FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Rozšíření mobilního klienta pro Android pro system Journal
Student:	Michael Kozel
Vedoucí:	Ing. Zdeněk Rybola, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Journal je webová aplikace pro vedení osobního deníku a zaznamevávání důležitých událostí. Aplikace poskytuje API, které je možné konzumovat mobilní aplikací.

Cílem bakalářské práce je rozšíření funkčnosti mobilního klienta pro Android pro tuto aplikaci se zaměřením především na podporu funkcí webového rozhraní chybějících v mobilním klientovi. Jde především o následující funkce:
- podpora označování osob a vyhledávání podle osob
- podpora hierarchických událostí
- integrace kalendářů

Pokyny k vypracování:
- po dohodě s vedoucím práce stanovte požadavky na rozšíření a podrobně je analyzujte
- navrhněte způsob řešení stanovených požadavků v souladu se stávající architekturou
- implementujte stanovená vylepšení na základě provedeného a schváleného návrhu
- provedené úpravy patričně otestujte a zdokumentujte
- v případě potřeby navrhněte a realizujte úpravy API na straně webové aplikace

---

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 30. prosince 2021 v Praze.

Bakalářská práce

ROZŠÍŘENÍ MOBILNÍHO KLIENTA PRO ANDROID PRO SYSTEM JOURNAL

Michael Kozel

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Zdeněk Rybola, Ph.D.
12. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat rodině za podporu během studia, a také vedoucímu práce panu Ing. Zdeňkovi Rybolovi, Ph.D. za cenné rady a vstřícnost při konzultacích během psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 12. května 2022 .....

Abstrakt

Tato bakalářská práce se zabývá rozšířením již existující mobilní aplikace pro deníkový portál Journal, která vznikla v rámci předcházející bakalářské práce – Mobilní klient pro aplikaci Journal, paní Martiny Fukalové. Rozšíření se týká funkcí, které jsou zatím dostupné pouze ve webové verzi. Jde především o funkce, jako jsou podpora hierarchických událostí, zobrazení adresy umístění události při upravování události, přehled osob, přidání osob do události, zobrazení událostí dané osoby, vylepšení přidávání lokace do události. Aplikace je naprogramována pomocí programovacího jazyka Kotlin a uživatelské rozhraní je definováno pomocí jazyka XML. Dále aplikace využívá sady knihoven Jetpack, jako jsou Room pro offline caching, Navigation component pro navigaci, LiveData pro načítání dat a Gson pro deserializaci odpovědí z API ve formátu JSON. Aplikace komunikuje s backendem systému Journal pomocí REST API.

Klíčová slova Journal, deníková aplikace, rozšíření mobilní aplikace, Android, Kotlin, XML, Gradle, REST

Abstract

This bachelor thesis deals with the extension of the existing mobile application for the diary portal Journal, which was created in the previous bachelor's thesis - Mobile client for the application Journal, Mrs. Martina Fukalová. The extension applies to features that are currently only available on the website version. These are functions such as support for hierarchical events, address display, especially the location of the event when editing events, managing of user friends, adding people to events, displaying events person updates, adding location to events. The application is programmed in the in the Kotlin programming language and the user interface is defined using XML. Further applications uses Jetpack library sets such as Room for offline caching, Navigation component for navigation, LiveData for retrieving data, and Gson for deserializing responses from the JSON API. The application communicates with the Journal backend using the REST API.

Keywords Journal, diary application, mobile application, mobile client, Android, extension, Kotlin, XML, Gradle, REST

Seznam zkratek

API	Application Programming Interface
CRUD	Create Read Update Delete
DAO	Database Access Object
DTO	Data Transfer Object
FAB	Floating Action Button
FP	Funkční požadavek
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVVM	Model View ViewModel
OS	Operační systém
POI	Point of interest
REST	Representational State Transfer
SQL	Structured Query Language
UC	Use Case
XML	Extensible Markup Language

Kapitola 1
Úvod

Mnoho lidí si někdy ve svém životě vedlo deník. Vedení deníku může mít pozitivní vliv na vývoj člověka. Lidský mozek mnoho věcí a myšlenek z každodenního života prostě zapomene, a proto může být dobré si deník vést a mít možnost se k událostem nebo myšlenkám z minulosti zpětně vrátit. Nejspíše každý, kdo si vedl někdy deník, si poznámky o svém životě, myšlenkách, náladách apod. vedl písemnou formou. Díky době digitalizace a dostupnosti internetu se nabízí možnost vedení deníku i elektronicky. Má to nespočet výhod. Deník můžeme mít pořád u sebe, vzhledem k tomu, že dnes již skoro každý vlastní smartphone a nosí ho pořád u sebe. Dále také může být výhoda digitálního deníku i nezávislost na zařízení. Vše, co pro vedení deníku potřebujete, je zařízení s přístupem k internetu.

V této práci se zabývám rozšířením nativní mobilní aplikace pro operační systém Android, systému Journal – webové aplikace pro vedení osobního deníku a zaznamevávání důležitých událostí. Tato práce navazuje na bakalářskou práci paní Martiny Fukalové – Mobilní klient pro aplikaci Journal, ve které aplikace vznikla, o několik dalších případů užití, které jsou zatím dostupné pouze přes webové rozhraní.

Součástí teoretické části je analýza současné aplikace, shrnutí již implementovaných požadavků a případů užití, použité architektury. Dále se zde zabývám srovnáním různých architektur často používaných ve vývoji nativních mobilních aplikací pro OS Android. Představuji zde souhrn požadavků zadaných zadavatelem, analyzuji je a navrhuji jejich implementaci.

V praktické části se zabývám implementací jednotlivých požadavků a jejich otestováním.

Výsledek práce může být použit pro edukativní účely všem, kteří hledají informace o nativním vývoji aplikací pro OS Android.

1.1 Cíle bakalářské práce

Hlavním cílem této bakalářské práce je rozšíření již existující nativní mobilní aplikace pro OS Android, systému Journal – webové aplikace pro vedení osobního deníku a zaznamevávání důležitých událostí. Mobilní aplikace je naprogramovaná v jazyce Kotlin, s backendem systému Journal komunikuje pomocí REST API. Rozšíření se bude týkat funkcí momentálně podporovaných pouze ve webové části s důrazem na podporu označování osob, vyhledávání podle osob, podporu hierarchických událostí a integraci kalendářů.

Cílem teoretické části je analýza funkčnosti existující mobilní aplikace, analýza webového klienta s důrazem na výše zmíněné funkce.

Cílem praktické části je návrh řešení jednotlivých požadavků, jejich implementace a řádné otestování.

Hlavním přínosem bakalářské práce je zpřístupnění dalších funkcí systému Journal pro uživatele mobilní aplikace.

Kapitola 2
Analýza

V následující kapitole budu analyzovat aktuální řešení. Stručně představím projekt Journal a vznik Android klienta tohoto projektu.

2.1 Představení projektu

„*Journal aplikace je aplikace umožňující si ukládat prožité události v elektronické podobě. K události je možné přidávat soubory, fotky, gps lokaci a další. Projekt vzniká v rámci předmětů BI-SP1, BI-SP2 na Fakultě informačních technologií ČVUT.*“ [2]

Prvotní nápad vytvořit Android aplikaci k tomuto systému byl v roce 2018, kdy vznikla první aplikace naprogramovaná v rámci předmětu BI-SP2. Pracovali na ní Igor Tsaregorodtsev, a Zhanybek Sadvakassov. Aplikace splňovala některé funkce, ale z důvodu nespokojenosti majitele vznikla v rámci bakalářské práce paní Bc. Marie Fukalové [3] nová, nativní Android aplikace, na kterou budu v této práci navazovat a rozšiřovat její funkce.

2.2 Aktuální funkce aplikace

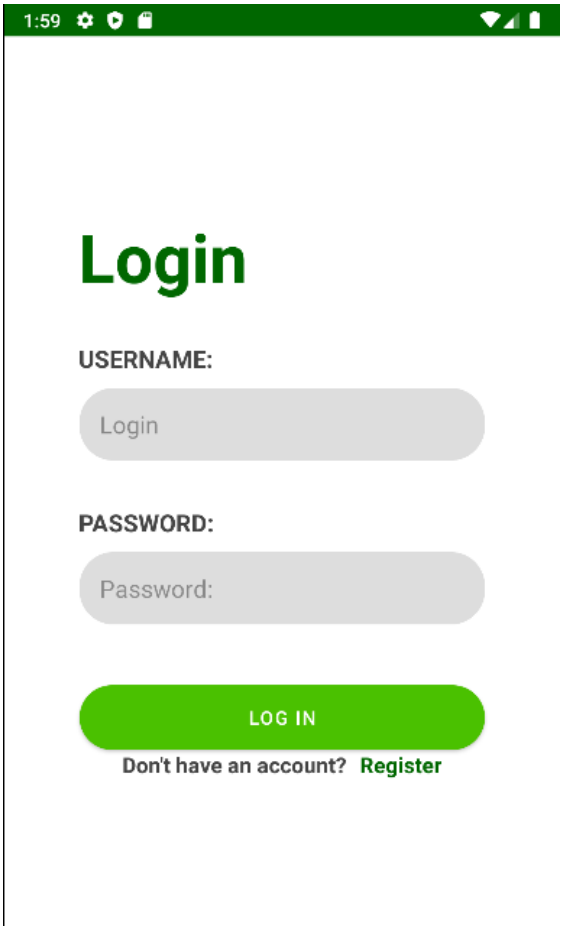
V této kapitole představím aktuální aplikaci. Co aplikace umí, jak toho lze dosáhnout, a také co aplikace neumí. Dále analyzuji funkce webového rozhraní, které aplikace nepodporuje a vytvořím seznam požadavků pro rozšíření, kterým se bude tato práce zabývat.

2.2.1 Obrazovka přihlášení

Jako první se uživateli zobrazí přihlašovací obrazovka (viz obrázek 2.1), kde může uživatel zadat uživatelské jméno a heslo a poté se kliknutím na tlačítko „Log in“ pokusit přihlásit. V případě úspěchu se dostaneme na seznam všech událostí. V případě neúspěchu se na displayi zobrazí `Toast` s textem „Invalid username or password“. V případě, že byl uživatel již na tomto zařízení přihlášen a neodhlásil se, tak se tato obrazovka přeskočí a uživatel je automaticky v přihlášeném stavu na další obrazovce. Pokud uživatel ještě nevlastní účet v systému, může se zaregistrovat na webu, kam ho odkáže tlačítko „Register“.

