aktivity a návratu uživatele do *PlaceDetailActivity*.

- *activity_navigation_map.xml* – Rozložení obsahující komponentu *MapView* k zobrazení načtených map na zařízení. Kromě této komponenty zde lze najít i tlačítko určené k navrácení do aktivity *PlaceDetailActivity* a tlačítko, které spustí navigaci k místu.




**Obrázek 7 Navigace na mapě – NavigationActivity Zdroj: Vlastní**

- *activity_camera_split.xml* a *activity_camera_panorama.xml* – Tato rozložení jsou obsažena v aktivitách *CameraSplitActivity* a *CameraPanoramaActivity*. Rozdíl v těchto dvou rozloženích představuje komponenta, která slouží jako podklad pro fotoaparát. Zatímco v *activity_camera_panorama.xml* je jako podklad pro kameru použita komponenta *AutoScalingTextureView*, pro *activity_camera_split.xml* je vyhrazena komponenta *FrameLayout*. Pro zobrazení historického obrázku je použita komponenta *SubsamplingScaleImageView*, pomocí které si může uživatel obrázek posouvat dle potřeby. V obou rozloženích se nachází tlačítka sloužící k návratu do *PlaceDetailActivity*, vyfotografování snímku a záměně pohledů.

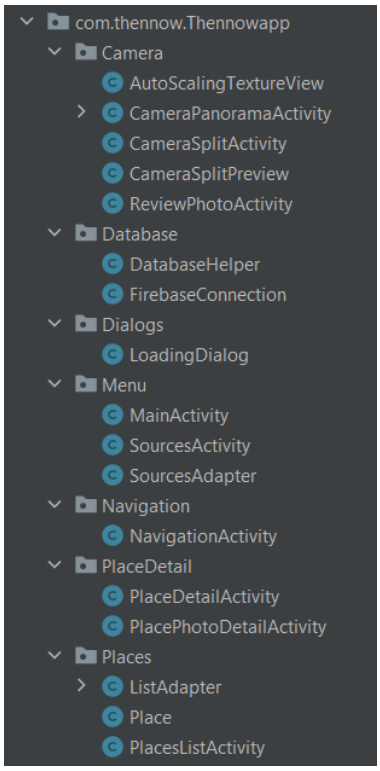
**Obrázek 8 Normální režim – CameraSplitActivity Zdroj: Vlastní**

**Obrázek 9 Panoramatický režim – CameraPanoramaActivity Zdroj: Vlastní**

- *activity_review_photo.xml* – Po vyfotografování snímku se spustí aktivita *ReviewPhotoActivity*, ve které je uplatněn právě tento layout. Nabízí uživateli dvě tlačítka. Tlačítko „zpět“ vrátí uživatele do režimu fotografování. Druhé tlačítko slouží k uložení vyfotografovaného snímku do uložiště telefonu.

### 2.2.4 Přehled a popis tříd

V této podkapitole je věnována třídám aplikace. Třídy jsou rozdělené do složek dle jejich funkce v aplikaci. V úvodu je rozebrána třída *MainActivity*, která je nastavena jako startovací aktivita v aplikaci. V další části podkapitoly jsou podrobně vysvětleny třídy určené k připojení do cloud databáze a načítací dialogy. Poté se pozornost zaměřuje na třídy spjaté se zobrazováním míst a detailu daného místa na zařízeních. V neposlední řadě se podkapitola bude věnovat třídě, která zajišťuje načtení mapových podkladů a řízení navigace. Na závěr jsou probrány třídy obsluhující kameru zařízení a její součásti.




**Obrázek 10 Přehled tříd Zdroj: Vlastní**

**Menu**

Tato složka obsahuje startovací aktivitu *MainActivity* a třídy spojené se zobrazením zdrojů použitých fotografií.

- MainActivity – Jedná se o první aktivitu, která se spustí při spuštění aplikace. Uživateli se v této aktivitě zobrazí úvodní menu a logo aplikace. Kromě obsluhy tlačítek v menu, vytváří tato aplikace statickou instanci třídy *DatabaseHelper*, která se poté používá v rámci celé aplikace. Tato instance při startu aplikace zároveň i připojuje uživatele do databáze. Aktivita je nastavena pouze do režimu „na výšku“. Toto nastavení má na starost metoda *setRequestedOrientation()* s předaným parametrem *SCREEN_ORIENTATION_PORTRAIT* od třídy *ActivityInfo*.

- SourcesActivity – Aktivita, ve které dochází k zobrazení zdrojů použitých fotografií. Hlavní komponentou třídy je *ExpandableListView*, která poskytuje zobrazení textů v podobě rozbalovacího menu. Po inicializaci všech vytvořených instancí se pomocí třídy *SourcesAdapter* vytvoří adaptér s předanými prvky. Tento adaptér se následně nastaví jako adaptér do instance třídy *ExpandableListView*.
- SourcesAdapter – Pomocí této třídy se realizuje zobrazení jednotlivých textů v komponentě *ExpandableListView*. Třída je nastavena jako potomek třídy *BaseExpandableListAdapter* a implementuje všechny její metody. Důležité je zde rozdělení prvků na hlavní skupiny a podskupiny. V konstruktoru třídy dochází k nastavení těchto dvou atributů, ke kterým se dále přistupuje přes přepsané metody, které obstarávají nastavení pohledu. Vytvořený pohled se následně vrátí a je připraven k zobrazení na příslušných komponentách.

**Database**

Ve složce Database jsou třídy obstarávající připojení do Firebase databáze.

- FirebaseConnection – V této třídě se pouze předává Android kontext do instance Firebase pomocí statické metody *setAndroidContext().* Tato metoda je nutná pro další volání metod z instance Firebase.
- DatabaseHelper – Jedná se o třídu, která zprostředkovává přístup a komunikaci s cloudovou databází Firebase. V kontruktoru třídy se vytváří nová instance třídy Firebase s parametrem, který představuje odkaz v podobě textového řetězce na adresu realtime databáze ve Firebase. Prostřednictvím této vytvořené instance se v metodě *connectDataSnapshotToFirebaseDatabase()* přidá posluchač (listener) na událost, který bude udržovat aktuální data z databáze prostřednictvím tzv. snapshotu. Ze získaného snapshotu dat se v dalších metodách předávají samotné informace o historickém místu uloženém v databázi k dalšímu zpracování. Z předané polohy místa se v metodě *calculateDistance()* s parametry zeměpisná šířka a zeměpisná délka místa vypočítá a vrátí vzdálenost historického místa od uživatelova zařízení. Tento výpočet zajišťuje metoda *distanceBeetwen(double startLatitude, double StartLongtitude, double endLatitude, double endLongtitude, float[] results)* nad objektem Location. S touto vypočítanou vzdáleností lze poté vytvořit instanci třídy Place s parametry:
    - název místa,
    - zeměpisná šířka (latitude),

- zeměpisná délka (latitude),
- obrázek (uložen v textovém řetezci),
- vypočítaná vzdálenost k místu,
- popisek místa.

