F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Adaptivní cestovní aplikace pro motoristy

Alvina Lushnikova
Softwarové inženýrství a technologie

Leden 2022
Vedoucí práce: doc. Ing. Ivan Jelínek, CSc.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lushnikova** Jméno: **Alvina** Osobní číslo: **495549**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Adaptivní cestovní aplikace pro motoristy

Název bakalářské práce anglicky:

Adaptive travel applications for motorists

Pokyny pro vypracování:

Cílem bakalářské práce je adaptivní individualizovatelná aplikace pro motoristy, která umožní naplánovat cestu, modifikovat plán podle zadaných kritérií, automaticky upravovat trasu podle aktuální polohy a preferencí, nabízet zajímavá a důležitá místa v okolí, možnosti ubytování apod. A v neposlední řadě umožní zaznamenávat zážitky z cesty, ve formě foto a poznámek. Kritériem pro cestovní plán bude i finanční rozvaha.
Proveďte analýzu požadavků cílové skupiny uživatelů a analýzu již existující řešení. Na základě této analýzy navrhněte koncept aplikace a low-fi prototyp. Proveďte a vyhodnoťte UI low-fi test prototypu a navrhněte "klikatelné" high-fi prototypu. Navrhněte strukturu a architekturu aplikace, namodelujte doménový model, případy užití, a diagram nasazení. Navrhněte a vytvořte provizorní serverovou část aplikace a vytvořte frontend aplikace. Následně pak implementujte obchodní logiku na backendu a propojení s databází. Navrhněte, proveďte a vyhoďte test backendu a frontendu. Na závěr analyzujte výsledky bakalářské práce a případně navrhněte její další rozvoj.

Seznam doporučené literatury:

Documentation for app developers Dostupné z: https://developer.android.com/docs?hl=fi
Iyanu Adelekan. Kotlin Programming By Example. Packt Publishing Limited. 2020, ISBN: 978-1-78847-454-2
Jenifer Tidwell. Designing Interfaces: Patterns for Effective Interaction Design. O'Relly 2011
ISBN: 978-1-449-37970-4
Google Inc.: Material Design 2022 Dostupné z: https://material.io/develop/android
Fundamentals of testing Android apps Dostupné z:
https://developer.android.com/training/testing/fundamentals?hl=fi

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Ivan Jelínek, CSc. kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.07.2022** Termín odevzdání bakalářské práce: **10.01.2023**

Platnost zadání bakalářské práce: **19.02.2024**

doc. Ing. Ivan Jelínek, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studentky

Poděkování / Prohlášení

Děkuji především vedoucímu mé bakalářské práce panu doc.Ing. Ivanu Jelínkovi, CSc. za ochotu a pomoc při vypracování. Děkuji také rodině za podporu a trpělivost v průběhu studia.

Prohlašuji, že jsem práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Motodickým pokynem a etické přípravě vysokoškolských závěrečných prací. Nemám závažný důvod proti užívání tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským.

Abstrakt / Abstract

V uvedené práci je představena aplikace pro chytrý telefon na platformě Android. Aplikace je určená pro motoristy a umožní naplánovat cestu, modifikovat plán, nabízet zajímavá a důležitá místa v okolí, možnosti ubytování apod. V práci se porovnávají podobné existující aplikace, analyzují se funkční požadavky, které jsou poté navrženy pomocí diagramu případů užití. Součástí práce je i serverová část na Kotlin Spring Boot. K samotné komunikaci s backend aplikací je použito rozhraní REST. Uložení fotografií a dokumentů implementováno pomocí službu Firebase Storage. Při vývoji byl hlavní důraz kladen na implementaci nového deklarativního přístupu Jetpack Compose na platformě Android. Výsledná aplikace byla otestována pomocí programátorských a uživatelských testů. Výsledkem bakalářské práce je funkční prototyp, který je připraven pro další rozvoj.

Klíčová slova: vývoj aplikace, Kotlin, Spring Boot, Firebase, REST

This bachelor thesis is about an application for android smartphone. The application is targeted for car travelers and allows planning or modifying routes. It suggests POI in surroundings. It helps to find accommodation and so on. This thesis contains comparison of similar applications, analysis of functional requirements and use cases based on them. The server part of this project is a Kotlin Spring Boot application. REST is used to communicate with the backend. Management of photos and documents is implemented via Firebase Storage. A very important part of this project is implementation of declarative UI with help of Jetpack Compose framework on Android platform. Final implementation of the application is tested with automated and manual tests. The outcome of this thesis is a functional prototype of the application, allowing further development.

Keywords: android Application development, Kotlin, Spring Boot, Firebase, REST

Obsah /

Kapitola 1
Úvod

Jistě se vám někdy stalo, že jste se rozhodli jet s rodinou či kamarády na výlet. Vaším úkolem bylo naplánovat zajímavou trasu. To je ta nejsnadnější část celého výletu. Jistě si pamatujete, že žádný výlet neproběhl podle plánu. Již při prvních kilometrech vaše kamarády napadne malá odbočka, změní se počasí, v restauraci mají zavřeno atd. A proto byla sepsána práce o aplikaci, která umí podobné situace řešit. Cílem je navrhnout mobilní aplikaci pro cestovatele se specifickými požadavky. Aplikace má za úkol umožnit naplánovat cestu, automaticky ji modifikovat podle zadaných kritérií a zdokumentovat. Dále umožní uživateli nejen naplánovat trasu, ale především podle aktuální situace a preferencí trasu upravovat, nabízet zajímavá a důležitá místa v okolí, stejně tak možnosti ubytování a v neposlední řadě zaznamenávat zážitky z cesty. Kritériem pro cestovní plán je i finanční rozvaha. Aplikace bude vycházet z dat API Google Map. Správa uživatelů bude tedy realizována pomocí backend aplikace na Kotlin String Boot. Uživatel se bude moct přihlásit pomocí vlastního e-mailu nebo účtu Google. Nejdříve bude provedena rešerše stávajících řešení. V další části dojde k analýze potenciálních uživatelů a jejich požadavků. Následuje návrh uživatelského rozhraní a implementace, které vycházejí z analýzy a doporučených postupů při vývoji na platformě Android. Řešena je také implementace serverové a klientské části. Výsledná aplikace je podrobena testům na úrovni programátora a uživatele. Na závěr jsou vyhodnoceny výsledky, kterých se podařilo dosáhnout.

Kapitola 2
Analýza konkurenčních aplikací

V dnešní době většina aplikací určených pro cestovatele existuje převážně ve tvaru mobilních aplikací, které nemají webové rozhraní, což má jednoduchý důvod: Na cestě člověk tráví víc času se svým smartphonem než s notebookem. Mobilní aplikace také může do velké míry fungovat offline a snadněji využívat další funkce telefonu, jako je GPS a polohové služby geofencing (spuštění akce, pokud se uživatel přiblíží k nějakému místu.) Hledání konkurenčních aplikací v Google Play Market bylo prováděno na základě klíčových slov pomocí výrazů: "travel planner", "trip planner" a "road planner". Pro hledání byl použit chytrý telefon Sony Xperia XZ 1 s operačním systémem Android 9. Analýza ukázala, že se aplikace pro cestovatele převážně člení na tři kategorie:

- Sociální sítě pro zveřejňování cestovatelských recenzí (Travel Buddy, TripAdvisor) V podstatě se takové aplikace zabývají uživatelskými recenzemi, tedy přidělováním hodnocení různým místům. Hlavním obsahem jsou recenze hotelů, volnočasových aktivit a restaurací. Mapy slouží pouze k nalezení různých objektů infrastruktury, aplikace neumožňují vygenerovat trasu nebo odhadnout předpokládanou dobu cesty.
- Aplikace pro rezervaci (Trip.com, KAYAK) Tyto aplikace se často používají k rezervaci hotelů nebo letenek či jízdenek na vlak. Někdy slouží jako agregátory pro půjčovny aut v různých zemích. Často obsahují cestovní návrhy pro inspiraci. S takovými aplikacemi není možné něco plánovat bez nákupu letenek nebo rezervace ubytování na hotelu. Není možné samostatně vygenerovat cestovní trasu pro jízdu autem.
- Aplikace pro plánování cest na několik dní. (WanderLog, TripIt, Roadtrippers, TripCase, SygicTravel).

Tyto aplikace se používají k plánování cest s nabídkou návštěvy zajímavých míst. Aplikace jsou hluboce integrovány v Google Maps. V průběhu této analýzy bude největší pozornost věnována aplikacím, které jsou určeny převážně pro milovníky cestování autem nebo na motorce. Podobné aplikace budou analyzované podle kritérií popsaných v následujících bodech:

- Plánování cesty
  Tato část bude věnována hodnocení funkčnosti aplikace při plánování cesty na několik dní a vyhledávání zajímavých míst pro uživatele.
- Změna cestovního plánu během samotné cesty
  Tato sekce bude věnována hodnocení funkčnosti aplikace při změně trasy a vyhledávání infrastrukturních objektů v průběhu cesty.
- Hluboká integrace s Google Maps
  Odstavec hodnotí integraci s Google maps a využití integračních schopností aplikací.
- Zaznamenávání zážitků z cesty
- Tato část hodnotí schopnost aplikací ukládat zážitky z cest, fotografie a poznámky.
- Finanční rozvaha
- Hodnocení schopnosti aplikací ukládat informace o plánovaných a nakonec skutečně realizovaných finančních výdajích.

- Doplňující funkce
  Další funkce poskytované v aplikaci pro pohodlí cestujících.
- Uživatelské rozhraní
  Vyhodnocení uživatelského rozhraní a intuitivnosti práce s aplikací.
- Sdílení itineráře mezi více uživateli

2.1 WanderLog

Aplikace WanderLog [1] poskytuje možnost plánovat itinerář na několik dní a také nabízí zajímavá místa cesty, ale pouze na začátku nebo na konci cesty. Veškerá odpovědnost za budování trasy mezi městy leží na Google maps, které se otevírají jako samostatná aplikace. Během cesty není možné na již vygenerované trase zadávat další průběžná místa před samotnou finální destinací. Je nutné původní trasu zcela vyrušit a všechny případné průběžné body zadat před vygenerováním nové trasy. Integrace s Google maps představuje pouze přenos dat od začátku do konce trasy, další pohyb po trase a geolokace mobilu nemá vliv na původní plán. Aplikace také umožňuje zobrazit místa k návštěvě na mapě bez nutnosti přechodu do prostředí Google maps. Aplikace žádá uživatele, aby zadal čas začátku a konce návštěvy různých míst, přičemž nezohledňuje možné časové překryvy návštěvy různých míst v témže čase a nezohledňuje, kolik času zabere samotná cesta na vybrané místo. Ke každému cíli může uživatel přiložit dokument nebo fotografii, chybí však cestovní zpráva a zaznamenávání zážitků z cesty. Aplikace má vynikající systém finančního účetnictví, ke každé položce je možné přidat hodnotu, přiřadit kategorii výdajů a také sdílet náklady s přáteli, pokud jsou do aplikace pozváni. Další zajímavou funkcí je integrace s e-mailem pro vyhledávání rezervací. Funkčnost aplikace je poměrně široká a není vždy intuitivní. Problematická se jeví např. ve funkci komunikace se všemi přáteli. Při prvotním používání aplikace se neobjeví základní průvodce orientace v aplikaci. Neomezenou funkčnost aplikace nabízí zpoplatněný tarif účtu “PRO”.

2.2 TripIt

Aplikace TripIt [2] slouží pro organizaci cestovních plánů. Aplikace poskytuje možnost plánovat itinerář na několik dní a nabízí snadnou správu všech provedených rezervací. Dále aplikace nevyžaduje přístup k emailu, ale žádá pouze o zaslání rezervačních údajů na konkrétní adresu, což svědčí o dobrém přístupu k nakládání s osobními údaji uživatele. Aplikace nenavrhuje místa k návštěvě, a to jak v samotných destinacích, tak v průběhu cesty spravované v itineráři. Integraci s Google maps využívá jenom při plánovaní samotné trasy. Ke každému cíli může uživatel přiložit dokument nebo fotografii, chybí však cestovní zpráva a zaznamenávání zážitků z cesty. Aplikace nabízí možnost přehledu veškerých finančních nákladů spojených s cestou. Další zajímavou funkcí je “Alert”, která připomíná důležité události. Přidávání aktivit v aplikaci není příliš pohodlné, jelikož je potřeba znát přesnou adresu míst, která se mají navštívit. V aplikaci chybí krok pro zaškolení uživatele. Některé funkce jsou k dispozici pouze za poplatek.

2.3 Road trippers

Aplikace Road trippers [3] je určena pro plánování krátkých tras, nelze plánovat trasu na více dní s přidáním zajímavých míst, aktivit a hotelů. Aplikace přitom perfektně

vyhledává aktivity podle kategorií po celé trase nebo v blízkosti aktuální polohy zařízení. Aplikace v sobě integruje Google maps, kdy nabízí pokročilý filtr aktivit. Nemá funkce ukládání fotografií a poznámek z výletu a rovněž postrádá funkci finančního plánování. Vzhledem k omezené funkčnosti má aplikace spíše uživatelsky přívětivé rozhraní. Obsah je v současné době zaměřen pouze na USA, Kanadu, Austrálii a Nový Zéland.

2.4 Sygic Travel

Pomocí plánovače [4] je snadné naplánovat každý den výletu. Aplikace čerpá data z OpenStreetMap.org a nepoužívá Google maps. Aplikace odhadne čas dopravy do místa destinace a také, kolik času zbývá pro návštěvu nějakého zajímavého místa. Aplikace dále nabízí již naplánované šablony pro zájezdy do turisticky nejoblíbenějších měst a rovnou integruje plán výletu na požadovaný den. Mezi návštěvami jednotlivých míst aplikace perfektně vygeneruje trasu a vyhodnotí předpokládaný čas pro přesun. Neumožňuje ovšem nastavit čas příjezdu do města ani začátek návštěvního dne. Nepočítá s možností zpoždění a stejně tak s rozdílnými otevíracími dobami pro jednotlivá místa. Uživatel tak nemá přesnou informace, která místa ještě reálně navštívit může a která nikoliv. Aplikace také umožňuje přidat poznámku k bodům cesty bez možnosti přiložit dokument nebo fotografii. Nelze zaznamenávat zážitky z cesty, a pak se podívat a sdílet cestovní zážitky s kamarády. Doplňkovými funkcemi jsou 360° videa nejznámějších památek světa a offline mapy celého itineráře, mapy metra a přehled počasí. Uživatelské rozhraní je velmi pěkné a intuitivní.

2.5 Srovnání aplikací

Zde je srovnání aplikací na základě jejich funkcionalit.

Name	Wander Log	TripIt	Road Trip	Sygic Travel
Počet stažení(v tisících)	>100	>5000	>1000	>1000
Uživatelské hodnocení	4.9	4.5	3.0	4.4
Jazyk	en	en, fr, de	en	de-foltní
Placené funkce v aplikaci	✓	✓	×	✓
Pohodlné plánování cesty na několik dnů	✓	✓	×	✓
Změna plánu v procesu cestování	×	×	✓	×
Hluboká integrace s Google Maps	×	×	✓	×
Zaznamenávání zážitků z cesty	×	×	×	×
Placené funkce v aplikaci	✓	✓	×	✓
Finanční rozvaha	✓	×	×	×
Doplňující funkce	✓	✓	×	✓
Pohodlné uživatelské rozhraní	✓	×	✓	✓
Sdílení itineráře mezi více uživateli	✓	✓	✓	✓

Tabulka 2.1. Srovnání aplikací.

Hlavním cílem analýzy konkurenčních řešení bylo zjistit silné a slabé stránky aplikací pro cestovatele a také jejich užitečnost pro milovníky cestování autem nebo na motorce. Analyzované aplikace mají spoustu zajímavých funkcí, např. nabídku míst v okolí trasy, změnu původního plánu, odhad času dopravy do místa destinace, šablonu pro zájezdy, zaznamenávání zážitků z cesty a přehled veškerých finančních nákladů. Aplikace jsou poměrně často stahované a dobře hodnocené. Bude se z nich vycházet během plánování a implementací vlastního řešení. Nicméně analyzované aplikace mají společnou chybu v generování rozvrhu pro uživatele, jelikož se nedá nastavit počáteční čas návštěvního dne a také se nezohledňují možné časové překryvy návštěvy různých míst v témže čase. Funkcionalita aplikací nepočítá s případným zpožděním a stejně tak s rozdílnými otevíracími dobami pro jednotlivá místa, na která neupozorní uživatele. Dalším důležitým poznatkem je to, že ani jedna aplikace neumožňuje porovnání s naplánovanými finančními náklady a skutečně realizovanými. Aplikace neumožňuje zveřejnit plán cesty s poznámkami a fotkami pro inspiraci dalších uživatelů. Jak analýza ukázala, žádná z aplikací plně neuspokojuje všechny požadavky. Neexistuje řešení, které dovolí snadně naplánovat a využít itinerář s jeho následným zveřejněním.

Kapitola 3
Analýza funkčnosti aplikací

3.1 Analýza potenciálních uživatelů

Aplikace je určena především pro lidi, kteří rádi cestují autem nebo na motorce. Funkce aplikace však budou vhodné i pro řidiče na kratších cestách za jakýmkoliv účelem, po městě nebo mezi městy, kde je důležité navštívit několik míst a mít přehledný plán, který sleduje polohu smartfonu a nabízí různé možnosti včetně úpravy plánu. Aplikace bude zajímavá i pro ty, kteří rádi cestují bez konkrétního plánu, a určí si pouze hlavní destinace a některá místa, která musí navštívit. Pro řidiče může být velmi praktická funkce vyhledání například nejbližší čerpací stanice přímo ve směru jízdy. Pro řidiče je k dispozici přepnutí na navigaci, která je součástí aplikace Google mapy. Funkce zaznamenávání zážitků z cesty je vhodná pro ty, kteří po cestě rádi sdílejí informace na sociálních sítích, ale neradi znovu vyhledávají všechny body trasy a rozpomínají se na detaily. Aby měli cestující všechny cestovní doklady, jízdenky, vstupenky, kupony atd. po ruce, je k dispozici funkce uložení (použití offline). Pro spořivé lidi bude funkce finančního přehledu velmi důležitá, zejména pro porovnání plánovaných výdajů se skutečnými. A pro skupinu lidí, kteří nejsou součástí stejné rodiny a mají tak různé rozpočty, bude nutné rozdělit náklady a na konci cesty mít přesný soupis všech výdajů.

3.2 Analýza požadavků

V kapitole bude pojednáno o sestavení analýzy funkčních požadavků. Při vývoji libovolného softwaru je důležité nejprve přesně definovat, co může uživatel se softwarem dělat, tj. funkčnost aplikace, která se musí nejen přesně popsat, ale také definovat její omezení. Vytvoří se tak ucelený seznam funkčních požadavků aplikace. Nefunkční požadavky poslouží jako doplněk funkčních. Popisují další nezbytné vlastnosti potřebné vzhledem k prostředí a kontextu.

3.2.1 Registrace

- FP1:
  Uživatel při prvním přihlášení vybere způsob registrace ze tří možných: Standardní přihlášení přes email + heslo, přihlášení pomocí Google Account.
- FP2:
  Uživatel si může obnovit zapomenuté heslo.

3.2.2 Plánování

- FP3: Aplikace nabídne uživateli naplánovat trasu, vybrat název, termín cesty a další potřebná data k jejímu výchozímu bodu. Pokud uživatel plánuje jít na výlet s kamarády, může se jednoduše přihlásit. Následně dojde k uložení cesty v účtu uživatele.
- FP4: Uživatel může smazat plán cesty, pokud je jejím vlastníkem.

- FP5: V cestovním plánu se uživatel může podívat na jednotlivé dny a přidávat k nim různé aktivity.
- FP6: Aktivity jsou rozděleny podle typů: Stravování, ubytování, zastávka na čerpací stanici, návštěva památek a jiné účely cesty.
- FP7: Uživatel může vyhledávat návštěvní místa několika způsoby: Podle názvu, své polohy a manuálně přímo na mapě.
- FP8: V aktivitách může uživatel zobrazit potřebné informace o místě, přidat dobu trvání návštěvy, přidat poznámku včetně nákladů, dále přidat fotografii místa nebo potřebný soubor.
- FP9: Každá další aktivita, kterou uživatelé přidají do rozvrhu, má čas zahájení shodný s časem ukončení předchozí aktivity s potřebnou časovou rezervou k přesunu. Pokud je cesta delší než jeden kilometr, je automaticky nastavena jízda pomocí dopravních prostředků. V případě kratší cesty se automaticky nastavuje pěší chůze.
- FP10: V části rozvrhu dne je uvedeno, kolik času vyžadují všechny činnosti a čas na přesun mezi nimi.
- FP11: U každého výletu si můžete prohlédnout všechny fotografie a dokumenty z akce, stejně jako seznam účastníků, včetně pozvání nového účastníka, a seznam všech dluhů.

3.2.3 Použití aplikace při cestování

- FP12: Pokud se otevírací doba pro jednotlivá místa nevejde do změněného časového rámce návštěvy místa, tak aplikace uživatele upozorní a navrhne změnu plánu.
- FP13: Aplikace umožní najít užitečná místa v okolí cesty ve směru k destinací, jako je např. čerpací stanice.
- FP14: Do plánu lze přidat také infrastrukturní objekty, aby bylo možné zohlednit všechny cestovní náklady.
- FP15: Během cestování aplikace umožní přidávat fotografie z alba telefonu.

3.2.4 Sdílení aplikace

- FP16: Uživatel může každou cestu sdílet s ostatními uživateli.
- FP17: Uživatel může přidat nového účastníka.

3.2.5 Reporty

- FP18: Aplikace dokáže zobrazit všechny výdaje rozdělené podle aktivit, a všechny dluhy budou vypočítány tak, aby uživatelé museli po cestě mezi sebou provádět co nejméně převodů.

3.3 Nefunkční požadavků

- NFP1: Grafické rozhraní bude vytvořeno podle standardu Android Material Design [5], a musí být jednoduché a intuitivní.
- NFP2: Aplikace bude dostupná pro zařízení se systémem Android.
- NFP3: Aplikace vyžaduje minimálně verzi 8.0
- NFP4: Aplikace musí splňovat bezpečnostní požadavky pro správu osobních dat uživatele.
- NFP5: Aplikace musí být škálovatelná pro co nejširší zastoupení přístrojů s rozdílnými poměry stran obrazovek.

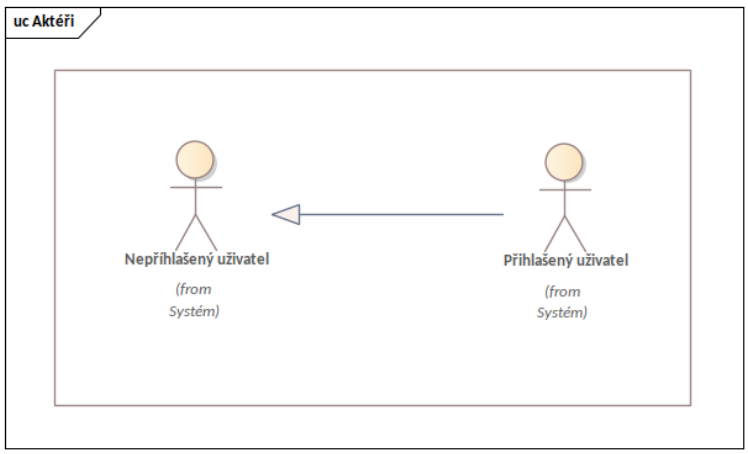
- NFP6: Aplikace bude vyžadovat přístup k poloze telefonu pro splnění požadavků F7 a F13, ale základní funkce aplikace budou dostupné i bez uvedeného povolení.
- NFP7: Aplikace bude vyžadovat přístup k úložišti souborů v telefonu pro splnění F7 a F13, ale základní funkce aplikace budou dostupné i bez povolení.
- NFP8: Data uložená v aplikaci budou dostupné i bez připojení k internetu.