2.2.2 Diary fragment

Po úspěšném přihlášení se uživateli zobrazí obrazovka se seznamem událostí. Zde může uživatel kliknutím na položku seznamu zobrazit detail dané události, nebo může události vyhledávat



Obrázek 2.1 Přihlašovací obrazovka

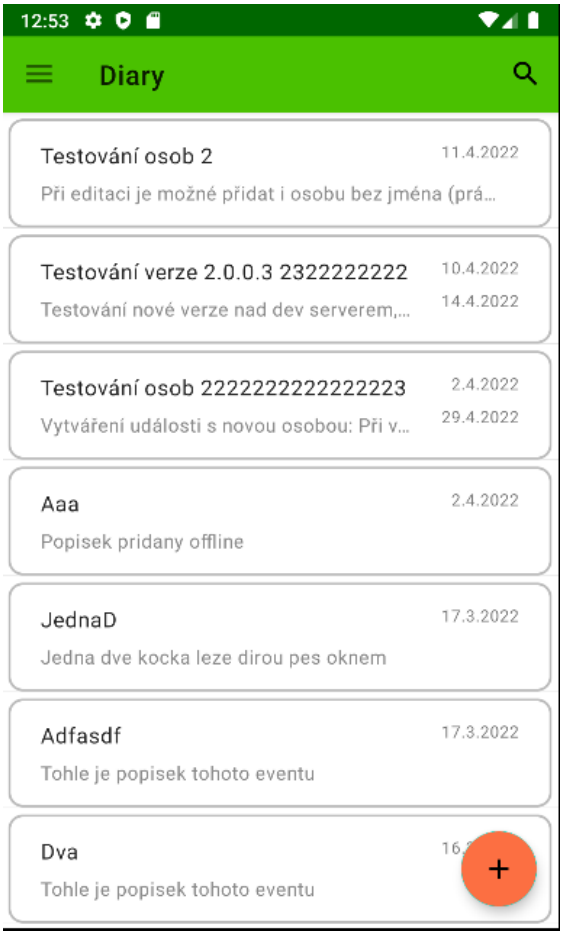
kliknutím na lupu v pravém horním rohu. viz obrázek 2.2 Pokud je uživatel přihlášen, má možnost zobrazit šuplíkové menu pomocí tzv. hamburger tlačítka v levém horním rohu obrazovky. Je to způsob navigace v aplikaci používaný ve starších aplikacích. Kliknutím na toto tlačítko vyvolá vyjetí šuplíkového menu viz obrázek 2.3. Stejného výsledku dosáhne uživatel i pomocí gesta tažení z levého okraje obrazovky směrem k pravému okraji. Zde může uživatel přepnout do ostatních sekcí aplikace Journal. A to jsou kalendář a mapa. Poslední možností je odhlášení uživatele z aplikace.

2.2.3 Kalendář

V kalendáři viz obrázek 2.4 uživatel může vidět v jakých dnech či rozmezích dní se udály jeho události. Při prvotním zobrazení kalendáře se načte aktuální měsíc. Gesty tažení doprava či doleva se může přesouvat o měsíc dopředu nebo na dozadu.

2.2.4 Mapa

Mapa viz obrázek 2.5 je poslední obrazovkou v aplikaci. V prvotním zobrazení obrazovky se uživateli ukáže detail na Českou republiku. Na mapě se zobrazují adresy událostí ve formě markerů. Pokud je na jednom místě více událostí, zobrazí se tzv. Cluster, který lze rozkliknout a zobrazit



Obrázek 2.2 Detail události

tak jednotlivé markery událostí. Po kliknutí na marker se nad markerem zobrazí box s titulkem události společně s jejím datem. Na tento box lze kliknout a přesunout se tak na detail dané události.

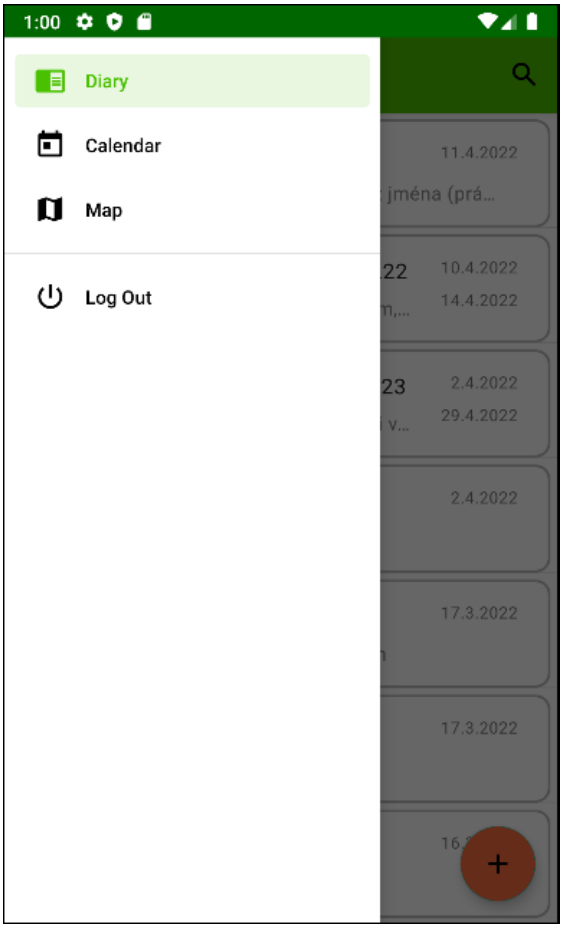
2.2.5 Odhlášení

Poslední možností v šuplíkovém menu je odhlášení. Po kliknutí na tuto možnost by se měl uživatel z aplikace odhlásit a vrátit zpět na obrazovku přihlášení. To se ale děje pouze v některých případech. V některých případech se odhlásit nepodaří. To je chyba, které se věnuje jeden z funkčních požadavků.

2.3 Nalezené problémy v aplikaci

V aplikaci bylo při analýze nalezeno několik chybných chování. Jsou jimi:

- Nefungující odhlášení – při odhlášení se aplikace odhlásí jen v části případů, kdy si uživatel zobrazil pouze obrazovku seznamu všech událostí. Jinak se odhlášením přesune o jednu obrazovku zpět, a pokud se pokusí navštívit další obrazovky, aplikace spadne.



Obrázek 2.3 Postranní šuplíkové menu

- Nemožnost přesouvání v mapě ve vertikálním směru na obrazovce detailu události – pokud se uživatel pokusí přesunout v mapě vertikálně, místo posunu mapy se posune všechen obsah detailu události.

2.4 Doménová a datová vrstva

Pro lepší představu přikládám doménový model z bakalářské práce paní Bc. Martiny Fukalové. [3] Doménový model je konceptuální model domény, který zahrnuje chování a data používaná v aplikaci.[4] Tento model v bakalářské práce není přesně doménový model podle specifikace UML. Posloužit ale může pro základní orientaci v systému. Doménovou vrstvou Android aplikace Journal je Event reprezentující událost. Dalšími entitami jsou atributy události, a to její umístění, tagy a přílohy. Datová vrstva momentálně sdílí její model, knihovna Room zastřešující objektově relační mapování využívá třídu *Event*, která je sdílena a používána taktéž jako DTO pro příjímání dat ze serveru, které se následně uloží do lokální databáze.

2.5 Stávající architektura aplikace

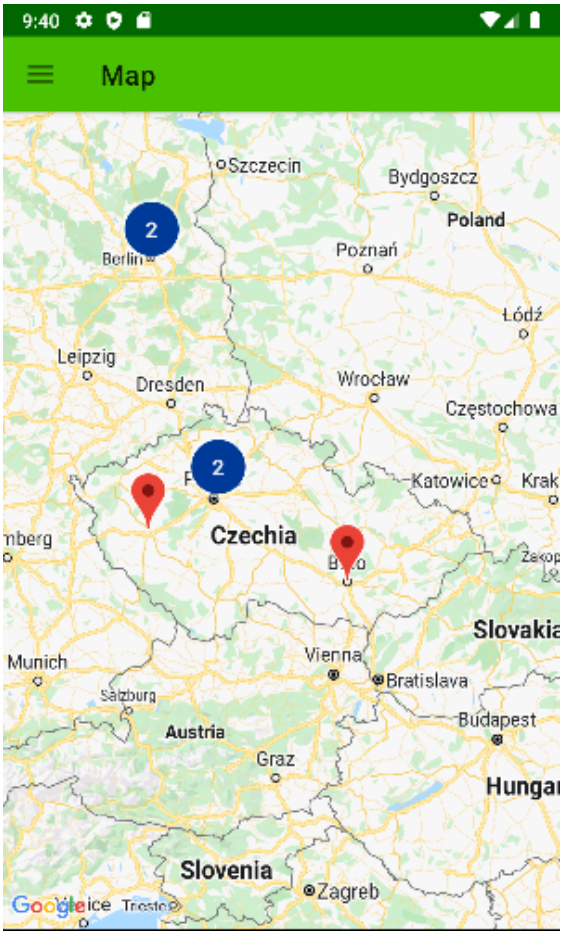
Současná aplikace je naprogramována s využitím architektury MVVM – Model View ViewModel. V minulosti se ve vývoji nativních aplikací pro OS Android používaly různé architektury.

Obrázek 2.4 Kalendář

Mezi nejrozšířenější patří Model View Controller, Model View Presenter nebo právě Model View ViewModel. MVVM je architektonický vzor, jehož hlavním účelem je dosáhnout oddělení zodpovědnosti skrz jasné oddělení mezi rolemi jeho tří vrstev.[5]

- View, která zajišťují zobrazování uživatelského rozhraní. Např zobrazování progress barů a animací. Dále se stará o příjímání vstupu od uživatele.
- ViewModel, které se starají o logiku zpracování dat, zpřístupňují informace Views a zdroji jejich dat je Repository, který abstrahuje datové zdroje, obsahuje logiku získávání dat, se kterými aplikace pracuje.
- Model, který dostává informace z datového zdroje a vystavuje je ViewModelům.