Po vytvoření instance se místo vloží do listu míst a list se setřídí podle vzdálenosti pomocí třídicího algoritmu bubble sort. Aplikace umožňuje uživateli zobrazit maximálně 20 míst, což znamená, že při naplnění listu 20 položkami se další příchozí historické místo nastaví na příslušný index a posune místa za sebou. Tímto cyklem získáme nenáročně list historických míst z databáze s nejmenší vzdáleností místa k uživateli. Nutno dodat, že pomocí takto zprostředkované logiky se data z databáze získávají rychle – každé místo s větší vypočítanou vzdáleností, než je vzdálenost 20. prvku v listu, se ihned zahodí. Navíc není potřeba využití velkého množství mobilních dat – obrázek se načítá jako odkaz v podobě textového řetězce a zpracovává se až v dalších částech aplikace, tudíž není nutné při každé aktualizaci míst stahovat několik desítek MB obrázků všech míst v databázi.

**LoadingDialog**

Tento dialog si lze představit jako načítací okno sloužící k zabavení uživatele při načítání komponent v aplikaci. Dialog lze použít v každé aktivitě, stačí předat danou aktivitu jako parametr v konstruktoru dialogu. Vzhled dialogu je zprostředkován pomocí speciálně upraveného layoutu skládajícího se z animace objektu a transparentního okna. Třída obsahuje metody na spuštění dialogu, vypnutí dialogu a zjištění, zda dialog stále běží či ne.

**Places**

V této skupině se nacházejí třídy, které pracují na zobrazení míst v seznamu na obrazovce zařízení.

- Place – Tato třída reprezentuje historické místo v aplikaci. Nabývá atributy typu textového řetězce name, image, decription, a atributy typu double longitude, latitude a distance. Součástí třídy Place jsou také parametrický konstruktor, gettery a settery jednotlivých atributů.
- ListAdapter – Představuje třídu, která slouží k přístupu k položkám a nastavení vzhledu seznamu míst v rámci adaptéru. ListAdapter dědí ze třídy *ArrayAdapter* s generickým typem *Place*. Díky této dědičnosti je adaptér schopen vrátit zobrazení pro každý objekt z kolekce dat. V parametrickém konstruktoru se nastaví aktuální používaný kontext

aplikace, list historických míst a rozložení (layout) vytvořené pro zobrazení míst ve třídě *ListView*. Pomocí přepsané metody *getView(int position, View convertView, ViewGroup parent)* dochází k nastavení jednotlivých prvků v adaptéru do vybraného rozložení zobrazení. Kromě nastavení jednotlivých textových pohledů názvem a vzdáleností místa, v této metodě probíhá také převod odkazu obrázku místa v podobě textového řetězce na obrázek skutečný. Tuto přeměnu zajišťuje třída Picasso, která je detailněji popsána v kapitole o použitých technologiích.

- PlacesListActivity – Jedná se o aktivitu, která zajišťuje komunikaci se službou GPS a interakci s načteným seznamem historických lokalit. Při spuštění aktivity se nejprve ověří, zda má aplikace přidělené oprávnění využívat službu GPS. Pokud jej nemá, uživatel je vyzván k povolení tohoto oprávnění, v případě zamítnutí nelze pokračovat v aplikaci dále. Jestliže zařízení nemá aktivní funkci pro určování polohy, zobrazí se uživateli dotaz, zda chce povolit přístup k poloze. Tato funkce je přístupná pouze pro zařízení s verzí Androdi 6.0 a vyšší. Jestli zařízení má povolen přístup k určování polohy a služba GPS je aktivní, zaregistruje se požadavek na získávání polohy zařízení. Požadavku je zapotřebí nastavit následující informace:
    - interval pro obnovování pozice v milisekundách,
    - způsob získávání pozice (pomocí GPS nebo ze sítě).

  Úspěšný požadavek obslouží vytvořená instance třídy *FusedLocationProviderClient*, která je prvkem Google API. Obsloužení probíhá pomocí této metody:

  *fusedLocationProviderClient.requestLocationUpdates(locationRequest, locationCallback, Looper.getMainLooper());*

    - locationRequest – Předaný úspešně vytvořený požadavek,
    - locationCallback – Instance abstrakní třídy *LocationCallback*, která za předpokladu úspěšného registrování požadavku u klienta poskytne pomocí přepsané metody *onLocationResult(LocationResult locationResult)* získanou polohu zařízení.
    - Looper.getMainLooper() – Vrací Looper, ve kterém je obsaženo hlavní vlákno aplikace.

Ze získané polohy zařízení se poté nastaví statické atributy *myLatitude* a *myLongitude* typu double, vysílání požadavku pro získání polohy se zastaví a přejde se k načtení seznamu historických míst.

V metodě *loadlist()* probíhá načtení listu míst a jejich zobrazení na obrazovce zařízení pomocí třídy *ListView* s nastaveným adapterém. Jelikož je již poloha zařízení zjištěna a uložena do statických atributů, lze pomocí instance třídy *DatabaseHelper* získat seznam historických míst seřazený od nejmenší vzdálenosti vůči uživatelově zařízení. Samotné zobrazení míst zprostředkovává instance třídy *ListAdapter*, která je po jejím správném vytvoření nastavena jako adaptér instance třídy *ListView*. Výběr položky ze seznamu je realizován pomocí posluchače na událost kliknutí na položku umístěném na instanci třídy *ListView*. Aby se obrázek mohl předávat mezi aktivitami bez nutnosti dalších požadavků, je třeba jej transformovat do statického pole bajtů. Tato transformace se provádí pomocí technologie Glide, která je detailněji popsána v kapitole o použitých technologiích.

Další aktualizace seznamu podle nové uživatelovy lokace je prováděna až na vyžádání uživatelem pomocí tlačítka na obnovení seznamu a to i v případě, že danou aktivitu opustí. Všechny údaje související se seznamem míst jsou totiž uloženy jako statické atributy a instance, tudíž při opětovném spuštění aktivity se nejdříve ověří, zda byl seznam v minulosti již načtený. Pomocí této logiky se zamezí neustálému zasílání požadavků do databáze a aktualizaci polohy při každém znovuspuštění aktivity.

**PlaceDetail**

Složka PlaceDetail nabízí aktivity obsluhující zobrazení detailu místa vybraného ze seznamu historických míst.

- PlaceDetailActivity – V této třídě dochází k předání parametrů historického místa vybraného ze seznamu míst. Předaný obrázek místa je převeden z pole bajtů do statické bitmapy, se kterou se dále v aplikaci pracuje. Jak název, tak i popis místa jsou načteny do textových pohledů, přičemž text popisu místa je zarovnán do bloku. Zarovnání do bloku ovšem podporují zařízení s verzí Android 8.0 a vyšší. Uživatelům s nižší verzí Android se text zobrazí zarovnán doleva. Kromě informací o vybraném místu, obsahuje aktivita i tlačítka k přepnutí do aktivity *NavigationActivity*, aktivit fotografování a při kliknutí na obrázek do aktivity *PlacePhotoDetailActivity*. V případě přepnutí do „fotografovací“ aktivity je zapotřebí pomocí vyskočeného dialogu povolit oprávnění

aplikace k fotoaparátu zařízení. Pokud uživatel zamítne oprávnění, nebude vpuštěn do fotografovacích aktivit. Jestliže jsou oprávnění povolena, dostane uživatel prostřednictvím dalšího dialogu na výběr, do jakého režimu fotografování chce přejít. Po vybrání příslušného typu se zařízení přepne do aktivit spojených s fotografováním popsaných ve skupině *Camera*.