3.4 Model případů užití

Use case model vyjadřuje, kdo bude jakým způsobem používat IT systém, a přesně definuje uživatele systému a jejich práva. Případy používání vychází výhradně ze zadání funkčních požadavků. V kapitole se bude řešit problematika tzv. mapování uživatelských požadavků pro jednotlivé modelové případy (use cases). Model zahrnuje: Aktéry, tj. entity, které mají roli v rámci systému; případy užití - jednotlivé funkčnosti, které popisují chování systému a vztahy mezi nimi. Diagram případů užití je znázorněn v příloze B.

3.4.1 Aktéři

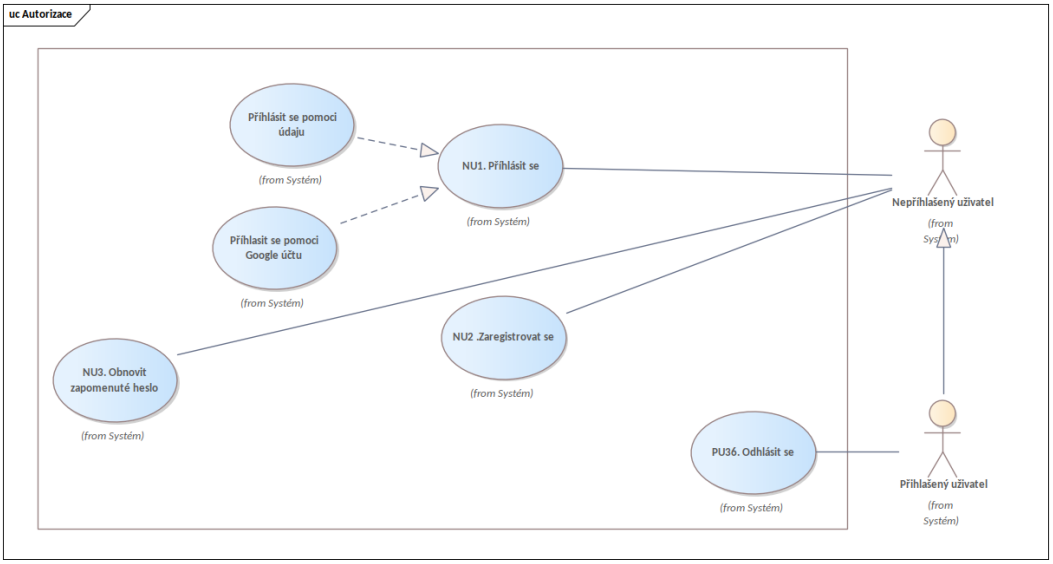
Systém rozlišuje dva typy uživatelů. Nepřihlášený uživatel se může registrovat, přihlásit a požádat o obnovení zapomenutého hesla, zatímco přihlášený uživatel má přístup ke všem funkcím aplikace.



Obrázek 3.1. Diagram případu užití - část 1 (Aktéři)

3.4.2 Autorizace

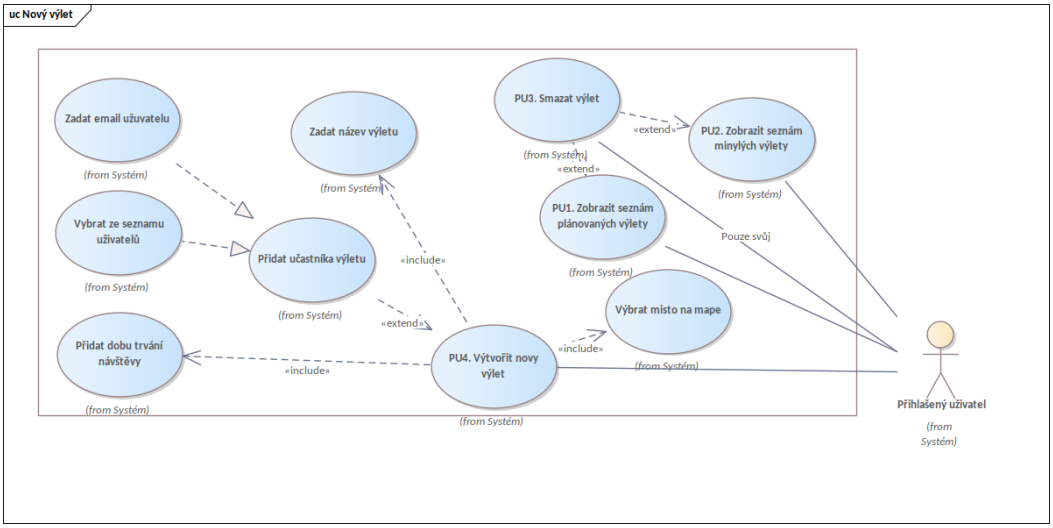
Aplikace umožní uživateli se přihlásit (NU1) pomocí e-mailu/hesla nebo účtu Google. Systém umožní registraci (NU2), pokud jsou splněna všechna potřebná pravidla (správné vyplnění polí registrace) a na serveru nebude zaregistrován žádný uživatel s danou e-mailovou adresou. Pokud uživatel zapomene své heslo (NU3), systém mu umožní získat přístupový kód pro změnu hesla e-mailem. Aplikace umožní přihlášenému uživateli se odhlásit (NU36).



Obrázek 3.2. Diagram případu užití - část 2 (Autorizace)

3.4.3 Novy výlet

Aplikace umožní prohlédnout si všechny absolvované nebo iniciované výlety. Výlety jsou seřazeny podle data naplánování (PU1) a uskutečnění (PU2). Uživatel dále může svůj výlet definitivně smazat (PU3). Aplikace umožní vytvořit nový výlet (PU4) tak, že jej pojmenuje a název se bude řídit pravidly systému. Může nastavit datum začátku a konce výletu, dále vybrat na mapě výchozí bod a volitelně přidat někoho ze seznamu přátel, se kterými již cestoval, případně zadat e-mail osoby, se kterou chce výlet sdílet.

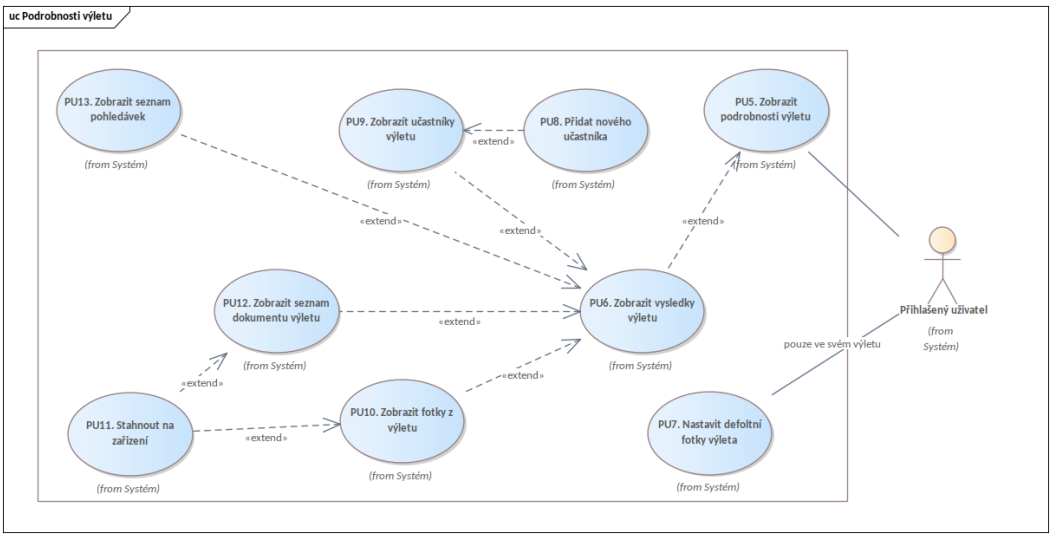


Obrázek 3.3. Diagram případu užití - část 3 (Výlet)

3.4.4 Podrobnosti výletu

Aplikace umožní prohlížet informace o výletu (PU5), prohlížet všechny fotografie a dokumenty (PU9-10) přiřazené k aktivitám na výletu a stahovat soubory do svého telefonu (PU11). Dále si prohlížet všechny účastníky výletu (PU6), a to včetně těch, kteří ještě nepřijali pozvání nebo se v aplikaci nezaregistrovali. Stejně tak zvát nové

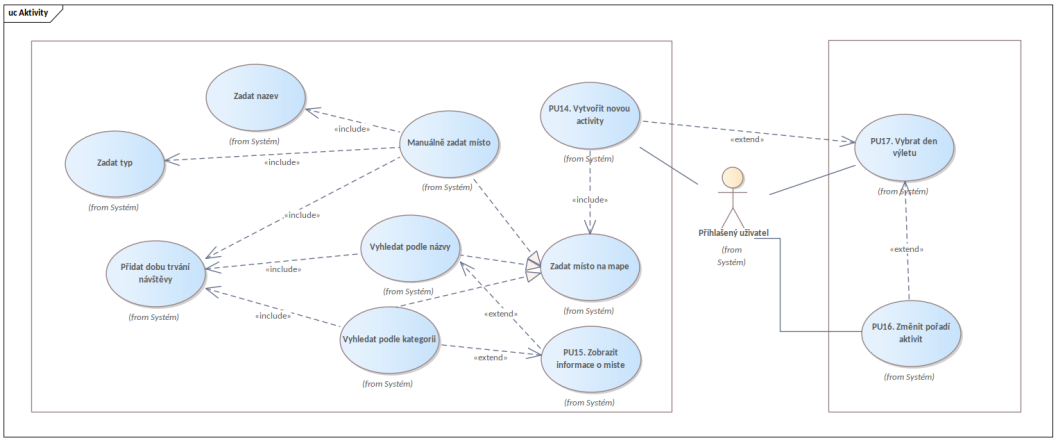
účastníky výletu (PU7), prohlížet přehled výdajů, včetně toho, kdo a kolik komu dluží v různých měnách (PU8). Aplikace umožní nastavit výchozí fotky pro každý výlet.



Obrázek 3.4. Diagram případu užití - část 4 (Podrobnosti výletu)

3.4.5 Den výletu a nová aktivita

Aplikace umožní zobrazení výletu a rozvrh jednotlivých aktivit seřazený podle dnů - výběr konkrétního dne (PU17) cesty a přidání nové aktivity (PU14) dvěma způsoby. Prvním je umístění bodu na mapu nebo samostatně, postupným uvedením názvu, kategorie a doby trvání události. Druhým způsobem je výběr z navržených objektů na mapě, a to buď v rámci navržených kategorií, nebo pomocí vyhledávacího dotazu. Aplikace umožní prohlížet dostupné informace o místě (PU15), fotografie nebo odkazy na stránky (s možností prokliku) a poté vybrat pouze dobu trvání události. Aplikace umožní změnit pořadí aktivit (PU16)

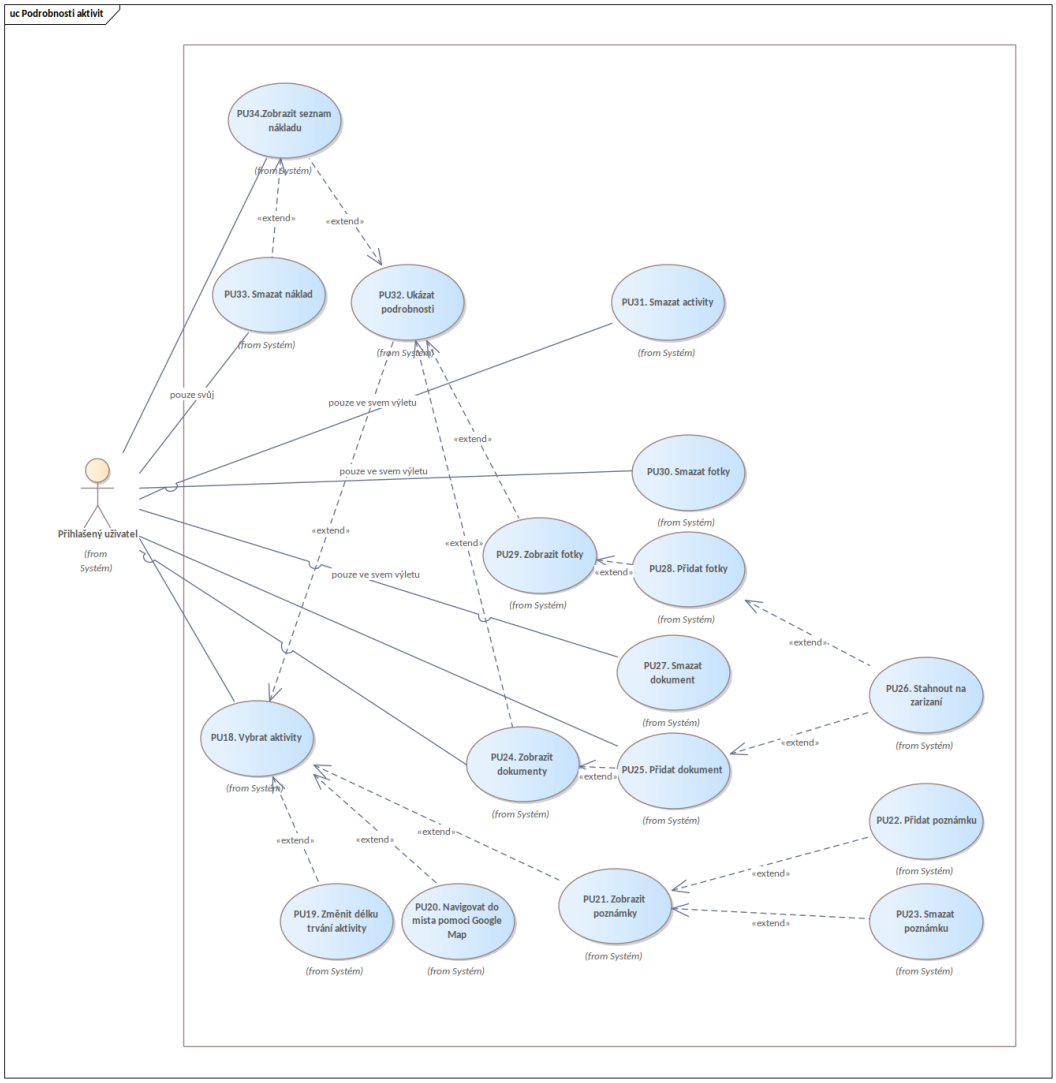


Obrázek 3.5. Diagram případu užití - část 5 (Aktivity)

3.4.6 Podrobnosti aktivit

Aplikace umožní vybrat aktivity (PU18) a prohlížet informace o aktivitě (PU32), přidat k nim dokument (PU25), fotografii (PU28) nebo stáhnout ty dokumenty fotografie (PU26), které do aktivity připojili oni sami nebo jiní uživatelé. Aplikace umožní procházet přehledem výdajů (PU34), zkontrolovat jejich výši v uživatelem definované měně.

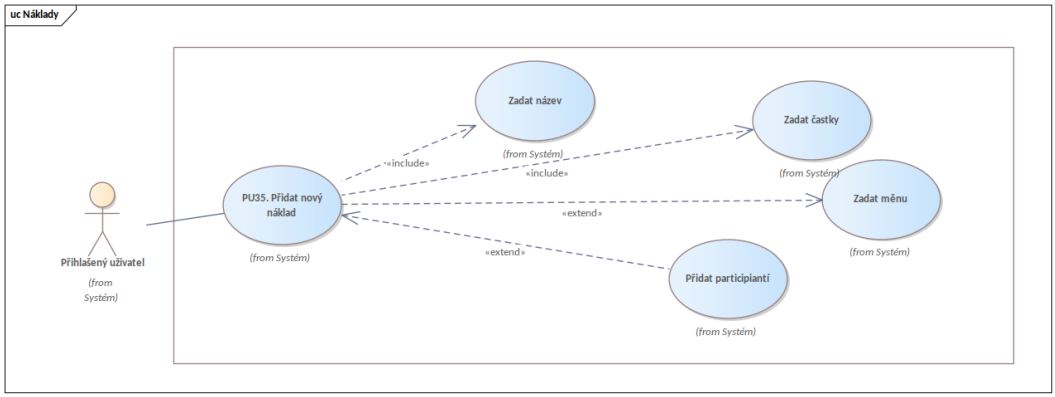
Aplikace umožní zobrazit všechny účastníky akce, mezi které jsou náklady rozděleny, a přesnou částku, kterou každý z nich dluží účastníkovi, který zaplatil na daném místě. Aplikace umožní navigovat (PU20) na zadané místo pomocí Google Map nebo změnit dobu trvání událostí (PU19). Aplikace umožní zobrazit všechny poznámky k aktivitám (PU21) nebo si napsat vlastní(PU22), a následně smazat pouze své poznámky (PU23).



Obrázek 3.6. Diagram případu užití - část 6 (Podrobnosti aktivit)

3.4.7 Náklady

Aplikace umožní přiřadit nový výdaj zadáním částky, měny, názvu a přiřadit(PU35) k němu účastníky, se kterými uživatel chce daný náklad sdílet. Aplikace dále umožní výdajovou položku smazat(PU33), ale pouze svoji.



Obrázek 3.7. Diagram případu užití - část 7 (Náklady)

Kapitola 4
Rozbor možných implementačních prostředků

V následující kapitole budou rozebrány možnosti implementace mobilní aplikace, včetně analýzy statistiky nejpoužívanějších operačních systémů, základních typů mobilních aplikací a programovacích jazyků.

4.1 Základní typu mobilních aplikací

- **Webová aplikace**
  Jedná se o jednoduchou webovou stránku, která se spouští prostřednictvím prohlížeče, přičemž podoba mobilního prostředí zůstává téměř nezměněna, což usnadňuje orientaci. Kód je tvořen za pomoci technologií HTML, CSS či JavaScrip. Výkon takové aplikace je přímo závislý na možnostech prohlížeče. Výhodou vývoje a údržby takové aplikace je jednoduchost a časová nenáročnost. Optimalizace standardní webové stránky pro smartphone není náročný proces. Snadná dostupnost je další výhodou, uživatel nic nemusí stahovat a instalovat bez schvalování. Protože není možné nahrát aplikaci do obchodů s aplikacemi, nemusíte ji ani podstoupit procesu schvalování. Tato varianta je proto vhodnou volbou zejména v případě vývoje malé a nenáročné aplikace, která nepracuje s tolika daty a nepotřebuje rozšířený přístup např. k fotoaparátu či čtečce otisku prstu.
- **Nativní** aplikace patří mezi nejstarší podoby aplikace. Jejím základním znakem je, že musí být vyvíjena pro každou platformu zvlášť. Vývoj takových aplikací ve dvou systémech najednou (Android a iOS) vyžaduje hodně času a také dva týmy vývojářů. Údržba takových aplikací je také poměrně náročná. Na druhou stranu ovšem nativní aplikace nabízejí nejlepší podmínky pro práci offline a rychlejší odezvu na uživatelské příkazy. Navzdory neustálému vývoji alternativních řešení nabízí největší výkon, protože vývojář může naplno využívat potenciálu hardwaru smartphonu či tabletu. Proto se v praxi používají nativní řešení při vývoji aplikací s vysokými nároky na uživatelské požadavky.
- **Hybridní** aplikace využívají jak nativní, tak webové technologie, které se používají pro vývoj webových aplikací. Umožňují jim fungovat na různých platformách. Na rozdíl od nativní ji vývojáři připravují pro obě platformy najednou, tedy píší pouze jeden kód, který se následně konvertuje pro iOS a Android. Výhodou je menší časová náročnost, vývoj je rychlejší a levnější než u nativních aplikací a mají stejné funkce i na odlišných operačních systémech. Na druhou stranu hybridní aplikace kombinuje nevýhody nativních a webových aplikací. Nemá přístup k hardwaru daného zařízení a potřebuje pluginy, které tuto komunikaci umožní. Použité pluginy, knihovny a frameworky pak musí být kompatibilní s aktuální verzí operačního systému, ať už je to Android nebo iOS. Není proto možné reagovat na novou verzi operačního systému, dokud nedojde k podpoře ze strany tvůrce frameworku. Výběr typu aplikace není jednoduchý proces, protože při výběru typu je třeba v první řadě zohlednit nejen stávající požadavky na funkčnost aplikace, ale také její budoucí vývoj, který může

vyžadovat maximální přístup k hardwaru daného zařízení. Aplikace, která je předmětem tohoto semestrálního projektu, má ve svých funkčních požadavcích nejen pristup k hardware, ale také spolupráci s aplikací Google Maps [6–8].

4.2 Operační systémy pro mobilní aplikace

Při vývoji nativní mobilní aplikace je důležité seznámit se se statistikami podílu mobilních operačních systémů na celosvětovém trhu [9]. Při výběru operačního systému je také nutné vycházet z předpokladu, kde se aplikace bude používat častěji, zda v telefonech, nebo v tabletech. Předpokládá se, že cestovní aplikace bude používána na telefonech, proto budu pracovat se statistikami zaměřenými pro operační systémy v mobilních telefonech. Statistiky uvádějí, že v roce 2022 operační systém Android používalo okolo 72,06 % uživatelů. Další nejrozšířenější systém je iOS s podílem přibližně 27,29 % uživatelů. Ostatní systémy Windows, Samsung, Harmony OS, KaiOS nedosahuje ani jednoho procenta, a proto je nebudu brát v potaz. V České Republice se statistiky příliš neliší od celosvětových údajů (Android 69,92 % a iOS 29,51 %) [10].

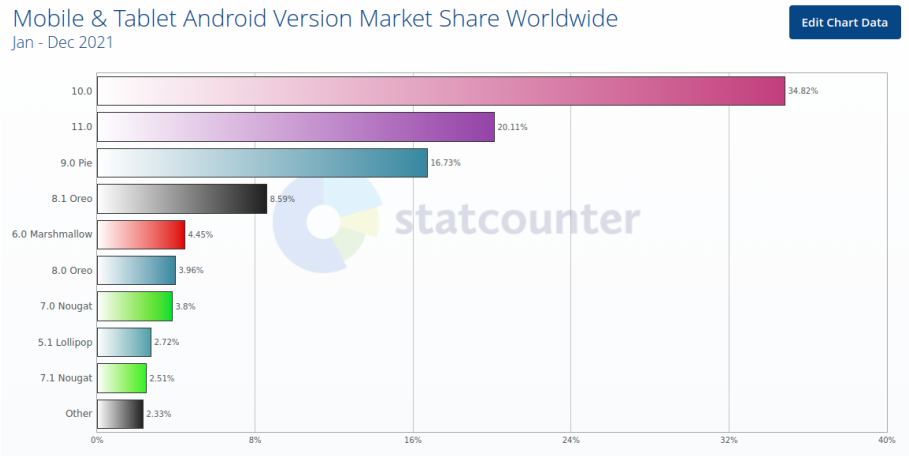
Historie systému Android začíná v říjnu 2003 – dlouho předtím, než se pojem smartphone stal všudypřítomným, a několik let před oznámením prvního iPhonu a iOS společností Apple. V roce 2005 se začala psát další významná kapitola historie systému Android, kdy původní společnost koupil Google. Zakládající členové pokračovali ve vývoji operačního systému pod novým vlastníkem. Poté se rozhodli použít jako základ operačního systému Android Linux. Klíčovým rozhodnutím v historii systému Android byl závazek společnosti Google učinit z Androidu operační systém s otevřeným zdrojovým kódem. To mu umožnilo získat velkou oblibu u výrobců telefonů třetích stran [11].