Hlavní rozdíl mezi MVVM a dalšími architekturami je, že MVVM striktně dbá na to, aby ViewModel neměl závislost na Views. To umožňuje použití více ViewModelů v jednotlivých Views. Součástí vrstvy s modely jsou také datové služby, které obsahují konkrétní kód, který obstarává příjímání dat, kterým může být vzdálený server nebo lokální data uložená v databázi či na filesystému v zařízení. Dobrým přístupem je princip jednoho zdroje pravdy. Je to princip, kde veškerá data, se kterými aplikace pracuje jsou načítána z jednoho zdroje. Pokud aplikace využívá více datových zdrojů, měly by se nejdříve uložit na jedno místo a odtud poté načítat do aplikace. V našem případě se data načítají z lokální databáze. Pokud se data aktualizují ze vzdáleného serveru, nejdříve se uloží do lokální databáze a uložená data se poté zobrazí v aplikaci. [6] [5]



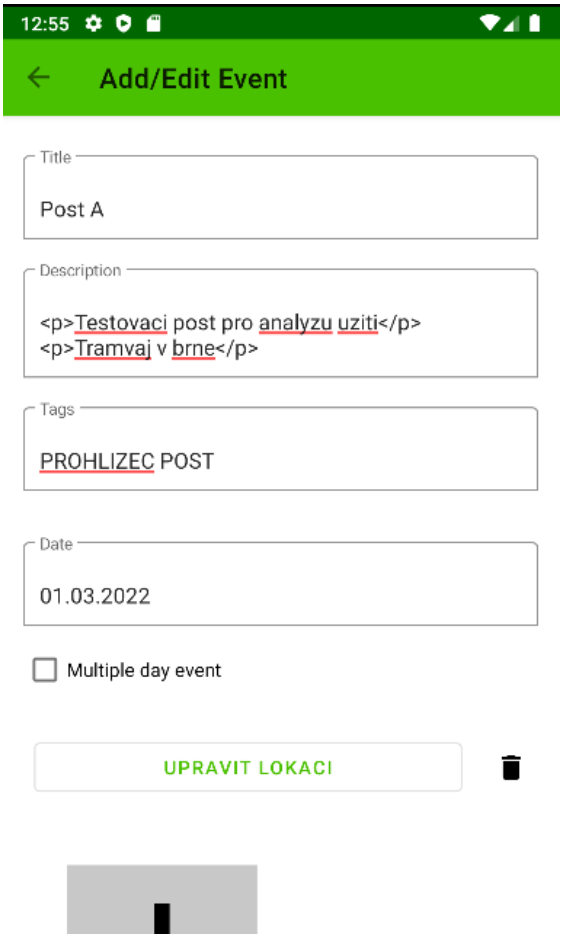
Obrázek 2.5 Mapa

2.6 Definování požadavků

Deníkový systém jsme se rozhodli rozšířit o několik případů užití, které je do této doby možné obsloužit pouze pomocí webového rozhraní. Ty byly zadány zadavatelem. Některé další požadavky byly přidány na základě analýzy stávající aplikace.

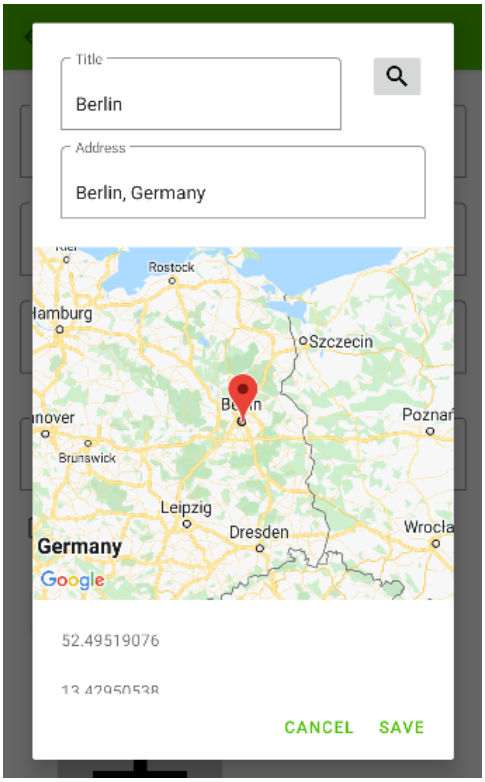
2.6.1 Funkční požadavky

1. **F1 – Chybný návrat po editaci události**
   - Tento požadavek si klade za cíl opravit chybu v navigaci aplikace. Aplikace se chová chybně v případě, že uživatel upravuje existující událost. Po potvrzení editace se správně zobrazí detail události. Pokud ale na detailu klikne na tlačítko šipky zpět v horním levém rohu aplikace, přesune se opět na editaci události. Správné chování by mělo uživatele po kliku na šipku zpět vždy přesounout na obrazovku se seznamem událostí.
2. **F2 – Formátování textu**
   - Texty v mobilní aplikaci importované z webového rozhraní mají rozbitou diakritiku a formátování. Je potřeba, aby se texty zobrazily správně.



Obrázek 2.6 Editace události

3. **F3 – Podpora hierarchických událostí**
   - Ve webové aplikaci je možné k existující události vytvořit podudálost. Dále je možné u událostí zobrazovat, jaké podudálosti či nadřízenou událost konkrétní událost má. Android aplikace ani jednu z těchto funkcí neumí.
4. **F4 – Přehled osob**
   - Webová aplikace umožňuje evidovat u profilu uživatele osoby, které souvisí s událostmi. Pojmem osoba se v tomto kontextu myslí přátelé či jiní lidé, se kterými souvisí konkrétní události tohoto uživatele. Ve webové aplikaci je možné zobrazit přehled osob, který zobrazuje všechny osoby daného uživatele. Zároveň webová aplikace umožňuje založit nové osoby, a také upravovat údaje existujících osob.
5. **F5 – Označování osob v událostech**
   - Ve webové aplikaci je možné ke každé události přidat osoby, se kterými daná událost souvisí. V Android klientovi není možné v události osoby označit ani je v detailu události zobrazit.
6. **F6 – Přechod z kalendáře na událost**



Obrázek 2.7 Přidání / editace polohy události

- Funkce kalendáře ve webové aplikaci umožňuje kliknout na událost v kalendáři, zobrazit základní údaje o této události. Dále bude mít uživatel možnost se přesunout z kalendáře na danou událost.

7. **F7 – Vyhledávání umístění s výběrem**

   - V aktuální verzi Android aplikace lze přiřadit adresu k dané události a uložit tak, kde se událost stala. Při zadání adresy se ale okamžitě potvrdí zadaná poloha i přes to, že to nemusí být ta správná. Dále není možné marker, který se vytvoří na mapě, přesunout přesně tam, kam by si uživatel přál.

8. **F8 – Podpora vztahů mezi událostmi**

   - Cílem tohoto požadavku je rozšířit práci s událostmi o další typy vztahů mezi událostmi. Předcházející, navazující, související atd.

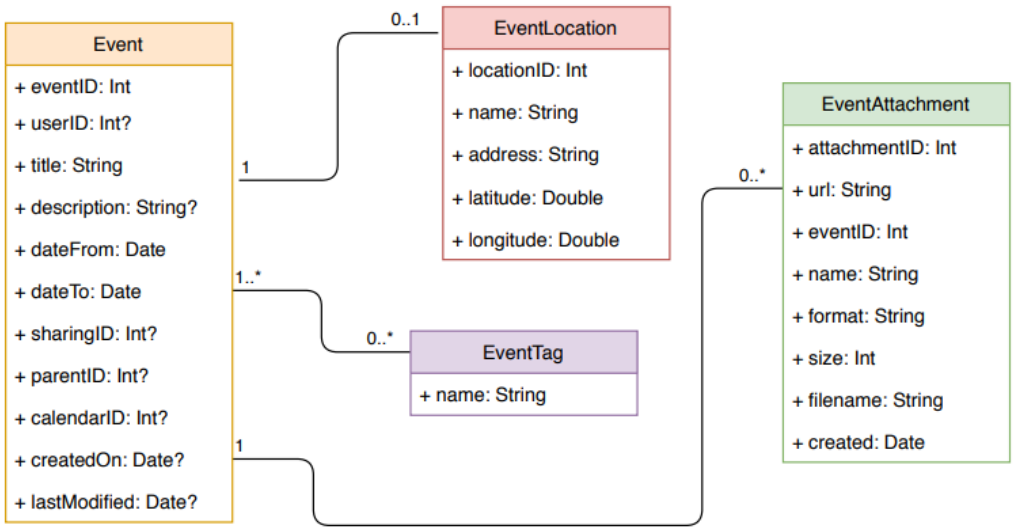
9. **F9 – Podpora více lokací v jedné události**

   - Cílem tohoto požadavku je přidat možnost evidovat více lokací pro danou událost.

10. **F10 – Přechod na obrazovku přihlášení, pokud se uživatel odhlásí**

    - V analýze byl odhalen problém, kdy se uživatel ne vždy přesune na obrazovku přihlášení, pokud klikne na tlačítko odhlášení v bočním menu.

11. **F11 – Mapa na detailu události a výběru lokace**



Obrázek 2.8 Doménový model [3]

- Tento požadavek si klade za cíl vylepšit mapový widget na obrazovce detailu a obrazovce zadání lokace při editaci události. Cílem je zajistit, aby se uživatel mohl posouvat po celé mapě.

2.6.2 Nefunkční požadavky

1. REST API – Aplikace bude komunikovat s backendem systému Journal pomocí REST API.
2. Cache – Aplikace bude schopna data přijatá ze serveru lokálně uložit a zobrazit je v případě, že zařízení nebude připojeno k internetu.

2.7 Analýza nových požadavků

V této sekci analyzuji nové požadavky. Některé požadavky jsou změnové a týkají se prvků, které již v aplikaci jsou, u těch zanalyzuji jejich chování. Pak jsou zde požadavky kompletně nových funkcí, pro které analyzuji tyto funkce ve webovém rozhraní.

2.7.1 Funkční požadavky

Funkční požadavky jsou definovány jako funkce systému nebo jeho částí, kde pojmem funkce je myšlena specifikace chování mezi vstupy a výstupy.[4] V této podkapitole analyzuji funkční požadavky.

2.7.1.1 F1 – Chybný návrat po editaci události

Toto je změnový požadavek, chyba aplikace kterou si zadavatel vyžádal opravit. Na problém narazil při editaci události. Vstup od něj zněl takto: „Po editaci události se zobrazí detail události. Šipka v záhlaví události však místo na přehled událostí vede zpět do editace.“ Další analýza se

skládala z vyzkoušení tohoto scénáře. Pokud uživatel klikne na tlačítko se šipkou zpět během editace, aplikace se chová nepředvídatelně. Někdy se zobrazí fragment kalendáře a někdy se aplikace celá restartuje. To se stává kvuli nesprávnému nastavení obrazovek v rámci navigace obrazovek použitím knihovny Navigation component. [7]

2.7.1.2 F2 – Formátování textu

Ve webové části je možné vkládaný text zformátovat. V aktuální verzi aplikace se formátované texty zobrazí přesně tak, jak jsou obdrženy v odpovědích z Rest API. Je potřeba, aby se zformátované texty ukázaly v aplikaci korektně.

2.7.1.3 F3 – Podpora hierarchických událostí

K tomuto požadavku je nutné analyzovat, jak funguje datová vrstva aplikace, abychom zjistili, co bude potřeba rozšířit. Jak již bylo zmíněno výše, je potřeba umožnit podudálosti zakládat a zobrazovat. Ve webovém rozhraní je možné u konkrétní události stisknout tlačítko a tím se přesunout do formuláře tvorby podudálosti. Ten je totožný, jako tvorba nové události. Dále bude nutné tento vztah uložit i lokálně do databáze. Aby bylo možné zobrazit informace o podudálostech, v případe, že nebude zařízení připojeno k internetu.