- PlacePhotoDetailActivity – Aktivita slouží k zobrazení obrázku historického místa v plné velikosti na celou obrazovku zařízení. Zobrazení je realizováno pomocí technologie SubsamplingScaleImageView, která je detailněji popsána v kapitole o použitých technologiích.

**NavigationActivity**

Hlavním úkolem této aktivity je zprostředkovat mapové podklady od Mapbox SDK a zobrazit je na uživatelově zařízení. Třída implementuje rozhraní *OnMapReadyCallBack* s metodou *onMapReady(MapboxMap mapboxMap)*, která poskytuje instanci mapy a zavolá se až tehdy, když bude mapa připravena k použití. Nejprve je zapotřebí předat třídě *MapBox* instanci skládající se z aktuálního kontextu aplikace a přístupového klíče, který si registrovaný vývojář vygeneruje na oficiálních stránkách společnosti Mapbox. Před samotným přístupem k poloze je nutno si opět ověřit, zda má aplikace povolení používat službu GPS. V případě nepřidělení oprávnění není možné mapy načíst a dále s nimi pracovat. Načítání map zajišťuje Mapbox API. Kromě zobrazení načtených map i s vyznačenou cestou k danému místu od uživatelova zařízení, má uživatel i možnost navigace k místu skrze navigační API společnosti Mapbox. Obě tyto rozhraní jsou detailněji popsány v kapitole o využívaných API.

**Camera**

Obsahuje třídy spojené s používáním fotoaparátu zařízení v aplikaci a ukládáním pořízeného snímku. Aplikace nabízí dva fotografické režimy – normální a panoramatický. V obou případech je kamera umístěna na jedné polovině zařízení, zatímco na druhé polovině je umístěn obrázek vybraného historického místa. Součástí obou režimů je tlačítko na pořízení fotografie a tlačítko na prohození oblasti obrázku s oblastí s fotoaparátem. Zařízení je v režimu fotografování nastaveno do režimu „na šířku“. Kromě aktivit těchto dvou režimů obsahuje tato skupina i třídy upřesňující nastavení parametrů kamery v režimech.

Normální fotografický režim obsluhují tyto třídy:

- CameraSplitPreview – Jedná se o třídu zajišťující spravné rozlišení a poměr stran kamery v normálním režimu kamery. Třída dědí ze třídy *SurfaceView* a implementuje

rozhraní *SurfaceHolder.Callback*, které poskytuje informace o změnách rozlišení pohledu, na kterém je umístěna služba fotoaparátu. V konstruktoru třídy se nastaví vytvořená instance třídy *SurfaceHolder* a pomocí parametrů se do atributů třídy předá aktuální kontext aplikace a vytvořená kamera. Pomocí předané kamery se zjistí podporovaná rozlišení kamerou zařízení a uloží do listu. V přepsané metodě *onMeasure(int widthMeasureSpec, int heightMeasureSpec)* s parametry šířka a výška vázaného pohledu se vypočítá a nastaví ideální poměr stran kamery. Tato metoda využívá další metodu, která z podporovaných rozlišení kamery vrátí nejoptimálnější rozlišení vůči vytvořenému pohledu. Tomuto vybranému rozlišení se následně upraví poměr stran tak, aby výsledná kamera pokryla celý pohled.

- CameraSplitActivity – Aktivita řídící služby kamery včetně přibližování a oddalování jak fotoaparátu, tak i historického obrázku. Nabízí režim fotografování, ve kterém fotoaparát zabírá celou jednu polovinu obrazovky zařízení. Druhou polovinu vyplňuje obrázek historického místa. Třída implementuje rozhraní *SensorEventListener*, pomocí kterého se zjišťuje, zda bylo zařízení otočeno. Díky této informaci je zabudovaná kamera schopna správně měnit svoji orientaci při otočení zařízení. Při spuštění aktivity se vytvoří instance třídy *CameraSplitPreview*, která vytvoří správný podklad pro umístění kamery do pohledu. Tato nastavená instance se následně přidá jako pohled do instance třídy *FrameLayout*, ve kterém je uložena samotná kamera.

  Spuštěné kameře je potřeba nastavit v parametrech funkci automatického zaostřování. To obstarává metoda *setCameraAutoFocusMode()*. Další potřebnou funkcí je povolení přibližování a oddalování fotoaparátu, které zajišťuje komponenta *ZoomControls*. Vytvořená instance této komponenty se přidá jako pohled do rozložení, kde je umístěna kamera a v parametrech fotoaparátu se nastaví limit na maximální přiblížení. Po stisknutí tlačítka určeného k fotografování se vytvoří dva snímky obrazovky – snímek z fotoaparátu a snímek z obrázku historického místa. Snímky se převedou do bitmapové podoby a předají se do aktivity *ReviewPhotoActivity*.

Režim panoramatický obstarávají tyto třídy:

- AutoScalingTextureView – Jedná se o potomka třídy TextureView, která slouží k zobrazení kamery v pohledu. Ve třídě se nastaví atributy určující šířku a výšku parametrů kamery. Pomocí přepsané metody *onMeasure(int widthMeasureSpec, int heightMeasureSpec)* s parametry šířka a výška pohledu se nastaví optimální rozlišení

pohledu s vypočítaným poměrem stran dle parametrů kamery. Rozdíl oproti třídě *CameraSplitPreview* spočívá v rozdílné realizaci nastavení poměru stran. Zatímco ve třídě *CameraSplitPreview* se optimální vybrané rozlišení fotoaparátu zařízení přizpůsobuje vázanému pohledu, tak v této třídě se pohled nastaví podle vybraného rozlišení z parametrů kamery i s přizpůsobením poměru stran pohledu.

- CameraPanoramaActivity – Tato aktivita zprostředkovává panoramatický režim fotografování v aplikaci, ve které fotoaparát vyplňuje celou jednu polovinu obrazovky zařízení. Ve druhé polovině obrazovky se nachází obrázek historického místa. Třída podobně jako aktivita *CameraSplitActivity* implementuje rozhraní *SensorEventListener,* které zjišťuje správnou orientaci integrované kamery při otáčení zařízením. Při startu aktivity se pomocí vytvořeného posluchače třídy *TextureView.SurfaceTextureListener* zajistí aktivace fotoaparátu až tehdy, když bude pohled *TextureView* plně nastaven a připraven k použití. Správné spuštění a nastavení fotoaparátu obstarávají tyto metody:
    - *createCameraPreview()* – Přiřazuje fotoaparát do pohledu, který je určen k zobrazení kamery.
    - *setUpCameraOutputs(int width, int height)* – Vybere podle parametrů zařízení optimální podporované rozlišení z parametrů kamery a přizpůsobí podle něj pohled v aktivitě. Cílem je vybrat rozlišení s vysokou kvalitou a nejpřesněji odpovídající poměru 16:9, aby vynikl panoramatický režim.
    - *configureTransform(int viewWidth, int viewHeight)* – Pomocí parametrů již nastaveného pohledu upraví případné nedokonalosti vybraného rozlišení (např. přečnívající výšku či šířku kamery z pohledu)
    - *openCamera(int width, int height)* – Po vykonání všech předešlých metod spustí v aktivitě fotoaparát pomocí instance třídy *CameraDevice*.