Stojí za zmínku, že rozmanitost zařízení, která používají platformu Android, je velmi široká. Systém Android je nainstalován v telefonech od celé řady výrobců z celého světa. Výrobci mohou volně upravovat jádro systému, protože Android je otevřená platforma. Na rozdíl od společnosti Apple, kde je systém iOS aktualizován pravidelně a na všech podporovaných zařízeních, může na systému Android fungovat více zařízení s různými verzemi, což představuje možný problém s podporou na různých zařízeních. Různé verze mají v mnoha ohledech odlišnou logiku. Například před verzí 6.0 nemusely aplikace žádat o každé oprávnění zvlášť (přístup k fotoaparátu, mikrofonu atd.). Oprávnění byla uvedena v Google Play a předpokládalo se, že je uživatel před stažením zná. Od verze 6.0 je třeba při spuštění aplikace požádat o každé povolení zvlášť. Pokud tedy při vývoji mobilní aplikace pro Android neimplementujete oba způsoby udělení oprávnění, nebude taková aplikace fungovat ani před verzí 6.0, ani později.

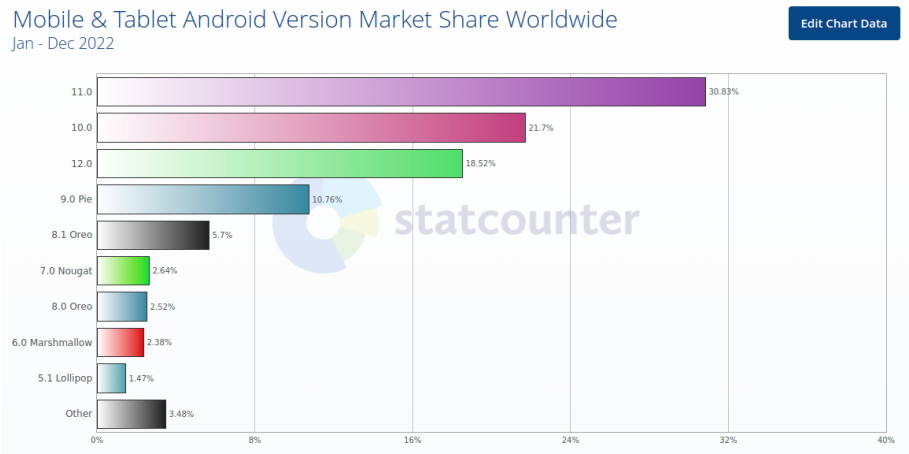
Při vývoji aplikací pro Android je třeba brát v úvahu architekturu samotné aplikace. Na rozdíl od systému iOS, kde aplikace tvoří architektonicky jeden celek, jsou v systému Android sestaveny z logicky oddělených a samostatných částí - aktivit a fragmentů. Jak ukazuje praxe, první verze aplikace by měla být vyvinuta na systému Android, protože přístup do obchodu s aplikacemi je pro vývojáře snazší a lze získat zpětnou vazbu od většího počtu uživatelů.

4.3 Výběr minimální podporované verze

Vývojáři aplikací pro systém Android se potýkají s vážným problémem, jak vybrat minimální podporovanou verzi. Tento výběr by se měl řídit statistikami používání jednotlivých verzí telefonu, náročností aplikace na systémové požadavky a také cílovou skupinou potenciálních uživatelů.



Obrázek 4.1. Statistiky verzí Androidu 2021 rok [12]



Obrázek 4.2. Statistiky verzí Androidu 2022 rok [9]

Jak je patrné ze statistik, starší verze systému Android rychle ztrácejí uživatele, zřejmě spolu se zastaráváním chytrých telefonů, na kterých byly nainstalovány. Podporovat verze nižší než 8 v 2022 roce už nemá smysl, protože jejich kumulativní podíl nepřesahuje 6 %. V důsledku toho byla jako minimální podporovaná zvolena verze 8.0 s rozhraním min API 26, která umožní použití aplikace na přibližně 94 % zařízení.

4.4 Výběr jazyka pro vývoj

K napsání aplikace pro Android lze použít několik programovacích jazyků: Java, Kotlin, Python (nástroje Kivy BeeWare), JavaScript (React Native framework), Dart. Všechny aplikace pro Android běží na VJM pomocí bytového kódu přímo zkompilovaného z jazyků Kotlin nebo Java. Programy napsané v jiných jazycích se spouštějí přes vrstvu, což zpomaluje jejich běh a komplikuje ladění. Proto je pro psaní aplikace nejlepší použít Javu nebo Kotlin. Java je zavedený programovací jazyk s rozsáhlými open-source nástroji a knihovnami, které pomáhají vývojářům. Přesto žádný jazyk není bez chyb a i v Javě se vyskytují komplikace, které mohou vývojářům ztížit práci. Jestli něco Kotlin přinese, tak řešení běžných programátorských problémů a zlepšení ekosystému Javy jako celek. Kotlin je silně typovaný, objektově orientovaný jazyk, který běží v JVM a lze jej použít k vývoji aplikací v mnoha problémových oblastech [13]. Komunita vývojářů

mobilních aplikací v jazyce Kotlin se neustále rozrůstá. V roce 2017 společnost Google povýšila jazyk Kotlin tím, že jej učinila druhým oficiálním jazykem pro vývoj aplikací pro Android. Od té doby Kotlin zaznamenal významný nárůst poptávky jak v komunitě vývojářů, tak v podnikové sféře. Poté co společnost Google oznámila, že Kotlin je nyní preferovaným jazykem pro vývojáře aplikací pro Android, se jazyk ukazuje jako pragmatický, moderní a intuitivní. Někteří vývojáři zřejmě věří, že Kotlin v příštích letech vytlačí Javu z vývoje pro Android. Jiní odborníci vidí koexistenci Kotlinu a Javy, aniž by jeden převážil nad druhým. Pro většinu z nich silné stránky jazyka Kotlin převažují nad jeho nevýhodami. V Javě existují určitá omezení, která brání návrhu rozhraní API pro Android. Kotlin je ze své podstaty lehký, čistý a stručný. Zejména pokud jde o zápis zpětných volání, datových tříd a getterů/setterů. Jinými slovy, jazyk Kotlin je speciálně navržen tak, aby vylepšil stávající modely jazyka Java tím, že nabízí řešení nedostatků [14]. Syntaxe Javy je v porovnání s Kotlinem těžkopádná. Takto by např. vývojář systému Android zadal text do pole v těchto dvou programovacích jazycích. Java:

```
final TextView helloTextView = (TextView)findViewById(R.id.text_view_id);
helloTextView.setText("Lorem ipsum");
```

Kotlin:

```
helloTextView.text = "Lorem ipsum"
```

Programovací jazyk Kotlin nabízí funkce, které lze ukládat do datových struktur a proměnných, takže s funkcemi lze pracovat všemi způsoby, které jsou možné pro jiné nefunkční hodnoty. Jedním z největších problémů, se kterými se vývojáři v Javě potýkají, jsou výjimky NullPointerExceptions. Kotlin pro vývojáře Androidu řeší tento nepříjemný problém tím, že všechny typy jsou ve výchozím nastavení nenulovatelné. Java:

```
String ptr = null;
println(ptr)
NullPointerException
```

Kotlin:

```
val b: String? = null
b = "Kotlin"
println(b?.length)
```

Korutiny jazyka Kotlin jsou návrhovým vzorem, který můžeme v systému Android použít ke zjednodušení kódu, který se vykonává asynchronně. Pomáhají spravovat dlouhotrvající úlohy, které by jinak mohly zablokovat hlavní vlákno a způsobit, že aplikace nebude reagovat [15] Jelikož je Kotlin staticky typovaný jazyk, využívá řadu funkčních typů včetně specializovaných jazykových konstrukcí, jako jsou výrazy Lambda. Ve frontendu potřebují vývojáři funkce, které neblokují hlavní obsah, zatímco se přistupuje k interní databázi nebo serveru. Asynchronní neboli neblokující programování je důležitou součástí vývojového prostředí. Kotlin problém řeší flexibilně tím, že poskytuje podporu coroutine na úrovni jazyka a většinu funkcí deleguje na knihovny. Kotlin:

```
viewModelScope.launch {
  withContext(Dispatchers.IO) {
    try {
     val result = tripRepository.getTripsFromWeb
```

```
        (appPreferences.accessToken as String)
      } catch (e: HttpException) {
         Log.e("getTrips", e.message.toString(), e)
      }
    }
  }
```

Z pohledu obchodu Google Play je Kotlin oficiálním programovacím jazykem pro vývoj aplikací pro Android. Z analýzy jasně vyplývá, že nejlepším řešením by v současné době bylo použít k vývoji aplikace pro Android jazyk Kotlin.

4.5 Výběr frameworku pro UI

Jedním z hlavních trendů v mobilním vývoji v posledních letech je deklarativní uživatelské rozhraní. Toto řešení se již dlouho úspěšně používá ve webových a multiplatformních řešeních a konečně se dostalo i do nativního vývoje. V systému iOS je to SwiftUI (představený na konferenci WWDC 2019) a v systému Android Jetpack Compose (uvedený o měsíc dříve na konferenci Google I/O 2019). Podstatou deklarativního uživatelského rozhraní je popsat jej jako sadu složitelných funkcí (tzv. widgetů), které nepoužívají „pod kapotou“ view, ale kreslí přímo na canvas. Výhody používání Android Jetpack Compose:

- „Unbundled toolkit“: JC není závislý na konkrétních verzích platformy, což znamená, že není třeba podpůrné knihovny.
- Pouze Kotlin: Už žádné přepínání mezi třídami a XML soubory – veškerá práce s uživatelským rozhraním se provádí v jednom souboru Kotlin.
- Složený přístup: Každá komponenta uživatelského rozhraní je běžná složitelná funkce, která je zodpovědná pouze za omezenou funkčnost, tj. žádná další logika a dědičnost.
- Jednosměrný tok dat: Jeden ze základních konceptů aplikace JC. V JC jsou všechny komponenty ve výchozím nastavení bezstavové. Díky jednosměrnému principu stačí poskytnout pouze datový model a o jakoukoli změnu stavu se postará samotný framework. Tímto způsobem se zjednodušuje logika komponenty a zapouzdření stavu zabraňuje chybám spojeným s jeho částečnou aktualizací.
- Zpětná kompatibilita: Pro použití aplikace Compose není nutné začínat projekt od nuly. Je možné jej vložit (pomocí ComposeView) do existujícího XML-wrap a naopak.

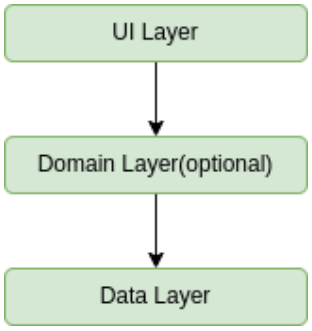
4.6 Výběr architektury frontendu aplikace

S rostoucí velikostí aplikací pro systém Android je důležité definovat architekturu, která umožní škálování aplikace, zvýší její robustnost a usnadní testování. Architektura aplikace definuje hranice mezi jednotlivými částmi aplikace a úkoly, které by každá část měla plnit. Pro splnění výše uvedené potřeby musí být architektura aplikace navržena podle několika konkrétních zásad:

- **Separation of concerns (SoC)** je princip návrhu pro rozdělení počítačového programu na samostatné části. Každá sekce řeší samostatný problém, soubor informací, které ovlivňují kód počítačového programu.
- Další důležitou zásadou je škálovatelnost **(scalability)** - možnost rozšířit projekt a implementovat nové funkce.
- Udržovatelnost **(maintainability)** lze definovat jako potřebu malých atomických změn po implementaci všech funkcí.

- Spolehlivost **(reliability)** je základním požadavkem mnoha aplikací.
- Pro kvalitní aplikaci je také velmi důležitá testovatelnost **(testability)**.

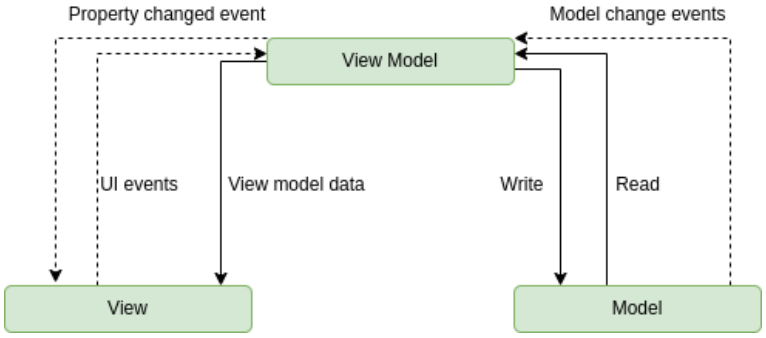
Všechny tyto zásady lze realizovat pouze oddělením odpovědnosti mezi jednotlivými vrstvami aplikace.



Obrázek 4.3. Schéma typické architektury aplikace [16]

V moderních architekturách mobilních aplikací se používá několik populárních přístupů, jedním z nich je pattern MVC. Vzor MVC rozděluje kód na 3 složky. Při vytváření třídy/souboru aplikace ji musí vývojář zařadit do jedné z následujících tří vrstev:
Model: Tato komponenta uchovává data aplikace. Nemá žádné znalosti o rozhraní. Model je zodpovědný za zpracování doménové logiky (reálných obchodních pravidel) a komunikaci s databází a síťovou vrstvou.
View: Modelová vrstva je tvořena dvěma vrstvami: Je to vrstva uživatelského rozhraní UI, která obsahuje komponenty viditelné na obrazovce. Kromě toho zajišťuje vizualizaci dat uložených v modelu a nabízí interakci uživateli.
Controller: Tato komponenta vytváří vztah mezi View a Modelem. Obsahuje základní logiku aplikace a získává informace o chování uživatele a aktualizuje Model podle potřeby. Nevýhodou této architektury je, že její části jsou příliš provázaný (coupling), přičemž ontroller ovládá ostatní části. To znamená, že komponenty nejsou zcela nezávislé, což ztěžuje práci s jejich dalším nakládáním. Další modernější pattern je MVVM, který překonává všechny nevýhody návrhového vzoru MVC. MVVM navrhuje oddělit logiku prezentace dat (Views nebo UI) od hlavní části business logiky aplikace.

Jednotlivé vrstvy kódu MVVM jsou:
Model: Vrstva je zodpovědná za abstrakci datových zdrojů. Model a ViewModel spolupracují při získávání a ukládání dat.
View: Je tvořen dvěma vrstvami, které jsou součástí modelu ViewModel. Účelem vrstvy je informovat ViewModel o akci uživatele, přičemž vrstva pozoruje ViewModel a neobsahuje žádný druh aplikační logiky.
ViewModel: Vystavuje datové toky, které jsou relevantní pro View. Kromě toho slouží jako spojovací článek mezi Modelem a View.



Obrázek 4.4. Schéma MVVM architektury aplikace [17].

4.7 Výběr architektury bekendu aplikace

Pro vývoj backendu aplikací pro Android se dají použít následující programovací jazyky: JavaScript, Python, PHP, Java, Kotlin, C++, C# Všechny jazyky mají své výhody a nevýhody. Jako vývojář mám největší zkušenosti s backendovými aplikacemi v Javě/Kotlinu se SpringBoot, proto použiji následující řešení pro vývoj backendové části aplikace. Obliba Spring jako systému se přičítá především úspěchu Javy/Kotlina jako jazyka. Co se týká výkonu a bezpečnosti, je nejlepší volbou virtuální stroj s Javou. Spring je aplikační kontextový řadič a framework pro inverzi výkonu. Obecně platí, že při vytváření nových entit v aplikaci Spring jsou objekty přímo spravovány daným systémem. Spring má rovněž moduly pro zápis do databází a vývoj online aplikací. Spring Boot je nadstavba Springu. Implementuje tedy veškeré funkce Springu, ale ještě k tomu nám poskytuje jednu významnou funkci navíc, která nám tvorbu projektu výrazně zrychlí a usnadní. Spring Boot má totiž v sobě server Tomcat, který nám při spuštění projektu vytvoří server, který nastaví přístupy pro frontend aplikace. Spring Boot je postavený na MVC architektuře. To znamená, že aplikaci rozděluje do 3 vrstev, a tím ji činí přehlednou a jednoduše rozšiřitelnou [18]. REST je architektonický styl komunikace mezi komponentami distribuované aplikace v síti. Návrh rozhraní RESTful API definoval Dr. Roy Fielding ve své doktorské práci z roku 2000. Aby byla webová služba skutečně RESTful API, musí dodržovat následujících šest architektonických omezení: Použití jednotného rozhraní (UI) znamená, že zdroje by měly být jednoznačně identifikovatelné prostřednictvím jediné adresy URL a mělo by být možné s nimi manipulovat pouze pomocí základních metod síťového protokolu, jako jsou DELETE, PUT a GET s protokolem HTTP.

Na bázi klient-server by mělo existovat jejich jasné vymezení. Uživatelské rozhraní a záležitosti týkající se shromažďování požadavků jsou doménou klienta. Přístup k datům, správa pracovní zátěže a zabezpečení jsou doménou serveru. Volné propojení klienta a serveru umožňuje vyvíjet a vylepšovat oboje nezávisle na druhém.

Bezstavové operace představují veškeré operace mezi klientem a serverem, kdy by měla probíhat požadovaná správa stavu výhradně na klientovi, nikoli na serveru. RESTful resource caching (ukládání zdrojů do mezipaměti). Všechny zdroje by měly umožňovat ukládání do mezipaměti, pokud není výslovně uvedeno, že ukládání do mezipaměti není možné.

Vrstvený systém REST umožňuje architekturu složenou z více vrstev serverů. Kód na vyžádání představuje statickou reprezentaci zdrojů serveru a je zasílán ve formátu XML nebo JSON. V případě potřeby však servery mohou klientovi zaslat spustitelný kód. Rozhraní RESTful API rozděluje transakci na řadu malých modulů. Každý modul řeší základní část transakce. Popisovaná modularita poskytuje vývojářům velkou

flexibilitu, ale může být náročné navrhovat rozhraní REST API od nuly. V současné době poskytuje několik společností modely, které mohou vývojáři používat. Mezi nejoblíbenější patří řešení poskytované společnostmi Amazon S3, Cloud Data Management Interface (CDMI) a OpenStack Swift. Rozhraní RESTful API používá k získání prostředků příkazy. Stav prostředku v daném časovém okamžiku se nazývá reprezentace prostředku. Rozhraní RESTful API využívá stávající metodiky HTTP definované protokolem RFC 2616, jako např:
GET pro získání prostředků;
PUT pro změnu stavu nebo aktualizaci prostředku, kterým může být objekt, soubor nebo blok;
POST pro vytvoření daného prostředku a DELETE pro jeho odstranění. nl V případě protokolu REST jsou síťové komponenty zdrojem, ke kterému uživatel požaduje přístup - jako černá skříňka, jejíž implementační detaily jsou nejasné. Všechna dotazování jsou bezstavová, služba RESTful nemůže mezi jednotlivými úkony nic uchovávat.

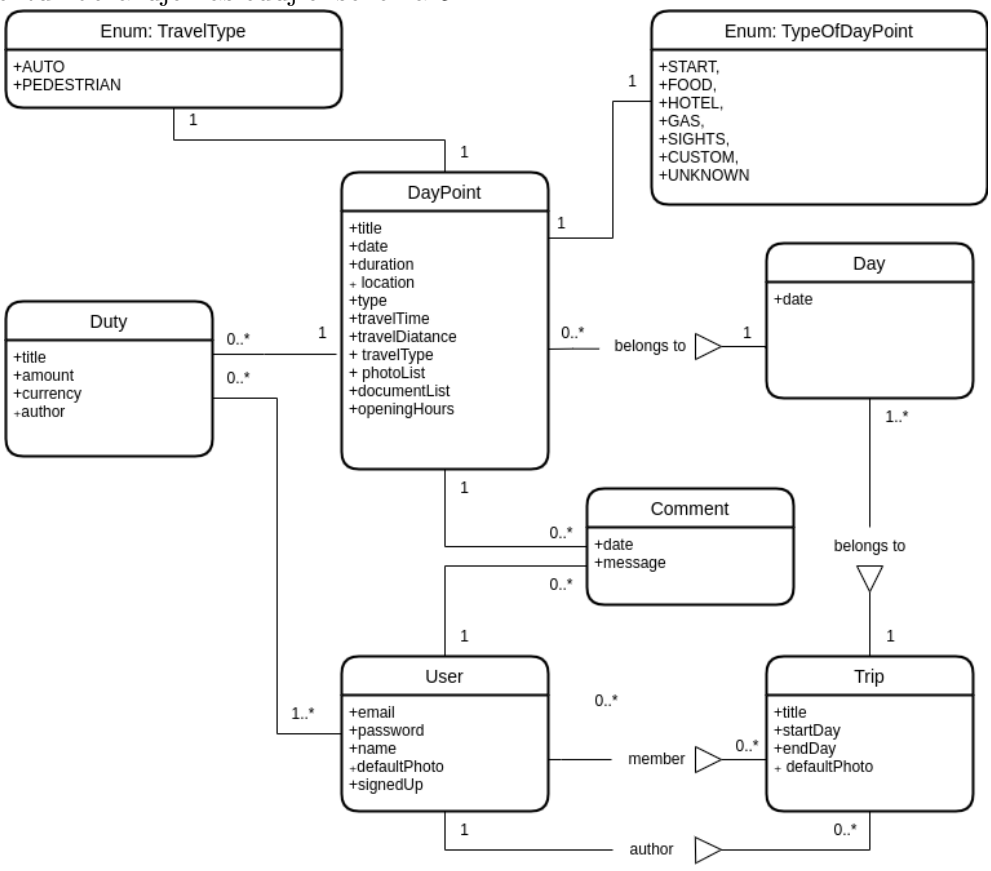
4.8 Výběr databáze

Aplikace využívá relační databázi PostgreSQL. Protože platforma PostgreSQL je robustní, bezpečná a rozšiřitelná, jelikož disponuje bohatým ekosystémem dostupných nástrojů. Vývojáři ji využívají pro celou řadu použití. Software je navržen tak, aby byl kompatibilní se všemi hlavními operačními systémy, včetně Linuxu, Windows a Mac OS, a podporuje text, obrázky, zvuky a video. Desítky let vývoje přispěly k tomu, že systém PostgreSQL je extrémně odolný vůči chybám. Je v souladu se zásadami ACID (z anglických slov Atomic, Consistent, Isolated, Durable – atomický, konzistentní, izolovaný, trvanlivý) pro databázové transakce. Kromě toho PostgreSQL podporuje více jazyků pro spouštěče, atributy cizích klíčů, spojení a uložené procedury. PostgreSQL umožňuje práci s nejběžnějšími datovými typy, a podporuje Unicode – mezinárodní znakové sady a vícebajtová kódování znaků. Dispozice PostgreSQL ve formě opensourcové licence dává uživatelům možnost větší flexibility a inovativnosti, než je tomu v případě komerčních databázových systémů. Díky tomu, že uživatelé nemají žádné licenční náklady, mohou volně zkoumat nekonečné možnosti vývoje a dokonce upravovat nebo implementovat zdrojový kód podle vlastního uvážení. Vzhledem k tomu, že PostgreSQL podporuje relační a nerelační dotazování, mohou uživatelé přistupovat k datům pomocí výrazů SQL a cest JSON. [19] Mezi aplikací Kolin a databází PostgreSQL existuje trojrstvý systém komunikace: Vrstva nejvyšší úrovně Spring data je sada nástrojů pro práci s daty založená na práci s aplikací Spring. Další vrstva JPA převádí SQL dotazy Kolin na databáze a odpovědi na dotazy převádí zpět na business entity. Třetí vrstva je JDBC protokol, pomocí kterého si aplikace Java vyměňuje data s relačními databázemi. JPA je Java Persistence API. Jedná se o specifikaci pro správu relačních databází pomocí jazyka Java/Kotlin. Popisuje abstrakční vrstvu mezi třídami jazyka Java/Kotlin a relační databází. Spring Data JPA je obal kolem poskytovatelů JPA, jako je například Hibernate. Umožňuje perzistenci tříd jazyka Java/Kotlin tak jednoduše, jako je přidání několika anotací a vytvoření jednoduchého rozhraní úložiště. Není třeba skutečně psát metody persistence nebo načítání. Další velkou výhodou je, že Spring Data JPA umožňuje transparentně měnit základní implementaci databáze, aniž by bylo nutné měnit jakýkoli kód. Všechny tyto výhody, které PostgreSQL databáze poskytuje, ji činí nejlepší volbou, abych ji začlenila pro účely své aplikace.