2.7.1.4 F4 – Přehled osob

Přehled osob je ve webové aplikaci řešen pomocí dvou částí, kde první se skládá ze seznamu všech osob uživatele. Druhá část se skládá z formuláře, kde uživatel může přidat osobu novou zadáním jména a volitelně i e-mailu.

2.7.1.5 F5 – Označování osob v událostech

Tento požadavek je ve webové aplikaci vyřešen textovým polem, do kterého uživatel píše jméno osoby. Systém vytváří pod textovým polem vyskakovací seznam, kde uživateli nabízí možnosti již existujících uživatelů, jejichž jména se podobají tomu, které uživatel píše do textového pole. Pokud na jedno z nich uživatel klikne, přidá se do události vytvořením šedivého boxu s tímto jménem, které je přidáno do vnitřku textového pole. Dále je na konci tohoto boxu tlačítko s ikonou křížku, pomocí kterého může uživatel danou osobu z události vymazat.

2.7.1.6 F6 – Přechod z kalendáře na událost

Ve webové aplikaci je tento požadavek řešen pomocí vyskakovacího okna, kde se zobrazí základní údaje o události a tlačítko s textem „Detail“, pomocí kterého je možné se přesunout na detail dané události.

2.7.1.7 F7 – Vyhledávání umístění s výběrem

V současné verzi aplikace je vyhledání umístění s výběrem řešeno pomocí Dialogu [8] s vlastním layoutem. (Viz obrázek 2.7) Hlavním problémem tohoto dialogu jsou jeho rozměry. Dialog není roztažen na celou obrazovku. Dalším problémem tohoto řešení je, že mapa není posouvatelná ve vertikálním směru. Ve webové aplikaci se uživateli zobrazí seznam navrhovaných poloh, ze kterého má uživatel možnost si vybrat. Při vybrání některé z nich se mapa přesune na místo zadané polohy, kde vytvoří marker, který může uživatel přesunout na přesné místo podle jeho potřeby.

2.7.1.8 F8 – Podpora vztahů mezi událostmi

Tento požadavek ještě není řešen ani ve webové aplikaci, cílem tohoto požadavku je umožnit více vztahů mezi událostmi a umožnit jednotlivé události více provázat. Cílem je, aby bylo pro uživatele možné události provázat dalšími vztahy, jako je předcházející událost, nadcházející událost nebo související událost.

2.7.1.9 F9 – Podpora více lokací v jedné události

Tento požadavek taktéž ještě není implementován ani ve webové části. Jeho cílem je uživateli umožnit přiřadit ke konkrétní události více poloh. Bude nutné navrhnout jak datovou část aplikace a rozšířit logiku tak, aby bylo možné tyto vztahy ukládat a zároveň zobrazovat.

2.7.1.10 F10 – Přechod na obrazovku přihlášení, pokud se uživatel odhlásí

Tento požadavek vyšel z analýzy. Pokud uživatel klikne na položku Logout v postranním menu, ne vždy se dostane na obrazovku přihlášení. Je zde opět problém s navigací mezi obrazovkami aplikace. Funkce odhlášení je v současné aplikaci vyřešen vymazáním přihlašovacích klíčů uživatele a zavoláním metody způsobující stejné chování, jako kdyby uživatel stiskl tlačítko zpět. Pokud tedy uživatel navštívil více než jednu obrazovku, vrátí se na předchozí obrazovku a ne na obrazovku přihlášení.

2.8 Případy užití

Na základě funkčních požadavků byly vytvořeny případy užití, které tyto požadavky pokrývají.

2.8.1 UC1 – Návrat po editaci události

Uživatel se nachází na obrazovce editace události 2.6

1. Uživatel klikne na šipku zpět.
2. Zobrazí se detail původně upravované události.

2.8.2 UC2 – Založení podřízené události

Uživatel se nachází na obrazovce Detail události 2.2.

1. Uživatel klikne na šipku zpět.
2. Zobrazí se detail původně upravované události.

2.8.3 UC3 – Přesun na podřízenou událost

1. Na obrazovce Detailu události 2.2 uživatel klikne na položku seznamu podřízených událostí.
2. Aplikace zobrazí detail dané podudálosti.

2.8.4 UC4 – Přesun na nadřízenou událost

UC3 je řešen podobně jako UC2. Jen se uživatel přesune místo na obrazovku podřízené události na nadřízenou událost

1. Na detailu události uživatel klikne na položku nadudálosti.
2. Aplikace zobrazí detail dané nadudálosti.

2.8.5 UC5 – Označení nové osoby v události

Tento případ užití popisuje způsob, jak do události přidat osobu. Hlavní scénář popisuje přidání osoby, která je již vedena v profilu uživatele. Alternativní scénář popisuje případ, kdy je přidávána osoba, která ještě v profilu evidována není.

1. Na obrazovce úpravy události uživatel napíše začátek jména osoby.
2. Aplikace uživateli nabídne možnost vybrat si z již existujících osob v profilu pomocí vyskakovacího seznamu. Nabídne mu takové osoby, jejichž jméno se podobá začátku jména.
3. Uživatel si jednu osobu vybere kliknutím na položku seznamu.
4. Vybraná osoba se zobrazí v seznamu přidaných osob u události.

Alternativní scénář popisuje případ, kdy uživatel zadává osobu, která ještě není evidována v profilu uživatele.

1. Na obrazovce úpravy události uživatel napíše celé jméno osoby.
2. Uživatel klikne na tlačíko s textem „Add person“.
3. Nová osoba se zobrazí v seznamu přidaných osob u události.

2.8.6 UC6 – Přidání nové lokace k události

1. Na obrazovce pro editaci události uživatel klikne tlačítko pro přidání další lokace.
2. Aplikace zobrazí dialog přidání lokace 2.7.
3. Uživatel vyplní formulář a uloží polohu.
4. Aplikace zobrazí polohu na obrazovce editace události a po uložení události i na obrazovce detailu události.

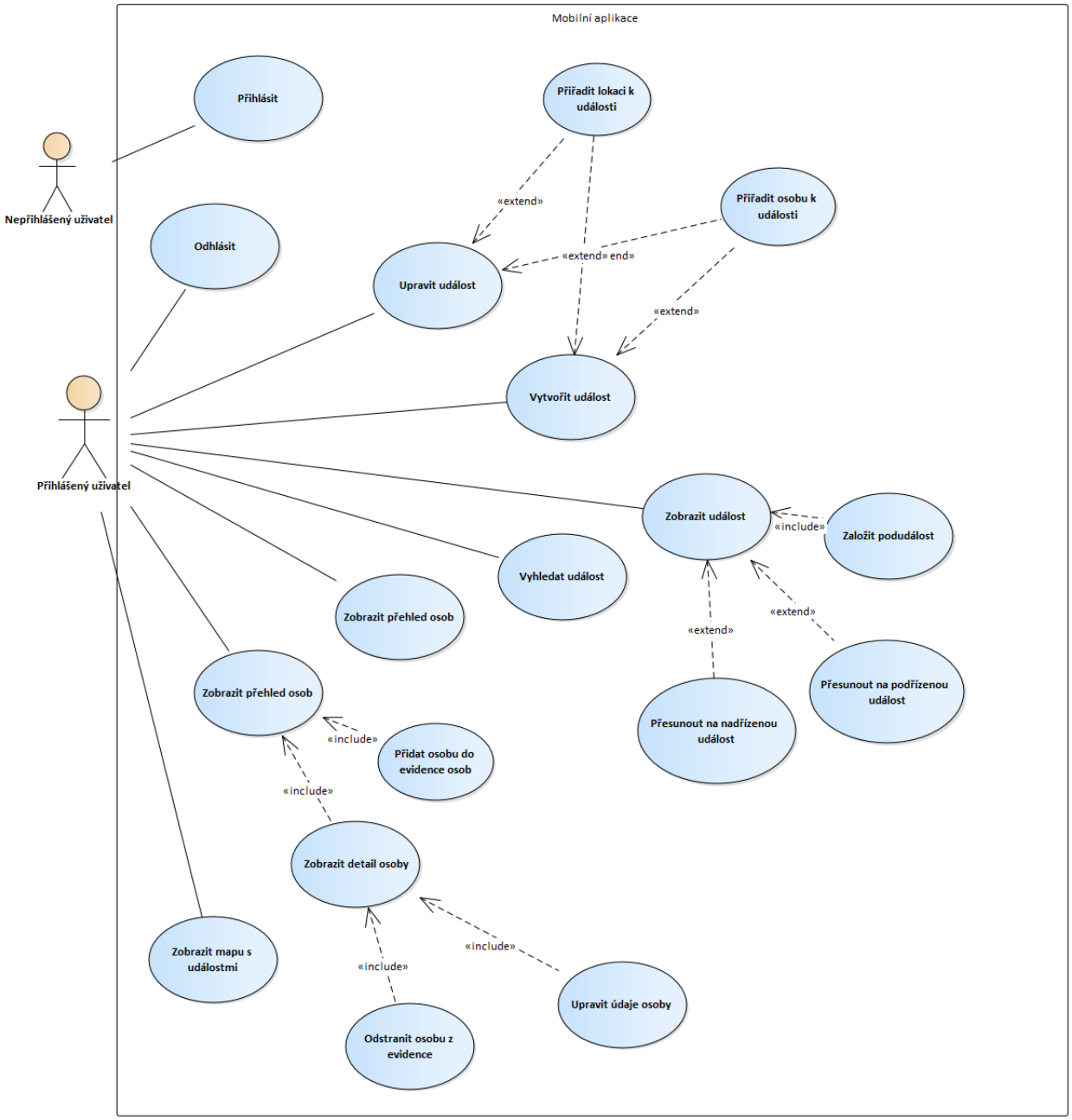
2.8.7 UC7 – Přesun z kalendáře na událost

1. Na obrazovce kaledáře uživatel stiskne položku události.
2. Aplikace zobrazí základní informace o události 2.7.
3. Uživatel klikne na tlačítko „Detail“.
4. Aplikace zobrazí obrazovku detailu události.

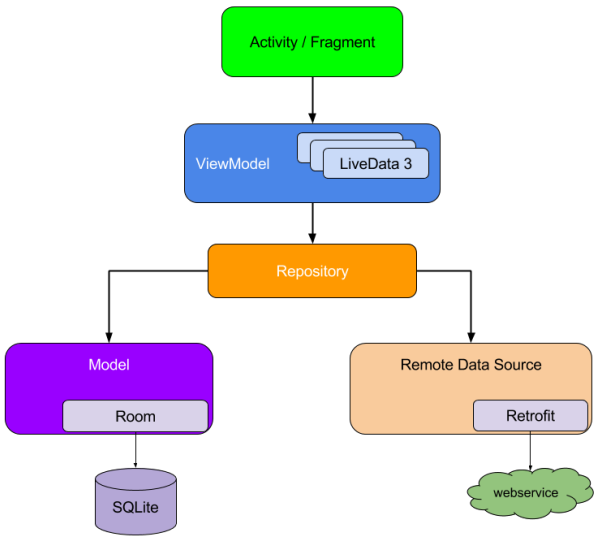
2.8.8 UC8 – Odhlášení

1. Uživatel stiskne tlačíto „Log out“ v bočním šuplíkovém menu.
2. Aplikace zobrazí obrazovku přihlášení.

Na základě nových případů užití jsem vytvořil nový Use Case diagram, který je znázorněný na obrázku 2.9.