Všechny výše zmíněné metody využívají funkcionalit Camera2 API, které jsou detailněji rozebrány v kapitole o využívaných API.

Přibližování a oddalování fotoaparátu je realizováno pomocí posluchače umístěném na pohledu, ve kterém se nachází kamera. Posluchač detekuje pohyb dvou prstů na zařízení a podle uživatelových pohybů je schopen fotoaparát přiblížit či oddálit. Při stisknutí tlačítka k vyfotografování místa se zavolá metoda *takePicture()*, která vytvoří fotografii a převede ji do bitmapy. Následně se vytvoří snímek obrazovky historického místa a obě bitmapy se předají se

do nově spuštěné aktivity *ReviewPhotoActivity*. Detailní popis procesu fotografování a přibližování je vylíčen v kapitole o využívaných API.

**ReviewPhotoActivity**

Poslední rozebíranou aktivitou je aktivita určená k zobrazení vyfotografovaného snímku a k jeho možnému uložení do telefonu. Bitmapy historického místa a snímku z fotoaparátu se nastaví do pohledů a spolu vytvoří jeden finální snímek připraven k uložení. Před samotným procesem ukládání se nejdříve musí ověřit oprávnění aplikace zapisovat do uložiště telefonu. Pokud uživatel používá zařízení s verzí Android 6.0 a vyšší, tak v případě nepovolení zápisu do uložiště je uživatel prostřednictvím vyskakovacího dialogu vyzván k udělení povolení.

V procesu ukládání se nejprve vytvoří jméno souboru ve formátu JPEG s názvem „thennow_“ + aktuální systémový čas. Následně se definuje cesta do adresáře „thennow“ v telefonu, kam se soubor uloží. Pokud adresář ještě neexistuje, tak se vytvoří. Poté se bitmapa převede do formátu JPEG a soubor je uložen do telefonu pomocí instance třídy *FileOutputStream*. Po úspěšném uložení je uživatel informován o dokončení procesu pomocí vyskakovací notifikace s textem „Saved“ a cestou k uloženému souboru.

### 2.2.5 Použité technologie

Pomocí technologií popsaných v této podkapitole aplikace zpracovává data a zjednodušuje práci s nimi. Pozornost je především zaměřena na technologie, které se věnují práci s obrázkem. Všechny níže popsané technologie je pro jejich správný chod potřeba definovat v *build.gradle* souboru v bloku *dependencies*.

**Picasso**

Jedná se knihovnu vyvinutou společností Square, která slouží k zobrazovaní obrázků z internetu. Technologie dokáže stáhnout obrázek z jeho URI textového řetězce a uložit ho do mezipaměti aplikace. Před jejím používáním je zapotřebí přidat internetové oprávnění do *AndroidManifest.xml* souboru, pomocí kterého se umožní aplikaci používat internet ke stahování obrázků. Načtený obrázek se poté zpravidla ukládá do komponenty *ImageView*. [22]

Základní načítání obrázku lze uskutečnit pomocí tohoto příkazu:

```
Picasso.get().load(imageURI).into(imageView);
```

- *get()* – Tato získaná instance je automaticky inicializována výchozími hodnotami, které jsou vhodné pro většinu implementací. Mezi tyto hodnoty patří: [23]
    - mezipamět LRU skládající se z 15 % dostupné RAM paměti,

    - mezipamět disku s kapacitou 2 % uložného prostoru (5 MB až 50 MB),
    - tři vlákna určená ke stahování na disku a přístup do sítě.

- *load()* – Spustí požadavek na stažení obrázku z parametru zadané cesty.
- *into()* – Určuje, do jaké instance třídy *ImageView* se stáhnutý obrázek uloží.

Knihovna Picasso nabízí i další vlastnosti, které lze použít při stahování obrázku. Mezi ty nejpoužívanější se řadí: [22], [23]

- *placeholder()* – Při čekání na stáhnutí obrázku se pomocí předaného parametru se mezitím nastaví jiný obrázek do instance třídy *ImageView*. Tento prozatímní obrázek zmizí v případě správného načtení požadovaného obrázku.
- *error()* – Podobná funkcionalita jako u vlastnosti *placeholder()* s tím rozdílem, že obrázek předaný v parametru této metody se nastaví až v případě neúspěšného načtení požadovaného obrázku.
- *resize()* – Umožňuje přenastavit velikost obrázku dle zadaných parametrů.

**Glide**

Technologie Glide je knihovna od společnosti Bumptech sloužící pro načítání a manipulaci s obrázky. Knihovna je podporovaná firmou Google, která ji používá v mnoha projektech s otevřeným zdrojovým kódem. Hlavními vlastnostmi technologie je podpora animací a ukládání načtených obrázků do mezipaměti aplikace. Technologie je velice podobná technologii Picasso s tím rozdílem, že v Glide je zapotřebí předat aktuální kontext aplikace a také nabízí více alternativ při ukládání obrázků. [24]

Základní funkcionalitu technologie Glide lze popsat pomocí této metody:

```
Glide.with(context).load("uri_address").into(imageView);
```

- *with()* – Pomocí této metody je Glide propojeno s danou aktivitou prostřednictvím předaného kontextu aplikace.
- *load()* a *into()* – Tyto metody, podobně jako u technologie Picasso, slouží k načtení obrázku u uvedené adresy a zobrazení jej pomocí komponenty *ImageView*.

Kromě vlastností *placeholder()* a *error()*, které pracují stejně jako u technologie Picasso, nabízí Glide funkcionality umožňující ukládat obrázky nejen do komponenty *ImageView*, ale i například jako bitmapu. Umožněno je i přidávat různé posluchače na jednotlivé vlastnosti. [24]

**SubsamplingScaleImageView**

Poslední probíranou technologií je komponenta SubsamplingScaleImageView. Jedná se o alternativu ke komponentě *ImageView*, určenou pro zobrazování obrázků. Zahrnuje všechny standardní funkcionality pro přiblížení a posouvání obrázků a poskytuje funkce pro animaci a změnu měřítka obrázku včetně otáčení obrázku. Komponenta je vysoce konfigurovatelná. Je umožněno například přidat vlastní překrytí obrázku ukotvené k jeho bodům. [25]

Tato knihovna řeší tyto běžně vyskytující problémy s komponentou *ImageView*: [25]

- Chyby, kdy je bitmapa příliš velká na to, aby mohla být nahrána do textury. Tyto chyby mohou vzniknout při pokusu o zobrazení obrázků o šířce nebo výšce 2048 pixelů.
- Problém při načtení obrázku do paměti bez převzorkování. Pomocí technologie se obrázek podvzorkuje a při přiblížení se zobrazí ve vysokém rozlišení ve viditelné oblasti.