Kapitola 5
Návrh

5.1 Návrh doménového modelu

Dalším krokem je definice hlavních entit aplikace a popis jejich vztahů a chování. Třídy v doménovém modelu jsou však značně zjednodušené, neobsahují metody a mají pouze důležité atributy. To lze provést pomocí standardu UML v programu Enterprise Architect nebo podobných. UML může sloužit jako užitečný nástroj k usnadnění návrhu a vývoje informačního systému. Diagramy mají velmi důležitou vlastnost abstrakce. Každý diagram poskytuje určitý pohled na systém, který akcentuje vybrané hledisko. Zbytek systému bude ignorován a zobrazí se pouze, co je v danou chvíli důležité. UML diagramy snižují riziko, že systém bude špatně navržený. Navržený doménový model projektu zobrazuje následující schema 5.1:



Obrázek 5.1. Doménový model.

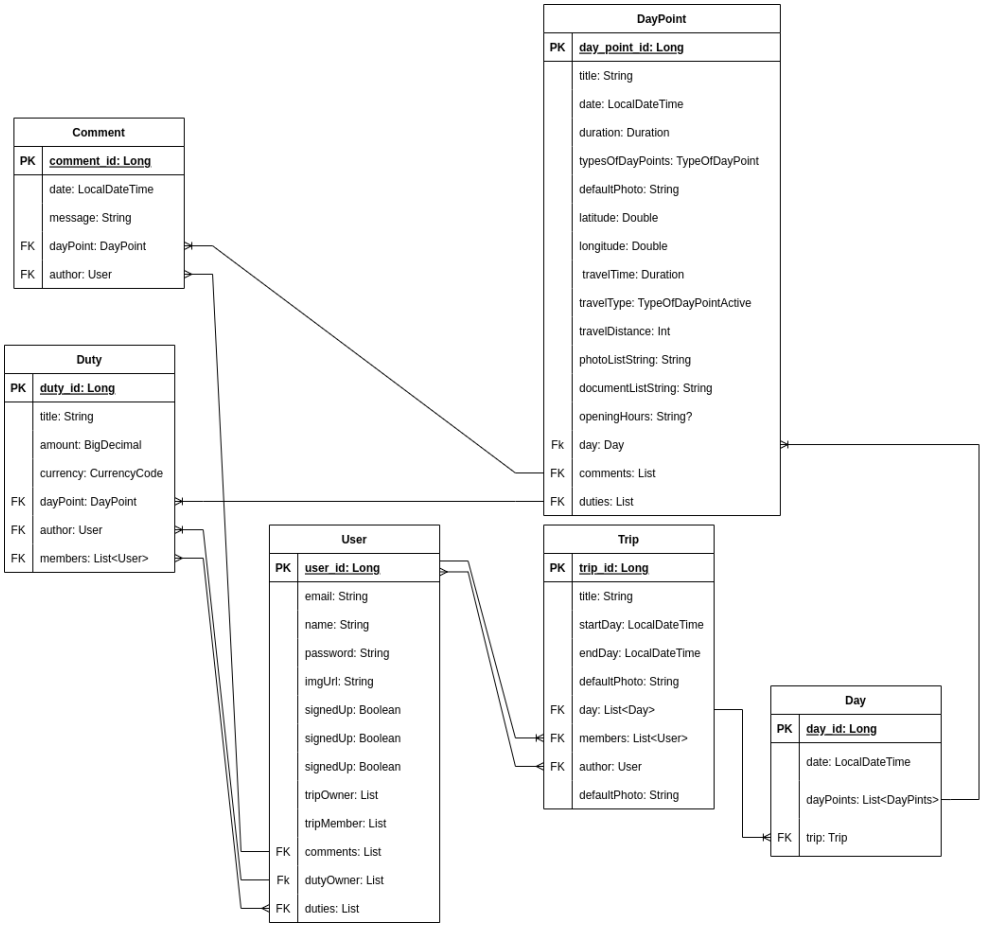
Základní entitou v navrženém systému je *Trip*, který se zde nemůže vyskytovat sám o sobě, jelikož je jeho zastoupení vázáno na majitele. Majitel muže sdílet *Trip* s kamarády a oní se stavají učastniky *Tripu*. Každý *Den* může mít *DayPoint*, které patří do jedné z kategorií, přičemž každý *DayPoint* má začátek, konec a udání zeměpisné polohy

provozované aktivity, a má typ cesty na místo, čas a vzdálenost. Každý uživatel může *Trip* upravovat, přidávat *DayPoint*, poznámky, fotky, soubory a výdaje.

5.2 Relační databáze

Serverová část aplikace obsahuje databázi, ve které jsou uložena data uživatelů. Správně navržená databáze umožňuje přístup k aktuálním a přesným informacím. Aplikace bude používat vlastní lokální databázi pro dočasné ukládání používaných dat, ale víceuživatelská aplikace vyžaduje databázi na serveru. Všechny obrázky budou uloženy odděleně pomoci vnější databáze, v interní databázi budou pouze odkazy na zdroje. Komentáře a výdaje, které uživatel vytvořil, budou rovněž uloženy samostatně. Při vytváření databáze byl použit přístup, který navrhuje Spring Boot a Kotlin, tedy popsat tabulky a pole databáze a vztahy mezi nimi pomocí třídy s anotací *@Entity*. To umožňuje neupravovat kód na dvou místech a nevytvářet ani neaktualizovat tabulky v datové databázi, ale pouze je vytvořit na jednom místě. Navržený databazový model zobrazuje schéma 5.2: Kotlin: Ukázka kódu

```
@Entity
@Table(name = "days")
class Day(
    @Id
    @Column(unique = true)
    @GeneratedValue(strategy = GenerationType.AUTO)
    val id: Long = 0,
    @Column(nullable = false)
    val date: LocalDateTime = LocalDateTime.now(),
    @ManyToOne(optional = false)
    @JoinColumn(name = "trip_id", referencedColumnName = "id")
    var trip: Trip,
    ): Serializable {
    @OneToMany(mappedBy = "day", targetEntity = DayPoint::class,
                  cascade = arrayOf(CascadeType.PERSIST))
    var dayPoints: MutableList⟨DayPoint⟩ = mutableListOf()
```



Obrázek 5.2. Relační databáze.

5.3 Návrh frontendu

V článku "10 Usability Heuristics for User Interface Design" byly popsány nejdůležitější principy tvorby user-friendly rozhraní [20].

- **Viditelnost stavu systému**
  Návrh by měl uživatele vždy informovat o tom, co se děje, a to prostřednictvím vhodné zpětné vazby v přiměřeném čase. Když uživatelé znají aktuální stav systému, dozví se výsledek svých předchozích interakcí a určí další kroky. Předvídatelné interakce vytvářejí důvěru v produkt i značku.
- **Shoda mezi systémem a reálným světem**
  Návrh by měl mluvit jazykem uživatelů. Používejte slova, fráze a pojmy, které uživatelé znají, a nikoliv interní žargon. Dodržujte konvence reálného světa, aby se informace zobrazovaly v přirozeném a logickém pořadí. Tento princip je velmi důležitý také pro překlady aplikace do jiných jazyků.
- **Kontrola a svoboda uživatele**
  Uživatelé často provádějí akce omylem. Potřebují jasně označený `nouzový východ`, aby mohli nechtěnou akci opustit, aniž by museli procházet zdlouhavým procesem. Když je pro lidi snadné z procesu vycouvat nebo akci vrátit zpět, podporuje to pocit svobody a důvěry. Východy umožňují uživatelům mít systém stále pod kontrolou a vyhnout se zaseknutí a pocitu frustrace.

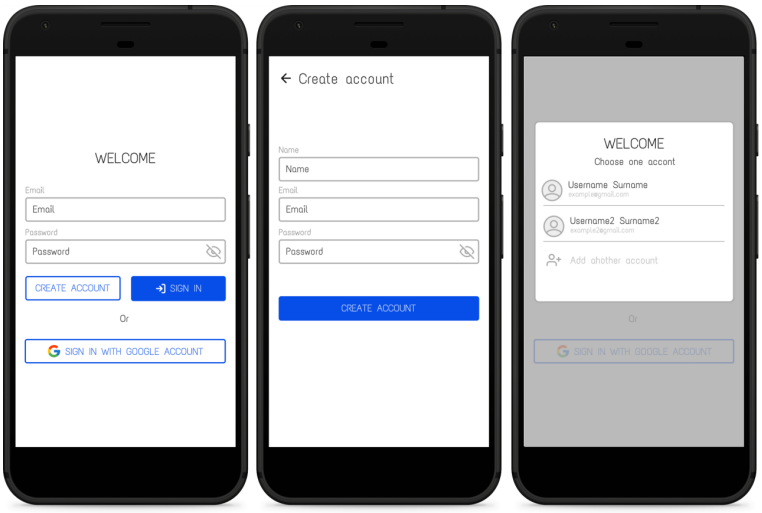
- **Konzistence a standardy**
  Uživatelé by neměli přemýšlet, zda různá slova, situace nebo akce znamenají totéž. Dodržujte konvence platforem a odvětví.
- **Prevence chyb**
  Dobrá chybová hlášení jsou důležitá, ale nejlepší návrhy pečlivě předcházejí vzniku problémů. Buď eliminujte podmínky náchylné k chybám, nebo je kontrolujte a nabídněte uživatelům možnost potvrzení předtím, než se k akci zaváží.
- **Spíše rozpoznávání než vzpomínání**
  Minimalizujte zatížení paměti uživatele tím, že zviditelníte prvky, akce a možnosti. Uživatel by si neměl pamatovat informace z jedné části rozhraní do druhé. Informace potřebné k používání návrhu (např. popisky polí nebo položky nabídek) by měly být viditelné nebo v případě potřeby snadno vyvolatelné.
- **Flexibilita a efektivita použití**
  Zkratky - skryté před začínajícími uživateli - mohou urychlit interakci pro zkušeného uživatele, takže návrh může vyhovět jak nezkušeným, tak zkušeným uživatelům. Umožněte uživatelům přizpůsobit si časté akce.
- **Estetický a minimalistický design**
  Rozhraní by neměla obsahovat informace, které jsou nepodstatné nebo které jsou potřeba jen zřídka. Každá další jednotka informací v rozhraní konkuruje relevantním jednotkám informací a snižuje jejich relativní viditelnost.
- **Pomáhá uživatelům rozpoznat, diagnostikovat a použít se z chyb.**
  Chybová hlášení by měla být vyjádřena jednoduchým jazykem (bez chybových kódů), přesně označovat problém a konstruktivně navrhovat řešení.
- **Nápověda a dokumentace**
  Nejlepší je, když systém nepotřebuje žádné další vysvětlení. Může však být nutné poskytnout dokumentaci, která uživatelům pomůže pochopit, jak dokončit jejich úkoly.

Při navrhování aplikace je velmi důležité dodržovat uvedená pravidla. Kromě toho budu ve své práci používat knihovnu Material Design vyvinutou speciálně pro aplikace pro Android. Material je designový systém vytvořený společností Google, který pomáhá týmům vytvářet vysoce kvalitní uživatelskou zkušenost při používání Android, iOS, Flutter a web. Všechny součásti designového systému jsou obsaženy v systému Figma. Figma je jedna z revolučních aplikací pro úpravu grafiky, která ovládla svět designu. Atraktivní ji činí skutečnost, že je zdarma k použití. Figmu jsem si vybrala, protože nejenže dokáže navrhnout aplikaci, ale je také skvělá pro uživatelské testování (Usability testing). Díky tomu, že Figma používá klikací pole, lze uživatelské testy provést před vlastním vývojem aplikace, což zkrátí dobu vývoje.

5.3.1 Přihlašovací stránka

První stránka, kterou uživatel uvidí, je přihlašovací stránka. Přihlašovací stránkaivatel obvykle nechce v aplikaci trávit příliš mnoho času procesem autorizace, proto jsou k dispozici funkce pro autorizaci pomocí účtů Google. Přihlašování pomocí účtů uvedené služby umožňuje uživatelům ověřit se pomocí svých stávajících přihlašovacích údajů. To znamená, že se uživatel může jednoduše přihlásit do aplikace/na webové stránky třetí strany, aniž by si musel vytvořit nový účet na těchto stránkách. Výhody přihlašování přes existující účty třetích stran: Snadná registrace - přihlášení nabízí zjednodušenou, rychlou a snadnou registraci. Uživatelům umožňuje pohodlnou registraci jediným kliknutím, což snižuje časovou náročnost při vytváření nového účtu. Svou jednoduchostí zvyšuje pravděpodobnost dokončení registrace, a tím i lepší počáteční zkušenost uživatelů s danou platformou.

Jak ukazují statistiky uvedené ve článku "Why Do Consumers Prefer Social Login" [21] 36 % uživatelů v evropském regionu dává přednost přihlášení do aplikace třetí strany pomocí 34 % s účtem Google. Zbylých 28 % dává přednost jiným sociálním sítím, ale problémem je, že uživatelé často zapomínají, přes který účet se do aplikace přihlašují, proto jsem ve své aplikaci ponechala pouze dva nejoblíbenější způsoby přihlášení přes Google. Aplikace podporuje také tradiční způsob přihlašování přes login/password. Navrženou přihlašovací stránku je možné vidět na obrázku 5.3



Obrázek 5.3. Přihlašovací stránky (Lo-fi prototyp).

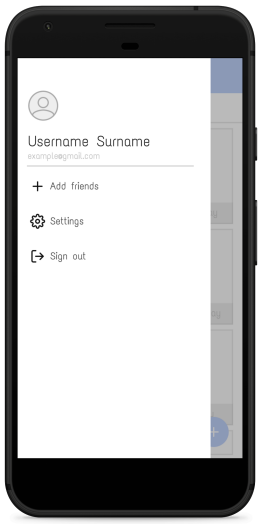
5.3.2 Hlavní stránka

Hlavní stránka obsahuje informace o plánovaných a uskutečněných výletech. Orientaci v aplikaci zajišťuje levé navigační menu. Kliknutím na výlet se otevře stránka samotného výletu, která je rozdělena podle dnů. Na domovské stránce můžete vytvořit novou trasu zadáním požadovaných informací, přičemž ostatní informace se doplní během plánování a v průběhu výletu. Navrženou hlavní stránku a stránku pro vytvoření trasy je možné vidět na obrázku 5.4

Obrázek 5.4. Hlavní stránka a stránka pro vytvoření trasy (Lo-fi prototyp)

5.3.3 Levé menu

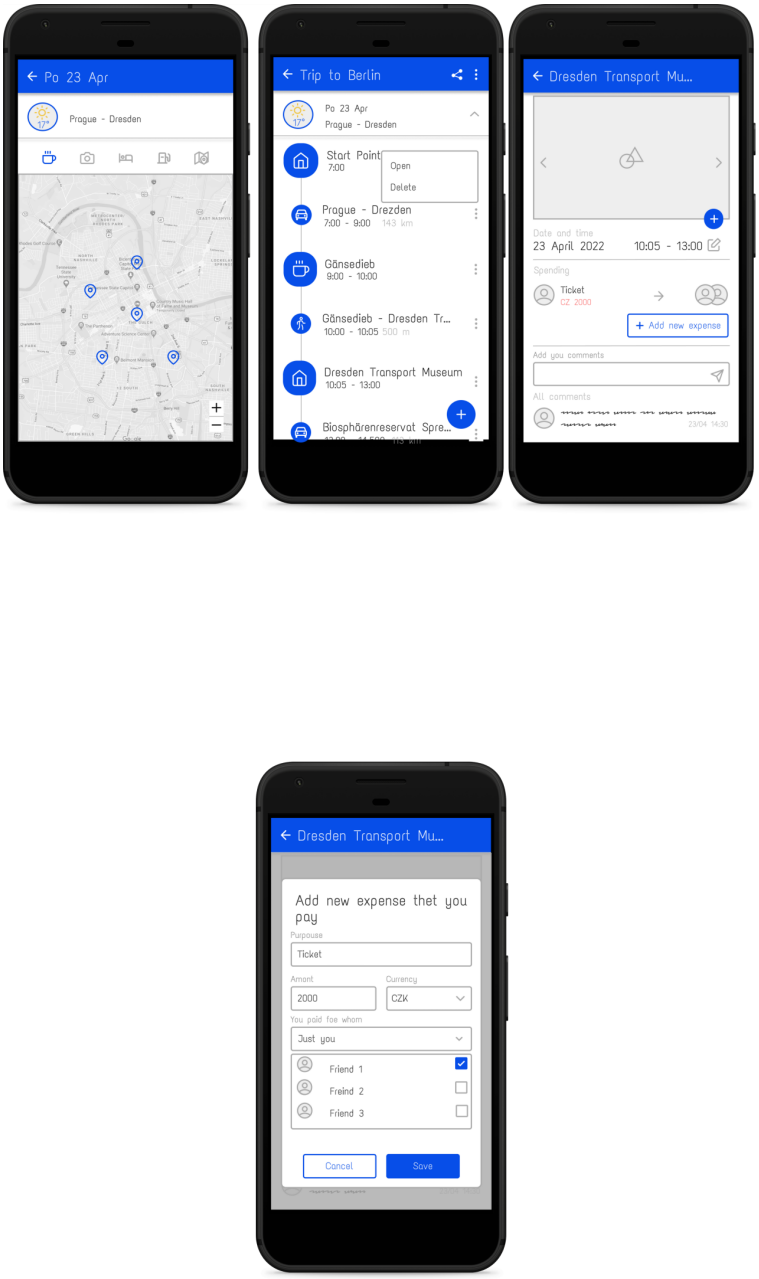
Levé menu obsahuje informace o uživateli a navigační menu, které slouží také jako rezerva pro další vývoj aplikace a rozšíření funkcí.



Obrázek 5.5. Obsah levého menu (Lo-fi prototyp)

5.3.4 Aktivity

Aktivity v aplikace jsou rozděleny podle typů: Stravování, ubytování, návštěva památek a jiné účely cesty. Takové rozdělení pomáhá při plánování a je jasně znázorněno různými ikonami. Jednotlivé kategorie lze použít k vyhledávání aktivit na mapách Google. Každou aktivitu můžete také upravovat, přidávat fotografie, dokumenty (např. vstupenky) a informace o ceně služeb na daném místě. Můžete také přidávat poznámky. Změny aktivity můžete provádět jak ve fázi plánování cesty, tak během cesty samotné.



Obrázek 5.6. Plánování cesty a stanovení aktivity (Lo-fi prototyp)

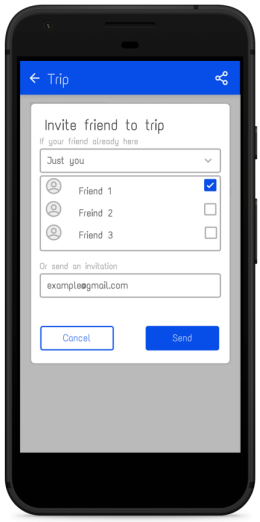
5.3.5 Využití plánované trasy

Následující stránky představují využití plánované trasy a přesuny mezi jednotlivými body. Tyto ikony jsou v současné době navrženy v uvedené podobě. Podrobnější a intuitivnější rozhraní bude vypracováno po obdržení zpětné vazby od uživatelů během testů uživatelského rozhraní. Bližší nastavení testování bude popsáno v další kapitole.

Obrázek 5.7. Využití plánované trasy (Lo-fi prototyp)

5.3.6 Sdílení cesty

Sdílení cesty se provádí přes ikonku v horní nabídce, která se mění podle toho, na které stránce se uživatel nachází. Obrazovka umožňuje uživateli nejen sdílet trasu s přáteli, ale také jim povolit určitá práva úpravy trasy a přidání komentáře, fotek, dokumentů a výdajů.



Obrázek 5.8. Plánování cesty a stanovení aktivity (Lo-fi prototyp)

5.3.7 Lo-fi prototyp

Pro Lo-fi prototyp jsem vytvořila pouze uživatelské rozhraní pro hlavní funkce aplikace. Další funkce, stejně jako oznámení, souhlas uživatele a chybová okna budou vyvinuty později v procesu implementace. Pro dané prvky bude použita knihovna Material Design od společnosti Google. Důležité respektovat obvyklé přiřazení tlačítek, protože

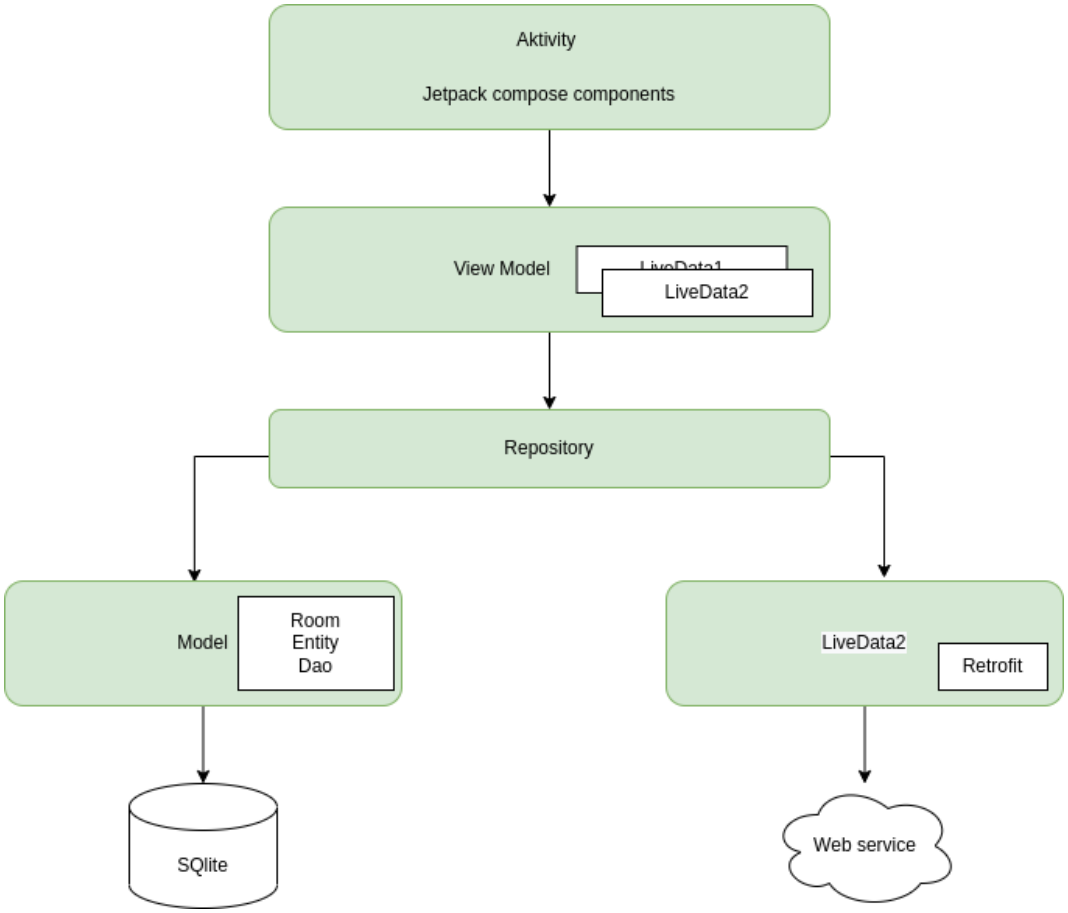
stisknutím tlačítka „zpět“ se uživatel musí dostat na předchozí stránku. [22]. Rozhraní bude po otestování Lo-fi prototypu mírně upraveno na základě zpětné vazby od uživatelů.