Obrázek 2.9 Rozšířený use-case model



Obrázek 2.10 Architektura MVVM

Kapitola 3
Návrh

3.1 Použité knihovny

V této sekci popíši využité komponenty a knihovny. Aplikace byla navržena a naprogramována v moderních technologiích pro nativní vývoj aplikací pro OS Android. Jelikož se jedná o rozšíření již existujícího projektu, volbu technologií jsem zvolil takřka stejnou jako ve stávajícím projektu. Taktéž s přihlédnutím k tomu, že byly použité doporučované knihovny společností Google a je veřejně dostupná jejich široká dokumentace. Mezi rozdíly ve využitých knihovnách patřily hlavně volba *Kotlin Coroutines* [9] pro asynchronní volání v nových definicích komunikace s REST API a přidání třídy *NetworkBoundResourceFlow*, pracující právě s *Kotlin Flows*. To umožňuje dodržet princip *Single source of truth* [10] při práci s daty podobně, jako tomu je ve stávající aplikaci ve třídě *NetworkBoundResource*, která pracuje s *LiveData*. Jako programovací jazyk byl zvolen Kotlin – moderní programovací jazyk běžící nad prostředím JVM. [11] V aplikaci je použito několik knihoven ze sady Android Jetpack, což je sada knihoven doporučená k vývoji Android aplikací samotnou společností Google. Slibovaná je redukce boilerplate kódu, méně úniků paměti a kód, který funguje konzistentně napříč různými verzemi OS Android. [12]

3.1.1 Aktivita a Fragment

Aktivita i Fragment představují jednu obrazovku, se kterou uživatel může pracovat. Oproti tomu Fragment představuje část uživatelského rozhraní, která může být znovu použita několikrát v aplikaci. Každý Fragment musí být hostován v nějaké aktivitě. Jako například v této aplikaci Journal, kde jedna Aktivita slouží pro přihlášení, a zbytek akcí obstarává aktivita druhá. V té jsou všechny obrazovky vytvořené jako Fragmenty hostované právě touto Aktivitou, zapouzdřující stav přihlášeného uživatele, dále se stará o zobrazení bočního šuplíkového menu a změny právě Fragmentů jako obsahu. [13] [14]

3.1.2 Navigation Component

Navigation component [7] je jedna z novějších knihoven sady Jetpack umožňující snadnou definici navigace mezi obrazovkami aplikace. Umožňuje definovat přechody a akce vyvolající přechod, a také je možné přenášet serializovaná data mezi obrazovkami. To jsem zde využil pro předání identifikátoru události v proměnné *eventId*. Zjednodušenou specifikaci navigace v aplikaci je ukázána na ukázce 4.2 Definice je programována v jazyce XML. Jsou zde definovány dva fragmenty pomocí tagu *<fragment>* s několika atributy, jako je identifikátor *android:id* a *label* fragmentu,

Výpis kódu 3.1 Příklad specifikace navigace

```
    <?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/diaryFragment">

    <fragment
        android:id="@+id/diaryFragment"
        android:name="com.example.journalkt.diary.fragment.DiaryFragment"
        android:label="@string/diary"
        tools:layout="@layout/fragment_diary">
        <action
            android:id="@+id/action_diaryFragment_to_detailFragment"
            app:destination="@id/detailFragment" />
    </fragment>
    <fragment
        android:id="@+id/detailFragment"
        android:name="com.example.journalkt.eventdetail.DetailFragment"
        android:label="@string/event_detail"
        tools:layout="@layout/fragment_detail">
        <argument
            android:name="eventId"
            android:defaultValue="0"
            app:argType="integer" />
        <action
            android:id="@+id/action_detailFragment_to_diaryFragment"
            app:destination="@id/diaryFragment" />
    </fragment>
</navigation>
```

což je textový řetězec, který se pak zobrazuje v navigační liště. Dále je zde definována akce mezi obrazovkou *diaryFragment* a *detailFragment* pomocí tagu *action*.

3.1.3 Koin

Koin [15] je knihovna pro injektování závislostí naprogramovaná v jazyce Kotlin.

3.1.4 Room

Room je knihovna ze sady Jetpack, která poskytuje abstrakční vrstvu nad databází SQLite [16]. Umožňuje robustní přístup k databázi objektovým způsobem, ale zároveň i využít její plný výkon. Umožňuje jednoduchou definici entit pomocí vhodných anotací a *data class*. Dále umožňuje vytvořit DAO pomocí definice rozhraní. [17]

3.1.5 LiveData

LiveData je datová třída sloužící jako kontejner pro přenášená data. Splňuje návrhový vzor Observer. Dále je pro tuto knihovnu klíčové, že je tzv. lifecycle aware – to znamená, že LiveData aktualizují pouze UI komponenty, které jsou v aktivním stavu – nejsou zaniklé, takže se vyloučí možnosti úniků paměti.

Výpis kódu 3.2 Ukázka Kotlin Flows [1]

```
fun makeFlow() = flow {
   println("sending first value")
   emit(1)
   println("first value collected, sending another value")
   emit(2)
   println("second value collected, sending a third value")
   emit(3)
   println("done")
}

scope.launch {
   makeFlow().collect { value ->
       println("got $value")
   }
   println("flow is completed")
}
```

3.1.6 Flows

Kotlin Flows je knihovna, která přináší typ, který umožňuje asynchronní vysílání přenášení dat. Kotlin flows vytvoří asynchronní kanál, kterým lze přenést více dat, oproti *suspend* funkcím, asynchronně spouštěným v Kotlin Coroutines. Pomocí Flow lze data asynchronně přenést a dále pracovat funkcionálně různými operátory. Například zavolat funkci *collect* s vhodnou lambda funkcí jako parametrem, která způsobí provedení této lambda funkce s každou přijatou hodnotou pomocí této Flow. (Viz ukázka 3.2)

3.1.7 Retrofit

Retrofit je knihovna, která umožňuje jednoduchou implementaci REST klienta pomocí definice rozhraní. Usnadňuje komunikaci s REST API ve formátu JSON nebo jiných strukturovaných dat využitím converters, které zajistí serializaci a deserializaci.

3.1.8 Google Places API

Places API je služba, která umožňuje získat informace o místech pomocí HTTP požadavků. Místa jsou v tomto rozhraní definována jako provozovny, geografické lokace nebo POI.

3.1.9 Google Maps SDK

Google Maps SDK umožňují snadno přidat mapu do Android aplikace. Dále je možné do mapy přidat informace jako markery a vyznačit oblasat mnohoúhelníky a více markerů shlukovat do klastrů. [18]

3.2 Vývojový proces a použité nástroje

V této kapitole představím především nástroje používané k vývoji rozšíření aplikace Journal. Dále krátce popíši jak probíhal vývojový proces.

3.2.1 Android Studio

Pro vývoj aplikace v rámci této bakalářské práce bylo použito vývojové prostředí Android Studio Bumblebee 2021.1.1 Patch 1, jelikož je oficiálním vývojovým prostředím pro vývoj nativních mobilních aplikací pro OS Android doporučeným společností Google. Je to vývojový nástroj vyvinutý společností JetBrains založený nad IntelliJ IDEA [19] a nabízí mnoho užitečných funkcí, jako je emulace reálného zařízení pro testování funkčnosti aplikací, integraci verzovacích systémů, podporu sestavení projektu s využitím sestasvovacího nástroje Gradle [**gradle**] nebo pokročilé napovídání kódu a zobrazování dokumentace. [20]

3.2.2 Git

Verzování již bývá nedílnou součástí většiny projektů vzniku software. Na tomto projektu sice nepracovalo více vývojářů, ale i přesto je verzování proces, který umožní vývojáři pohybovat se v historii verzí a zároveň, pokud je použit vzdálený hosting, funguje také jako záloha pro případ ztráty zařízení užívaného k vývoji.

Pro verzování byl použit GIT – verzovací systém používaný ve většině software projektů.[21] Pro správu vzdáleného repozitáře byl zvolen systém Gitlab [22]. Důvody pro zvolení byly hlavně podpora fakulty, možnost spravování jednotlivých úkolů a jejich stavu, a také možnost tzv. Continuous Integration. Díky funkci Continuous integration bylo možné vytvářet testovací verze aplikace instalovatelné na reálné zařízení, čímž bylo umožněno testování na reálném zařízení a validace se zákazníkem – v tomto případě vedoucím práce.

3.2.3 Continuous Integration

Continuous Integration je postup automatizované integrace změn ve zdrojovém kódu. Její hlavní výhody jsou změnšení rizika přenesení chyb do produkčního kódu, zmenšení repetitivních procesů konaných manuálně, generace nasazovatelného kódu v jakýkoliv čas a odkudkoliv.[23] [24]

3.2.4 Postman

Nástroj `Postman` je nástroj využívaný pro testování API. Umožňuje testovat REST, SOAP a GRAPH QL API. Umožňuje i například vytvoření API testů nebo generování kódu pro využití v jíné aplikaci. Já jsem ho využil pro volání REST API a zobrazení odpovědí. Díky tomu jsem mohl lépe zpracovat očekávané odpovědi ze serveru a analyzovat funkci využívaného REST API. [25]

3.2.5 Validace a změnové požadavky

Validace produktu probíhala formou konzultací, kde byly ukázány průběžné nedostatky a připomínky k vylepšení v aktuální testované verzi.

3.3 Architektura aplikace

Architektura aplikace je navržená jako MVVM. Popsal jsem ji v kapitole 2.5. Tato architektura umožňuje oddělení vrstev aplikace, a při využití vhodných způsobů, jako je například použití rozhraní a Dependency injection, umožní i snadnou testovatelnost kódu pomocí Unit testů. V rozšíření aplikace jsem se rozhodl v této architektuře pokračovat.

3.4 Návrh tříd

3.4.1 Správa osob

3.4.1.1 PersonsListFragment

PersonsListFragment je třída reprezentující obrazovku seznamu osob. Obsahuje metodu pro načtení seznamu, kde se inicializuje **PersonListAdapter** pro zobrazení tohoto seznamu. Dále je zde observována proměnná persons z třídy **personList**, která v případě změny aktualizuje adaptér pro zobrazení seznamu osob.