Komponenta obsahuje funkcionality pro nastavení vlastností obrázku. Mezi ty nejpoužívanější lze zařadit tyto metody: [25]

- *setOrientation(int degree)* – Nastavuje orientaci obrazu. Parametrem metody je počet stupňů, o které se má obraz otočit.
- *setMinScale(int min)* a *setMaxScale(int max)* – Určuje minimální a maximální možné přiblížení obrazu.
- *setZoomEnabled(boolean isEnabled)* – Pomocí parametru typu boolean se v této metodě buď povolí, nebo zakáže možnost přiblížení obrazu.

## 2.3 Využívané API

API představuje jednoduché rozhraní pro vývoj aplikací. Skládá se z knihoven, procedur, funkcí a tříd. Funkce rozhraní je možno používat bez potřeby dalšího programování dané funkce. Třídy a knihovny zaručují komunikaci s dalšími softwary. Hlavním úkolem API je zajištění komunikace mezi aplikacemi, která se dá rozdělit na jednosměrnou a obousměrnou komunikaci. [26]

Princip fungování API je založen na poskytnutí dat a funkcí aplikaci, se kterou se primární aplikace hodlá propojit. Tyto data a funkce lze následně použít a dále s nimi pracovat. Z popisu dat je poté tvořena dokumentace, kterou poskytuje samotný vlastník daného API. [26]

Aplikace používá řadu menších i rozsáhlejších API. Detailněji zde bude popsáno API pro zprovoznění fotoaparátu na zařízení a API sloužící k nahrání map a navigaci.

### 2.3.1 Mapbox API

Rozhraní Mapbox API se dělí na čtyři odlišné služby – mapové a navigační služby, vyhledávácí služby s využitím geocoding a služby sloužící ke komunikaci účtů Mapbox. Pomocí těchto webových služeb lze přistupovat k Mapbox nástrojům a dalším službám. Aby programátor měl možnost pracovat s rozhraním Mapbox API, musí v aplikaci použít platný přístupový token, který připojuje požadavky příslušného API k jeho účtu. Detailněji zde budou popsány služby, které realizovaná aplikace využívá – služby map a navigace, které realizují své API pomocí přímé interakce s daným API nebo pomocí SDK. [8]

Služba Mapbox Maps Service obsahuje rozhraní API pro vytváření a vyžádání map. Součástí rozhraní jsou Mapbox styly, které umožňují načítat a měnit styl map, písem a obrázků. Kromě předem definovaných stylů si může registrovaný vývojář vytvořit vlastní styl pomocí nástroje Mapbox Studio. Mezi základní styly vlastněné Mapboxem, které jsou k dispozici všem registrovaným účtům pomocí jejich přístupového tokenu, patří: [8]

- Mapbox Light – Všeobecný styl, který má za cíl zobrazení geografického kontextu se zvýrazněním názvů a vizuálních dat.
- Mapbox Dark – Totožný styl jako Mapbox Light s tím rozdílem, že pozadí tohoto stylu tvoří tmavá barva.
- Mapbox Outdoors – Jedná se o styl uzpůsobený zvláště pro pěší turistiku a cykloturistiku.
- Mapbox Streets – Styl kladoucí důraz na přesné zobrazení a dobrou orientaci v silničních sítí.
- Mapbox Satellite – Představuje satelitní vyobrazení map.
- Mapbox Satellite Streets – Tento styl je kombinací stylu Mapbox Satellite s vektorovými daty ze stylu Mapbox Streets.

Mimo stylů map zde lze najít i další rozhraní, která mohou sloužit například k upravování map či nastavování parametrů jednotlivých snímků map. Integrování tohoto API do aplikace je následně realizováno pomocí Mapbox Maps SDK. [8]

Služba Mapbox Navigation Service představuje několik API sloužících k orientaci a navigaci na mapě. Zahrnuje služby využívající požadavky na směr cesty včetně přiřazení cesty na konkrétní silniční síť. Integraci API v rámci aplikace zajišťuje Mapbox Navigation SDK. Základ navigační služby tvoří následující součásti: [8]

- Mapbox Directions API – Základní API pro zobrazení vypočítané cesty na mapě i s ohledem na aktuální provoz a jiné mimořádné události.
- Matrix API – Slouží ke kalkulaci času a vzdálenosti nejrychlejší možné trasy. API vrací trvání v sekundách a vzdálenost v metrech. Čas i vzdálenost mezi body nemusí být symetrické, protože trasy se mohou lišit podle směru v důsledku jednosměrných ulic nebo omezení odboček. Například navigace z bodu A do bodu B může nabývat jiného trvání než navigace z bodu B do bodu A.
- Mapbox Optimization API – Rozhraní zajišťující optimalizaci tras pro různé dopravní prostředky. Pomocí tohoto API lze načítat trasy pro automobil, jízdu na kole či chůzi.
- Mapbox Isochrone API – Stará se o vykreslení obrysů a linií požadované cesty.

### 2.3.2 Camera2 API

Camera2 API je obsažena v balíčku *android.hardware.camera2* a poskytuje rozhraní k jednotlivým funkcím fotoaparátu zařízení. API vychází a rozšiřuje třídu *Camera* ze stejného balíčku jako Camera2 API.

Balíček modeluje kamerové zařízení jako kanál, který přijímá vstupní požadavky na zachycení snímku a vytváří obrázek podle požadavku na vyfotografování. Obdržené požadavky jsou zpracovávány systematicky v pořadí, kde v jednu chvíli může být spuštěno i více požadavků. Pro udržení plné snímkové frekvence na zařízeních je zapotřebí několik běžících, nevyřízených požadavků. [27]

Mezi nejzákladnější třídy pro konstrukci a obsluhu fotoaparátu obsažené v API patří: [27]

- *CameraDevice* – Reprezentuje vybranou kameru zařízení. Většina zařízení obsahuje alespoň dvě základní kamery (přední a zadní). Třída také obsahuje vnořenou třídu *CameraDevice.StateCallback*, která se stará o aktualizace stavu spuštěné kamery.
- *CameraManager* – Spravuje systémové služby pro detekování kamerového zařízení včetně uchovávání charakteristiky vybraného fotoaparátu,

- *CameraCaptureSession* – Nakonfigurovaná relace snímání obrazu, která je primárně určená pro třídu *CameraDevice*.
- *CaptureRequest* – Třída, která obsahuje nastavení potřebné pro zachycení jednoho snímku.
- *TotalCaptureResult* – Zahrnuje celkové výsledky zachycení obrazu z obrazového snímače.