5.3.8 Hi-fi prototyp

Vzhledem k tomu, že zpětná vazba od uživatelů se týkala spíše funkčnosti aplikace a sotva se dotkla navigace a grafiky, po otestování prototypu Lo-fi byl vyvinut Hi-fi prototyp hlavních stránek aplikace, který sloužil především k upřesnění návrhu jednotlivých prvků aplikace. Výsledné stránky prototypu jsou znázorněny v příloze C.

5.4 Architektura frontendu aplikace

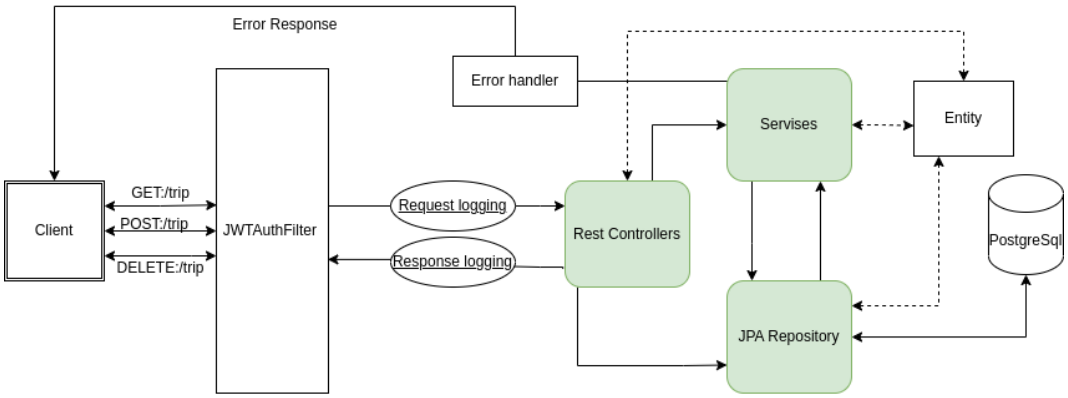
Výsledkem je, že po analýze populárních řešení a seznámení se s konceptem čisté architektury bylo rozhodnuto použít architekturu MVVM s interní databází a přistupovat k interním datům prostřednictvím ROOM[29] a k externím datům prostřednictvím RETROFIT. Vytvoření interní databáze vyžadovalo popis datového modelu aplikace a vztahů mezi nimi. Pak bylo nutné napsat DAO pro každý atribut a úložiště, přes které se bude přistupovat k databázi a externí webové službě. Konečné schéma architektury je vidět na obrázku 5.9.



Obrázek 5.9. Výsledná schéma architektury frontendu aplikace

5.5 Architektura backendu aplikace

Backend aplikace napsán v prostředí užitečného nástroje Spring Boot, jehož cílem je zjednodušit tvorbu aplikací založených na platformě Spring. Umožňuje nejjednodušší způsob vytvoření webové aplikace, který vyžaduje minimální úsilí vývojářů při jejím nastavení a vytváření kódu. Pro urychlení procesu správy závislostí Spring Boot implicitně balí potřebné závislosti třetích stran pro každý typ aplikace založené na platformě Spring a zpřístupňuje je vývojáři prostřednictvím tzv. startovacích balíčků (spring-boot-starter-web, spring-boot-starter-data-jpa atd.). Startovací balíčky jsou sadou praktických deskriptorů závislostí, které je možno zahrnout do aplikace. Poskytne komplexní řešení pro všechny technologie související se Springem a ušetří čas zbytečným hledáním ukázek kódu a stahováním závislostí. Aplikace rozdělena na vrstvy Controller a Model, přičemž databázová data přímo mapovaná na tento Model. V backendu jsem přidala JWTAuthTokenFilter, ve kterém jsem přesně definovala, které požadavky na službu mohou probíhat bez ověření uživatele. Aplikace má vlastní zpracování chyb, služba může vyhodit chybu a zpracování chyb (error handler) ji zachytí, přiřadí jí kód, zabalí ji do odpovědi HTTP a odešle ji klientovi. Konečné schéma architektury je vidět na obrázku 5.10.



Obrázek 5.10. Architektura Backend Spring Boot Aplikace

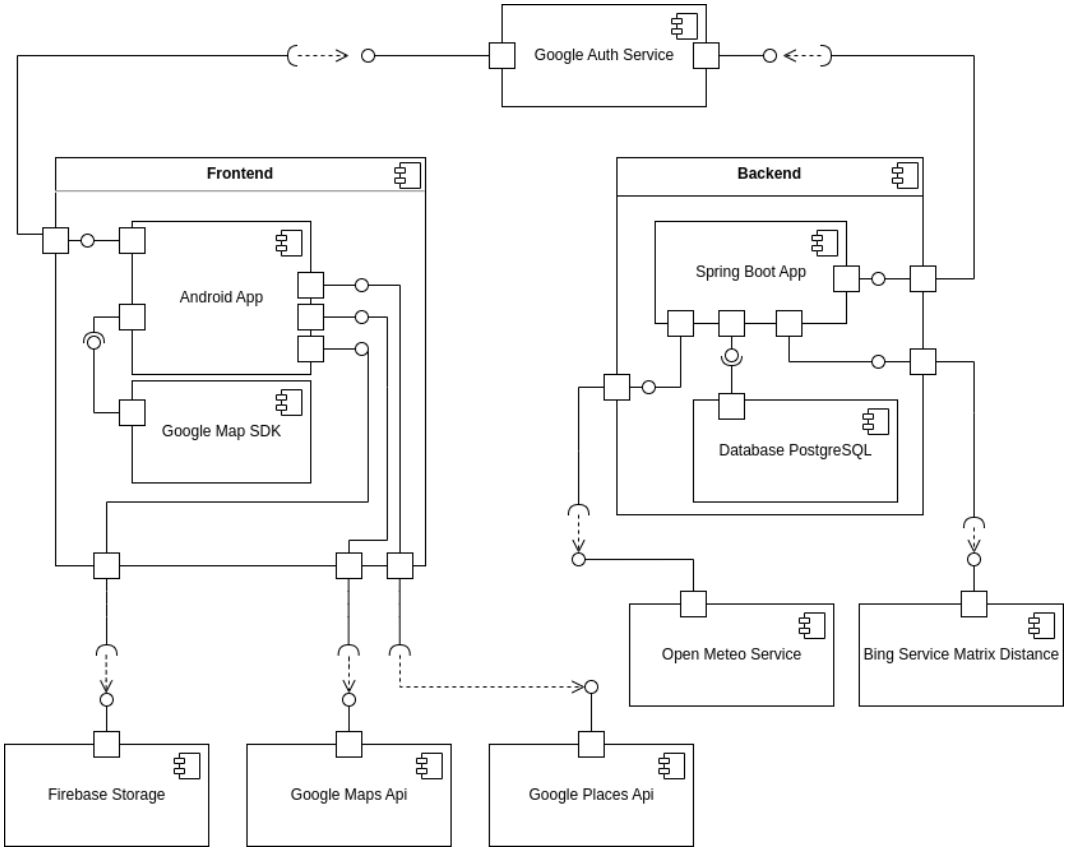
Vzhledem k tomu, že frontendová aplikace bude napsána v jazyce Kotlin a vzhledem k výhodám Kotlinu oproti Javě bylo rozhodnuto napsat v Kotlinu i backendovou aplikaci.

5.6 Komunikace mezi základními prvky projektu

Následný systém se skládá z frontendu (mobilní aplikace pro operační systém Android), backendové části (aplikace Spring Boot na Kotlinu poskytující RESTful API) a služeb třetích stran:

- **Google Auth Service** umožňuje autorizovat uživatele jediným kliknutím bez zadání e-mailu nebo hesla. Ke službě mají přístup frontend a backend.
- **Firebase Storage** slouží k ukládání souborů a fotografií, přistupuje k ní pouze frontend a databáze na backendu obsahuje výhradně odkazy na soubory.
- **Google Places API** se používá pro vyhledávání místa podle kategorií a podle textového dotazu vedle bodu, který zadá uživatel.
- **Google Maps API** se používá k zobrazení míst, o která uživatel požádal v předchozí službě.

- **Bing Service Matrix Distance** se používá v backendu pro výpočet času na přesun mezi body. Zvolen byl namísto Google Matrix Distance s omezeným počtem bezplatných dotazů.
- **Open Meteo Service** se používá v backendu k předpovědi počasí na příštích 5 dní v daném bodě.



Obrázek 5.11. Architektura komunikace mezi subsystémy

Kapitola 6
Implementace

6.1 Implementace backendu

Serverová část aplikace byla implementována pomocí frameworku Spring Boot, který umožňuje poměrně rychle napsat RESTful API. Rozhraní REST API je způsob, jakým webové stránky a webové aplikace komunikují se serverem. Nazývá se také RESTful. Termín se skládá ze spojení výrazů Rozhraní API a REST. Spring Boot na Kotlinu byl vybrán hlavně kvůli osobní zkušenosti s tímto frameworkem a obrovskému množství knihoven, které vývojářům zjednodušují práci a umožňují psát méně kódu.

6.1.1 Komunikace s serverem

Komunikace se serverem je rozdělena do několika řadičů, z nichž každý je zodpovědný za jinou část práce, a proto je vhodné je mít setříděné podle účelu použití.

- POST: /user/signup Přijme požadavek na registraci a odešle kladnou odpověď, pokud zadané údaje splňují všechny systémové požadavky, včetně formátu e-mailové adresy.
- POST: /user/signin Přijme požadavek na ověření uživatele, v kladné odpovědi odešle JWTAuthToken pro další komunikaci s uživatelem.
- POST: /user/sign-google přijme požadavek na ověření uživatele pomocí autorizačního tokenu Google a zjistí, zda je uživatel nový. Jako kladnou odpověď odešle JWTAuthToken pro další komunikaci s uživatelem.
- POST: /user/reset-password Přijme požadavek na resetování hesla uživatele, odešle kladnou odpověď, pokud takový uživatel existuje, a odešle na e-mail uživatele kód pro resetování hesla.
- POST: /user/update-password Přijme požadavek na aktualizaci hesla uživatele a odešle kladnou odpověď, pokud jsou zadané údaje správné. Body výše jsou v nástroji WebSecurityConfig k dispozici jako dostupné bez tokenu, všechny ostatní požadavky jsou nejprve kontrolovány z hlediska autorizace uživatele.
- GET: /user/friends je služba sloužící k vyhledání všech kontaktů uživatelů v systému, tj. když se uživatelé již zúčastnili společné cesty.

TripController endpoints: Všechny dotazy v tomto řadiči jsou nejprve zkontrolovány na oprávnění uživatele a poté na jeho práva cokoli v daných výletech měnit.

- GET: /trip/all vrátí uživateli všechny výlety, jichž byl autorem nebo se jich zúčastnil, ale s omezenými informacemi, pouze s názvy a datem zahájení a ukončení.
- POST: /trip/new Požadavek slouží k vytvoření nového výletu
- GET: /trip/tripId Žádost slouží k získání určitého výletu s harmonogramem a seznamem dlužných částek, komentářů, fotografií a dokumentů.
- GET: /trip/tripId/info Dotaz vrátí souhrn určitého výletu se všemi údaji jako v předchozím dotazu, ale především s konečnými informacemi o dlužných částkách v různých měnách.

- DELETE: /trip/tripId Odstraní výlet, pokud je uživatel, který zadal požadavek, jeho vlastníkem.
- POST: /trip/user/invite Jedná se o dotaz, který zapíše uživatele do konkrétního výletu jako účastníka.
- POST: /trip/default_photo Dotaz měnící výchozí fotografii konkrétního výletu.

DayPointController endpoints:

Všechny dotazy v tomto řadiči jsou nejprve zkontrolovány na oprávnění uživatele a poté na jeho práva cokoli v daných výletech měnit.

- GET: /day_point/dayPointId Žádost slouží k získání určité aktivity se seznamem výdajů, komentářů, fotografií a dokumentů.
- POST: /day_point/add_photo Přidá fotku k aktivitě.
- POST: /day_point/add_document Přiřadí dokument k aktivitě.
- DELETE: /day_point/dayPointId Odstraní aktivity.
- DELETE: /day_point/image Odstraní fotky, pokud je uživatel, který zadal požadavek, vlastníkem výletu.
- DELETE: /day_point/document Odstraní dokument, pokud je uživatel, který zadal požadavek, vlastníkem výletu.
- POST: /day_point/new Požadavek slouží k vytvoření nové aktivity.
- POST: /day_point/change Požadavek slouží ke změně doby trvání činnosti.

DutyController endpoints:

Všechny dotazy v tomto řadiči jsou nejprve zkontrolovány na oprávnění uživatele a poté na jeho práva cokoli v daných výletech měnit.

- POST: /duty/new Požadavek slouží k vytvoření nové dlužné částky.
- GET: /duty/calculation/tripId Dotaz vrátí výpočet všech dlužných částek vzniklých na cestě tak, aby cestující mezi nimi provedli co nejméně převodů mezi účty.
- DELETE:/duty/dutyId Odstraní pohledávku, pokud je uživatel, který zadal požadavek, jeho vlastníkem.

CommentController endpoints:

Všechny dotazy v tomto řadiči jsou nejprve zkontrolovány dle oprávnění uživatele a poté na jeho práva cokoli v daných výletech měnit.

- POST: /comment/new Požadavek slouží k vytvoření nového komentáře.
- DELETE: /comment/dutyId Odstraní komentář, pokud je uživatel, který zadal požadavek, jeho vlastníkem.

6.1.2 Error Handling

Zpracování výjimek je jedním z nejdůležitějších témat při vytváření rozhraní REST API. Spring poskytuje skvělou podporu pro zpracování výjimek pomocí nativních funkcí, které mohou pomoci přizpůsobit odezvu rozhraní API. Některé chyby vznikají voláním metody s neplatnými argumenty, jiné se stávají jen proto, že se něco provede špatně. Některé výjimky však služba může vyhodit sama, protože to, co jsme po ní chtěli, není možné provést. Pro zpracování chyb na jednom místě a pro pohodlnější implementaci používám třídu ControllerErrorHandler, která zpracovává chyby vzniklé při požadavcích od klienta a odpovídá klientovi jasnými kódy a komentáři, co se stalo. Ukázka kódu ControllerErrorHandler:

```
@ControllerAdvice
class ControllerErrorHandler {
companion object {
        private val LOG = LoggerFactory
        .getLogger(ControllerErrorHandler::class.java)}

@ExceptionHandler(Exception::class)
    fun unexpected(ex: Exception): ResponseEntity⟨ErrorResponse⟩ {
        LOG.error(ex.message, ex)
        val response = ResponseEntity
                  (ErrorResponse
                  (ResponseConstants.UNKNOWN.value,
                  "Unexpected an error"),
                  HttpStatus.INTERNAL_SERVER_ERROR)
        LOG.error("unexpectedException: {response.body.toString()}")
        return response
    }

@ExceptionHandler(ControllerException::class)
    fun processableException(controllerException: ControllerException):
            ResponseEntity⟨ErrorResponse⟩ {
        val res = ErrorResponse
        (controllerException.code, controllerException.message ?:
         "Empty message")
        val response = ResponseEntity.status
        (controllerException.status).body(res)
        LOG.error("Code: {controllerException.code}
        message: {controllerException.message}
        status: {controllerException.status}
        {response.body.toString()}")
        return response
    }
}
```

Všechny zpracovávané chyby jsou shromažďovány na jednom místě a všechny kódy chyb jsou shromažďovány ve enum, který pak lze použít i ve frontendové aplikaci, aby se zabránilo chybným zápisem. Ukázka kódu App Exceptions:

```
class DenyAccessException(message: String = "User credential invalid"):
ControllerException(message, ResponseConstants.INVALID_CREDENTIAL.value)
```

Ukázka kódu ResponseConstants:

```
enum class ResponseConstants(val value: String) {
    INVALID_CREDENTIAL("USR001"),
    INVALID_EMAIL("USR002"),
}
```

6.1.3 AuthService

Pro usnadnění kontroly oprávnění uživatelů byla v aplikaci použita knihovna „org.springframework.security.web“ Všechny požadavky odeslané na server jsou nejprve filtrovány

v JwtAuthTokenFilter, a pokud k nim není povolen přístup neautorizovaným uživatelům, je takový požadavek zachycen a zkontrolována autorizace uživatele. Takový postup se musí provést v každém ovladači zvlášť a veškerý ověřovací kód být na jednom místě. Také není nutné sledovat a kontrolovat datum expirace tokenu, veškerá práce probíhá automaticky. Vzhledem k tomu, že se jedná o uživatelskou aplikaci, datum vypršení platnosti tokenu je měsíc a automaticky se neobnovuje. Nutí tak uživatele, aby znovu zadal přihlašovací údaje. Uvedená knihovna je také výhodná, protože v budoucnu umožňuje rozšířit funkčnost práce s uživateli, přidělovat jim role, blokovat účty atd., což umožňuje rozšířit funkčnost aplikace bez časově náročných změn kódu. Ukázka kódu UserCredentialsBuilder:

```
return org.springframework.security.core.userdetails.User
            .withUsername(username)
            .password(user.password)
            .accountExpired(false)
            .accountLocked(false)
            .credentialsExpired(false)
            .disabled(false)
            .authorities("user")
            .build()
```

6.1.4 Bing Service API

Využití Bing API [23] je zaměřeno hlavně pro výpočet vzdáleností a cestovních časů mezi body zahrnutými v harmonogramu výletu. Nejprve bylo pro tyto účely používáno Google Geo API, které má ovšem velmi malé limity požadavků, jež byly v první fázi testování velmi rychle vyčerpány, takže musel být použita bezplatná služba Bing. Aplikace Bing je vyhovující, protože umožňuje nastavit nejen souřadnice výchozího a cílového bodu, ale také čas odjezdu, což umožňuje přesnější výpočet doby cesty s ohledem na dopravní zácpy, roční období atd. Nejprve je zpracován dotaz na odhad času, za který se do cíle uživatel dostane pěšky, a pokud tento čas přesáhne 5 minut, tak server naplánuje cestu autem.

6.1.5 Výpočet pohledávek

Během vývoje aplikace jsem musela vyvinout vlastní službu vypořádání dlužných částek, která spočívá v tom, že uživatelům usnadňuje vzájemné vypořádání po cestě. V aplikace může uživatel přiřadit výdaj k libovolné aktivitě a přidat je uživatelům na této cestě, nebo si jednoduše celý výdaj napíší na sebe, aby měli přehled o výdajích. Pro uživatele by však nebylo komfortní, kdyby se v případě účasti na jednotlivých akcích musel zobírat zjišťováním jednotlivých nákladů. Správce může vidět všechny výdaje na jednom místě a jednoduše získat dlužné částky napříč uživateli. Za tímto účelem byla vyvinuta služba pro výpočet pohledávek. Princip spočívá v tom, že nejprve se všechny dlužné částky rozdělí na uživatelské účty podle měny, přičemž na účtu může být záporná částka. Poté se dlužníci a příjemci rozdělí do dvou skupin a vyrovnání dluhu se provádí na základě největších částek. Tímto způsobem lze zjistit minimální počet převodů peněz na konci cesty.

6.1.6 MailService

Poštovní služba z knihovny org.springframework.mail.javamail.JavaMailSender slouží k oznámení uživateli, že byl pozván k účasti na cestě a k zaslání kódu pro obnovení hesla.

6.1.7 Open Meteo Service API

Meteorologické API se používá k zobrazení počasí na několik následujících dní, což není popsáno v požadavcích uživatele, ale slouží jako bonusová funkce. Výpočet se provádí pouze v případě, že den cesty není vzdálen více než 5 dní od dnešního data, a vypočítává se pro konkrétní místo a slouží jako rychlý odkaz na počasí během samotné cesty.

6.1.8 Hostování

Backendová část aplikace je umístěna na cloudovém serveru Heroku, kde je také umístěna databáze aplikace. Heroky se osvědčil jako spolehlivá služba pro hosting takových aplikací. Jedná se cloudovou, vícejazyčnou platformu, jakožto službu založenou na spravovaném kontejnerovém systému s integrovanými datovými službami a vyspělým ekosystémem pro nasazování a provoz aplikací. Heroku je Platform as a Service, což znamená, že platforma funguje jako služba: Poskytuje uživateli určité funkce a vlastnosti, přístup k systémům a softwaru. Zároveň je jeho infrastruktura zcela skrytá. Zaměstnanci služby dělají vše za uživatele - práce zůstává „pod kapotou“ a mnoho procesů je automatizováno. Za zabezpečení, architekturu a konfiguraci serveru zodpovídají specialisté platformy. Heroku je výhodné z následujících důvodů:

- Hostování aplikací a webových služeb;
- zjednodušení a zrychlení vývojového cyklu;
- snížení potřeby složité obsluhy serveru;
- zpracování vysoce zatížených aplikací;
- rychlé rozšiřování projektů.

6.2 Implementace frontendu

Hlavním tématem diplomové práce je vývoj funkční a uživatelsky přívětivé aplikace pro plánování cestování. Tomu byla věnována maximální pozornost a většina času. Při přípravě projektu jsem vytvořila několik menších na platformě Android. Vyzkoušela jsem klasický přístup pomocí XML atributu pro uživatelské rozhraní pomocí aktivit a fragmentů. Druhá aplikace pak již byla napsána pomocí relativně nového frameworku Jetpack Compose založeného na přístupu jedné aktivity, který v posledním roce nabyl na popularitě. Aplikace Jetpack Compose je založena na composable funkcích, protože ty jsou „základními“ kameny, z nichž se skládají stromy composable funkcí. Jakákoli funkce v jazyce Kotlin se stává složitelnou funkcí pomocí anotace @Composable. Každá funkce s anotací @Composable je kompilátorem (Jetpack Compose Compiler) doplněna o další parametr Composer-context, který je předáván všem podřízeným composable funkcím uvnitř této funkce atd. Nejdůležitějším, co nám uvedené funkce poskytují, je rekompozice, tedy odvolání composable funkcí při změně dat, na kterých jsou závislé. To znamená, že takový přístup odděluje data a jejich reprezentaci a umožňuje uživatelskému rozhraní neukládat žádný stav sám v sobě, ale pouze vykreslovat změny dat. Tento framework se ukázal být výhodnější a podle mého názoru má více výhod než klasický přístup. Proto jsem se ho rozhodla použít ve své diplomové práci.