3.4.1.2 PersonsListViewModel

PersonsListViewModel se stará o načítání dat pro seznam osob. Závislostí této třídy je rozhraní **PersonRepository**.

3.4.1.3 PersonRepository

PersonRepository je rozhraní popisující operace s evidencí osoby. Zajišťuje její založení, získání všech osob uživatele, získání konkrétní osoby, upravení konkrétní osoby a smazání konkrétní osoby. Poté je zde i třída **PersonRepositoryImpl**, která toto rozhraní implementuje a pomocí dependency injection knihovny Koin [15] je injektována do všech tříd, které závisí na *PersonRepository*. Tím je dodržen jeden z pricipů SOLID [26], a to je dependency inversion principle.

3.4.2 Událost

3.4.2.1 DetailFragment

DetailFragment je *fragment* starající se o zobrazení detailu události. DetailFragment observuje proměnné ze třídy *EventDetailViewModel*

3.4.2.2 EventDetailViewModel

EventDetailViewModel je *ViewModel* zajišťující získání dat o události, které se mají zobrazit. Jde především o metody *getEvent()* a *fetchSubEvents()*. Dále je zde metoda *deleteEvent()*, která zajišťuje smazání události.

3.4.3 EventsDao

EventsDao je rozhraní zajišťující komunikaci s databází prostřednictvím knihovny Room [17]. Je složena s definicemi metod s vhodnými anotacemi, ze kterých knihovna Room pomocí builderu vytvoří instanci, která je pak použita uvnitř aplikace. Obsahuje definice metod, které zajišťují CRUD operace s entitou Event. **EventRepository** je rozhraní, které reprezentuje datový zdroj zajišťující získávání dat o událostech. Jsou zde CRUD metody pro získávání událostí, a také podřízených a nadřízených událostí konkrétní události. Opět je zde i její implementace ve třídě **EventRepositoryImpl**

3.4.4 Editace události

3.4.4.1 AddEditFragment

AddEditFragment je *fragment* starající se o zobrazení obrazovky editace události.

3.4.4.2 AddEditEventViewModel

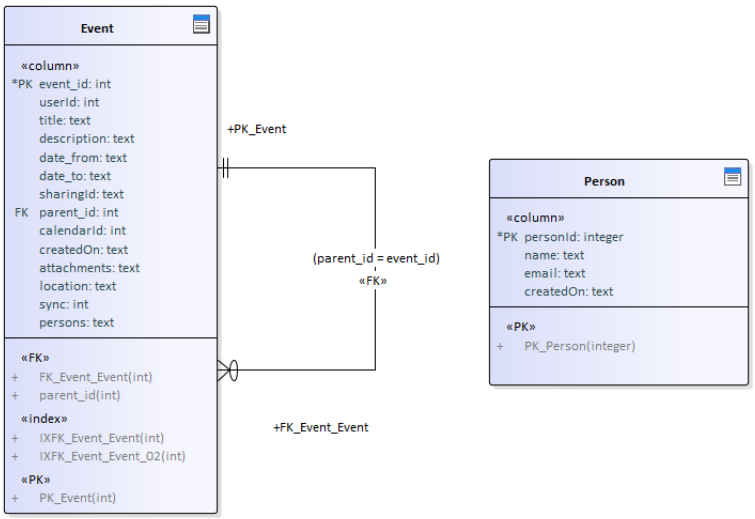
AddEditEventViewModel je *ViewModel* starající se o přípravu dat pro zobrazení editace události. Obsahuje metody zajišťující data všech osob uživatele, stejně jako CRUD operace s jednotlivými osobami a taktéž označování jednotlivých osob v události. Dále obsahuje stávající implementaci práce s událostmi, a také metodu pro odeslání přílohy v události.

3.4.4.3 ChooseLocationFragment

ChooseLocationFragment je *fragment* dědící ze třídy *BottomSheetDialogFragment* a obstarává označení polohy do události. Obsahuje dva textové vstupy. První z nich je *Autocomplete fragment* z knihovny Places SDK umožňující zobrazit nabízené lokace během psaní adresy. Po zvolení konkrétní polohy má uživatel možnost tuto lokaci pojmenovat. Dále se na této obrazovce nachází mapa zobrazující marker, kde navíc uživatel může dlouhým stiskem marker přesunout a zpřesnit tak zadání lokace.

3.5 Návrh databáze

Všechny informace přijaté ze serveru, které se zobrazují uživateli, jsou uloženy do lokální mezipaměti. Ta je implementována pomocí lokální databáze. V současné implementaci jsou všechny entity závislé na entitě *Event*, reprezentující událost, implementovány jako serializované objekty. Nyní je ale potřeba navrhnout rekurzivní vztah právě v entitě *Event* pro zprostředkování vztahu o hierarchických událostech. Vztah nadřízené a podřízených událostí je reprezentován jako vztah

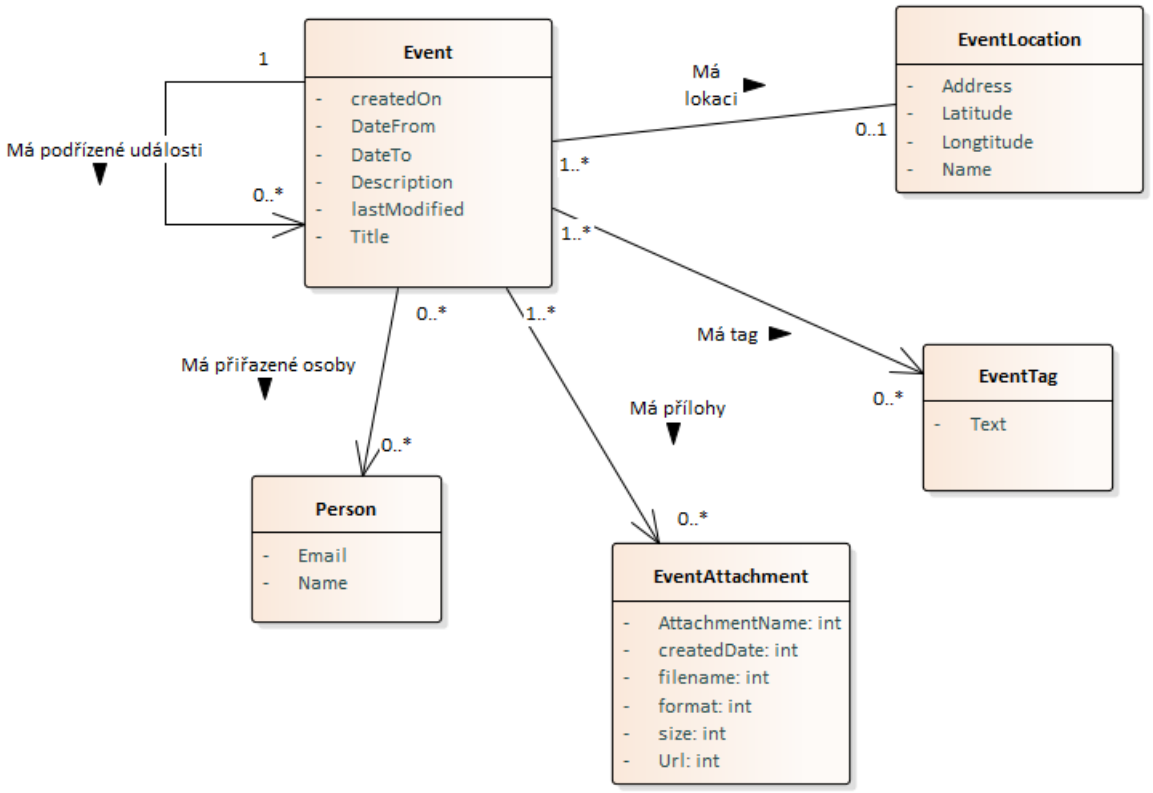


Obrázek 3.1 Upravený databázový model

One to many, to znamená, že jedna nadřízená událost může mít žádnou a nebo více podřízených událostí. Dále zde nově přibyla entita Person, reprezentující uloženou osobu z obrazovky Persons.

3.6 Doménový model

Na obrázku 3.2 je znázorněn rozšířený doménový model aplikace. Od toho stávajícího se liší tím, že přibyl rekurzivní vztah u entity Event popisující událost a její podudálosti. Dále přibyla entita Person, která popisuje osobu přiřazenou k události.



Obrázek 3.2 Rozšířený doménový model

Kapitola 4
Implementace

Požadavků na změny v této aplikací bylo mnoho. Bude zde popsána implementace čtyř stěžejních požadavků rozšíření aplikace. Nejprve představím použité nástroje a stručně popíši vývojový proces.

4.1 Podpora hierarchických událostí

V této sekci popíšu implementaci požadavku pro podporu hierarchických událostí. Stěžejní částí pro tento požadavek byla datová vrstva pro zajištění získání vztahu o podřízené a nadřízené události, aby bylo možné se z detailu původní události přesunout nas nadřízenou událost, či jednu z podřízených událostí.

4.1.1 Datová vrstva

Pro možnost zobrazit lokálně hierarchické události bylo nutné vztahy mezi událostmi uložit do databáze. V té bylo potřeba zajistit relace mezi událostmi v lokální databázi pomocí knihovny Room [17]. Docíleno toho bylo použitím cizího klíče *parentId* v entitě události. Tento cizí klíč byl poté použit pro získání nadřízených událostí i podřízených událostí vhodnými dotazy v jazyce SQL. Příklad získání podřízených událostí je v ukázce 4.3.

4.1.2 View vrstva

Uživatelské rozhraní bylo rozšířeno o funkci přidání podudálosti. Toho bylo docíleno přidáním tlačítka se symbolem hierarchie do navigační lišty.

Funkce přidávání poté byla docílena rozšířením Fragmentu přidání události o nový parametr při přidávání události, a to *parentEventId*. Ten nabývá číselné hodnoty *-1*, pokud se zakládá klasická událost. Nebo má hodnotu identifikátoru nadřazené události.

Dalším krokem bylo vypsat, jaké podřízené a nadřízené události daná událost má. Aby bylo možné se přesunout z jedné události na jinou, bylo potřeba upravit navigaci. Přidána proto byla akce do souboru nav_graph.xml, kde je specifikována navigace v aplikaci (viz 4.5). Toho bylo docíleno rozšířením navigace aplikace přidáním akce pro přesun z události do události.

4.2 Přehled osob

V této sekci popíši realizaci přehledu osob.

4.2.1 View vrstva

Byla vytvořena nová obrazovka se seznamem osob, kde je možné zobrazit všechny osoby daného uživatele. Dále byla vytvořena obrazovka přidání osoby, kam se dá dostat pomocí FAB z obrazovky seznamu osob. Vyplněním formuláře a potvrzením tlačítkem *Save* je možné novou osobu vytvořit. Kliknutím na položku seznamu je poté možné se přesunout na obrazovku detailu osoby, kde je možné zobrazit jméno, email a také události této osoby.