K otevření fotoaparátu zařízení a ovládání jeho funkcí zařízení je potřeba vytvořit instanci třídy *CameraManager*. Tato instance nese informace o vlastnostech popisující hardwarové zařízení, dostupná nastavení a výstupní parametry zařízení. Tyto informace jsou poskytovány prostřednictvím vytvořené instance třídy *CameraCharacteristics* a její metody *getCameraCharacteristics(String cameraId)*, kde parametr *cameraId* představuje unikátní identifikátor vybrané kamery. Vybrané kamerové zařízení se následně uchovává v instanci třídy *CameraDevice*. [27]

K zachycení a promítání snímků z kamerového zařízení na obrazovce aplikace je zapotřebí nejprve vytvořit objekt typu *CameraCaptureSession*, který obsahuje metodu *createCaptureSession(SessionConfiguration config)* nesoucí parametr s konfigurací. V této konfiguraci se musí nacházet podporované velikosti a formáty dostupné ve fotoaparátu. Zpracovaný náhled kamery se zpravidla odesílá do instancí nebo potomků komponent *SurfaceView* a *TextureView* (skrze třídu *SurfaceTexture*). [27]

Zachycení snímku ve formátu *JPEG* nebo *RAW* lze provést pomocí instance třídy *ImageReader* s danými typy formátů. Aplikace potřebuje zkonstruovat požadavek na pořízení snímku pomocí instance třídy *CaptureRequest*, která definuje parametry snímání potřebné pro zachycení daného snímku. Jakmile je požadavek nastaven, lze jej předat relaci pro buď jednorázové zachycení snímku (fotografování), nebo pro nekonečně opakujícího se zachytávání snímků (natáčení). [27]

Po zpracování požadavku na zachycení snímku vytvoří kamerové zařízení objekt *TotalCaptureResult*, který obsahuje informace o stavu kamerového zařízení v době zachycení a použité nastavení. Tyto informace se mohou od požadavku lišit, pokud bylo nutné zaokrouhlování nebo použití odlišných parametrů. [27]

## 2.4 Testování aplikace

V rámci vývoje aplikace je zapotřebí neustále kontrolovat správnost nejen přidaných komponent a technologií, ale i těch stávajících. Při vytváření častých změn v projektu se může totiž kdykoliv stát, že některá funkcionalita ovlivní jinou funkcionalitu bez vývojářova vědomí. Je potřeba tyhle případné nedostatky odhalit a opravit co nejdříve. Pomocí testování se programátor v průběhu vývoje snaží vyvíjet aplikaci ve stabilním stavu. Mezi možnosti určené na testování aplikace lze zařadit: [19]

- Emulátor – Emulátor slouží jako pomůcka pro programátora při vývoji aplikací. Nahrazuje fyzické zařízení v podobě chytrého telefonu nebo tabletu a je využíván především k testování a ladění aplikací. Pomocí emulátoru lze testovat aplikaci na více různých verzích operačního systému a na obrazovkách s různým typem rozlišení. [19]

  Používané vývojářské prostředí Android Studio disponuje velmi kvalitními emulátory jako např. Nexus One či Galaxy Nexus. Lze zde nalézt i další zařízení s různými rozlišeními. Samotný emulátor si poté může programátor dále konfigurovat, případně si může takový emulátor sám vytvořit. Emulátoru lze nastavit příslušnou verzi Androidu, určit velikost RAM či zadat parametry přední nebo zadní kamery, kterou může uživatel propojit s kamerou počítače nebo kameru emulovat úplně.

- Fyzické zařízení – Po připojení mobilního zařízení k počítači a povolení vývojářského režimu na zařízení je možné otestovat aplikaci přímo na fyzickém zařízení. Pomocí tohoto typu testování si lze lépe představit fungování aplikace na reálném zařízení.

# 3 DISTRIBUCE APLIKACE

Po dokončení a úspěšném otestování aplikace lze aplikaci publikovat a zpřístupnit pro veřejnost. V rámci distribuce aplikace musí vývojář vzít v úvahu všechny dostupné cesty publikování a vybrat z nich pro něj tu nejvýhodnější.

Nejdříve je zapotřebí z projektu aplikace vytvořit balíček s příponou .apk. V tomto balíčku musí být v souboru AndroidManifest.xml definované atributy spojené s ikonou, názvem a verzí aplikace. Nakonec, před poslední kompilací aplikace, je nutné podepsat aplikaci vývojářským privátním klíčem. [19]

Mezi nejvyužívanější bezplatné formy distribuce patří:

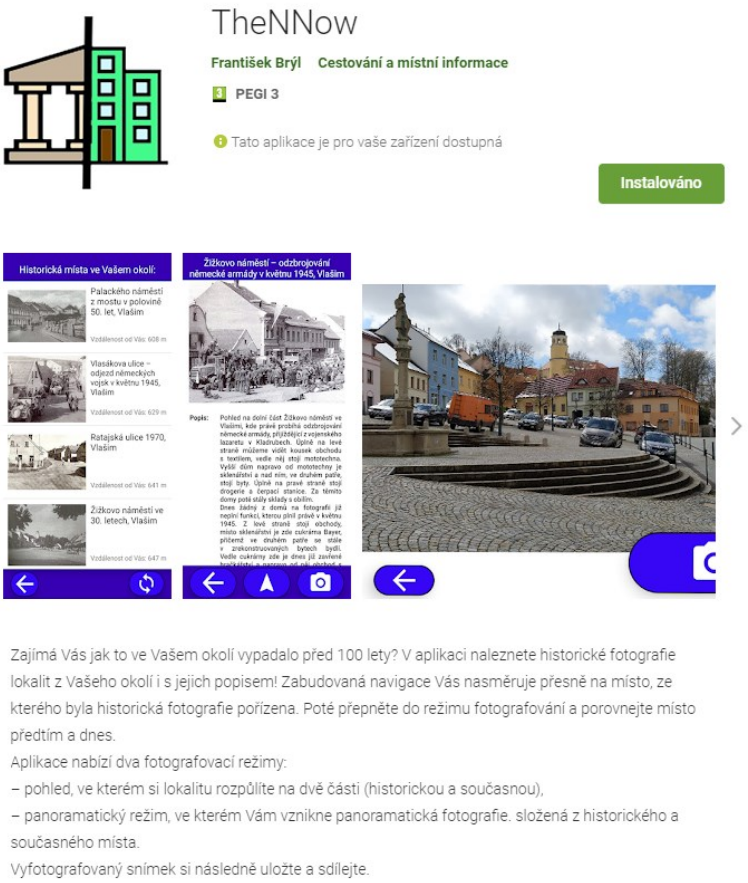
- Webové stránky – Pokud vývojář vlastní webové stránky, může je použít k distribuci aplikace. Alternativně lze aplikaci umístnit i na webové servery určené k nahrávání souborů (např. služba uloz.to). Uživatel si následně z těchto stránek stáhne do svého mobilního zařízení aplikační balíček APK. K instalaci balíčku je v zařízení nutné povolit instalaci aplikací z neznámých zdrojů.
- E-mail – Aplikaci lze šířit i prostřednictvím e-mailu pomocí rozesílání zpráv přátelům či jiným osobám. Do přílohy e-mailu se vloží aplikační balíček APK, který si uživatel stáhne a nainstaluje. Stejně jako u webových stránek, je i v této formě distribuce koncový uživatel povinen na svém mobilním zařízení zpřístupnit instalování aplikací z neznámých zdrojů.