6.2.1 Implementace stránkování ve frontendu

Jelikož Jetpack Compose ve většině případů nepoužívá přístup založený na více aktivitách a fragmentech, musela jsem nejprve vyřešit otázku navigace mezi jednotlivými obrazovkami aplikace. Pomohla mi s tím knihovna „androidx.navigation:navigation-compose:2.5.1“. Přístup k navigaci v aplikaci Jetpack Compose je poměrně jednoduchý. A spočívá ve vytvoření zapečetěné třídy Routes, která obsahuje informace o všech

cestách v aplikaci a má funkce pro vytváření cest s předáváním parametrů. Dale je uveden příklad kódu.

```
sealed class Routes(val route: String) {
    object MainRoute : Routes("main")
    object AuthRoute : Routes("auth")
    object CreateNewTripRoute : Routes("main/new")
    object TripRoute : Routes("trip/{tripId}"){
        fun createRoute(tripId: Long)
        = "trip/{tripId}"
    }
    object TripMainInfoRoute : Routes("trip/{tripId}/info"){
        fun createRoute(tripId: Long)
        = "trip/{tripId}/info"
    }
    object TripMapRoute : Routes("trip/map/{dayId}/{dayTitle}"){
        fun createRoute(dayId: Long, dayTitle: String)
        = "trip/map/{dayId}/{dayTitle}"
    }
    object DayPointRoute : Routes("trip/{tripId}/{dayPointId}"){
        fun createRoute(tripId: Long, dayPointId: Long)
        = "trip/{tripId}/{dayPointId}"
    }
}
```

Dále byl v rámci architektury aplikace vytvořen objekt NavController, který je obsažen v AppState celé aplikace. Následuje předání požadovaným komponentám aplikace a vyvolání navigace pomocí tlačítek nebo událostí. Některé přechody mezi stránkami jsou samozřejmě vytvořeny tak, že přechod na ně vymaže předchozí zásobník stránek, například při přechodu na stránku autorizace nebo naopak. Za tímto účelem jsou napsány samostatné funkce, které vymažou předchozí zásobník a umístí požadovanou stránků na přední pozici. Dale je uveden příklad kódu:

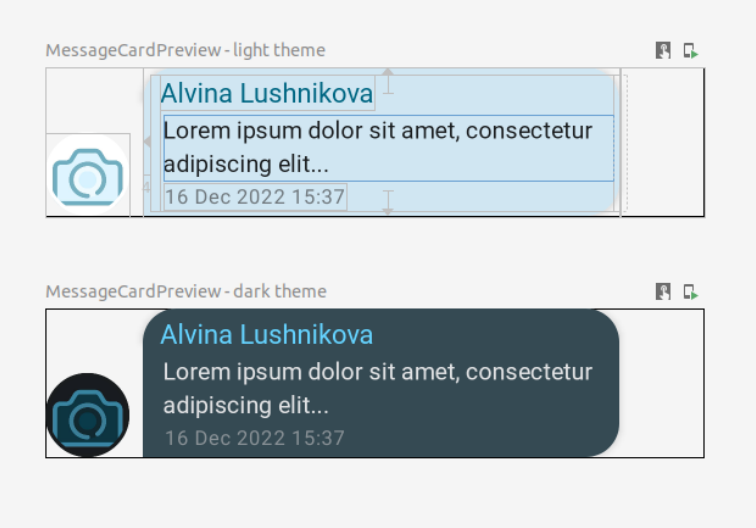
```
 fun navigateToAuth(){
        navController.navigate(Routes.AuthRoute.route){
            launchSingleTop = true
            popUpTo(Routes.MainRoute.route) {inclusive = true}}}
```

6.2.2 Vytváření grafických prvků

Hlavní výhodou použití Jetpack Compose je, že všechny komponenty lze použít několikrát, například jednou napsaná komponenta uživatele se zaškrtávacím políčkem byla v projektu použita několikrát. Velmi pohodlné bylo také použití Lazy Columns nebo Lazy Rows pro komponenty seznamu namísto psaní těžkopádných adaptérů. Komponenty Lazy se liší od většiny rozložení v aplikaci Compose. Místo toho, aby přijímaly parametr @Composable content block, který umožňuje aplikacím přímo emitovat složené objekty, poskytují komponenty Lazy blok - LazyListScope. Tento blok LazyListScope umožňuje popsat obsah položek. Komponenta Lazy je pak zodpovědná za přidání obsahu každé položky podle požadavků rozvržení a pozice posouvání. [24] ConstraintLayout je rozvržení, které umožňuje umístit na obrazovku Composable prvky vzhledem k jiným Composable prvkům. Je to alternativa k použití více vnořených řádků, sloupců,

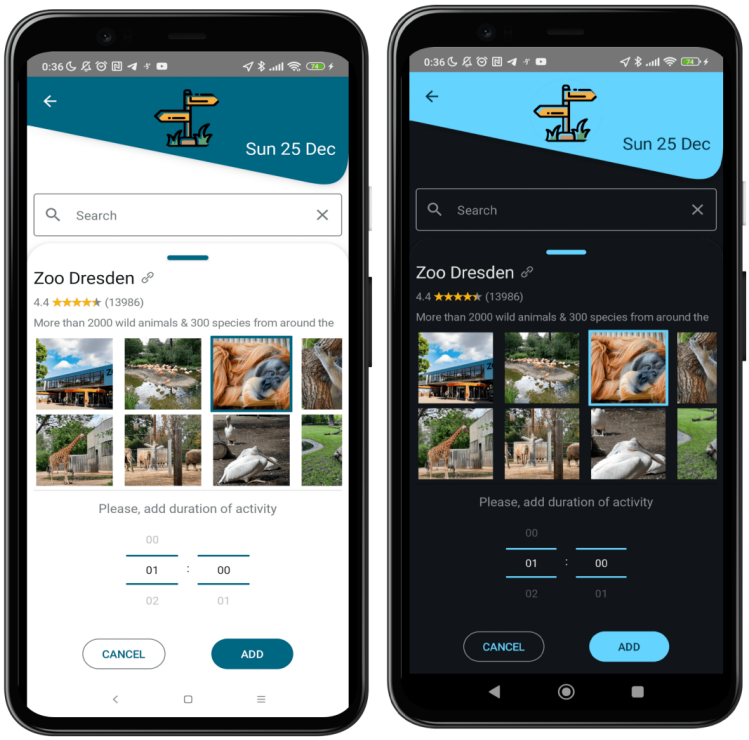
rámečků a dalších prvků vlastního rozvržení. ConstraintLayout je užitečný při implementaci větších rozvržení s komplikovanějšími požadavky na zarovnání. Ve frameworku Jetpack Compose se používá Constraint Layout, který v sobě vytváří organismy pro grafické komponenty. Omezení definuje vztah mezi komponentami v rámci nadřazené komponenty a s nadřazenou komponentou. V níže uvedeném příkladu kódu je možné vidět, jak má textový kontejner ohraničující rámeček ve vztahu k horní, dolní a počáteční části nadřazené komponenty. Ukázka kódu Constraint Layout:

```
ConstraintLayout() {
 val (name, text, time) = createRefs()
     Column(modifier = Modifier.constrainAs(name) {
           top.linkTo(parent.top, 4.dp)
           start.linkTo(parent.start, 4.dp)
           bottom.linkTo(parent.bottom, 4.dp)})}
```



Obrázek 6.1. Ukázka grafickeho prvků.

Jetpack Compose a Material 3 Design poskytuje velmi dobrou možnost nastavení téma aplikace. Ve standardních verzích je to pouze tmavé a světlé téma, ale obecně je možné modifikovat různé části aplikace. Všechny barvy, tvary a písma jsou definovány v několika proměnných, které jsou pak spojeny do motivu, který lze použít pro celou aplikaci. V budoucnu bude velmi výhodné změnit komponentu na jednom místě a změnit ji v celé aplikaci. To dává určitou flexibilitu při vývoji aplikace a provádění změn v jejím průběhu, což ušetří spoustu času. Ukázku výsledné aplikace lze vidět v příloze D.



Obrázek 6.2. Ukázka nastavení téma aplikace.

Pro zobrazení fotografií ve své práci používám knihovnu „io.coil-kt:coil-compose:2.2.0“. Umožňuje asynchronně nahrávat fotografie z různých zdrojů. V rámci mé práce se jedná o fotografie, které jsou uloženy ve Firebase, Google Photos, dále soubory v telefonu.

6.2.3 Ukládání uživatelských údajů

Při práci na aplikaci jsem potřebovala ukládat údaje o uživateli, jako je ID a token pro přístup k datům na serveru. Mohla jsem k tomu použít interní databázi, ale to není příliš pohodlné z hlediska přístupu k databázi pouze z asynchronních funkcí a nutnosti přistupovat k ní z úplně jiných komponent kódu, například pro zjištění, zda se uživateli zobrazí ikona pro odstranění komentáře, tj. pro kontrolu, zda je komentář jeho. Jak doporučuje oficiální dokumentace systému Android: *„Pokud máte relativně malou sbírku párů klíč-hodnota, kterou byste chtěli uložit, měli byste použít SharedPreferences API. Objekt SharedPreferences ukazuje na soubor obsahující páry klíč-hodnota a poskytuje jednoduché metody pro jejich čtení a zápis.“* [25] Pro aplikaci jsem vytvořila třídu App Preferences, která se řídí programovacím patternem Singleton a má pouze jednu instanci objektu, ve které jsou uložena všechna uživatelská data a token podle klíče a hodnoty. Při odhlášení se data vymažou.

6.2.4 Ukládání a validace stavy formulářů

Tato aplikace používá poměrně mnoho formulářů, které musí uživatel vyplnit jak při registraci nebo přihlášení do aplikace, tak při vytváření cesty nebo aktivity. Celý formulář je platný, když jsou platná všechna jeho pole a je lepší zobrazit chybu ne hned v průběhu vyplňování pole formuláře, ale jakmile pole přestane být aktivní. Pro tento účel jsem vytvořila TextFieldState, ze kterého se dědí například EmailFieldState s vlastními pravidly validace a textovou chybou, která se zobrazí, pokud je pole neplatné. Tyto stavy

se ukládají pomocí composable funkce „remember“, jejíž hlavní účelem je ukládání dat objektu i přes rekompozici hlavní funkce. Ukázka kódu ukládání stavů pole:

```
val passwordFocusRequest = remember { FocusRequester() }
val repeatPasswordFocusRequest = remember { FocusRequester() }
val emailState = remember { EmailState() }
val passwordState = remember { PasswordState() }
fun repeatPasswordChange(password: String): Boolean {
        return !(password != passwordState.text)
}
val repeatPasswordState
= remember { RepeatPasswordState(::repeatPasswordChange) }

     Email(emailState, onImeAction
     = { passwordFocusRequest.requestFocus() })
     Password(
            label = stringResource(id = R.string.password),
            passwordState = passwordState,
            imeAction = ImeAction.Next,
            onImeAction = {repeatPasswordFocusRequest.requestFocus() },
            modifier = Modifier.focusRequester(passwordFocusRequest))
     RepeatPassword(
            label = stringResource(id = R.string.repeat_password),
            passwordState = repeatPasswordState,
            imeAction = ImeAction.Next,
            onImeAction = { },
            modifier = Modifier
            .focusRequester(repeatPasswordFocusRequest))
```

Ve výše uvedeném příkladu kódu je vidět, že když jsou všechna pole vyplněna shora dolů, kurzor se po dokončení pole přesune do pole níže a v poli výše se zobrazí chyba, pokud informace není validní. Možnost kliknout na potvrzovací tlačítko formuláře je možná až po ověření všech polí formuláře, což umožní vyhnout se odesílání neplatných údajů na server, které server stejně odmítne.

6.2.5 Implementace skupinového nahrávání obrazovky

Bylo pro mě velmi důležité, aby si aplikace průběžně vybírala fotografie a dokumenty. Problémů s dokumenty bylo méně, protože uživatelé obvykle nepřipojují k jedné činnosti více než jeden dokument. Na druhou stranu fotografií obvykle chtějí vložit několik. Za tímto účelem jsem se inspirovala funkcemi aplikace Telegram a rozhodla jsem se implementovat výběr fotografií prostřednictvím zobrazení alba přímo ve své aplikaci. Existuje několik knihoven, ve kterých již byla tato funkce implementována, nicméně to nepomohlo, protože docházelo ke konfliktu s verzemi knihoven v aplikaci. Pokud se implementace týká přístupu k uživatelským datům, jako jsou kontakty, fotografie, videa, dokumenty atd. Vývojáři by s tím měli počítat a vždy si vyžádat svolení uživatele k úkonům nezbytným během provozu aplikace. Oprávnění aplikace pomáhají podporovat soukromí uživatelů tím, že chrání přístup k následujícímu: Omezená data, jako je stav systému a kontaktní informace uživatelů. Omezené akce, jako je připojení ke spárovanému zařízení a nahrávání zvuku. Pro přístup k internímu úložišti souborů v zařízení si aplikace okamžitě vyžádá několik oprávnění k prohlížení interního úložiště a přístupu k

fotoaparátu, protože v budoucnu se plánuje rozšíření funkčnosti nejen o nahrávání hotových fotografií, ale také pomocí fotoaparátu a výsledky fotografování jsou okamžitě nahrány na server. Pro ukladaní fotografie pomocí FirebaseStorage se vytvoří seznam úkolů a teprve po nahrání všech fotek do databáze serveru odešle požadavek na uložení odkazů. Podobná práce ve DayPointViewModel nastává s dokumenty, rozdíl je pouze v počtu souborů. Pro stahování dokumentů a fotografií je uživatel požádán o povolení zápisu do vnitřního úložiště telefonu.

Obrázek 6.3. Ukázka nastavení skupinového nahrávání obrazovky.

6.2.6 Ovládání stavu dat

Při odesílání požadavku na server může dojít k chybě. V této aplikaci jsou všechny stavy uvedené v aplikaci spravovány z ViewModelu, přičemž pro každou stránku existuje příslušný ViewModel. Tato třída uchovává stav dané stránky uživatelské rozhraní a sleduje tento stav. Při požadavku na data ze serveru se stav změní, a teprve když server odpoví, uživatelské rozhraní zobrazí data nebo chybu. Všechny požadavky jsou prováděny jako součást asynchronní funkce, aby nedošlo k zablokování běhu hlavního vlákna aplikace, k čemuž Kotlin poskytuje Coroutines. Ukázka kódu asynchronního dotazu na server:

```
private val _loadingState = MutableStateFlow(LoadingState.IDLE)
val loadingState: StateFlow⟨LoadingState⟩
    get() = _loadingState.asStateFlow()
fun logInWithEmailAndPassword(email: String, password: String)
= viewModelScope.launch {
        withContext(Dispatchers.IO) {
            try {
```

```
                _loadingState.value = LoadingState.LOADING
                userRepository
                .signInWithEmailAndPassword(email, password)
                _loadingState.value = LoadingState.LOADED
                withContext(Dispatchers.Main) {
                    appState.navigateFromAuthToMain()
                }

            } catch (e: Exception) {
                _loadingState.value
                = LoadingState.error(e.localizedMessage)
            }
        }
    }
```

V aplikaci se všechna data nečtou přímo ze serveru, ale nejprve se uloží do databáze a poté se zobrazí v uživatelském rozhraní. Provádí se to z důvodu, aby měl uživatel v zařízení nejnovější data a pro případ problémů s internetovým připojením. Za tímto účelem má aplikace interní databázi dat, datové modely a úložiště pro čtení a ukládání dat. Databáze v systému Android vyžaduje spoustu práce s psaním kódu a ručním vytvářením cross references tabulek. Zatímco odkazy typu one-to-many se vytvářejí poměrně snadno, odkazy typu many-to-many vyžadují samostatné tabulky a DAO pro ukládání dat. Jedná se o přístup navržený v oficiální dokumentaci systému Android pro interní databázi využívající knihovnu „androidx.room:room-runtime:2.4.3“. Ukázka kódu Model dat:

```
@Entity(
    primaryKeys = ["dutyId", "userId"],
    indices = [Index("dutyId"), Index("userId")],
    foreignKeys = [
        ForeignKey(
            entity = Duty::class,
            parentColumns = arrayOf("dutyId"),
            childColumns = arrayOf("dutyId"),
            onDelete = ForeignKey.CASCADE,
        ), ForeignKey(
            entity = User::class,
            parentColumns = arrayOf("userId"),
            childColumns = arrayOf("userId"),
            onDelete = ForeignKey.CASCADE
        )
    ]
)
data class UserDutyCrossRef(
    val dutyId: Long,
    val userId: Long
)

data class DutyWithUsers(
    @Embedded val duty: Duty,
```

```
    @Relation(
        parentColumn = "dutyId",
        entityColumn = "userId",
        associateBy = Junction(
            value = UserDutyCrossRef::class,
            parentColumn = "dutyId",
            entityColumn = "userId",
        )
    )
    val users: List⟨User⟩
)
```

6.2.7 Implementace spojení se serverem

Retrofit je typově bezpečný klient REST pro Android, jehož cílem je usnadnit konzumaci webových služeb RESTful. Retrofit 2 z knichoven „com.squareup.retrofit2:retrofit:2.9.0“ a „com.squareup.retrofit2:converter-jackson:2.9.0“ ve výchozím nastavení využívá OkHttp jako síťovou vrstvu a je postaven nad ní. Retrofit automaticky serializuje odpověď JSON pomocí POJO, který musí být pro strukturu JSON definován předem. Pro serializaci JSON je třeba nejprve použít konvertor, který jej převede na Gson, který dovoluje knihovny „de.grundid.opendatalab:geojson-jackson:1.6“ a „com.fasterxml.jackson.module:jackson-module-kotlin:2.12.0“. Příklad deklarace POST požadavky a Web API pro aplikaci:

```
private val retrofit = Retrofit.Builder()
    .baseUrl(BASE_URL)
    .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
    .addConverterFactory(JacksonConverterFactory
    .create(jacksonObjectMapper()))
    .registerModule(JavaTimeModule())
    .client(OkHttpClient.Builder()
    .readTimeout(60, TimeUnit.SECONDS)
    .connectTimeout(60, TimeUnit.SECONDS).build())
    .build()
interface AppWebApi {
    @POST("user/signup")
    fun signUp(@Body user: UserRegistrationDto):
     Single⟨UserDto⟩
    @POST("user/signin")
    fun signIn(@Body user: UserLoginDto):
    Single⟨JwtResponse⟩
}
```

6.2.8 Implementace vyhledávání míst v Google Places API

Google Places API slouží k vyhledávání míst v blízkosti bodu definovaného uživatelem. API je popsáno v oficiální dokumentaci pro vývojáře systému Android [26]. Výsledky poskytované službou Google Places API ukládá aplikace do databáze. Aby se snížil počet dotazů na tuto službu, jsou podrobné informace o místě vyžadovány až po požadavku uživatele. Google Places API umožňuje vyhledávat místa podle kategorií i slovních dotazů. Služba byla použita, protože pro ni existuje mnoho výukových programů. Příklad deklarace GET požadavky a Web API dotazy k Google Places API:

```
@GET("nearbysearch/json")
    suspend fun getCandidatePlacesByType(
        @Query("location") location: String,
        @Query("radius") radius: Int,
        @Query("type") type: String,
        @Query("key") key: maps-key, ): Response
    @GET("details/json")
    suspend fun getCandidatePlaceDetails(
        @Query("place_id") place_id: String,
        @Query("key") key: String = maps-key, ): PlacesDetailsResponse
```

6.2.9 Další použité knihovny a služby

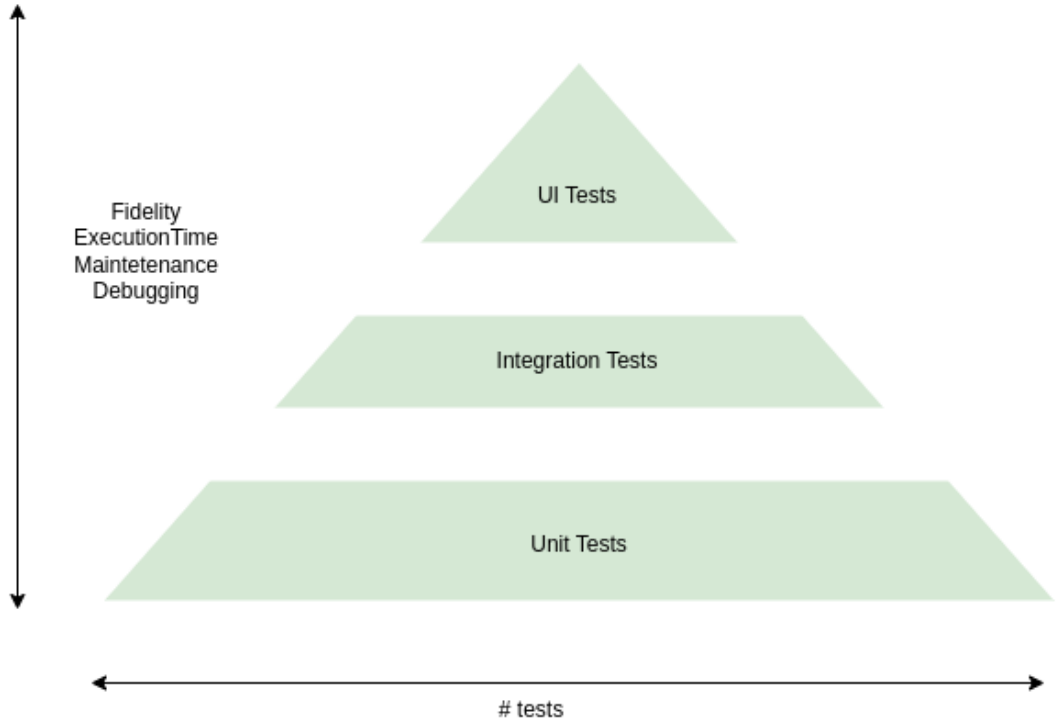
- Android support libraries - Sbírka vývojových knihoven pro Android, pomocných tříd a nástrojů.
- Material Design 3 je přizpůsobitelný systém pokynů, komponent a nástrojů, které podporují osvědčené postupy návrhu uživatelského rozhraní.
- Google Map API - pomocí dat Map Google, zobrazí mapy a reakcí na mapová gesta. Také poskytne další informace o místech na mapě a podpoří interakci uživatelů přidáním značek, polygonů a překryvů do mapy. Službu jsem použil , protože pro ni existuje mnoho návodů a také díky vlastním zkušenostem s implementací.
- Google Auth Service - umožňuje ověřit uživatele pomocí účtu Google.
- Wheel picker - malá knihovna pro nastavení doby trvání události. Použita byla také různá řešení s otevřeným zdrojovým kódem s licencí Apache 2.0, například z demo aplikací pro Android s příklady čistého kódu. Odkazy na ně jsou uvedeny přímo v kódu.

Kapitola 7
Testování aplikace

Testování softwaru je empirický technický výzkum kvality testovaného produktu nebo služby prováděný za účelem poskytnutí těchto informací všem zainteresovaným (=stakeholderům). Testování je tedy zejména o hledání určitých informací o produktu a jeho dalším zkoumáním. Dimenzí kvality jsou:

- Funkčnost (Functionality) – správné chování funkcí systému, jak je definováno funkční specifikací.
- Použitelnost (Usability) – zda vůbec a jak lze dosáhnout požadovaného cíle, zda je systém uživatelsky přívětivý a zda se s ním dobře pracuje. I když se SW chová podle uživatele „špatně“, stále může jít o „správné chování“ podle FS, pak jde ovšem o chybu samotného FS a smlouvy, kterou je vázán.
- Spolehlivost (Reliability) – zda se chová stejně za všech okolností, zvláště po přetížení, výpadku či chybě a zda takové stavy umí detekovat a hlásit.
- Výkon (Performance) – zda systém není pomalý a zvládne větší množství současně připojených uživatelů, nebo naopak, zda si i při naplnění všech požadavků na obsluhu uživatelů nebere příliš systémových zdrojů.
- Podporovatelnost (Supportability) – zda se systém dobře instaluje, nemá problémy s cílovými hardwarovými a softwarovými konfiguracemi a další vlastnosti související s údržbou systému a upgradovatelností.
- Bezpečnost (Security) – zda jsou systém i jeho data bezpečné a mohou jej používat pouze oprávněné osoby.
- Kompatibilita (Compatibility) – zda je možné jej používat i s ostatními programy a systémy.
- Přenositelnost (Portability) – zda je možné jej přenášet například na jiný operační systém nebo HW [27]

Další způsob dělení testování podle fáze testování. Pyramida testů by se dala shrnout jako strategie rozdělení testů. V tomto dělení obsahují nižší vrstvy rychlejší, levnější a izolované testy. V nejvyšší úrovni pyramidy jsou pomalejší, dražší a integrované testy. V návaznosti na uvedený koncept je důležité zmínit, že jak se stoupá po jednotlivých vrstvách pyramidy, mělo by docházet ke snižování počtu testů, jak je znázorněno na ilustračním příkladu 7.1.



Obrázek 7.1. Pyramida testů [27]

Společnost Google má také vlastní definici testovací pyramidy pro systém Android, která se dělí na malou, střední a velkou vrstvu. Podle nich by ideální rozdělení testování pro Android bylo následující: „ Ačkoli se podíl testů pro jednotlivé kategorie může lišit v závislosti na případech použití vaší aplikace, obecně doporučujeme následující rozdělení mezi kategorie: 70 % malých, 20 % středních a 10 % velkých“ [28].

7.1 Unit testy

Testování jednotek (unit testing) testuje jednotlivé elementární části kódu. Unit testování musí pokrývat nejen frontend, ale také backend aplikace. Nejpopulárnější přístup v testování Kotlin aplikací je používání knihovny Junit. Elementárním testem v JUnit je testovací metoda s přidanou anotací @Test. Pro svou aplikaci potřebuji otestovat funkce odpovědné za obchodní logiku a přidávání objektů do databáze. Kromě toho bude třeba testovat rozhraní API, včetně zpráv s chybnými vstupními údaji. Stejně tak testování frontendu aplikace vyžaduje použití Unit testů. Jednotkové testy v systému Android lze rozdělit na 2 typy: Lokální unit testy ke svému běhu využívají pouze JVM. Jsou určeny k testování obchodní logiky, která neinteraguje s operačním systémem. Instrumentované unit testy testují logiku „svázanou“ s rozhraním Android API. Provádějí se na fyzickém zařízení - emulátoru, což zabere podstatně více času než místní testy.

7.2 Integrační testy

I když jsou hranice mezi jednotlivými kategoriemi testů neostré, klíčovou vlastností integračního testu je, že se zabývá více částmi aplikace. Zatímco jednotkové testy vždy berou výsledky z jedné jednotky, například z volání funkce, integrační testy mohou agregovat výsledky z různých částí a zdrojů. V integračním testu není třeba oddělit části

aplikace. Možné je nahradit externí systémy, ale aplikace funguje nadále integrovaně. Dále se provádí externí testování dotazů na server. Integrační testy probíhají jako série požadavků na server v podobě několika scénářů, včetně chybných požadavků, které musí zachytit ErrorHandler a vrátit správný kód chyby.

7.3 UI testy

Testování interakcí s uživateli pomáhá zajistit, aby se uživatelé při interakci s aplikací nesetkali s neočekávanými výsledky nebo neměli špatné zkušenosti. Jeden z přístupů k testování uživatelského rozhraní spočívá v tom, že tester jednoduše provede sadu uživatelských operací s cílovou aplikací a ověří, zda se aplikace chová správně. Takový manuální přístup však může být časově náročný a náchylný k chybám. Druhý přístup umožňuje napsat testy uživatelského rozhraní tak, aby se uživatelské akce prováděly automatizovaně. Automatizovaný přístup umožňuje provádět testy rychle a spolehlivě opakovatelným způsobem. Při práci na své aplikaci buly používaný oba přístupy. Oficiální dokumentace společnosti Google navrhuje, abychom pro automatizované testování použili knihovny UI tests in Compose „androidx.compose.ui:ui-test-junit4:1.2.1“, které umožňují vytvářet testy uživatelského rozhraní aplikace přímo na testovacím zařízení [29]. Manuální testy jsou nutné pro testování aplikací pomocí klikacího prototypu dlouho předtím, než proběhne implementace. Použitelnost je míra, do jaké používání produktu vypadá efektně a navozuje spokojenost - radost z používání. Použitelnost není náhodná a zajišťují ji metody:

- Formativní (obvykle kvalitativní) se zabývají chováním uživatelů a s náhledem na problémy uživatelů s použitelností.
- Sumativní metody (obvykle kvantitativní) slouží především k ověření hypotéz o chování uživatelů.

7.4 Automatizované testování uživatelského rozhraní

Testování uživatelských rozhraní nebo obrazovek slouží k ověření správného chování kódu Compose a zlepšuje kvalitu aplikace tím, že zachytí chyby v rané fázi vývoje. Compose poskytuje sadu testovacích rozhraní API pro vyhledávání prvků, ověřování jejich atributů a provádění uživatelských akcí. Zahrnují také pokročilé funkce, jako je manipulace s časem. V aplikaci byly testovány některé funkce, jako je přihlašování, vyplňování některých formulářů a polí a také navigace.

Zařízení	Verze OS	Velikost displeje	Rozlišení displeje	Operační paměť
Redmi Note 11	Android 11	6.67	2040x1080	8GB
Ulefone Armor 9	Android 10	6.3	1080x2340	8GB
Xperia Z1	Android 8	5.2	1920x1080	4GB

Tabulka 7.1. Technické parametry testovacích zařízení.

7.5 Testování Lo-fi prototypu uživatelského rozhraní

Jak říká Boehmův první zákon: „Chyby jsou nejčastější ve fázích požadavků a návrhu, a čím později odhalíme chybu v průběhu vývoje systému, tím je dražší“. Vzhledem k tomu,

že na projektu pracuji sama s využitím metodiky vývoje softwaru podle V-modelu, budu požadovat test uživatelského rozhraní co nejdříve. Pro test uživatelského rozhraní mohu použít kvantitativní nebo kvalitativní metodiku testování. U oblíbeného A/B testování, kdy je uživateli nabídnuta možnost výběru z několika variant interakce s uživatelem a ve finále jsou akceptovány pouze pozitivní varianty, ale taková možnost je vhodnější spíše pro poslední fáze vývoje, možnost testování na reálných uživatelích a využití nástrojů, jako je Google Analytic. Naopak kvantitativní testování lze provádět na malé skupině lidí, pokud jsou z cílové skupiny a mají dostatečné zkušenosti s používáním mobilních aplikací. Tato studie bude moderovaná a já budu pracovat s testovaným a vyplňovat testovací protokoly. Cíle testování: Zjistit, zda jsou uživatelé schopni úspěšně dokončit určité úkoly (např. vytvořit cestu, najít informace). Určit, jak dlouho trvá dokončení konkrétních úkolů. Zjistit, zda jsou uživatelé s produktem spokojeni a jak jej lze zlepšit.

7.5.1 Příprava testovacího scénáře

Pro uživatelské testy je třeba vytvořit scénáře použití aplikace. Například scénář přidání aktivit do cesty by zahrnoval následující kroky -> otevření aplikace -> otevření cesty s názvem Londýn -> přidání aktivity návštěva London Eye dne 5. května v 9:00. Postup tvorby jednotlivých testovacích scénářů zahrnuje zodpovězení následujících otázek: Otázka k definici úlohy:
Q0 „Čeho chce uživatel dosáhnout?“
Otázky pro každý krok:
Q1 „Bude uživatelům zřejmá správná akce?“
Q2 „Spojí si uživatelé označení akce s tím, co je správné s jejich cíli?“
Q3 „Dostane uživatel rozumnou zpětnou vazbu?“
Podle scénáře budou činnost provádět nezkušení uživatelé. Po spuštění budou hodnotit, jak snadné je najít určité prvky a jak rychle pochopí rozhraní a dostanou se k cíli. Po obdržení výsledků testu klikacího prototypu je nutné jej upravit včetně designu, aby byla aplikace uživatelsky co možná nejpřívětivější. Poté je nutno testování zopakovat. Teprve po získání pozitivních výsledků UI testů lze v implementaci aplikace pokračovat. Scénáře interakce uživatele s produktem jsou navrženy na základě případů použití. Protože mohou být uživatelé v aplikaci při interakci s cestou v různých rolích (vlastník cesty a pozvaný uživatel), je nutné otestovat obě role. Za tímto účelem byly připraveny dva scénáře se základními soubory možných interakcí. Požadavky na funkčnost uživatelského rozhraní low fidelity prototype byly zpracovány ve formě případů užití (use-cases). Jejich přehled je v následující tabulce 7.2:

Testovací scénáře čislo 1. Uživatel založil výlet sam

- Úkol 1
  Chcete si vyjet na výlet. Využijte prototyp aplikace „Way planner“ a zkuste se přihlásit pomocí Google nebo e-mail účtu.
- Úkol 2
  Chcete si naplánovat výlet. Přidejte první cestu, pojmenujte ji, zadejte na mapě první a koncový bod cesty, nastavte datum a čas. Uložte výsledek.
- Úkol 3
  Chcete přidat první cíl. Například kavárnu v Drážďanech. Přejděte na cestu, otevřete první den a přidejte kavárnu výběrem na mapě.
- Úkol 4
  Chcete jet s přáteli na výlet autem, ve kterém máte dvě volná místa, a dohodli jste se, že s sebou vezmete další dva přátele. S jedním z nich jste již pomocí této aplikace na

Id	User	Admin	Use case
UC1	Not Logged	-	Po startu aplikace se uživatel přihlásí pomocí Google/e-mail.
UC2	Logged	-	Uživatel si zobrazí dokončené a budoucí výlety
UC3	Logged	-	Uživatel přihlásil kamarády a sdílí s nimi určité výlety
UC4	Logged	-	Uživatel přidá nový výlet
UC5	Logged	-	Uživatel nastaví název výletu, vybere počáteční a koncový bod na mapě, datum a čas a určí další uživatele, kterým bude výlet k dispozici.
UC6	Logged	-	Uživatel přidává nové aktivity podle kategorií do každého z cestovních dnů tak, že označí na mapě bod a nastaví dobu trvání.
UC7	Logged	ano	Uživatel upravuje jednotlivé aktivity, prohlíží/přidává fotografie, výdaje a komentáře.
UC8	Logged	ne	Uživatel si prohlédne jednotlivé aktivity, prohlíží/přidává fotografie, výdaje a komentáře.
UC9	Logged	ano	Uživatel upravuje výlety, prohlíží/přidává kamarády a fotografie. Uživatel se podívá na celkové náklady, na plátce a na to, kolik dluží každý člen cestovní skupiny.
UC10	Logged	ano	Uživatel prohlíží výlety, prohlíží/přidává fotografie. Uživatel se podívá na celkové náklady, plátce a na to, kolik dluží každý člen cestovní skupiny.
UC11	Logged	-	Uživatel si zobrazí plán cesty a zjistí, jak dlouho mu bude trvat cesta mezi jednotlivými body.
UC12	Logged	-	Uživatel se odhlásí

Tabulka 7.2. Požadavky na funkčnost uživatelského rozhraní low-fidelity prototypy

výlet vyrazili, s druhým ještě ne. Pozvěte své přátele, aby si mohli plán prohlédnout, sledovat jeho změny, přidávat výdaje, fotografie a komentáře.

- Úkol 5
  Na výletě jste navštívili Muzeum dopravy v Drážďanech a koupili vstupenky pro všechny své přátele. V aplikaci je zadejte, abyste věděli, jaký je rozpočet na cestu a kdo komu kolik dluží.
- Úkol 6
  hcete přidat fotky, který uvidí i vaši přátelé. Přidejte fotky.
- Úkol 7
  Muzeum se vám opravdu líbí, chcete napsat komentář, který uvidí i vaši přátelé. Zanechte komentář.
- Úkol 8
  Až cesta skončí, budete chtít vidět všechny fotografie. Jděte si prohlédnout fotografie z celého výletu.
- Úkol 9
  Po skončení cesty se chcete podívat na výdaje a na to, kdo vám dluží peníze. Přejděte na cestu a prohlédněte si výdaje.
- Úkol 10
  Chcete se odhlásit. Odhlaste se.

Testovací scénáře čislo 2. Uživatel dostal od kamaráda pozvánku na výlet

- Úkol 1
  Dostali jste od kamaráda emailem pozvánku na výlet, na který vám nabídne naplánovat si společný výlet prostřednictvím aplikace. Zaregistrujte se do aplikace.
- Úkol 2
  Chtěli byste vidět plán na první den. Podívejte se na plán prvního dne vaší nadcházející cesty.
- Úkol 3
  Do Technického muzea v Drážďanech se mi nechce, ale kamarád to naplánoval. Přidejte poznámku, že nejste dítě, abyste chodili do dětských muzeí.
- Úkol 4
  Během cesty jste si zaplatili společnou večeři v restauraci, připočtěte tyto výdaje k požadované položce a uveďte pouze svého přítele, protože jste v restauraci byli jen vy dva. Zbytek skupiny se vydal na fotbal.
- Úkol 5
  Až cesta skončí, budete chtít vidět všechny fotografie. Jděte si prohlédnout fotografie z celého výletu.
- Úkol 6
  Po skončení cesty se chcete podívat na výdaje a na to, kdo vám dluží peníze. Přejděte na cestu a prohlédněte si výdaje.
- Úkol 7
  Chcete se odhlásit. Odhlaste se.

7.5.2 Příprava testovaní uživatelského rozhraní

Dále jsem vytvorila klikací low-fidelity prototypy pro testování každého scénáře. Nejvhodnějším nástrojem pro tento úkol je Figma. Tento program umožňuje připravit klikací rámečky a umístit na ně informace o scénářích interakce. Obrazovky zahrnují pouze základní případy použití a zabývají se různými chybami, oznámeními a dalšími scénáři interakce. Prototyp je možné vidět v příloze C

Dalším krokem bylo najít respondenty, kteří by prototyp otestovali. Testování s pěti uživateli obvykle odhalí 85 % všech hlavních problémů s použitelností. Jak psal Jakob Nielsen ve svém článku o testování softwaru. V tomto případě však testujeme dvě různé role a zkušenosti získané v jedné roli by neměly ovlivnit testování druhé role. Pro test je proto třeba najít alespoň 10 osob, které mají řidičský průkaz skupiny A nebo B a aktivně cestují autem nebo na motorce. Respondenti musí být samozřejmě starší 18 let a musí aktivně používat telefony se systémem Android a aplikace na nich. Za účelem provedení testování a další analýzy výsledků jsem vytvořila testovací šablonu, která je uvedena v tabulce 7.3.

Kód	Věk	Telefon na bázi	Řidičský průkaz	Scénář
Úkol číslo	Kód	Poznámka		

Tabulka 7.3. Šablona pro uživatelské testování (vyplní moderátor)

Otázky k dokončení testu:
Bylo vám používání aplikace zcela jasné? (ohodnoťte od 1 do 10)
Jaký je váš celkový dojem z aplikace? (ohodnoťte od 1 do 10)
Máte nějaké nápady na vylepšení aplikace?

Protože čas na testování je omezený, vyvinula jsem kódy pro rychlý záznam výsledků:
X - Problém s použitelností
D - Duplicitní problém použitelnosti (popsaný dříve)
C - Komentář (obecný komentář účastníka)
P - Pozitivní názor vyjádřený účastníkem
N - Negativní názor vyjádřený účastníkem
B - Chyba
A - Pomoc moderátora
M - Různé (obecný postřeh zapisovatele)
K - Uživatel dokončil akci bez problémů.
Výsledky testů jsou uvedeny v příloze E

7.5.3 Výsledky a analýza testovaní uživatelského rozhraní

Testování dvou interakčních scénářů proběhlo hladce, respondenti nejčastěji pochopili, jak s prototypem pracovat, a většinu úkolů splnili bez problémů. Test však odhalil některé problémy, které je třeba v této fázi práce vyřešit. Podívejme se na problémy podrobněji. Závažnost problému se posuzuje na pětibodové stupnici, kde 5 znamená nejvyšší závažnost.

- Problém 1 - Respondent by si přál, aby byl přechod ze stránky s itinerářem na stránku s cestami zřetelnější a naopak. (3 podobné komentáře)
  Závažnost: 5 bodů
  Řešení: Přidání karty na stránku itineráře s odkazem na hlavní stránku cesty.
- Problém 2 - Respondent by si přál naplánovat čas aktivity ihned po příjezdu do destinace. (2 podobné komentáře)
  Závažnost: 4 body
  Řešení: Požádat uživatele pouze o nastavení doby trvání aktivity, nikoli však počáteční a koncové hodiny. Čas začátku se nastaví automaticky.
- Problém 3: Respondent by chtěl, aby registrace účtu a přihlášení byly na stejné stránce. (2 podobné komentáře)
  Závažnost: 3 body
  Řešení: Tento problém je poměrně snadno řešitelný, ale další výzkum ukázal, že jednostránkový přístup k registraci a přihlášení je velmi vzácný. A může být matoucí pro uživatele, kteří jsou již zvyklí na standardní stránky.
- Problém 4 -Respondent nechápal, jak se o výlet podělit s přáteli. (1 podobný komentář)
  Závažnost: 1 bod
  Řešení: Přidání možnosti sdílet cestu do nabídky na každé interní stránce cesty.

Vzhledem k tomu, že hodnocení prototypu bylo poměrně vysoké, provedu tyto změny již ve fázi Hi-Fi prototypu a nebudu opravovat současný Lo-Fi prototyp.

Kapitola 8
Závěr

Cílem práce bylo navrhnout mobilní aplikaci pro cestovatele se specifickými požadavky, která umožňuje naplánovat cestu. Dále měla uživateli poskytnout (podle aktuální situace a preferencí) úpravu trasy, nabízet zajímavá a důležitá místa v okolí, stejně tak možnosti ubytování a v neposlední řadě zaznamenávat zážitky z cesty. Kritériem pro cestovní plán byla finanční rozvaha. V rámci navrhované aplikace bylo dosaženo všech cílů definovaných v úvodu. Dle zadání byla provedena analýza stávajících řešení včetně analýzy funkčnosti aplikace. Počáteční návrh uživatelského rozhraní vychází z nejúspěšnějších řešení. Další vývoj návrhu vycházel z hodnocení a požadavků potenciálních uživatelů získaných během testování uživatelského rozhraní. Celkem aplikace obsahuje tři zakladní jednotky: Frontend na platformě Android v jazyke Kotlin, backend Kotlin Spring Boot a databázi PostgreSQL. Dále používá data z několika dalších otevřených API. V práci je navržen doménový model a struktura databáze, přičemž se posuzují možnosti komunikace mezi hlavními částmi aplikace. Následně byly podle návrhu vyvinuty serverová a klientská část aplikace. Při vývoji byl hlavní důraz kladen na implementaci nového deklarativního přístupu Jetpack Compose na platformě Android. Výsledná aplikace byla otestována pomocí programátorských a uživatelských testů. Největším problémem bylo navržení uživatelského rozhraní, u kterého bylo před samotnou implementací nutno počítat s UI testováním a upravit návrh podle požadavků uživatele. Práce na projektu mi umožnila využít znalosti, které jsem získala během studia, a zdokonalit se v nejnovějších přístupech vývoje aplikací pro Android. Výsledkem bakalářské práce je funkční prototyp, který je připraven pro další rozvoj. V budoucnu bych ji ráda rozšířila o další funkcionality, například možnosti zveřejnění absolvovaného výletu, soukromé zprávy, push notifikace pro oznámení uživatele o nové událostí, apod. Nezbytne také poskytnout nástroje pro analýzu chování uživatelů, napřiklad Google Analytics nebo Firebase Analytics. Také je plánováno vytvořit porty této aplikace i pro systém iOS a také webovou aplikaci. Toto podstatně rozšíří potenciální uživatelskou základnu.

Literatura

[1] *WanderLog App*. 2019.
https://play.google.com/store/apps/details?id=com.wanderlog.android.

[2] *TripIt App*. 2022.
https://play.google.com/store/apps/details?id=com.tripit.

[3] *Road trippers App*. 2022.
https://play.google.com/store/apps/details?id=com.roadtrippers.

[4] *Sygic Travel App*. 2022.
https://play.google.com/store/apps/details?id=com.tripomatic.

[5] Google Inc.:. *Material Design*. 2022.
https://material.io/develop/android.

[6] Barbora Koďousková. *Webová, nativní a nybridní aplikace: srovnáváme pro a proti*. 2021.
https://www.rascasone.com/cs/blog/webova-nativni-hybridni-aplikace-klady-zapory.

[7] In. Jaroslav Pavlíček. *Nativní vs. hybridní aplikace. Která se vám víc vyplatí?* 2021.
https://inited.cz/2021/12/08/nativni-vs-hybridni-aplikace-ktera-se-vam-vic-vyplati/.

[8] In. Jaroslav Pavlíček. *Nativní, hybridní nebo webová aplikace? Která je nejlepší?* 2021.
https://www.mediar.cz/nativni-hybridni-nebo-webova-aplikace-ktera-je-nejlepsi/.

[9] *Mobile Operating System Market Share Worldwide StatCounter*. 2022.
https://gs.statcounter.com/os-market-share/mobile/worldwide/2022.

[10] *Mobile Operating System Market Share Worldwide StatCounter*. 2022.
https://gs.statcounter.com/os-market-share/mobile/czech-republic/2022.

[11] John Callaham. *The history of Android: The evolution of the biggest mobile OS in the world Androidauthority*. 2022.
https://www.androidauthority.com/history-android-os-name-789433/.

[12] *Mobile Operating System Market Share Worldwide StatCounter*. 2021.
https://gs.statcounter.com/os-market-share/mobile/worldwide/2021.

[13] Iyany Adelekan. *Kotlin Programming By Example*. 1 vyd.. Packt Publishing Ltd., 2018. ISBN 978-1-78847-454-2.

[14] Patrick Bollhoff. *Kotlin vs Java: strengths, weaknesses and when to use which*. 2022.
https://kruschecompany.com/kotlin-vs-java/.

[15] Hiral Atha. *Java Vs Kotlin – Which Should You Choose For Android Development*. 2018.
https://www.moveoapps.com/blog/java-vs-kotlin/.

[16] Google Inc.:. *Guide to app architecture*. 2021. https://developer.android.com/jetpack/guide.

[17] *MVVM (Model View ViewModel) Architecture Pattern in Android*. 2022. https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/.

[18] *Spring Boot – Architecture*. 2022. https://www.geeksforgeeks.org/spring-boot-architecture/.

[19] Microsoft Inc.:. *PostgreSQL*. 2022. https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-postgresql/.

[20] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. 2022. https://www.nngroup.com/articles/ten-usability-heuristics/.

[21] Davide Pacilio. *Social login — Do users prefer Google, Twitter, or Facebook?* 2019. https://www.indiehackers.com/post/social-login-do-users-prefer-google-twitter-or-facebook-here-is-what-we-ve-learned-on-cruip-com-9e98cc9bbc.

[22] Jenifer Tidwell. *Designing Interfaces*. 2 vyd.. O'Really Media Inc., 2006. ISBN 978-1-449-37970-4.

[23] Microsoft Inc.:. *Calculate a Distance Matrix*. 2021. https://learn.microsoft.com/en-us/bingmaps/rest-services/routes/calculate-a-distance-matrix.

[24] Google Inc.:. *Lists and grids*. 2022. https://developer.android.com/jetpack/compose/lists.

[25] Google Inc.:. *Save key-value data*. 2021. https://developer.android.com/training/data-storage/shared-preferences.

[26] Google Inc.:. *Places SDK for Android*. 2021. https://developers.google.com/maps/documentation/places/android-sdk.

[27] Fernando Sproviero. *Android Unit Testing with Mockito*. 2018. https://www.kodeco.com/195-android-unit-testing-with-mockito.

[28] *Fundamentals of testing Android apps*. 2022. https://developer.android.com/training/testing/fundamentals?hl=fi .

[29] Google Inc.:. *Testing your Compose layout*. 2021. https://developer.android.com/jetpack/compose/testing.

[30] *Docker Engine installation overview*. https://docs.docker.com/engine/install/.

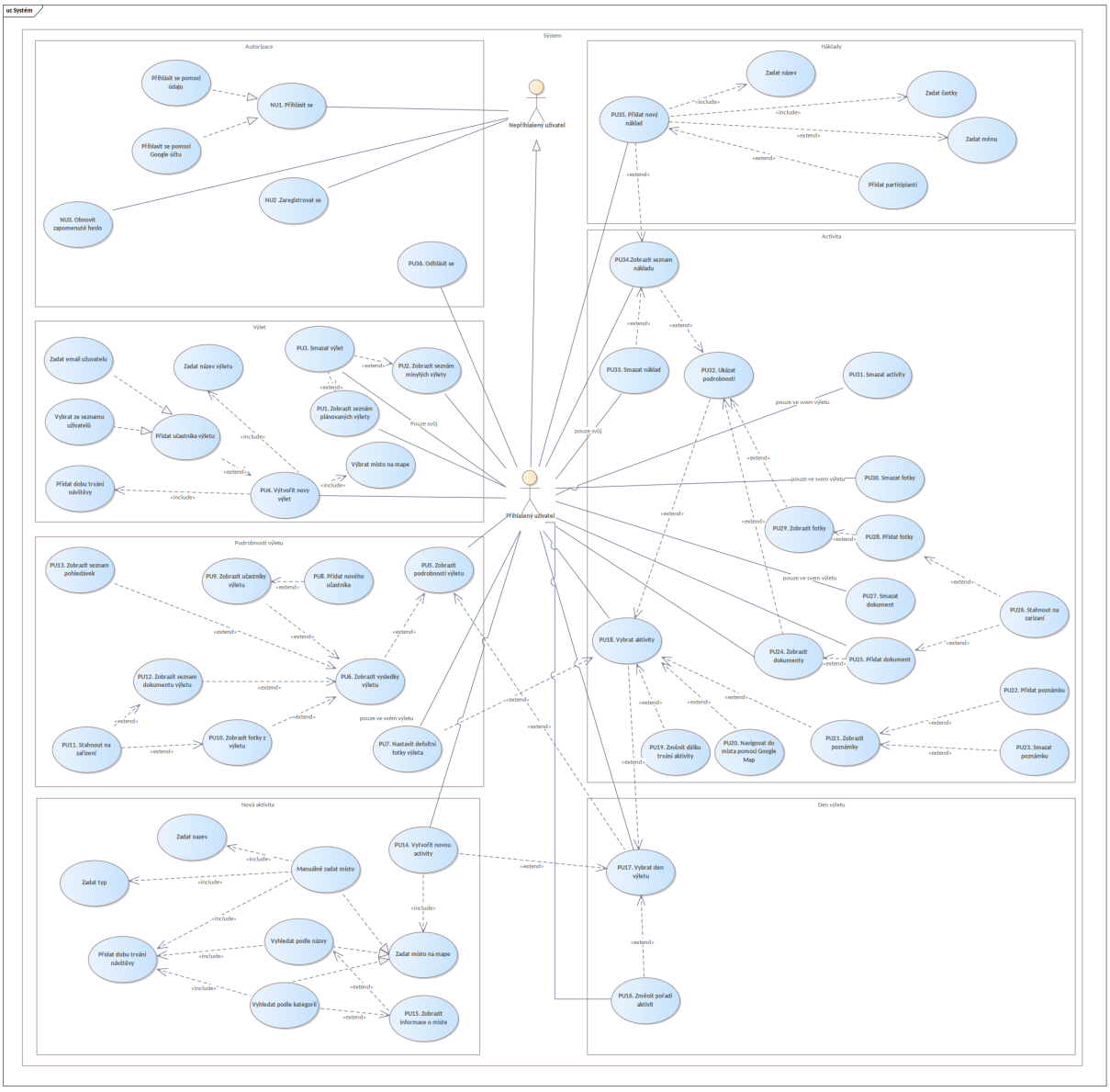
[31] *Overview*. https://docs.docker.com/compose/install/.

[32] *Docker Engine post-installation steps*. https://docs.docker.com/engine/install/linux-postinstall/.

Příloha A
Seznam použitých zkratek

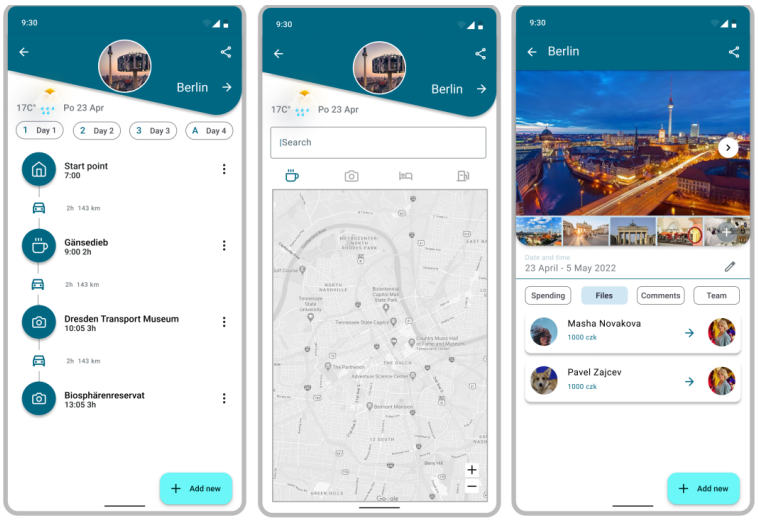
- API — Application Programming Interface
- CSS — Cascading Style Sheets
- DAO — Data Access Object
- FP — Funkční Požadavek
- GPS — Global Positioning System
- Hi-fi — High fidelity prototype
- HTML — HyperText Markup Language
- HTTP — Hypertext Transfer Protocol
- iOS — iPhone Operational System
- JC — Jetpack Compose
- JDBC — Java Database Connectivity
- JPA — Java Persistence API
- JSON — JavaScript Object Notation
- JVM — Java Virtual Machine
- Lo-fi — Low fidelity prototype
- MVC — Model View Controller
- MVVM — Model View Viewmodel
- NFP — Nefunkční Požadavek
- NU — Nepřihlášený Uživatel
- OS — Operational System
- POI — Point Of Interest
- POJO — Plain Old Java Object
- PU — Přihlášený uživatel
- REST — Representational State Transfer
- SQL — Structured Query Language
- UC — Use Case
- UI — User Interface
- UML — Unified Modelling Language
- URI — Uniform Resource Identifier
- XML — Extensible Markup Language

Příloha B
Diagram případů užití

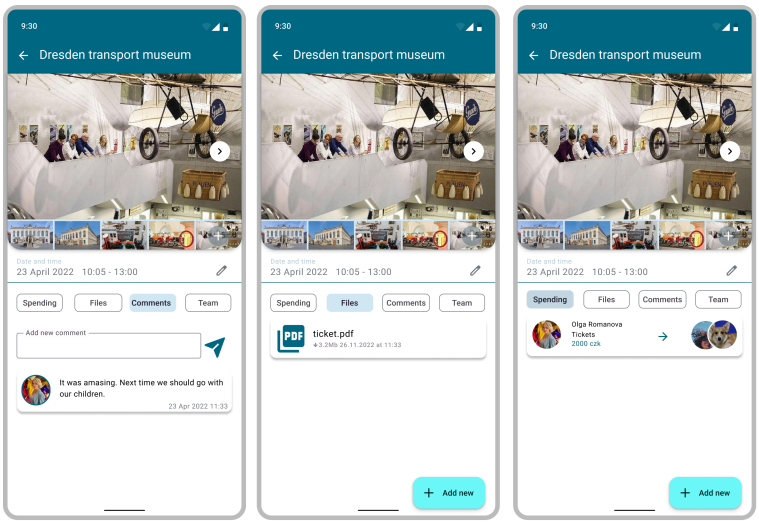


Obrázek B.1. Diagram případů užití

Příloha C
Hi-Fi prototyp

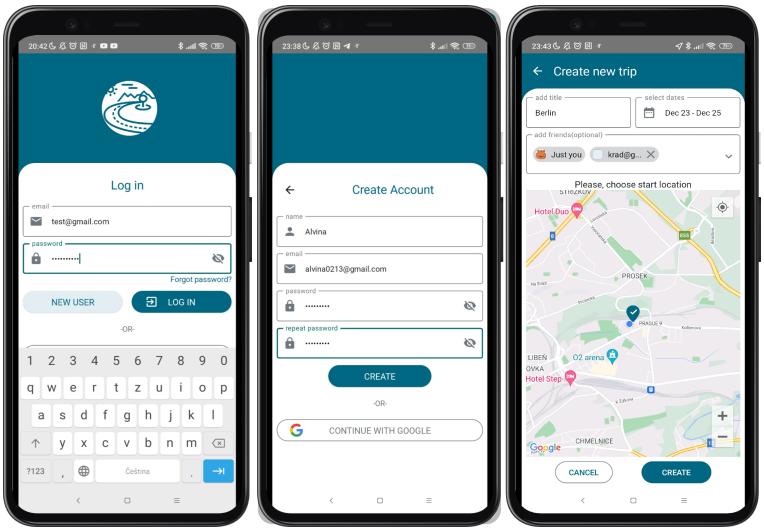


Obrázek C.2. Plánování cesty a stanovení nové aktivity(Hi-Fi prototyp)

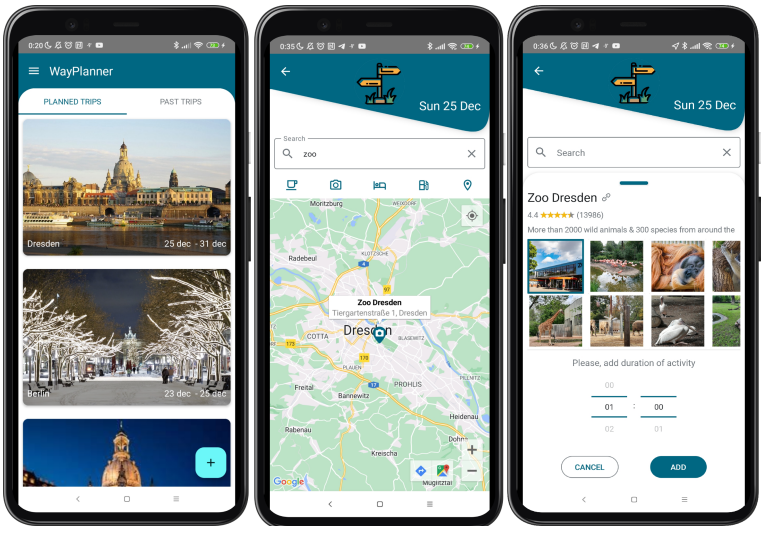


Obrázek C.3. Podrobnosti aktivit (Hi-Fi prototyp)

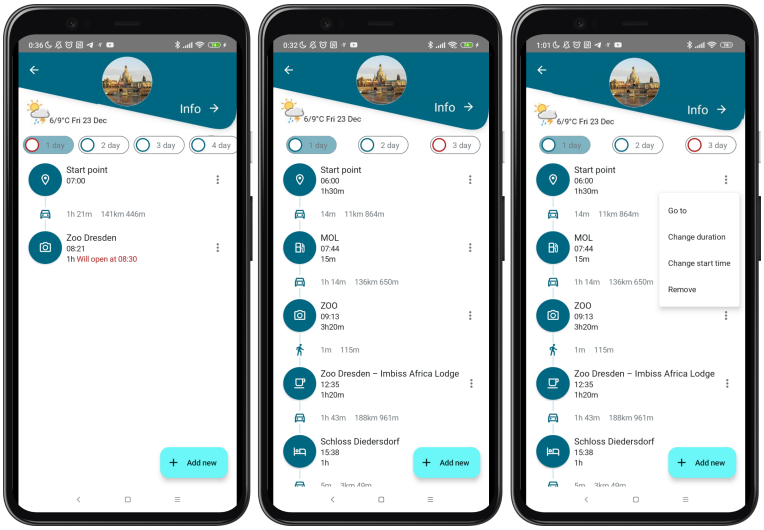
Příloha D
Ukázka výsledné aplikace



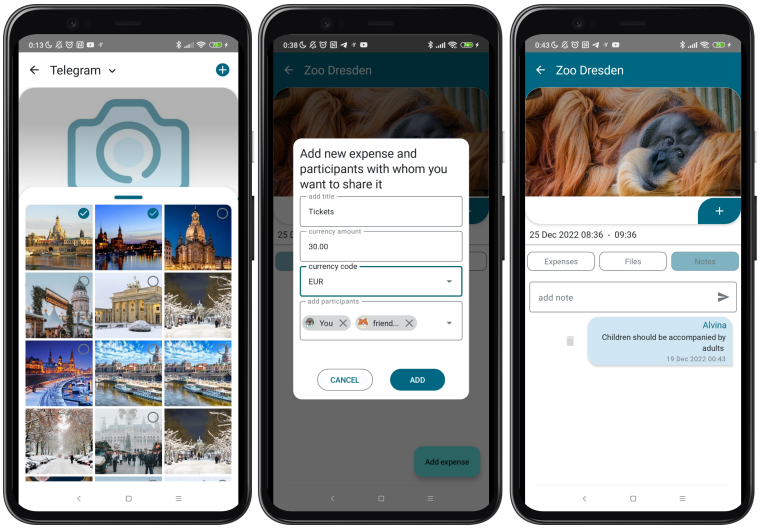
Obrázek D.4.



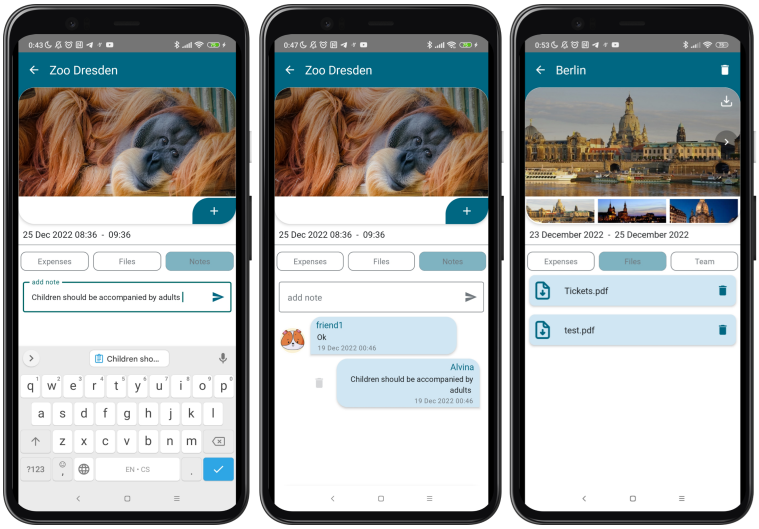
Obrázek D.5.



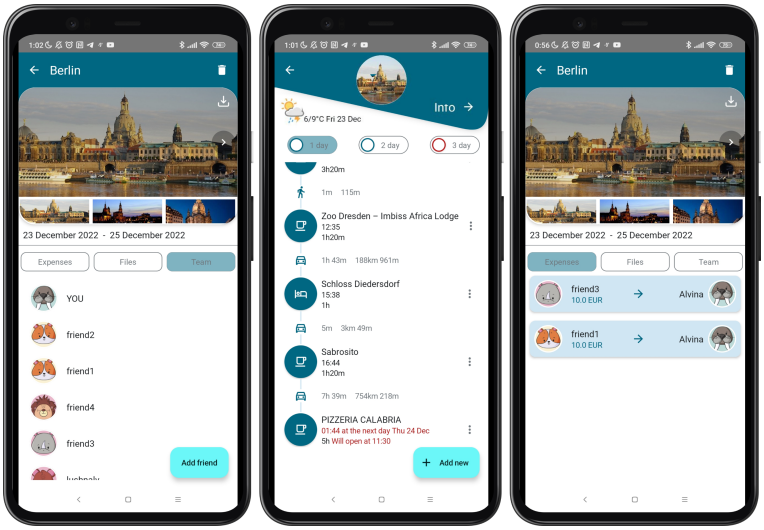
Obrázek D.6.



Obrázek D.7.



Obrázek D.8.


Berlin
23 December 2022 - 25 December 2022
Expenses
Files
Team
YOU
friend2
friend1
friend4
friend3
Add friend
Into
6/9°C Fri 23 Dec
1 day
2 day
3 day
3h20m
1m 115m
Zoo Dresden – Imbiss Africa Lodge
12:35
1h20m
1h 43m 188km 961m
Schloss Diedersdorf
15:38
1h
5m 3km 49m
Sabrosito
16:44
1h20m
7h 39m 754km 218m
PIZZERIA CALABRIA
01:44 at the next day Thu 24 Dec
5h Will open at 11:30
Add new
Berlin
23 December 2022 - 25 December 2022
Expenses
Files
Team
friend3
10.0 EUR
Alvina
friend1
10.0 EUR
Alvina

Obrázek D.9.

Příloha E
Výsledky testovaní uživatelského rozhraní

E.1 Otázky a odpovědi

Otázky a odpovědi k dokončení testu:
1. Bylo vám používání aplikace zcela jasné? [10 - respondenty 2.1, 2.4; 9 - respondenty 1.1, 1.3, 2.2, 2.3, 2.5; 8 - respondenty 1.2, 1.4; 7 - respondent 1.5]
2. Jaký je váš celkový dojem z aplikace? [10 - respondenty, 2.4, 2.3; 9 - respondenty 1.1, 1.4, 2.1, 1.3, 2.2, 2.5; 8 - respondenty 1.5; 7 - respondent 1.2]
3. Máte nějaké nápady na vylepšení aplikace?

- Respondent 1.1 - Respondent by chtěl, aby registrace účtu a pozvánka byly na stejné stránce.
- Respondent 1.2 - Potřebuju zřetelnější přechod mezi stránkami itineráře a stránkou s cestami.
- Respondent 1.3 - Chci, aby byl čas zahájení aktivity nastaven automaticky.
- Respondent 1.5 - Chci naplánovat čas aktivity ihned po příjezdu do destinace.
- Respondent 2.5 - Chci aby byl přechod ze stránky s itinerářem na stránku s cestami zřetelnější a naopak.

E.2 Testovací data

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
1.1	31	Android	4-5	B	1
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registrace pomocí emailu			
1	X	Respondent si chtěl založit účet na stránce přihlášení			
2-8	K				
9	P	Užitečná funkce			
10	X	Respondent nenašel tlačítko odhlášení, ale nemám s tím problém, odhlášení vůbec neřeší			

Tabulka E.1. Uživatelské testování. Respondent 1.1

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
1.2	39	Android	3-4	A,B	1
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí Google účtu			
1	X	Respondent nepochopil, proč potřebuje zadat jméno			
2	N	Respondent nepochopil, proč je na stránce všech cest, chtěl přejít k úpravě cestovního plánu			
3	N	Respondent si přál, aby byl čas zahájení aktivity nastaven automaticky			
4-10	K				

Tabulka E.2. Uživatelské testování. Respondent 1.2

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
1.3	23	Android	12	B	1
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí emailu			
2	K				
3	N	Respondent by si přál, aby byl přechod ze stránky s itinerářem na stránku s cestami zřetelnější a naopak			
4-9	K				
10	C	Respondent je překvapen, že ve scénáři je klauzule o odhlášení			

Tabulka E.3. Uživatelské testování. Respondent 1.3

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
1.4	22	Android	8	B	1
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí Google účtu			
2-3	K				
4	A	Respondent nechápal, jak se o výlet podělit s přáteli			
5	P	Hezká funkce			
6-10	K				

Tabulka E.4. Uživatelské testování. Respondent 1.4

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
1.5	32	Android	3-2	B	1
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí Google účtu			
2	K				
3	A	Respondent nerozuměl tomu, jak naplánovat čas aktivity ihned po příjezdu do destinace			
4	K				
5	P	Hezká funkce			
6-10	K				

Tabulka E.5. Uživatelské testování. Respondent 1.5

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
2.1	21	Android	8	B	2
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí Google účtu			
2	X	Respondent byl trochu v rozpacích, když na titulní straně viděl již připravenou cestu			
3-5	K				
6	P	Užitečná funkce			
7	K				

Tabulka E.6. Uživatelské testování. Respondent 2.1

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
2.2	21	Android	8	B	2
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí Google účtu			
2	K				
3	X	Respondent nerozuměl tomu, jak přejít z itineráře na hlavní cestovní stránku			
4-7	K				

Tabulka E.7. Uživatelské testování. Respondent 2.2

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
2.3	27	Android	5-6	B	2
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí Google účtu			
2-4	K				
5	X	Respondent nemohl najít, jak přejít na stránku s fotografiemi			
6-7	K				

Tabulka E.8. Uživatelské testování. Respondent 2.3

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
2.4	39	Android	3-2	A	2
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí emailu.			
2-7	K				

Tabulka E.9. Uživatelské testování. Respondent 2.4

Kód	Věk	Telefon na bázi	Čas který tráví v mobilu	Řidičský průkaz	Scénář
2.5	39	Android	2-3	A	2
Úkol číslo	Kód	Poznámka			
1	M	Respondent vybral registraci pomocí emailu			
2	X	Respondent by si přál, aby byl přechod ze stránky s itinerářem na stránku s cestami zřetelnější a naopak			
3-7	K				

Tabulka E.10. Uživatelské testování. Respondent 2.5

Příloha F
Instalační návod

Aplikaci lze otestovat několika způsoby:

1. Jde o vyhodnocení výkonu aplikace bez instalace a kompilace frontendu nebo backendu. Pokud k dispozici je chytrý telefon se systémem Android ve verzi vyšší než 8.0, aplikaci nejsnáze stáhnout z Google Play, protože je distribuovaná tam v interním testovacím režimu. Backend v tomto případě nadále běží na platformě Heroku (trvá asi 15 až 60 sekund min pro probuzení aplikace, pokud dlouho ji nikdo nepoužíval). Pro provedení, je nutné požádat o přidání e-mailové adresy, na kterou je telefon registrován, do seznamu testerů. To lze provést zasláním požadavku na adresu lushnalv@fel.cvut.cz.

2. Jde o vyhodnocení výkonu frontendu spuštěním aplikace na fyzickém zařízení nebo emulátoru pomoci Android Studio, backend v tomto případě nadále běží na platformě Heroku (trvá asi 15 až 60 sekund min pro probuzení aplikace, pokud dlouho ji nikdo nepoužíval). Pro spuštění je však nutné připojit soubor google-services.json, který obsahuje přístupové klíče k různým službám Google. Soubor pro testování aplikace lze dostat odesláním požadavku na adresu lushnalv@fel.cvut.cz.

3. Jedná se o provozování nejen frontendové aplikace, ale také backendu na vlastním hardwaru(Linux). V takovém případě je třeba spustit frontend stejně jako ve výše uvedeném příkladě a změnit základní adresu URL v souboru AppWebApi.kt na IP adresu hardwaru, na kterém je spuštěn backend. Pro spuštění backend je však nutné zadat klíče „bing.api.key“, „spring.mail.username“, „spring.mail.password“ do souboru application-local.properties. Údaje pro testování aplikace lze dostat odesláním požadavku na adresu lushnalv@fel.cvut.cz. Implementace využívá balíčkovací systém Docker-compose Inicializační postup pro spuštění backend:

- Nainstalovat docker podle pokynů [30]
- Nainstalovat docker-compose podle pokynů [31]
- Povolit běžným uživatelům používat docker podle pokynů [32]
- Spustit příkazem “docker-compose up” v složce /Docker-compose