4.2.2 Datová vrstva

Pro tento požadavek bylo nutné v databázi vytvořit novou tabulku pro evidenci osob. Dále byla datová vrstva rozšířena o Repository, které obstarává CRUD operace s osobami.

4.3 Označení osob v události

Označení osob bylo naprogramováno pomocí widgetu Autocomplete Textview. Tento widget splňuje funkcionalitu textového okna s vyskakovacím seznamem návrhů. Dále byl seznam osob přidán jako třídní proměnná entity Event. Není tedy vytvořena M:N vazba mezi tabulkami Event a Persons, ale osoby jsou ukládány pomocí serializace a deserializace. Objekty osob jsou tedy deserializovány do tabulky událostí. Serializaci a deserializaci zajistí knihovna Room pomocí *type converters* (viz obrázek 4.4). Tím se docílilo i synchronizace s backendem systému Journal.

Druhou nutností pro splnění tohoto požadavku bylo přidat funkci zobrazení osoby na obrazovce úpravy události. Toho bylo docíleno přidáním seznamu s osobami v události, kde je i křížek, čímž lze docílit odebrání osoby z této události. Layout obrazovky editace události je možné vidět na obrázku 4.3.

4.3.1 Editace osoby

Obrazovka editace osoby byla naprogramována podobně, jako vypadá editace události. V navigační liště je tlačítko s ikonou tužky, které vyvolá obrazovku editace osoby, což je stejná obrazovka jako obrazovka pro přidání nové osoby s tím rozdílem, že údaje o osobě jsou předvyplněné.

4.4 Přiřazení polohy události

V rámci tohoto požadavku byl Dialog přepracován na Bottom Sheet Dialog – Dialogové okno, při vyjíždějící ze spodního okraje obrazovky nahoru. Pro vytvoření seznamu nabízených událostí byla využita knihovna Google Places SDK [27]. Dále byl místo jednoduchého widgetu pro vstup textu přidán *Autocomplete fragment*. Splněn byl požadavek zobrazení možností adres, ze kterých si může uživatel vybrat. Vybranou polohu bylo nutné předat zpět Fragmentu editace události. Toho bylo docíleno využitím *SavedStateHandle*, což je objekt Fragmentu, do kterého se dají uložit informace ve tvaru klíč—hodnota. Poté je možné zavolat metodu *getLiveData* s generickým parametrem typu uloženého objektu. Tím lze obdržet LiveData a poté je observovat v předchozím fragmentu a hodnotu získat. Na obrázku 4.1 je informace vepsána do *SavedStateHandle* předchozího fragmentu. Na obrázku 4.2 se nachází úryvek kódu přijmutí lokace z fragmentu zadání polohy.

Výpis kódu 4.1 Implementace View Mapy

```
class MapViewInScrollView(context: Context?, attrs: AttributeSet?) :
    MapView(context!!, attrs) {
    override fun dispatchTouchEvent(ev: MotionEvent): Boolean {
        when (ev.action) {
            MotionEvent.ACTION_UP -> {
                println("unlocked")
                this.parent.requestDisallowInterceptTouchEvent(false)
            }
            MotionEvent.ACTION_DOWN -> {
                println("locked")
                this.parent.requestDisallowInterceptTouchEvent(true)
            }
        }
        return super.dispatchTouchEvent(ev)
    }
}
```

4.4.1 Mapa uvnitř dialog fragmentu

V rámci tohoto požadavku byl upraven taktéž samotný widget mapy, který je použit zde v dialogovém okně, ale také na detailu události. Po mapě se lze nyní posouvat jak vertikálně, tak i horizontálně. Bylo nutné vytvořit vlastní View třídu dědící z původního MapView. Poté bylo využito přetěžování a přepsána byla metoda *dispatchTouchEvent*, která se zavolá právě v případě, že se uživatel rozhodne posunout mapu. Docílí to správného chování mapy a posunu mapy uvnitř pokud ji uživatel posouvá vertikálním směrem.

```
binding.saveLocation.setOnClickListener { it: View!
    location?.name=binding.name.editText?.text.toString()
    findNavController().previousBackStackEntry?.savedStateHandle?.set(
        AddEditEventFragment.EDIT_LOCATION,
        location
    )
    dismiss()
}
```

Obrázek 4.1 Předání informace o poloze z dialogu

4.4.2 Chybný návrat po editaci události

Tento požadavek byl vyřešen změnou parametrů navigace Fragmentu diáře *diaryFragment* [7]. Definování obrazovek jsem popsal v části 4.5. V kódu se objevila chyba, kdy bylo docíleno toho, že v případě kliknutí na tlačítko zpět se ze zásobníku obrazovek vymaže obrazovka editace, obrazovka detailu i obrazovka seznamu událostí. Bylo to způsobeno těmito řádkami kódu. 4.4

Parametr `popUpTo` způsobuje, že z vrcholu zásobníku obrazovek se odstraní všechny obrazovky až do obrazovky s identifikátorem `"diaryFragment"`. Pokud přidáme i parametr `popUpToInclusive="true"` způsobí to, že se odstraní také obrazovka zapsaná do parametru `popUpTo` To způsobí nepředvídatelné chování, protože záleží na tom, jestli byla před obrazovkou Diary zobrazena nějaká jiná obrazovka. V tom případě se zobrazí tato obrazovka. Pokud ne, aplikace se restartuje.

Problém byl tedy odstraněn vymazáním zmíněných dvou řádek, které problém způsobovaly.

```
findNavController().currentBackStackEntry?.savedStateHandle?.getLiveData<EventLocation>(
    EDIT_LOCATION
)?.observe(viewLifecycleOwner) { it: EventLocation!
    it?.let { it: EventLocation
        setUpLocation(it)
    }
}
```

Obrázek 4.2 Přijmutí informace o poloze v předchozím fragmentu

4.5 Definice navigace v aplikaci

Knihovna Navigation component umožní jednoduché navržení navigace napříč obrazovkami. Umožňuje definovat přechody, a také je možné přenášet data mezi obrazovkami. To jsem zde využil pro předání identifikátoru události v proměnné *eventId*. Zjednodušená část specifikace navigace v obrazovce je ukázána na ukázce 4.2

V aktuální verzi aplikace není možné vytvářet podřízené události ani zobrazovat informace o tom, jaké podřízené nebo nadřízené události konkrétní událost má. Tuto funkcionalitu jsem rozdělil do několika případů užití.

4.6 Aktualizace knihoven

Aktualizace knihoven patří k neoblíbeným činnostem softwarového vývoje. Mohou se objevit nové problémy, pokud jsou verze navzájem nekompatibilní. Je to ale možnost, jak udržet fungující software delší dobu aktuální. V prvotní fázi vývoje jsem se věnoval aktualizaci použitých knihoven, verze jazyka Kotlin, a také jsem odstranil repositář JCenter ze zdrojů pro repozitáře, jelikož tento repozitář byl oznámen jako zastaralý a dále by se jako zdroj závislostí neměl používat. Později bylo oznámeno, že repositář bude dále dostupný jako read only, ale pro nové knihovny a pro nahrání nových verzí knihoven musí vývojáři použít jiné repozitáře.[28]

4.7 Shrnutí

V části implementace byly implementovány požadavky na rozšíření klienta Journal. Je třeba zmínit, že tři požadavky splněny nebyly. A to podpora více lokací v události, podpora vztahů mezi událostmi a podpora externích kalendářů a přesun z kalendáře na detail události. Návrh a implementace především datových částí lokálního cachování ostatních požadavků byly více náročné, než bylo očekáváno. Proto tyto požadavky zůstávají jako možnost pro další rozvoj této aplikace.

Výpis kódu 4.2 Příklad specifikace navigace

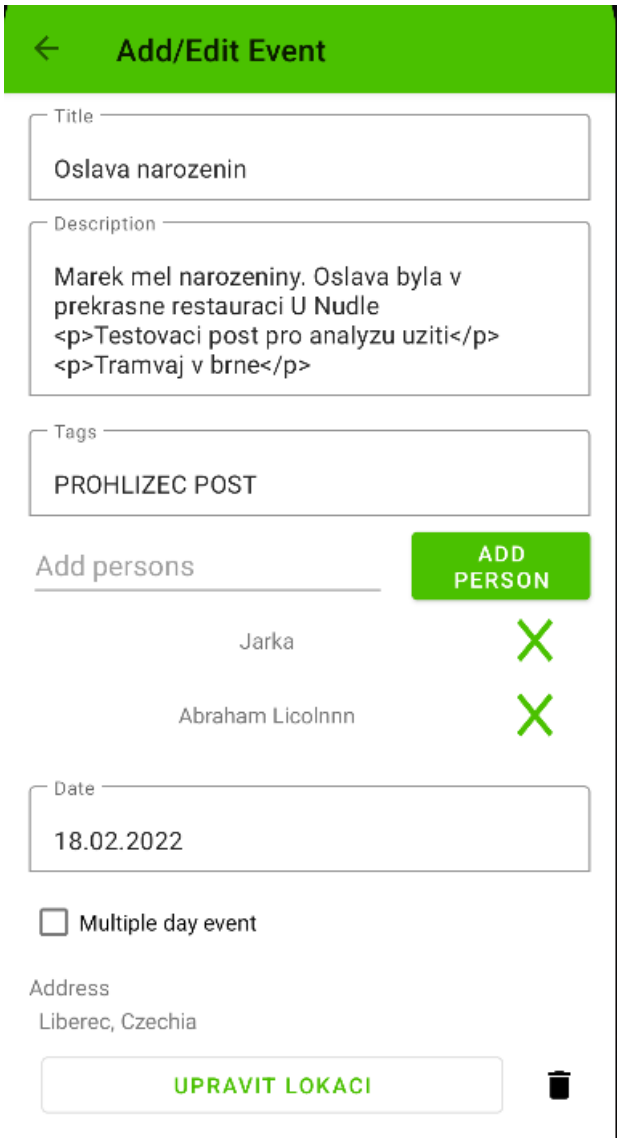
```
    <?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/diaryFragment">

    <fragment
        android:id="@+id/diaryFragment"
        android:name="com.example.journalkt.diary.fragment.DiaryFragment"
        android:label="@string/diary"
        tools:layout="@layout/fragment_diary">
        <action
            android:id="@+id/action_diaryFragment_to_detailFragment"
            app:destination="@id/detailFragment" />
    </fragment>
    <fragment
        android:id="@+id/detailFragment"
        android:name="com.example.journalkt.eventdetail.DetailFragment"
        android:label="@string/event_detail"
        tools:layout="@layout/fragment_detail">
        <argument
            android:name="eventId"
            android:defaultValue="0"
            app:argType="integer" />
        <action
            android:id="@+id/action_detailFragment_to_diaryFragment"
            app:destination="@id/diaryFragment" />
    </fragment>
</navigation>
```

Výpis kódu 4.3 Získání podřazených událostí

```
@Transaction
    @Query("SELECT * FROM events where parent_id =:id")
     fun getSubEvents(id: Int): List<Event>
```

Výpis kódu 4.4 PopUpTo

```
    app:popUpTo="@+id/diaryFragment"
    app:popUpToInclusive="true"
```



Obrázek 4.3 Nový layout obrazovky editace události

```kotlin
@TypeConverter
fun personsToJson(persons: List<Person>?): String? {
    return if (persons == null) null
    else Gson().toJson(persons)
}

@TypeConverter
fun jsonToPersons(personsJson: String?): List<Person>? {
    return if (personsJson == null) null
    else Gson().fromJson(personsJson, Array<Person>::class.java).toList()
}
```

Obrázek 4.4 Type converters

Kapitola 5
Testování

5.1 Testy zdrojového kódu

Testy zdrojového kódu jsou typicky spouštěné ve vývojovém prostředí. Testují funkčnost částí zdrojového kódu a umožňují co nejvíce zabránit regresním chybám nebo chybám při refactoringu. V nativním programování aplikací pro OS Android rozlišujeme dva typy unit testů. A to Unit testy a Instrumented unit testy.