Nejpopulárnější metodou k publikování aplikací je umístění aplikace do obchodu s aplikacemi Google Play. Obchod zahrnuje aplikace pro operační systém Android. K publikování aplikací na této platformě je potřeba se zaregistrovat na Google Play jako vývojář. Za zřízení tohoto typu účtu je vývojář nucen zaplatit 25 dolarů a souhlasit s distribučními podmínkami společnosti Google. Po vytvoření účtu může vývojář poskytovat aplikace ke stažení bezplatně nebo za vývojářem stanovenou cenu, přičemž společnost Google vývojáři účtuje 30% poplatek z každého prodeje. Podmínkou stanovení cen je vytvořený účet ve službě Google Wallet Mechant Center. Vývojáři je následně umožněno stanovit cenu aplikace, vkládat do ní reklamy či zavést do aplikace další formy prodeje. [19]

Nahrávání aplikace na platformu je realizováno ve formě balíčku APK. Před publikováním je potřeba vyplnit údaje o aplikaci. Po úspěšném nahrání aplikace do obchodu projde aplikace ještě řadou ověření, která kontrolují, zda je aplikace v souladu se smlouvami společnosti

Google. Publikovat lze pouze soubory APK, které nepřevyšují velikost 50 MB. Pokud vývojář potřebuje nahrát soubor s větší velikostí, musí soubory rozdělit na více souborů, přičemž každý soubor by měl být zacílen na určitou konfiguraci zařízení. [19]

Realizovaná aplikace s názvem TheNNow je umístěna v obchodě Google Play, ve kterém je poskytnuta zdarma ke stažení. V záznamu aplikace je uveden její popis a verze. Připojeny jsou i ukázky aplikace v podobě snímků obrazovek. Aplikaci je možné hodnotit či diskutovat v komunitním fóru.




**Obrázek 11 Aplikace TheNNow na Google Play Zdroj: Vlastní**

# ZÁVĚR

Cílem bakalářské práce bylo v teoretické části prezentovat návrh aplikace. Pomocí stanovení funkčních a nefunkčních požadavků na aplikaci se podařilo před samotnou realizací stanovit přesný směr, jakým by se aplikace měla udávat. V části o gelokačních metodách bylo vysvětleno základní fungování služeb GPS a GALILEO. Navíc bylo prezentováno i porovnání obou služeb a jejich využití v aplikacích. Následně byly popsány vybrané mapové podklady včetně vysvětlení principu zobrazování mapových dat na mobilních zařízeních. Na závěr první kapitoly byly představeny bezdrátové lokační technologie s jejich rozdělením a porovnáním.

V další kapitole bylo hlavním předmětem vytvoření mobilní aplikace pro operační systém Android. Na úvod kapitoly bylo představeno vývojové prostředí Android Studio, ve kterém byla aplikace realizována. V další části se prezentovala struktura aplikace, ve které byly zmíněny nezbytné soubory aplikace včetně porovnání dvou hlavních pohledů na strukturu aplikace ve vývojovém prostředí Android Studio. Následně se přešlo už k popisování samotné realizované aplikace, kde se vysvětlovaly třídy, technologie, použité API a grafické rozhraní aplikace. Závěr kapitoly se zaobíral možnostmi testovaní vyvtvořené aplikace.

Úspěšně vytvořená aplikace splňuje všechny požadavky, které byly navrženy před její realizací. Databáze aplikace v současné době disponuje přibližně 80 uloženými historickými lokalitami včetně jejich důvěryhodného popisu. Historické fotografie se vztahují zejména k městům Pardubice a Vlašim a lokalitám v oblasti Podblanicka. Jedno ze zamýšlených rozšíření aplikace do budoucna je rozhodně přidání více historických lokalit do databáze včetně rozšíření pole historických míst na další města a oblasti. Další funkcionalitou by mohla být i možnost registrace uživatelů v aplikaci. Registrovaní uživatelé by následně mohli komunikovat s vývojářem ohledně nepřesností popisů u daných lokalit či osobně přidávat další historické fotografie lokalit ze svého okolí do databáze.

# ZDROJE


[1] ERIKSSON, Ulf. Functional vs Non-Functional Requirements – Understand the Difference. *ReQtest: Requirements, Test Management, Bug Tracking Tool* [online]. 2012. [cit. 05.02.2021]. Dostupné z*:* https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

[2] KOVÁŘ, Pavel. *Družicová navigace: od teorie k aplikacím v softwarovém přijímači*. 1. vyd. Praha: ČVUT, 2016. ISBN 9788001059890.

[3] GENTILE, Camillo, Nayef ALSINDI, Ronald RAULEFS a Carole TEOLIS. *Geolocation techniques principles and applications*. Springer, 2013. ISBN 9781461418368.

[4] BABČANÍK, Jan. Jak funguje GPS? *Vývoj.HW.cz* [online]. 2006. [cit. 01.04.2021]. Dostupné z: https://vyvoj.hw.cz/teorie-a-praxe/jak-funguje-gps.html

[5] ŠRAJER, Michal. Co je to Galileo, proč je to u nás ta nejpřesnější navigace v telefonech a v jakých zařízeních jej najdete? *huramobil.cz* [online]. 22.08.2018. [cit. 1.04.2021]. Dostupné z: https://www.huramobil.cz/co-je-to-galileo--proc-je-to-u-nas-ta-nejpresnejsi-navigace-v-telefonech-a-v-jakych-zarizenich-jej-najdete-/blog-360/

[6] Mapbox. *Wikipedia: the free encyclopedia* [online]. [cit. 10.04.2021]. Dostupné z: https://en.wikipedia.org/wiki/Mapbox

[7] Getting started | Help. *Mapbox: Documentation* [online]. [cit. 10.04.2021]. Dostupné z: https://docs.mapbox.com/help/getting-started/

[8] API. *Mapbox: Documentation* [online]. [cit. 10.04.2021]. Dostupné z: https://docs.mapbox.com/api/

[9] SALAZAR, Jordi. *Bezdrátové sítě*. 1. vyd. Praha: ČVUT, 2017. ISBN 9788001061961.

[10] ŽÁRA, Ondřej. Firebase: krátké seznámení. *Zdroják – o tvorbě webových stránek a aplikací* [online]. 20.03.2015. [cit. 10.04.2021]. Dostupné z: https://zdrojak.cz/clanky/firebase-kratke-seznameni/

[11] Firebase. *Wikipedia: the free encyclopedia* [online]. [cit. 10.04.2021]. Dostupné z: https://en.wikipedia.org/wiki/Firebase

[12] Firebase Realtime Database. *Firebase: Documentation* [online]. [cit. 10.04.2021]. Dostupné z: https://firebase.google.com/docs/database

[13] Cloud Storage for Firebase. *Firebase: Documentation* [online]. [cit. 10.04.2021]. Dostupné z: https://firebase.google.com/docs/storage

[14] *JSON* [online]. [cit. 10.04.2021]. Dostupné z: https://www.json.org/json-cz.html

[15] Meet Android Studio. *Android Developers* [online]. [cit. 05.04.2021]. Dostupné z: https://developer.android.com/studio/intro

[16] Configure your build. *Android Developers* [online]. [cit. 05.04.2021]. Dostupné z: https://developer.android.com/studio/build

[17] *Gradle Build Tool* [online]. [cit. 05.04.2021]. Dostupné z: https://gradle.org/features/

[18] VÁVRŮ, Jiří a UJBÁNYAI, Miroslav. *Programujeme pro Android*. Praha: Grada, 2013. ISBN 9788024748634.

[19] LACKO, Ľuboslav. *Vývoj aplikací pro Android*. 1. vyd. Brno: Computer Press, 2015. ISBN 9788025143476.

[20] Projects overview. *Android Developers* [online]. [cit. 05.04.2021]. Dostupné z: https://developer.android.com/studio/projects

[21] Apache License. Dostupné z: http://www.apache.org/licenses/LICENSE-2.0.txt

[22] Displaying Images with the Picasso Library. *CodePath Android Cliffnotes* [online]. [cit. 10.04.2021]. Dostupné z: https://guides.codepath.com/android/Displaying-Images-with-the-Picasso-Library

[23] Picasso 2.71828 API. *Square Open Source* [online]. [cit. 10.04.2021]. Dostupné z: https://square.github.io/picasso/2.x/picasso/

[24] Displaying Images with the Glide Library. *CodePath Android Cliffnotes* [online]. [cit. 10.04.2021]. Dostupné z: https://guides.codepath.com/android/Displaying-Images-with-the-Glide-Library

[25] GitHub davemorrissey/subsampling-scale-image-view. *GitHub* [online]. [cit. 10.04.2021]. Dostupné z: https://github.com/davemorrissey/subsampling-scale-image-view/wiki

[26] API. *Česká logistika* [online]. [cit. 10.04.2021]. Dostupné z: https://www.ceskalogistika.cz/api/

[27] android.hardware.camera2. *Android Developers* [online]. [cit. 10.04.2021]. Dostupné z: https://developer.android.com/reference/android/hardware/camera2/package-summary

# PŘÍLOHY

Příloha A – *Zdrojový kód metody pro získávání dat z databáze ve třídě DatabaseHelper*

```
private ArrayList<Place> loadDataFromDatabase(DataSnapshot dataSnapshot) {
    int id = 1;
    Place place;
    int distance;
    int counter = 0;
    String title;
    placeList = new ArrayList<>();
    while(true) {
      title = (String)dataSnapshot.child(String.valueOf(id)).child("name").getValue();
      if(title == null){
        break;
      }
      String latitudeStr = dataSnapshot.child(String.valueOf(id)).child("latitude").getValue().toString();
      String longitudeStr = dataSnapshot.child(String.valueOf(id)).child("longitude").getValue().toString();
      String description = (String)dataSnapshot.child(String.valueOf(id)).child("description").getValue();
      if(description == null){
        description = "popis_mista";
      } else {
        if(description.contains("_n")){
          description = description.replace("_n","\n");
        }
      }
      double longitude = Double.parseDouble(longitudeStr);
      double latitude = Double.parseDouble(latitudeStr);
      distance = calculateDistance(longitude, latitude);
      if (distance == 0) {
        break;
      }
      if (counter < MAX_PLACES_TO_SHOW) {
        String image = dataSnapshot.child(String.valueOf(id)).child("image").getValue().toString();
        place = new Place(title, longitude, latitude, distance, image, description);
        placeList.add(place);
        counter++;
        if (counter == MAX_PLACES_TO_SHOW) {
          bubbleSort(placeList);
        }
        id++;
        continue;
      }
      for (int i = 0; i < counter; i++) {
        int oldDistance = calculateDistance(placeList.get(i).getLongitude(), placeList.get(i).getLatitude());
        if (oldDistance > distance) {
          String image = dataSnapshot.child(String.valueOf(id)).child("image").getValue().toString();
          place = new Place(title, longitude, latitude, distance, image, description);
          addPlaceToIndexInList(i, place, placeList, counter);
          break;
        }
      }
      id++;
    }
    if (counter < MAX_PLACES_TO_SHOW) {
      bubbleSort(placeList);
    }
    return placeList;
  }
```

Příloha B – *Zdrojový kód metod pro určení polohy uživatele ve třídě PlacesListActivity*

```
private void checkSettingsAndStartLocationUpdates() {
    if (hasGPSPermission) {
      LocationSettingsRequest request = new
LocationSettingsRequest.Builder().addLocationRequest(locationRequest).build();
      SettingsClient client = LocationServices.getSettingsClient(this);
      Task<LocationSettingsResponse> locationSettingsResponseTask = client.checkLocationSettings(request);

      locationSettingsResponseTask.addOnSuccessListener(locationSettingsResponse ->
startLocationUpdates());

      locationSettingsResponseTask.addOnFailureListener(e -> {
        if (e instanceof ResolvableApiException) {
          ResolvableApiException apiException = (ResolvableApiException) e;
          try {
            apiException.startResolutionForResult(PlacesListActivity.this, 1001);
            if (loadingDialog.isRunning()) {
              loadingDialog.dismissDialog();
            }
          } catch (IntentSender.SendIntentException ex) {
            ex.printStackTrace();
          }
        }
      });
    } else {
buttonRefresh.setBackgroundResource(com.thennow.Thennowapp.R.drawable.bg_light_blue_round_corners);
    }
  }

private void startLocationUpdates() {
    fusedLocationProviderClient.requestLocationUpdates(locationRequest, locationCallback,
Looper.getMainLooper());
  }

private void stopLocationUpdates() {
    fusedLocationProviderClient.removeLocationUpdates(locationCallback);
  }

private final LocationCallback locationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
      super.onLocationResult(locationResult);
      if (locationResult == null) {
        return;
      }
      for (Location loc : locationResult.getLocations()) {
        updateLatitudeAndLongitude(loc);
        stopLocationUpdates();
      }
    }
  };
```

Příloha C – *Zdrojový kód metody pro výběr optimálního rozlišení ve třídě CameraSplitActivity*

```
private Camera.Size getOptimalCameraPreviewSize(List<Camera.Size> sizes, int width, int height) {
    final double ASPECT_TOLERANCE = 0.15;
    double targetRatio = (double) height / width;

    if (sizes == null){
      return null;
    }

    Camera.Size optimalSize = null;
    double minDiff = Double.MAX_VALUE;

    int targetHeight = height;

    for (Camera.Size size : sizes) {
      double ratio = (double) size.height / size.width;
      if (Math.abs(ratio - targetRatio) > ASPECT_TOLERANCE){
        continue;
      }
      if (Math.abs(size.height - targetHeight) < minDiff) {
        optimalSize = size;
        minDiff = Math.abs(size.height - targetHeight);
      }
    }

    if (optimalSize == null) {
      minDiff = Double.MAX_VALUE;
      for (Camera.Size size : sizes) {
        if (Math.abs(size.height - targetHeight) < minDiff) {
          optimalSize = size;
          minDiff = Math.abs(size.height - targetHeight);
        }
      }
    }

    return optimalSize;
  }
```