5.1.1 Klasické unit testy

Klasické unit testy jsou stejné jako v ostatních programovacích jazycích a technologiích. Měly by testovat funkčnost malých částí – metody a třídy. Lokální testy tedy umožňují verifikovat logiku aplikace rychleji, než testy na zařízení s OS Android. Přesto ale to, že v lokálních unit testech není možné interagovat s jakýmikoliv komponentami z OS Android, tyto testy značně limituje. Podle [29] by se měla otestovat hlavně datová vrstva. Dále by se měla otestovat vrstva doménová. Stejně jako doménová vrstva by měla být nezávislá na platformě. Dále by se měly testovat tzv. utility classes. Jsou to znovupoužitelné třídy zajišťující manipulaci s daty. Může to být například třída zajišťující porovnávání nebo řazení prvků.

Unit testy jsou v projektu v balíčku **cz.cvut.fit.journalkt**.

5.1.2 Instrumented testy

Instrumented testy jsou druhým typem testů. Od standardních unit testů se liší tím, že do této kategorie spadají testy, které potřebují prostředí OS Android. Mohou to být například UI testy, tedy testy uživatelského rozhraní nebo testy navigace.

5.2 Testy použitelnosti

Pro uživatelské testy jsem vytvořil testovací scénáře, které ověřují několik nových případů užití aplikace z této bakalářské práce. V průběhu práce byly jednotlivé funkcionality testovány programátorem. Zároveň byly některé funkce konzultovány se zadavatelem. K testu použitelnosti byli vybráni tři testeři.

5.2.1 Tester 1

Tester 1, je ve věku 20—30 let, studuje informatiku na vysoké škole a používá systém Android na svém smartphone.

5.2.2 Tester 2

Tester 2, je ve věku 20—30 let a sice používá systém Android na svém smartphone, ale vykonat s telefonem složitější operace pro něj bývá problém.

5.2.3 Tester 3

Tester 3 je ve věku 50—60 let s taktéž spíše základními schopnostmi řešit problémy na elektronických zařízeních.

5.2.4 Uživatelské scénáře

Každému uživateli byly zadány tyto úkoly. Popisují hlavní požadavky této bakalářské práce.

1. Založte k libovolné události podřízenou událost „Slavnostní přípitek“ s libovolným datem a popiskem.
2. Do evidence osob přidejte novou osobu se jménem „Jan Novák“.
3. Do libovolné události přidejte osobu se jménem „Jan Novák“.
4. Do libovolné události přidejte polohu města Liberec.

5.2.5 Poznatky z testů použitelnosti

Testy použitelnosti odhalily některé chyby v uživatelském rozhraní. Především uživatelé s menšími znalostmi systému Android nedokázali vykonat danou operaci napoprvé. Nejzávažnější problém byl s přidáním podřízené události, kde nebylo spatřeno vstupní pole pro zadání adresy. To bylo opraveno podbarvením tohoto okna pro lepší rozpoznatelnost.

Kapitola 6
Závěr

Hlavním cílem této bakalářské práce bylo rozšíření již existující nativní mobilní aplikace pro OS Android. A to o případy užití zatím obsloužitelné pouze z webového rozhraní. Nově je možné pohodlněji zadávat polohu události, jelikož se po zadání adresy polohy zobrazí i nabídka míst, ze kterých si uživatel může vybrat. Dále je uživateli umožněno po zadání adresy posunout marker a zadat tak umístění přesněji. Dále se mi povedlo implementovat správu osob se zobrazením událostí jednotlivých osob, přidávání osob do událostí. Taktéž je možné pracovat s hierarchickými událostmi, a to zakládat podřízené události a u jednotlivých událostí zobrazovat nadřízenou událost, a také události podřízené. S možností přesunout se na konkrétní podřízenou událost nebo nadřízenou událost. Byly opraveny různé chyby z předchozí verze aplikace, a to nevhodné chování navigace, špatné formátování textu a nefunkční odhlášení. Čtyři z požadavků tohoto rozšíření aplikace implementovány nebyly, a to integrace externích kalendářů, podpora více lokací v jedné události a podpora vztahů mezi událostmi. Bylo to z důvodu nepodpory backendu systému Journal, a také složitosti ostatních požadavků, kterým byla věnována větší pozornost. V první části práce jsem analyzoval současnou aplikaci a její funkce, užité technologie a architekuru. Dále jsem specifikoval požadavky pro implementaci rozšíření. V praktické části jsem většinu z požadavků stanovených v této kapitole implementoval.

Výsledkem je funkční aplikace, která byla uživatelsky otestována. Zároveň bylo zavedeno Continuous Integration na službě GitLab umožňující jednoduché vyprodukování aplikace ze zdrojového kódu.

Bibliografie

1. *Learn advanced coroutines with Kotlin Flow and LiveData* [online]. [B.r.] [cit. 2022-05-11]. Dostupné z: https://developer.android.com/codelabs/advanced-kotlin-coroutines#7.
2. *Journal - O aplikaci*. Praha: Praha, 2021. Dostupné také z: https://journal.zryb.eu/about/about/.
3. FUKALOVÁ, Marie. *Mobilní klient pro aplikaci Journal* [online]. 2020 [cit. 2022-03-19]. Dostupné z: https://dspace.cvut.cz/handle/10467/90387.
4. FOWLER, Martin; RICE, David; FOEMMEL, Matthew; HIEATT, Edward; MEE, Robert; STAFFORD, Randy. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.
5. CHENG, Yun; DOMÍNGUEZ, Olivares. *Advanced Android App Architecture: Real-world app architecture in Kotlin 1.3*. First Edition. Razeware LLC, 2019. ISBN 9781942878698.
6. SHAHBAZ, Ahmed. *Getting started with android architecture components and MVVM Part-1*. San Francisco, CA: A Medium Corporation, 2017. Dostupné také z: https://medium.com/android-news/getting-started-with-android-architecture-components-and-mvvm-156a96a1bd05.
7. [Online] [cit. 2022-03-19]. Dostupné z: https://developer.android.com/guide/navigation/navigation-getting-started.
8. [Online] [cit. 2022-05-01]. Dostupné z: https://developer.android.com/guide/topics/ui/dialogs.
9. [Online] [cit. 2022-05-01]. Dostupné z: https://developer.android.com/kotlin/coroutines?gclid=CjwKCAjwve2TBhByEiwAaktM1BIBddMgMD8vYsW-TVtJCVuPQ92EG4AeG_JwkqPPl33dEL2rN6JjMBoCSWMQAvD_BwE&gclsrc=aw.ds.
10. *Concept Single Source of Truth - Dragon1 Software* [online]. [B.r.] [cit. 2022-05-11]. Dostupné z: https://www.dragon1.com/concepts/single-source-of-truth.
11. *Kotlin Programming Language* [online]. [B.r.] [cit. 2022-05-11]. Dostupné z: https://kotlinlang.org/.
12. [Online] [cit. 2022-03-19]. Dostupné z: https://developer.android.com/jetpack.
13. [Online] [cit. 2022-05-01]. Dostupné z: https://developer.android.com/reference/android/app/Activity.
14. [Online] [cit. 2022-05-01]. Dostupné z: https://developer.android.com/guide/fragments.
15. KOIN & KOTZILLA. *Koin*. 2022. Ver. 3.1.5. Dostupné také z: https://insert-koin.io/.

16. HIPP, Richard D. *SQLite*. 2020. Ver. 3.31.1. Dostupné také z: https://www.sqlite.org/index.html.
17. [Online] [cit. 2022-05-01]. Dostupné z: https://developer.android.com/jetpack/androidx/releases/room.
18. [Online] [cit. 2022-05-01]. Dostupné z: https://developers.google.com/maps/documentation/android-sdk/overview.
19. JETBRAINS. *IntelliJ IDEA*. JetBrains, 2019. Dostupné také z: https://www.jetbrains.com/idea/.
20. STUDIO, Android. *Download Android Studio and SDK tools*. 2019. Dostupné také z: https://developer.android.com/studio.
21. [Online] [cit. 2022-05-01]. Dostupné z: https://git-scm.com/.
22. [Online] [cit. 2022-05-01]. Dostupné z: https://about.gitlab.com/.
23. DUVALL, Paul M; MATYAS, Steve; GLOVER, Andrew. *Continuous integration*. Boston, MA: Addison-Wesley Educational, 2007.
24. REHKOPF, Max. *What is continuous integration*. [B.r.]. Dostupné také z: https://www.atlassian.com/continuous-delivery/continuous-integration.
25. *Postman: What is Postman?* San Francisco: Postman, Inc., 2022. Dostupné také z: https://www.postman.com/product/what-is-postman/.
26. OLORUNTOBA, Samuel. *SOLID the First 5 Principles of Object Oriented Design* [online]. 2020 [cit. 2022-05-01]. Dostupné z: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.
27. [Online] [cit. 2022-05-01]. Dostupné z: https://developers.google.com/maps/documentation/places/android-sdk/overview.
28. CHIN, Stephen. *Into the sunset: Bintray, JCenter, GoCenter, and Chartcenter*. 2022. Dostupné také z: https://jfrog.com/blog/into-the-sunset-bintray-jcenter-gocenter-and-chartcenter/.
29. [Online] [cit. 2022-05-01]. Dostupné z: https://developer.android.com/training/testing/fundamentals/what-to-test.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `apk` ..... adresář se spustitelnou formou mobilní aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF