FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Android aplikace pro podporu rodin v rozvodovém řízení
Student:	Martin Beran
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem této práce je realizace android aplikace pro podporu rodin v rozvodovém řízení. Backendovou část (API) řeší souběžně ve spolupráci s autorem této práce bakalářský student Iaroslav Kolodka.

Postupujte v těchto krocích:
1. V návaznosti na Softwarový Týmový Projekt analyzujte současný stav vývoje. Proveďte revizi požadavků klienta a řádně nastudujte API navržené Iaroslavem Kolodkou.
2. Na základě analýzy navrhněte vhodné úpravy tak, aby byl dosažen kvalitnější výsledek a zároveň byly splněny požadavky klienta.
3. Implementujte android aplikaci dle analýzy a návrhu.
4. Při implementaci věnujte řádnou pozornost testům - navrhněte a aplikujte vhodné testy. Dále řádně konzultujte řešení s klientem.
5. Zhodnoťte použitelnost výsledné aplikace a navrhněte vhodné budoucí kroky (např. rozšíření, technologické směřování, apod.).

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 31. ledna 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Android aplikace pro podporu rodin v rozvodovém řízení

Martin Beran

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Hunka

3. června 2020

Poděkování

Chtěl bych poděkovat panu Ing. Jiřímu Hunkovi za vedení práce. Dále Bc. Lukášovi Talpovi a Mgr. Bohuslavě Janků za přiblížení problematiky rozvodů. Všem spolužáků, kteří se mnou spolupracovali v předmětech BI-SP1 a BI-SP2 na vývoji aplikace. A konečně i rodině za podporu při studování na škole.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu 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 předpisu. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ustanovení § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisu.

V Praze dne 3. června 2020 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Beran, Martin. *Android aplikace pro podporu rodin v rozvodovém řízení.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

V této práci je zpracován postup vytváření Android aplikace. Aplikace má za úkol pomoci rodinám ulehčit rozvod a hlavně nastalou situaci po rozvodu. Aplikace vznikla na požadavek klienta Bc. Lukáše Talpy. Pan Talpa má v tomto odvětví širokou praxi. Aplikace by měla ulehčit řešení porozvodových problémů a poskytne důkazy pro případná soudní řízení.

Klíčová slova Android aplikace, pomoc při rozvodech, právní ochrana, kotlin

Abstract

This thesis describes a creation of the Android application. The main goal of the application is help families with divoreces. The application has been created on demand of Bc. Lukas Talpa. Mr Talpa has large experience in this sector. The application should help solving problems connected with divorces and it will provide evidence for possible trials.

Keywords Android application, help with divorces, legal protection, kotlin

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

Rozvod je nepříjemná události, kterou si nikdo nepřeje zažít. Nicméně občas tato událost nastane, a potom je potřeba se s tím popasovat. V dnešní době internetu a rozsáhlých možností, kdy každý buduje kariéru a na vztahy nezbývá tolik prostoru, se může stát, že k rozvodu dojde velice rychle.

Když už k rozvodu dojde, obvykle to nebývá klidný a spořádaný proces, ale vyhrocené utkání, ve kterém se každá strana snaží získat co nejvíce. Bohužel většinou na úkor dětí. A proto zde přišel nápad vytvořit mobilní aplikaci, která by rodičům usnadnila řešení případných porozvodových sporů a pokusila se děti odstínit od rozvodových hádek rodičů.

Proto přišla myšlenka, jejímž cílem bylo vytvořit mobilní aplikace, která pomůže rodinám v rozvodovém řízení a v době po něm. Aplikace vznikala pod vedením pana Ing. Jiřího Hunky jako reakce na požadavek od zadavatele Bc. Lukáše Talpy, který s myšlenkou na vytvoření aplikace přišel.

Vývoj začal v letním semestru B182 v předmětu BI-SP1. V tomto předmětu jsme s týmem pod mým vedením zjišťovali první poznatky o této problematice na pravidelných týdenních schůzkách s klientem.

Společně jsme analyzovali požadavky a vymýšleli možné postupy jak požadavky splnit. Z analýzy vyplynulo, že celá aplikace se bude točit kolem několika funkcionalit, které by měly pomoci lidem usnadnit rozvody, a nebo si aspoň ukládat důkazy pro pozdější soudní řízení mezi zúčastněnými stranami.

Na konci BI-SP1 začala prvotní implementace v jazyce Kotlin. V počátku na implementaci Aplikace pracovali společně se mnou další tři lidé. Později v BI-SP2.1 jsem na implementaci pracoval sám. V mojí práci jsem poznatky z toho předmětu použil jako odrazový můstek.

Cílem mojí práce je vytvořit Android aplikaci, která splní zadání od klienta a poskytne vhodný základ pro budoucí vývoj projektu. Aplikace by měla plnit požadavky klienta, ale ve zjednodušené formě. Aplikace bude fungovat pouze offline bez napojení na Backend. Implementace Backend aplikace není předmětem této práce.

Backend aplikace vzniká jako další bakalářská práce Iaroslav Kolodky. Android aplikace by se ve finální verzi měla napojit na rozhraní této BE(Backend) aplikace a společně by měly vytvořit fungující systém s požadovanou funkcionalitou.

V mojí práci jsem měl analyzovat současný stav vývoje aplikace po předmětu BI-SP2.1 a na něj navázat, navrhnout vhodné úpravy předešlé analýzy a na jejím základě vytvořit Android aplikaci, která splní zadání klienta. Tuto aplikaci následně otestovat a navrhnout vhodná místa pro zlepšení.

Kapitola 1

Vývoj aplikace v kostce

Zde se seznámíme, jak probíhal celkový vývoj aplikace od jejích počátků v předmětu BI-SP1 až do dnešních dnů. Stručně zde vysvětlím hlavní milníky, které ovlivnily vývoj samotné aplikace. Povíme si něco o lidech, kteří se mnou na aplikaci začínali pracovat a shrneme si v jakém stavu jsem aplikaci převzal.

1.1 Vývoj v BI-SP1

Úplný začátek vývoje aplikace nastal v předmětu BI-SP1, kde jsme se sešli v poměrně rozsáhlém týmu 7 lidí. Naši vedoucí byli Ing. Jiří Hunka a Ing. Oldřich Malec. Na první poradě bylo rozhodnuto, že budu dělat vedoucího tohoto týmu. Bylo tak rozhodnuto na základě mé zkušenosti z praxe, kterou nikdo v týmu nedisponoval. Na první schůzce jsme se dále dozvěděli, že budeme dělat mobilní aplikace pro rodiny po rozvodu, ale v té době jsem si opravdu nebyl schopný představit, co taková aplikace bude obnášet. Nakonec jsme zjistili, že aplikace vzniká jako nápad klienta pana Hunky.

Na druhé schůzce jsme se již seznámili s klientem Bc. Lukášem Talpou. Zde jsme se dozvěděli, jak takový rozvod probíhá a co jsou v této problematice nejčastější problémy. Schůzky se opakovaly každý týden a postupně jsme zde budovali pohled na tuto problematiku. Na pár schůzkách byla Mgr. Bohuslava Janků, která nás uvedla do této oblasti očima ženy.

Podle osnovy jsme tvořili scénáře, usecase diagramy, class diagramy a wireframy. Dále zde vznikla architektura aplikace. Všechny tyto věci jsem později použil jako odrazový můstek k analýze samotné Android aplikace.

Na konci předmětu jsme měli analýzu z velké části hotovou a vznikly první realizace Android aplikace a serverového řešení. Tento prototyp Android aplikace se později stal základem mojí aplikace. Na prvotní verzi jsem pracoval společně z dalšími 3 lidmi z týmu. Zbylí členové týmu tvořili implementaci BE aplikace.

1.2 Vývoj v BI-SP2.1

V předmětu, který navazuje na BI-SP1, s názvem BI-SP2.1 jsme pokračovali v tvorbě systému. Bohužel nás zde opustili dva členi, kteří se mnou pracovali na prvotní verzi aplikace. Nakonec jsme se v týmu dohodli, že na Android aplikaci budu pracovat sám a zbytek členů se přesune na BE aplikaci. Zde jsem se taky s panem Hunkou domluvil, že android aplikaci použijeme jako mojí bakalářskou práci.

Od této chvíle jsem tedy začal na aplikaci aktivně pracoval. Kvůli rozsahu aplikace jsem zjednodušil analýzu, kterou jsme společně vytvořili v BI-SP1, do méně pracné podoby.

Po ořezání analýzy jsem se pustil do implementace. Zde jsem plynule navázal na prvotní verzi aplikace, kterou jsme vytvořili. Nicméně tato verze obsahovala jenom základní navigaci skrz aplikaci. Na některých obrazovkách se objevovaly základní komponenty jako tlačítka, textview atd. Aplikace byla téměř holá.

Aplikaci jsem postupně začal tvořit podle analýzy z předchozího předmětu. Svůj postup jsem pravidelně konzultovali s členy týmu a s klientem.

Na konci předmětu již existovala aplikace se základní funkcionalitou vyžadovanou klientem. Aplikace sice ještě neobsahovala všechny požadované části, ale hlavní části již v aplikaci existovaly a fungovaly.

1.3 Vývoj po BI-SP2.1

Po skončení předmětu vývoj plynule pokračoval. Pracoval jsem na dokončení všech požadovaných funkcí. Aplikaci jsme na schůzkách pravidelně ukazovali zadavateli. Při těchto konzultací jsme se vždy dohodli na dalších postupech. Dále zde pomalu vyplouvalo na povrch požadované rozhraní API. Toto API jsem dále konzultoval s Iaroslavem Kolodkou, aby jeho BE rozhraní plnilo potřebu mojí Android aplikace.

Schůzky s klientem probíhaly přibližně jednou měsíčně. Obvykle jsme zde probrali dosavadní postup práce a její budoucí vývoj. Dále jsme zde dělali revizi požadavků, jestli aktuální aplikace odpovídá jeho představám.

1.4 Popis problematiky rozvedených rodin

Zde si zjednodušeně popíšeme problematiku, kterou by aplikace měla řešit, jak takový rozchod vypadá a co ho doprovází. Zde bych rád podotkl, že žádnou osobní zkušenost s tímto tématem nemám a veškeré informace jsou předány od Bc. Lukáše Talpy a Mgr. Bohuslavy Janků na pravidelných konzultacích v rámci předmětů BI-SP1 a BI-SP2.1. [1]

Rozvod je nepříjemná součást života, kdy se z milované osoby stane nepřítel. Bývalí partneři spolu nekomunikují a nejsou schopni se pořádně domluvit na péči o děti. V daném sporu potom nejvíce trpí právě děti.

V rámci soudního řízení a nebo po domluvě rodičů se sestavuje rozvrh, kdy který rodič bude pečovat o děti. Toto je jeden z nejčastějších problémů. Ani tak ne sestavení rozvrhu jako jeho dodržování. Stává se, že se rodiče nedomluví a dorazí na místo předání v jiných časových intervalech. Jeden z partnerů tedy musí čekat na druhého a to si samozřejmě nenechá líbit. A asi není potřeba rozvádět, kam toto chování vede. Nejhorší ovšem je, že toto všechno se děje za přítomnosti dětí.

Mimo péči o dítě jsou dalším velice častým zdrojem sporů peníze. Při rozvodu se standardně určí pečovatel, což je obvykle matka. Pečovateli musí druhá strana odvádět alimenty určené soudem. Druhá strana nemusí s uvedenou částkou souhlasit a vše možně se snaží peníze neposílat. Což pro matku bez finanční zajištěnosti může být fatální.

Popřípadě se druhá strana snaží alimenty za určité období vůbec neposlat, a nebo poslat nižší částku s argumentem, že dítěti koupila například nové boty. Tyto situace často končí u soudu. V takovém případě se hodí uchovávat účtenky a záznamy o koupi daného předmětu.

Konečně zde dochází i ke kupování přízně dítěte. Dítě má nějaké přání a bohatší z rodičů mu ho vždy vyplní. Tím si takzvaně koupí jeho lásku a ono ho začne mít radši než druhého rodiče.

1.5 Analýza

1.5.1 Původní návrh

Analýza a návrh aplikace vznikala primárně v předmětech BI-SP1 a BI-SP2.1. Její podstatná část vznikla v předmětu BI-SP1 a podíleli se na ní všichni členové skupiny. Bylo při ní myšleno primárně na děti a jejich ochranu.

Se zadavatelem bylo dohodnuto, že moje verze poslouží jako testovací prototyp funkcionalit a jejich provedení, a nebude použita v reálném prostředí. Bude se tedy jedna o testovací prototyp, který bude analyzován a testován uživateli. Z tohoto testování vznikem analýza, která bude implementována ve finální aplikaci určené pro uživatele a produkci.

Z problémů, které nastávají v rozvodech popsaných v 1.4 jsme vydefinovali základní funkce, kterými by aplikace měla disponovat. Zde je připomenu a rozvedu.

Pro začátek rozvedu základní požadavek, který je společný pro všechny aplikace. To je správu uživatelů. Ti se do aplikace budou moci buď zaregistrovat, a nebo použít nějaký již existující účet třetích stran, například Google účet. Uživatel si v aplikaci může měnit jméno, heslo a jeho profilový obrázek.

Dále uživatel bude moci založit několik rodin. Uživatel si nadefinuje rodinu, popřípadě rodiny, přímo v aplikaci. Bude přidávat další uživatele, kte-

rým přijdou pozvánky. Tito uživatelé v systému mohou, ale i nemusí, existovat. Rodina musí fungovat i přesto, že ostatní uživatelé v rodině pozvánky nepřijmou. V rodině bude existovat systém rolí, kdy každý člen rodiny bude mít svojí roli a tím se určí jeho práva v rodině.

S funkcemi, které jsou dostupné pouze po přihlášení do rodiny, začnu pečovatelskými dny. Aplikace musí umět uživateli připomínat, kdy má péči. Dítě by taktéž v aplikaci mělo vidět, u jakého rodiče je v určitý den. Nastavení mohou provádět pouze rodiče a jakoukoliv změnu musí schválit všichni rodiče v aktuální rodině.

Nastavení bude probíhat pomocí pravidel. Toto nastavení pečovatelských dnů by se mělo zobrazit v kalendáři aplikace. Kromě toho by zde měl mít uživatel možnost vytvářet vlastní události a taktéž si své nastavení exportovat do například Google kalendáře.

Abychom zabránili „kupování lásky“, popsané v 1.4, a také hádkám rodičů o tom, kdo vlastně určitou věc koupil, vznikla kniha potřeb. Zde by jakýkoliv člen rodiny mohl vytvořit potřebu a ostatní by se mohli k dané věci vyjádřit. Například by se jednalo o koupi nových bot pro dítě. Nebo samotné dítě by si přálo kytaru a rodiče by se zde dohodli, kdo kytaru koupí a za jakou částku. Dále by tato potřeba prošla schvalováním u obou rodičů. Potom bude vidět, že s danou věcí souhlasili oba rodiče a toto ulehčí pozdější dohadování partnerů o dané věci. Diskuze rodičů bude probíhat přes diskuzní vlákno, kam budou oba rodiče psát své názory na danou věc. Budou zde existovat dvě vlákna. Jedno z nich nebude pro dítě dostupné, aby nedocházelo k jeho zbytečnému stresování. Druhé vlákno bude dostupné pro celou rodinu.

Aplikace by měla taktéž pomoci uživateli se správou alimentů. Uživatel bude mít možnost nastavit den v měsíci, kdy má platit nebo mu mají přijít alimenty. Aplikace bude také kontrolovat, zde alimenty skutečně byly odeslány. Pokud odeslány nebyly, tak bude upozorňovat uživatele, že alimenty má poslat. Konečně zde bude existovat automatické validace příchozích alimentů napojená přímo na bankovní účet uživatele.

Dále by aplikace měla mít možnost uchovávat účtenky v rámci rodiny a tím dokazovat nákupy v rodině. K účtence půjde pořídit i foto daného předmětu a daňový doklad.

Navíc vše v aplikaci by se mělo ukládat, a to včetně změn všech nastavení. Tyto údaje by měly jít exportovat a uživatel by je dále mohl použít k dokazování v soudních procesech.

1.5.2 Zjednodušený návrh

Z důvodů časové náročnosti jsem byl nucen provést revizi analýzy a některé funkce zjednodušit. Klient s tímto řešením souhlasil.

V mojí verzi bude dostupná registrace a přihlášení do aplikace. Avšak nebude možné přihlášení pomocí účtů ostatní aplikací. Připojování za pomoci

jiných účtů by bez serverové části bylo velice náročné na implementaci. To byl hlavní důvod jeho vyřazení z mého návrhu.

Ohledně správy rodin jsem se rozhodl implementovat pouze základní role Dítě a Rodič. Dále nebude zde žádný systém oprávnění. Takže všichni členové budou mít přístup do všech částí aplikace v rámci rodiny. Jediné místo s omezeným přístupem na základě role bude rodičovské chat v Knize potřeb.

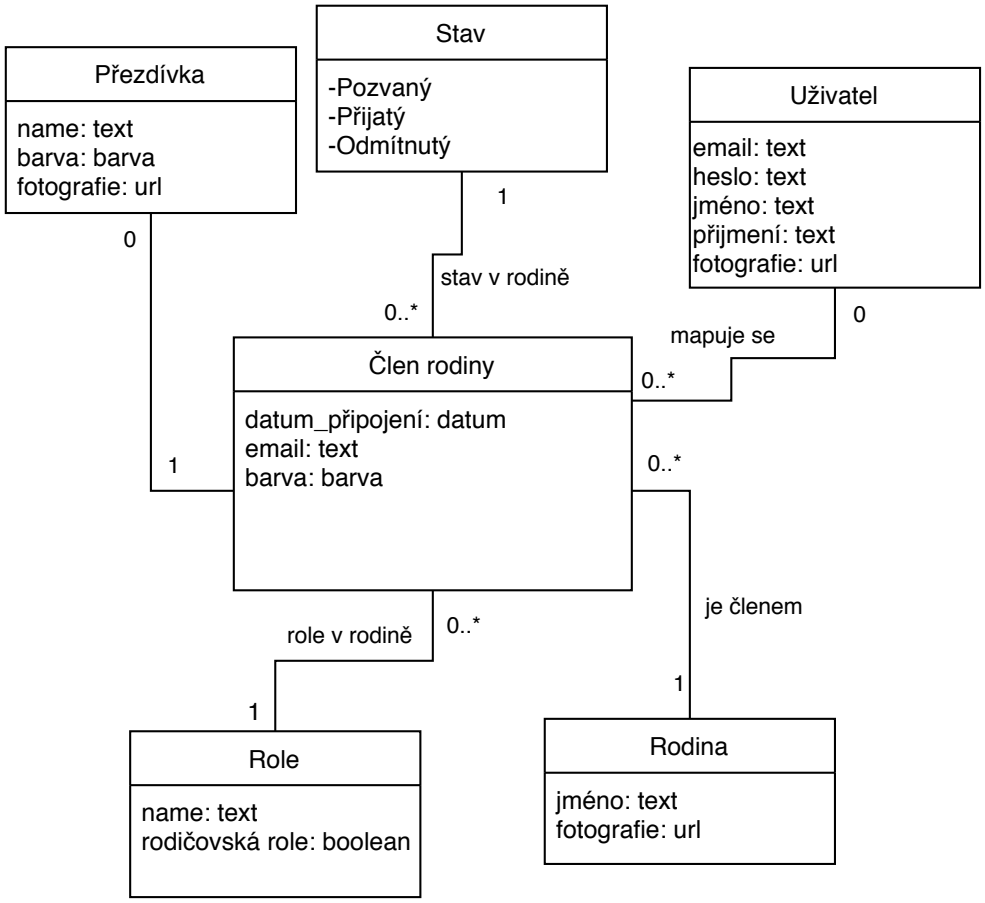
V pečovatelských dnech u nastavení dlouhodobých pravidel jsem taktéž provedl zjednodušení. Prvotní analýza toto nastavení počítala i s výjimkami na svátky a speciální pravidla pro různé dny v roce. Toto jsem později zjednodušil na nastavení, kdy si uživatele vybere počet týdnů a ke každému dnu nastaví pečovatele. Toto nastavení se potom periodicky opakuje. Kromě dlouhodobých pravidel půjde nastavit i jednorázovou výjimku na určitý interval.

Dále integraci s Google kalendářem taktéž nebude řešit. Toto řešení by bez serverové části bylo opět velice pracné.

Kniha potřeb jako taková nepotřebovala žádné zjednodušení. Avšak během konzultací s klientem byl vydefinován diagram průchodu a schvalování potřeby. Při implementaci jsem však narazil na chybná místa v tomto diagramu, o kterých jsem při jeho tvorbě neuvažoval, a proto jsem od jeho implementace nakonec upustil.

Automatickou validaci příchozích alimentů proti bankovnímu účtu jsem v mojí prototypové aplikaci neimplementoval. Moje zjednodušená forma počítá s tím, že uživatel příchozí a odchozí alimenty zadá v aplikaci manuálně.

Exportování z aplikace pro pozdější dokazování u soudů jsem se rozhodl taktéž vynechat. Pro prvotní testovací verzi, kterou moje aplikace má představovat, by byla tato funkce zbytečná. Navíc si osobně myslím, že tato funkce by měla být dodělána později, až bude jasné, jaká data může aplikace poskytnou, a tudíž co by tento export měl obsahovat.



Obrázek 1.1: Doménový model pro správu rodiny

Kapitola 2

Průběh vytváření Android aplikací

V této kapitole si detailně rozebereme, jak probíhal samotný implementační proces všech Android aplikací. Prvně si představíme Android SDK. Android SDK je soubor knihoven, které jsou vytvořeny pro vývoj Android aplikací. Popíšu zde možné architektury Android aplikací a možné postupy, jak tyto aplikace vytvářet. Dále si zde představíme strukturu android projektů a nejdůležitější soubory této struktury. Nakonec si představíme programovací jazyk a vývojové prostředí. Popíšu mnou použitý jazyk a IDE a popřípadě jejich alternativy.

2.1 Android SDK

Android SDK (software development kit) je soubor knihoven pro vývoj Android aplikací. Zde si představíme komponenty, které obsahuje, respektive komponenty, které jsem zvažoval pro použití v aplikaci a jejich alternativy. Samotné Android SDK je natolik rozsáhlé, a tak zde popíši jen nepatrnou část.[2]

Nejprve zde uvedu základní skupiny knihoven nutných pro vývoj. Představíme si mnou použité a jejich alternativy. V další části si představíme nejdůležitější komponenty těchto knihoven.

2.1.1 Soubory knihoven

2.1.1.1 Android platform

Android platform je základní soubor knihoven každého Android projektu. Obsahuje základní komponenty jako Activity, Context, logování, přístup k systémovému času atd. Je nezbytnou součásti každé Android aplikace. Právě

přes tuto knihovnu dochází ke komunikaci s HW jako s bluetooth, Wi-fi, display atd. [3]

2.1.1.2 Android support library

Android support library je soubor knihoven, který obaluje komponenty rozšiřující základní funkcionalitu z Android Platform, například práce s grafickým rozhraním aplikace. Dále zde nalezneme rozhraní pro práci na pozadí v jiném vláknu.

Tento soubor knihoven v Androidu existuje prakticky od začátku Androidu. To se na něm projevilo spíše negativně. Díky obrovským změnám, kterým Android procházel, se tento soubor stal velice nepřehledným. Často lze zde najít funkce, které řeší stejný problém, akorát pro různé verze Androidu. Tato vlastnost způsobila, že časem došlo k nahrazení Android support library pomocí AndroiX. [4]

2.1.1.3 AndroiX

Androix je soubor knihoven, jehož primárním účelem je spojení knihoven z Android support library do přehledné struktury. Avšak obsahuje i nové komponenty. Dovolím si vypíchnout dvě. Jsou jimi LiveData a ViewModel, které společně tvoří páteř prakticky všech nových Android aplikací.

Dále obsahuje komponenty pro práci na pozadí. Ty jsou přizpůsobeny požadavkům nových Android aplikací, které mají velice striktní omezení, které se týče práce na pozadí jednotlivých aplikací.

Díky těmto vlastnostem je AndroiX ideální volbou v dnešní době, a proto jsem se i já rozhodl jej použít. [5]

2.1.1.4 Architecture Components

Architecture Components je soubor, který obstarává ORM v Android aplikacích. Stará se o rozhraní mezi objekty a jejich perzistentním ukládáním do databáze. Jedná se o takzvanou Room Database. Room Database neužívá hibernate ani žádný jemu podobný framework. Ale pomocí systému anotací vygeneruje kód, který by jinak programátor musel psát ručně. Kvůli využití starého rozhraní cursorů při generování kódu je zde velice špatná práce s vazbami mezi objekty. Na rozdíl od Hibernate se zde musíme na ostatní tabulky odkazovat pomocí ID.

Avšak to je jenom drobná vada na kráse a určitě je lepší použít tuto funkcionalitu než psát ručně SQL dotazy. Navíc je zde celkem dobře vyřešený systém migrací mezi verzemi databáze. [6]

2.1.2 Hlavní komponenty

2.1.2.1 Manifest

Manifest je konfigurační soubor každé aplikace. Aplikace zde definuje, jaká bude vyžadovat práva po uživateli a jaké HW komponenty potřebuje ke svému běhu. Dále definuje třídu Application, jejíž instance má být vytvořená při spuštění aplikace, a seznam activit, které aplikace používá. Dále jaká activita se pouští jako první. Může také definovat Servici a Content providery. Vlastně se jedná o takový přehled pro systém čím vším aplikace disponuje a co požaduje. [7]

2.1.2.2 Application A MultiDexApplication

Android aplikace jako takové nemají žádnou zaváděcí main funkci. Místo toho zde existují funkce, které jsou volané systémem a programátor jejich přetížením dostane možnost se o těchto událostech dozvědět. Základní komponenta každé aplikace je třída, která programátorovi oznámí, že uživatel spustil, popřípadě ukončil aplikaci. Zde je to třída Application.

Application se používá pro vytváření věcí, které chceme použít po celý běh aplikace. Inicializuje se zde databáze, logovací systém a knihovny, jenž aplikace používá.

Jak aplikace postupně roste, roste i její velikost. V jistém okamžiku se nám stane, že aplikace přestane fungovat. V takové případě jsme dosáhli limitu aplikace a je potřeba povolit MultiDex, a nebo místo Application použít MultiDexApplication. [8]

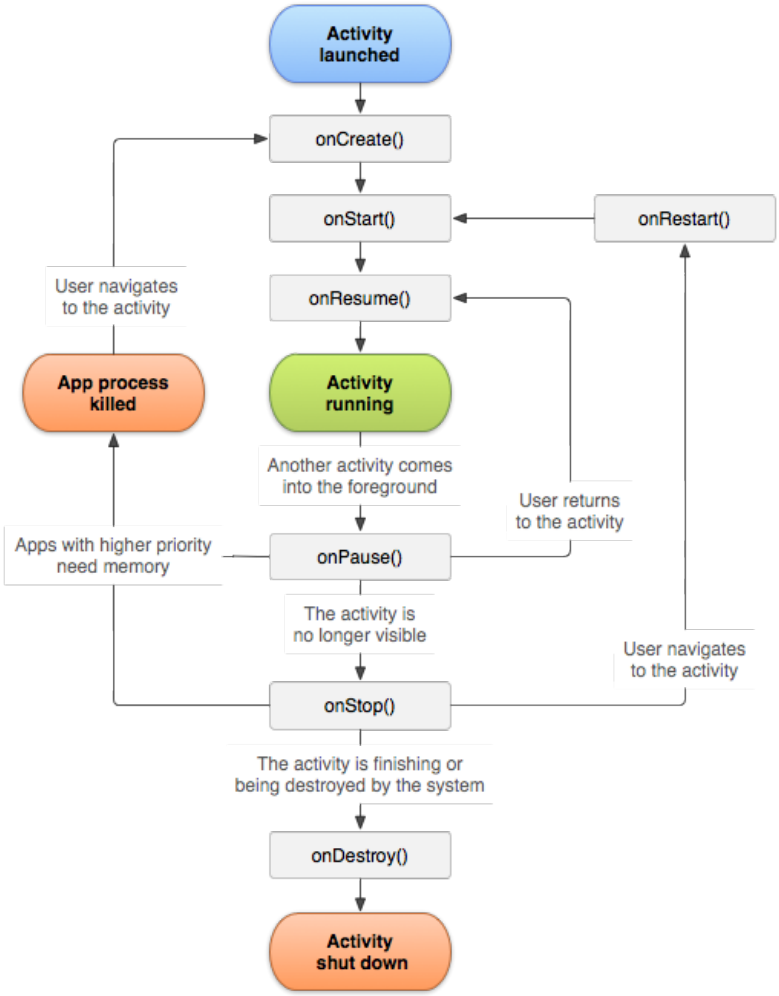
2.1.2.3 Activity

Další velice důležitou součásti je activita. Jedná se vlastně o samotnou obrazovku, která zabírá celý display mobilu. Activity mají napojení na systém a systém může za správných okolností spustit jakoukoliv z nich. Každá activita musí být deklarována v Manifestu aplikace. V základu při startu aplikace systém spouští activitu, která je nastavená v Manifestu aplikace jako výchozí.

Samotná activita má vlastní životní cyklus, který oznamuje, kdy uživatel spustí activitu, popřípadě ukončí. Dále pokud je aktivita zapnutá a uživatel z aplikace odejde nebo se do ní vrátí, čili detekci, jestli je aktuální aktivita zobrazena na obrazovce zařízení. Celý graf nalezneme na obrázku 2.1. [9]

2.1.2.4 Fragmenty

Velice důležitou část vývoje tvoří také fragmenty. Jejich struktura je podobná jako již u zmíněné activity. Avšak jejich použití je jiné. Vždy musí být součástí nějaké activity, ale jedna activita může obsahovat více různých fragmentů. Každý fragment potom obsahuje různé komponenty jako tlačítka, textová pole atd. Fragmenty se nemusí deklarovat v Manifestu aplikace.



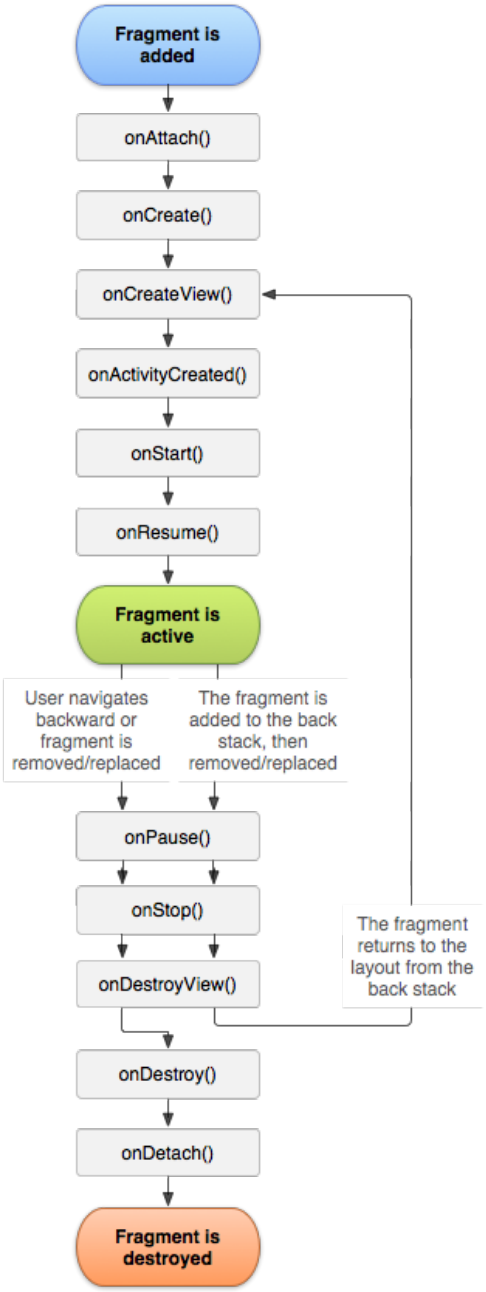
Obrázek 2.1: Životní cyklus activity [9]

Podobně jako activity i fragmenty mají vlastní životní cykly. Avšak jejich cyklus je složitější, což můžeme vidět na obrázku 2.2. [10]

2.1.2.5 Service

Service se používají v androidu pro dlouhotrvající operace na pozadí. Jedná se o komponentu, která je deklarovaná v manifestu a systém ji můžeme zapínat a vypínat dle potřeby. Na rozdíl od activity se nijak nezobrazuje uživateli a ten většinou nemá ani ponětí, že service v aplikaci existuje.

Dneska jsou postupně nahrazovány workery a joby, avšak pro dlouho trvající úlohy jsou stále service jedinou možnou volbou. Nicméně pro úlohy, které trvají v rámci sekund existují lepší možnosti. [11]



Obrázek 2.2: Životní cyklus fragmentu [10]

2.1.2.6 ViewPager

ViewPager je komponenta, která se zobrazuje v activitě popřípadě v Fragmentu. ViewPager je svázaný s PagerAdapter. PagerAdapter se dále dělí na FragmentPagerAdapter a FragmentStatePagerAdapter. Tyto adaptéry slouží jako datový sklad pro ViewPager. Starají se o inicializaci a udržení instancí fragmentů, které ViewPager zobrazuje. A tím se dostávám k tomu, co vlastně ViewPager dělá.

Jeho úkolem je zobrazit libovolný počet fragmentů vedle sebe. tak aby uživatel mezi nimi mohl přepínat pomocí gest. Tuto komponentu známe třeba z Galerie nebo čteček knih, kdy jednoduchým táhnutím zobrazíme další obrazovku ať už s novou fotkou, a nebo se stránkou textu. [12]

2.1.2.7 ViewModel a AndroidViewModel

ViewModel a AndroidViewModel vznikly jako mezivrstva pro jednotlivé fragmenty, respektive activity. Instance activity i fragmentu je zrušena při otočení obrazovky a programátor s tím musí počítat. ViewModel tento problém řeší za nás. Jeho instance přežije i otočení obrazovky, a proto je ideálním místem pro komunikaci s BE aplikace a uchovávání dat.

A pomocí LiveData, která podporují Observer pattern, předává data do activity nebo fragmentu. Díky tomu můžeme velmi pohodlně zobrazovat aktuální data.

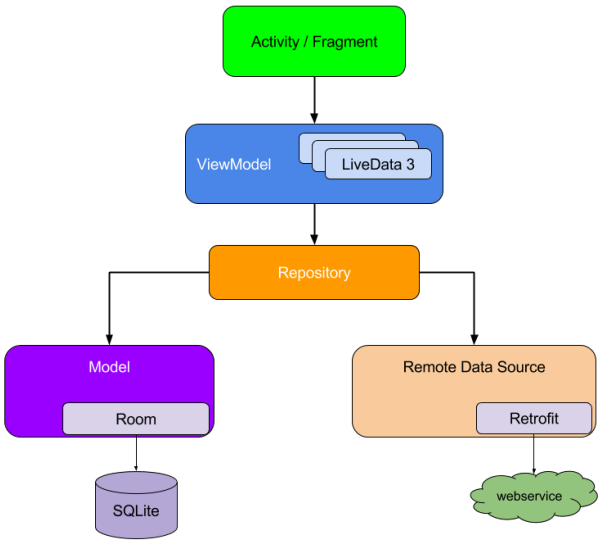
Rozdíl mezi ViewModel a AndroidViewModel je pouze v rozhraní konstruktoru. V AndroidViewModel se v konstruktoru předává odkaz na aplikaci, který si používá pro získání kontextu. ViewModel má prázdný konstruktor a nemá přístup ke kontextu aplikace. [13]

2.2 Architektura

Navrhnout architekturu u Android aplikací byl do nedávna velice obtížný úkol. Neexistoval žádný doporučený postup, jak jí navrhnout, natož aby v Android SDK byly nějaké podpůrné funkce pro architekturu. To se změnilo s příchodem ViewModel. Společně s tím vydal Google doporučený postup jak tvořit architekturu v aplikacích. [14]

Jak ukazuje obrázek 2.3 ,využívá se zde ViewModel, který slouží jako komunikační vrstva pro view. Zde se jedná o fragment nebo activitu. ViewModel nabídne view LiveData. View sleduje LiveData pomocí Observer patternu. Pokud z view přijde požadavek na načtení dat, ViewModel se zeptá repository na aktuální data. Ta mu je poskytne a díky observer patternu se data okamžitě propíšou do view. Repository zde funguje jako rozcestí, která rozhoduje, jestli data poskytne z lokální databáze, a nebo je stáhne ze vzdáleného serveru.

Jiné architektury nejsou ze strany SDK podporovány, a proto prakticky nemá smysl je zde zmiňovat. Navíc ze zkušeností z implementace musím říci, že



Obrázek 2.3: Architektura android aplikace [14]

tato architektura je vyhovující. Uživatel vidí data, která má mobil k dispozici bez zbytečných závislosti mezi komponentami. View zde neřeší, jestli jsou data k dispozici, jenom žádá a zobrazuje. Díky tomu nevznikají ohromné activity, která mají přes 1000 řádků, což byl velký problém v době, kdy žádný podobný doporučený postup neexistoval.

2.3 Programovací jazyky

Zde si představíme programovací jazyky, které lze využít pro programování Android aplikace. Samotné SDK je z velké části napsané v Javě a některé části v C++. Nicméně jeho rozhraní bylo převedeno do spousty programovacích jazyků. Zde představím ty nejznámější a nepoužívanější, ve kterých lze Android aplikaci vytvořit.

2.3.1 Java

Jedná se o jeden z nepoužívanější jazyků vůbec. [15] Jelikož SDK je taktéž napsané v tomto programovacím jazyce, tak v minulosti byla Java jedinou možnou volbou. Díky tomu existují stovky knihoven, které programátor může použít. Taktéž Java je velice jednoduchý jazyk, který programátora nezradí. [16]

Bohužel Java v poslední době pomalu upadá a je nahrazována novými jazyky, které jsou přehlednější a dle mého názoru na použití i příjemnější. [15] Avšak Java jako taková nejspíše nikdy nezmizí. Její ekosystém je ohromný

a existují stovky knihoven včetně samotného SDK, které jsou z velké části napsané právě v Javě. [16]

2.3.2 Kotlin

Kotlin je celkem nový programovací jazyk, který oproti Javě přináší velké množství inovací. Mezi hlavními je určitě podpora takzvané null save, čili kontrola proměnných, zdali může dojít k jejich nastavení na null hodnotu. Toto kontrolu provádí kompilátor, a proto při správném použití je chyba odhalena ještě při kompilaci. Dále Kotlin může běžet s Javou ve stejném projektu. Dokonce tyto kódy lze i kombinovat. Klidně můžeme mít jednu třídu v Javě a další v Kotlinu. Tyto třídy potom spolu mohou navzájem komunikovat, jakoby byly napsané ve stejném jazyce. Díky tomu může Kotlin využívat veškeré knihovny, kterými Java disponuje. [17]

Nakonec kotlin disponuje knihovnou pro asynchronní programování zvanou Coroutines. Toto asynchronní programování je podporováno přímo na úrovni samotného Kotlinu. [18]

Kotlin je taky doporučovaný společností Google, která se stará o Android SDK a v dnešní době už se bere jako primární programovací jazyk pro platformu Android. [19] S Kotlin mám již praktické zkušenosti z jiných projektů, ve kterých se ukázal jako vzorný nástupce Javy. Z těchto důvodu jsem taky v mojí práci zvolil jako programovací jazyk právě Kotlin.

2.3.3 JavaScript

JavaScript se primárně používá pro webové aplikace, avšak lze ho s určitými omezeními využít i pro naprogramování mobilních aplikací včetně Android aplikace a to díky frameworkům, kterými svět JavaScriptu disponuje, jako jsou Apache Cordova či React Native. Tito frameworky umožňují mít jeden projekt, ve kterém lze vytvořit multiplatformní aplikace. V praxi tedy máme jeden projekt, který vytvoří aplikaci pro IOS i Android zároveň. [20]

2.4 Vývojové prostředí

Volba správného vývojového prostředí je nedílnou součástí vytváření softwaru. Je potřeba vybrat vhodné IDE (Vývojové prostředí), které bude programátorovi vyhovovat. Zde si představíme IDE, které jsem zvažoval použít pro vývoj Android aplikace.

2.4.1 Android Studio

Android Studio je IDE vytvořené přímo pro vývoj Android aplikací. Nabízí rozsáhlou paletu nástrojů, které vývojáři pomohou se samotným vývojem, například kvalitní formátování kódu, návrhář uživatelského rozhraní, debugo-

vací a diagnostické nástroje pro ladění aplikací, android emulátory atd. IDE je zdarma ke stažení zde: `https://developer.android.com/studio`.

Toto IDE bylo také moje finální volba. Zvolil jsem ho, jelikož nabízí vše potřebné již v samotném základu a nic není potřeba složitě instalovat. Po celou dobu vývoje jsem s ním neměl jediný problém, který by mě nějak výrazně zbrzdil v práci na aplikaci.

2.4.2 IntelliJ IDEA

Další IDE, které je možno použít pro vývoj pro Android je IntelliJ IDEA: `https://www.jetbrains.com/idea/`. Samotné Android studio používá jako základ toto IDE. Proto podobnost mezi těmito dvěmi vývojovými prostředími je značná.

Avšak IDEA je specializovaná na větší záběr jazyků. Prakticky veškerý programovací jazyk, který lze spustit na JWM, lze napsat v tomto vývojovém prostředí. Avšak základní verze neobsahuje všechny nástroje pro vývoj na Android. Proto je dle mého názoru lepší použít Android Studio, které vše potřebné má už v samotném základu.

2.4.3 Eclipse

Eclipse je z uvedených IDE nejstarší. Používalo se v počátcích samotného vývoje pro Android. Jedná se o IDE primárně určené pro Javu, avšak díky komunitě je zde velké množství pluginů, které se dají využít.

Avšak pro dnešní potřeby je Eclipse zastaralé a Android Studio ho překonává ve všech směrech, minimálně co se do počtu funkcí pro podporu vývoje Android týká. Výběr tohoto nástroje již v dnešní době nedává smysl.

Kapitola 3

Tvorba aplikace

V této kapitole popíšu samotný implementační proces, můj postup verzování a vydávání nových verzí aplikace. Popíšu zde postupný vývoj aplikace od prvních obrazovek v BI-SP1 až po současnost. Rozepíšu zde mnou použité knihovny a moje úpravy těchto knihoven. Zvláště úpravy provedené v knihovně zobrazující kalendář.

Projekt android aplikace jsem založil 8. dubna 2019 v rámci BI-SP1. Již od počátku byla aplikace psaná v čistém Kotlinu. Jako šablonu projektu jsem použil předem připravené nastavení v Android Studiu. Hned po založení projektu byl projekt napojen na verzovací systém Git, kde jsem využil faktulní Gitlab jako server pro zálohování.

Po celou dobu práce jsem využíval větvení, které Git umožňuje. Větev Dev jsem používal pro samotný vývoj. Při vytvoření nové verze jsem udělal merge request do větve Master. Dále jsem si novou verzi v Mastru označil štítkem, který měl stejný název jako verze aplikace například v1.0.9.

Tyto nové verze jsem potom nahrával na Google Play. Zde si novou verzi mohli ostatní členové otevřít a zhodnotit změny, popřípadě mi dát zpětnou vazbu. Aplikace je nastavená na minimální verzi SDK 21. Target verze je 29. Application id je *cz.cvut.fit.sp.rozvody.android*. Aplikace lze stáhnout v obchodě Google Play na adrese `https://play.google.com/store/apps/details?id=cz.cvut.fit.sp.rozvody.android`. Momentálně je aplikace dostupná jenom v režimu interního testování. To znamená, že k ní mají přístup jenom konkrétní uživatelé.

3.1 Použité knihovny

V této sekci popíšu knihovny, které jsem v aplikaci použil, a mnou provede úpravy těchto knihoven. Budu zde zmiňovat hlavně nestandardní knihovny. Knihovny popsané v kapitole 2.1.1 jsem samozřejmě použil také. Ale jejich použití je nezbytné pro každou Android aplikaci. Zde se zaměřím na knihovny

specificky použité pro mojí aplikaci. Některé z těchto knihoven jsem musel lehce upravit, respektive rozšířit, aby splnily mnou požadovanou funkcionalitu.

3.1.1 Stetho

Jedná se o soubor diagnostických nástrojů. Vytváří most mezi zařízením a prohlížečem v připojeném počítači. Po vytvoření spojení je možné přes prohlížeč přistupovat k databázi aplikace, která tuto knihovnu používá. [21]

Díky tomuto živému spojení může vývojář sledovat dění v databázi. Dále může i měnit data a dokonce strukturu samotné databáze a to vše pomocí SQL příkazů. Rozhraní je velice spolehlivé a osvědčilo se mi v minulosti i na velkých datech, kde jiná podobná řešení selhala.

Samotná knihovna umí mnohem více funkcionalit, ale ty jsem ve svojí práci nevyužil. Použil jsem jí jenom na kontrolu a práci s databází aplikace během vývoje.

3.1.2 Knihovny pro práci s obrázky

Pro práci s obrázky jsem v mojí aplikaci využil dvě knihovny. Jedná se o Picasso a SimpleCropView. Obě knihovny jsem zvolil hlavně ze zkušenosti. Již jsem s nimi v minulosti pracoval a velice se mi osvědčily.

3.1.2.1 Picasso

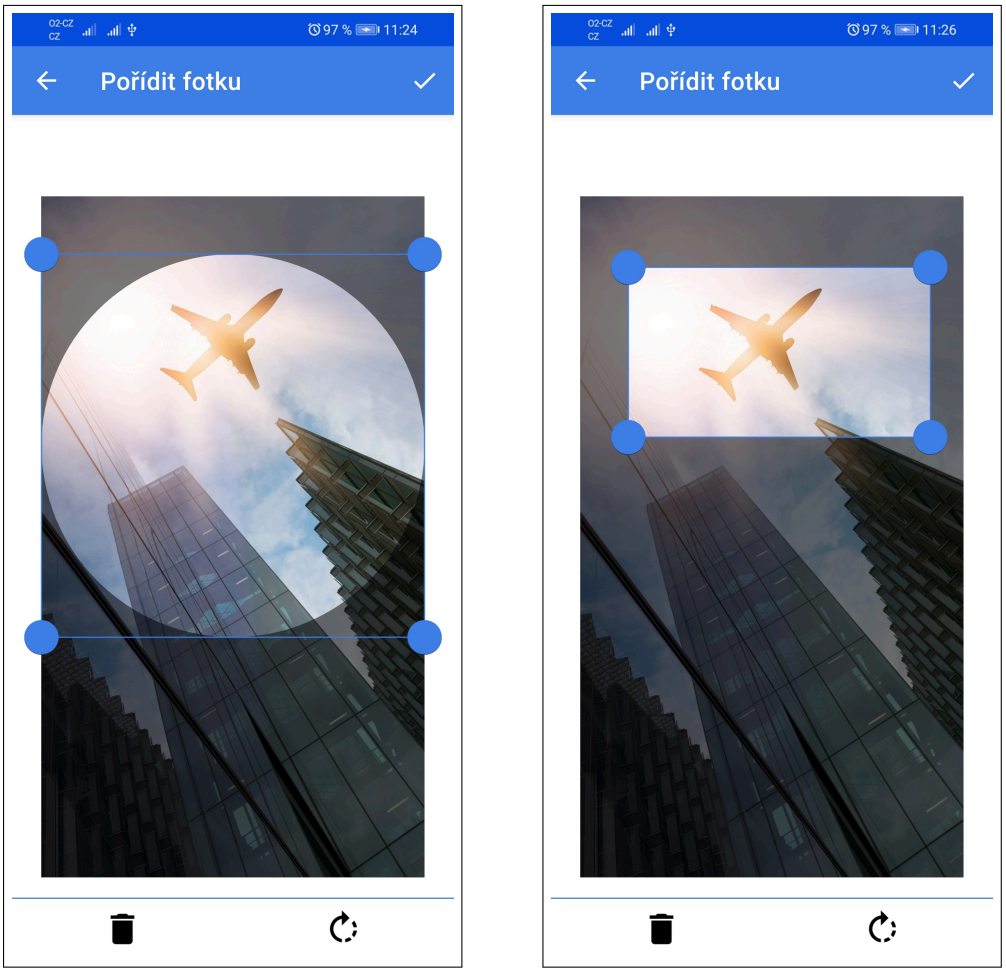
Díky limitované paměti může aplikace padnout na OutOfMemoryException. [22] Programátor s tím musí počítat a přizpůsobit práci s obrázky tomuto faktu. Tato knihovna řeší právě tento problém a mnoho dalších. Knihovna umí načíst obrázky z lokálního úložiště, popřípadě z internetu, pomocí URL adresy. Obrázky potom ukládá, aby další zobrazení bylo výrazně rychlejší. [23]

Knihovna je v aplikaci použitá pro načítání všech obrázků napříč aplikací. Prozatím aplikace pracuje jenom s lokálními obrázky. Pro budoucí použití bude potřeba knihovnu upravit a předávat jí pouze URL adresy obrázků uložených na serveru.

Alternativou této knihovny je knihovna Glide. S touto knihovnou jsem v době psaní práce neměl žádné zkušenosti, a proto jsem ji využil po Picasso. Avšak Glide je lepší varianta Picassa. Hlavní výhodou je práce s GIF formátem, kterou Picasso neumí. Další výhodou je také lepší práce s pamětí. [24]

3.1.2.2 SimpleCropView

Jak již název napovídá, knihovna umožňuje uřezávat obrázky. Konkrétně implmentuje komponentu pro uživatele, který díky ní může uříznout jím vybraný obrázek. Jedná se tedy o komponentu, která se vloží do activity. Její vstup je odkaz na soubor obrázků. Její výstup je potom uživatelem oříznutý obrázek.



Obrázek 3.1: Ořez obrázků v aplikaci pomocí SimpleCropView

Knihovna umí oříznou obrázek v několika tvarech. V aplikaci jsem použil kolečko a obdélník. Kolečko využívám pro fotky uživatelů a členů rodiny. Obdélník využívám na fotku rodiny. Ukázky vidíme na obrázcích 3.1. [25]

3.1.2.3 Android Week View

Jedná se o knihovnu, která umí zobrazit uživateli detailní přehled na jednotlivé týdny. Původní knihovna na Githubu je primárně určena pro zobrazení školního rozvrhu. [26].

Knihovna se dělí do dvou hlavní složek, do WeekView a WeekBackgroundView. WeekView se stará o zobrazení událostí a interakci s nimi. WeekBackgroundView kreslí pozadí. Jedná se o čáry a jednotlivé hodiny. WeekView dědí od RelativeLayout. Do layoutu je přídáno WeekBackgroundView, které potom vytváří pozadí samotné WeekView. Já jsem knihovnu musel ve své apli-

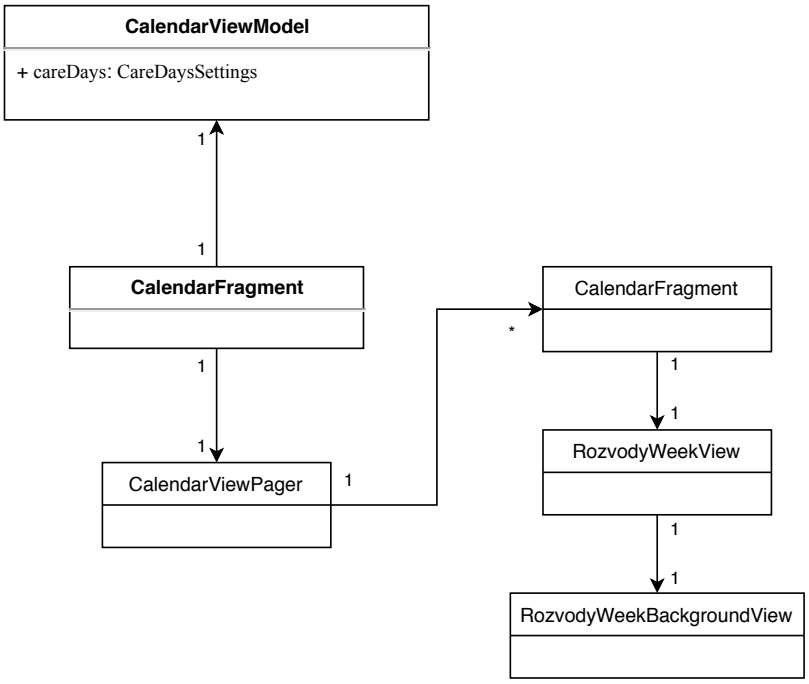
```
fun shouldDrawDayHighlight(
    column: Int,
    drawDayHighlight: (Int, Int) -> Unit
) {
    val date = weekStart.plusDays(column.toLong())
    val listener = weekBackgroundViewListener ?: return
    listener.getCareDayColor(date) { color ->
        drawDayHighlight(column, color)
    }
}

fun drawDayHighlight(
    canvas: Canvas, column: Int,
    highlightColor: Int
) {
    val left2: Int = getColumnStart(column, true)
    val right: Int = getColumnEnd(column, true)
    val rect = Rect(left2, 0, right, bottom)
    val paint = Paint().apply {
        strokeWidth = DIVIDER_WIDTH_PX.toFloat() * 2
        color = highlightColor
        alpha = 32
    }
    canvas.drawRect(rect, paint)
}
```

Výpis kódu 3.1: Vykreslení pečovatelských dnů

kaci rozšířit o některé mnou vyžadované funkce. Vytvořil jsem dvě třídy, RozvodyWeekView a RozvodyWeekBackgroundView.

RozvodyWeekBackgroundView bylo třeba naučit pracovat s daty. RozvodyWeekBackgroundView si tedy nově drží začátek týdne, který vykresluje. Konkrétně jeho datum, například pondělí 30. března 2020. Tato hodnota musí být vždy začátek týdne, čili pondělí. Další dny se potom dopočítají na základě indexu. Díky tomu umí komponenta zobrazit nejen den, ale i jeho datum. Dále bylo potřeba, aby uměla zobrazit barvu pečovatelských dnů. Toho jsem docílil úpravou kódu vykreslování jednotlivých dnů týdne, kdy se komponenta dotazuje WeekViewListener, jestli pro daný den existuje pečovatelský den a pokud ano, jakou má barvu. Algoritmus vidíme v ukázce kódu 3.1.

RozvodyWeekView funguje potom jako komunikátor, který nastavuje hodnoty v RozvodyWeekBackgroundView. Dále také zprostředkovává API pro funkce dostupné právě v RozvodyWeekBackgroundView. Celá komponenta se tedy pro zbytek aplikace prezentuje jako RozvodyWeekView a veškeré funkce



Obrázek 3.2: Diagram hierarchie komponent kalendáře

jsou dostupné právě přes API v RozvodyWeekView.

RozvodyWeekView je zaobalená fragmentem WeekViewFragment. Protože dále již pracuji jenom s tímto fragmentem, je možné použít ViewPager. ViewPager může obsahovat neomezené množství těchto fragmentů. Uživatel se potom může přesouvat mezi jednotlivými týdny kalendáře jednoduchými gesty táhnutí. Samotný ViewPager je potom umístěn v CalendarFragment. CalendarFragment je napojen na CalendarViewModel. Díky tomuto návrhu je možné zobrazit CalendarFragment v různých částech aplikace a postkytnou mu různé datové zdroje, které zobrazí WeekViewFragment. Ukázka hierarchie je na obrázků 3.2.

3.2 Implementace aplikace

Již od samotného počátku jsem se snažil o jednoduchý design aplikace. Jelikož se mělo jednat pouze o prototyp, tak jsem se rozhodl udržet v aplikaci jednoduchost na úrok designu. Proto v aplikaci nejsou žádné složité komponenty a obrazovky. V průběhu celé práce jsem počítal, že se aplikace bude ještě ra-

pidně měnit, a proto jsem se snažil veškeré věci centralizovat, aby nedocházelo ke zbytečným chybám v pozdějším vývoji.

Jelikož aplikace zatím není napojená na BE rozhraní, veškeré požadavky jsou zpracovány lokálně. Avšak rozhraní aplikace je již připraveno a práci BE simuluji takzvanými mocky. To jsou třídy, které mají za práci simulovat jednotlivé rozhraní BE serveru. Jejich volání je úmyslně zpomaleno, abych co nejlépe simuloval požadavky přes internet na rozhraní serveru.

Veškeré texty v aplikaci jsem se snažil umístit do centrálního souboru, aby aplikace byla připravená i pro budoucí překlady. Barvy jsou taktéž umístěny centrálně a je na ně odkazováno napříč aplikací, a to z důvodu možné budoucí grafické změny. To má za důvod, že budoucí změna grafiky nebude tak náročná na provedení.

3.2.1 Obrazovky před přihlášením

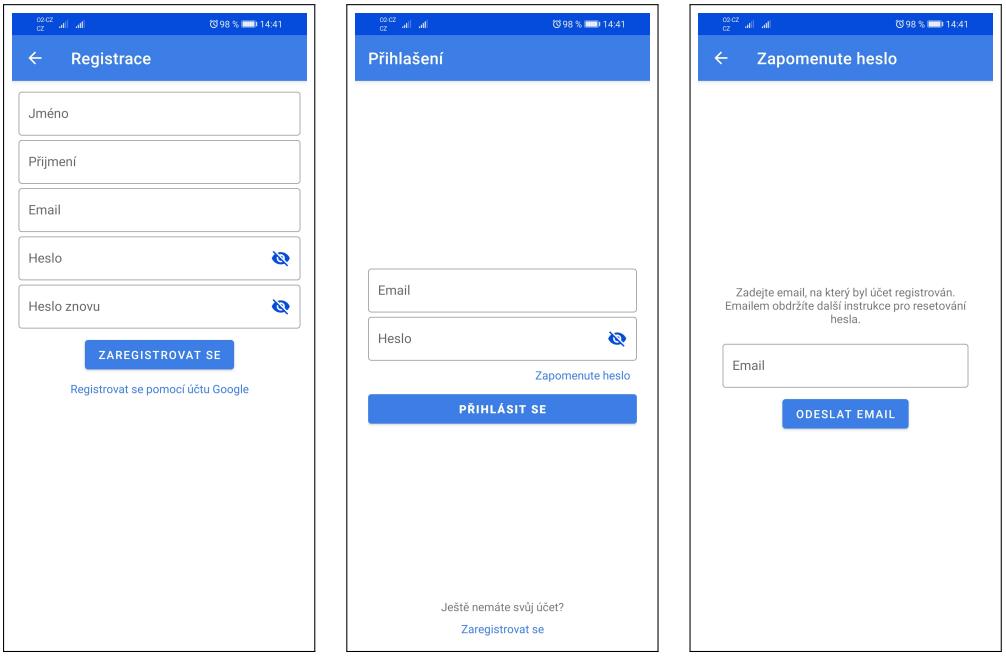
Před přihlášením má uživatel možnost dostat se ke třem obrazovkám. Jako první po spuštění aplikace se zobrazí obrazovka pro přihlášení. Jedná se o normální obrazovku, kde uživatel může vyplnit jméno a heslo a přihlásit se. Pokud heslo zapomněl, může překliknout na obrazovku zapomenuté heslo. Ta je momentálně nefunkční a je potřeba jí napojit na BE rozhraní. Nebo může kliknout na zaregistrovat se. Po vyplnění svých údajů se může zaregistrovat do aplikace. Tím vytvoří svůj účet v aplikaci, kterým se následně může přihlásit. Příklady vidíme na obrázku 3.3. Po přihlášení se zobrazí hlavní obrazovka. Ta se také bude zobrazovat po spuštění aplikace dokud se uživatel neodhlásí.

3.2.2 Hlavní obrazovka

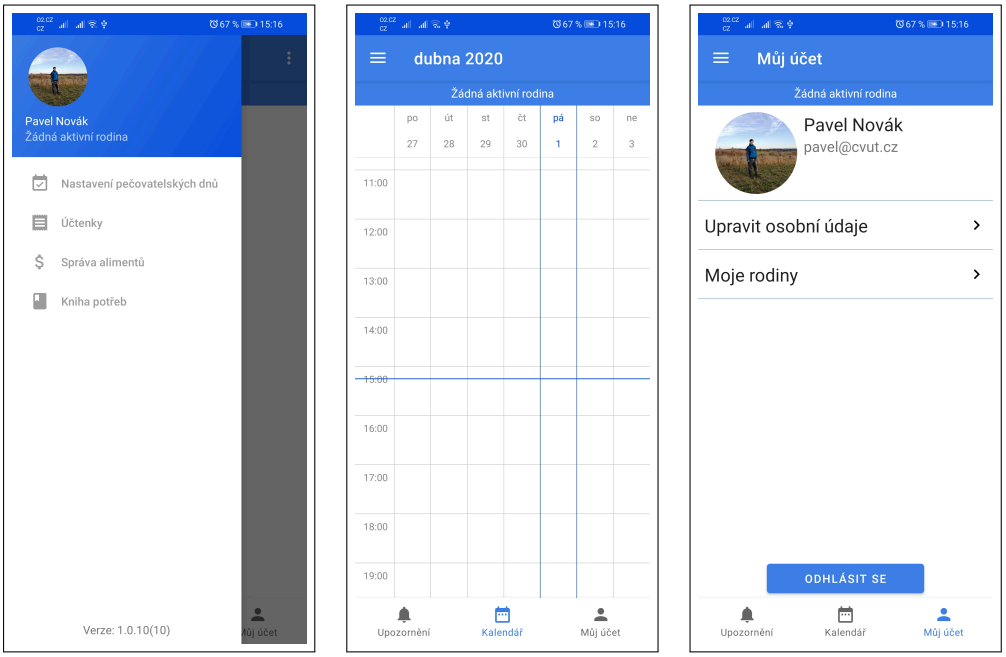
Hlavní obrazovka je centrum celé aplikaci. Pokud uživatel je již přihlášen, je to právě ta obrazovka, kterou uvidí po zapnutí aplikace. Najdeme zde tři hlavní obrazovky a menu, přes které se můžeme dostat do dalších částí aplikace.

Na této obrazovce uživatel vidí upozornění, kalendář a podrobnosti o jeho účtu. V upozorněních najde veškeré změny, které se udály v jeho rodině. Jsou zde znázorněny přečtené a nepřečtené novinky. Po kliknutí na novinku se dostane na místo změny v aplikaci. Dále je zde kalendář. Zobrazuje pečovatelské dny v aplikaci. Vidí zde i přehledy do minulosti a budoucnosti. Takže ví, kdo má péči například za dva týdny. Poslední obrazovka je pro informace o aktuálně přihlášeném uživateli. Může se přes ní dostat do podrobností o jeho profilu, nebo na přehled jeho rodin, popřípadě se může odhlásit.

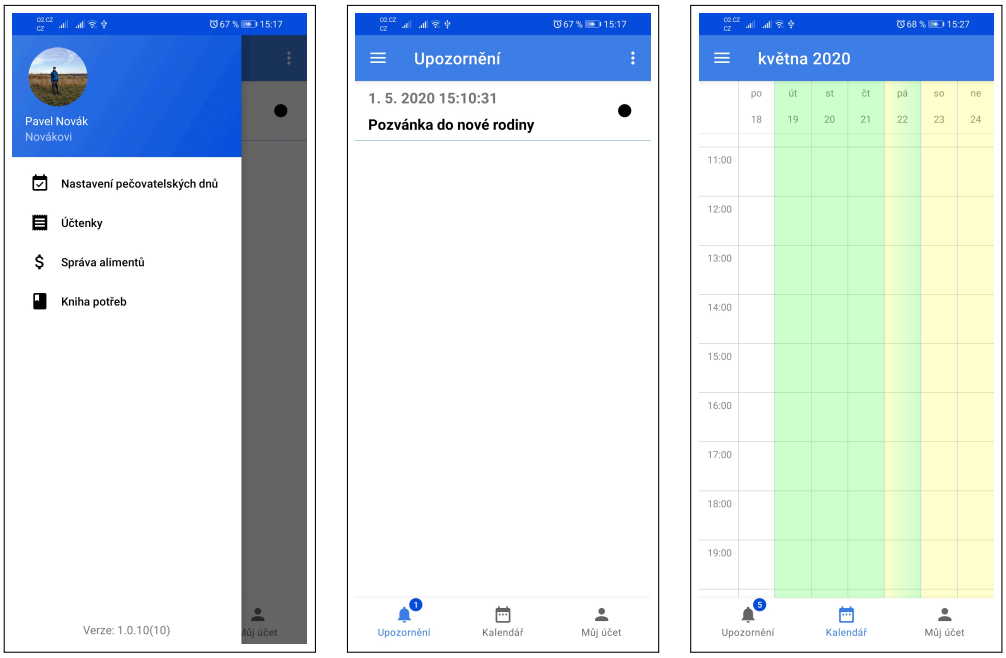
Pokud uživatel není připojen k žádné rodině, jsou některé funkce zablokované. Dále mu aplikace oznamuje nápisem „Žádná aktivní rodina“, že není přihlášen do žádné rodiny. Ukázku najdeme na obrázku 3.4. Pokud se uživatel přihlásil do rodiny, a nebo vytvoří novou rodinu, obrazovka se lehce změní a má možnost pokračovat do všech funkcí aplikace. Ukázka na obrázku 3.5.



Obrázek 3.3: Obrazovky před přihlášením do aplikace



Obrázek 3.4: Hlavní obrazovka, když uživatel není přihlášen do rodiny



Obrázek 3.5: Hlavní obrazovka po přihlášení uživatele do rodiny

3.2.3 Úprava osobních údajů

Na tuto obrazovku se uživatel dostane z hlavní obrazovky ze sekce Můj účet. Slouží ke změně jména a příjmení uživatele, dále ke změně hesla a obrázku uživatele.

3.2.4 Obrazovky pro správu rodiny

První obrazovka je „Moje rodiny“. Na této obrazovce uživatel vidí své dostupné rodiny. Může zde přepínat mezi jeho rodinami, popřípadě vytvořit novou rodinu.

Při vytváření nebo upravování rodiny může uživatel přidávat nebo upravovat stávající členy v rodině, jejich role atd. Ale jenom do okamžiku než přijmou pozvánku do rodiny. Po přijetí pozvánky je možné upravovat pouze barvu pečovatelských dnů. Po přijení člena již člena nelze smazat. Je třeba nutné udržet integritu rodiny a nejsou povolena mazání a změny členů rodiny, kteří již mohli s rodinou být nějak svázáni. Dále je zde také možné měnit obrázek upravované, respektive vytvářené rodiny.

Přidání člena do rodiny probíhá velice jednoduše. Pro pozvání nového člena je třeba vyplnit email, na který bude odeslána pozvánka do rodiny. Uživatel tohoto emailu může, ale nemusí být v aplikaci registrován. Dále se členu přiřadí jeho role a barva jeho pečovatelských dnů.

Po vytvoření rodiny si můžeme přepnout na detail rodiny. Zde vidíme

členy v dané rodině, jejich role a barvy jejich pečovatelských dnů. Dále zde taky přijímáme a odmítáme pozvánky do dané rodiny. Po kliknutí na člena rodiny se zobrazí obrazovka přizpůsobení daného člena.

Na obrazovce přizpůsobení si můžeme přizpůsobit člena. Tyto změny vidíme jenom my. Ostatní členové rodiny je nevidí. Můžeme měnit jméno, které uvidíme u dané člena napříč aplikací. Dále můžeme upravovat fotografie a nebo měnit barvu jeho pečovatelských dnů. Přehled obrazovek vidíme na obrázku 3.6.

3.2.5 Pečovatelské dny

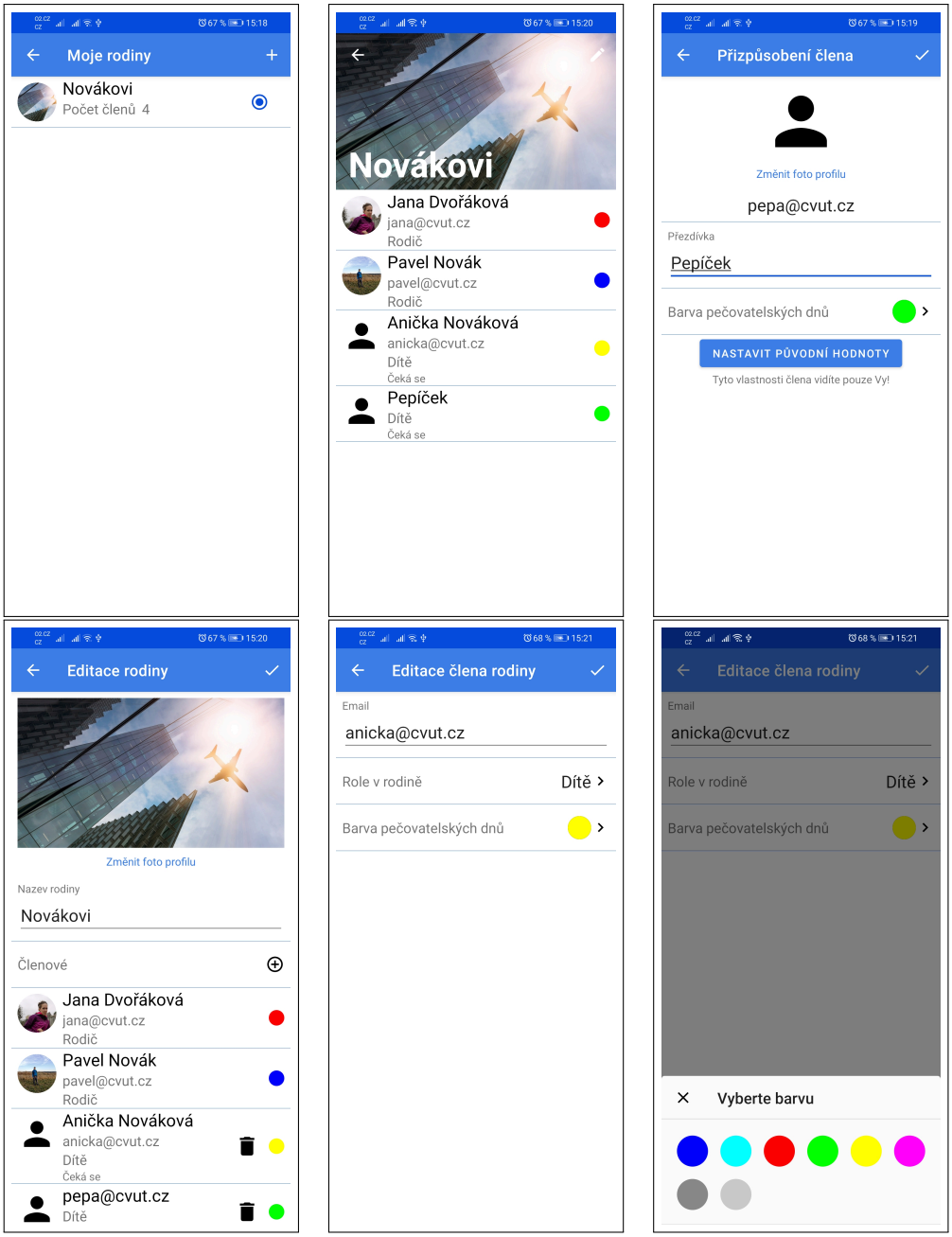
Asi nejdůležitější funkcionalita aplikace je správa pečovatelských dnů. Aplikace umí zobrazit přehledně v kalendáři, o jaké děti má přihlášený uživatel péči, pokud je přihlášený uživatel rodič. Dětem se zobrazuje rodič, který o ně bude pečovat. Toto nastavení péče se nastavuje na obrazovce pečovatelských dnů dostupné z hlavní obrazovky aplikace.

Na úvodním přehledu vidíme aktuální dlouhodobá pravidla. Ta se dělí na jednotná a podrobná. Jednotná nastavení jsou nastavení pravidel pro všechny děti. Toto nastavení je automaticky nastavené pro všechny děti v dané rodině. I pro později přidané děti se toto nastavení aplikuje. Při tomto nastavení uživatel specifikuje pouze pečovatele a dny, kdy pečovatel má v péči všechny děti v rodině. Podrobné nastavení specifikuje pečovatele jednotlivým dětem v rodině a neaplikuje se na nově přidané děti. Zde je potřeba pro každé dítě zvlášť nastavit pečovatele na dané dny v týdnu. Pokud si uživatel přeje upravit dlouhodobá pravidla je třeba kliknout na ikonku tužky.

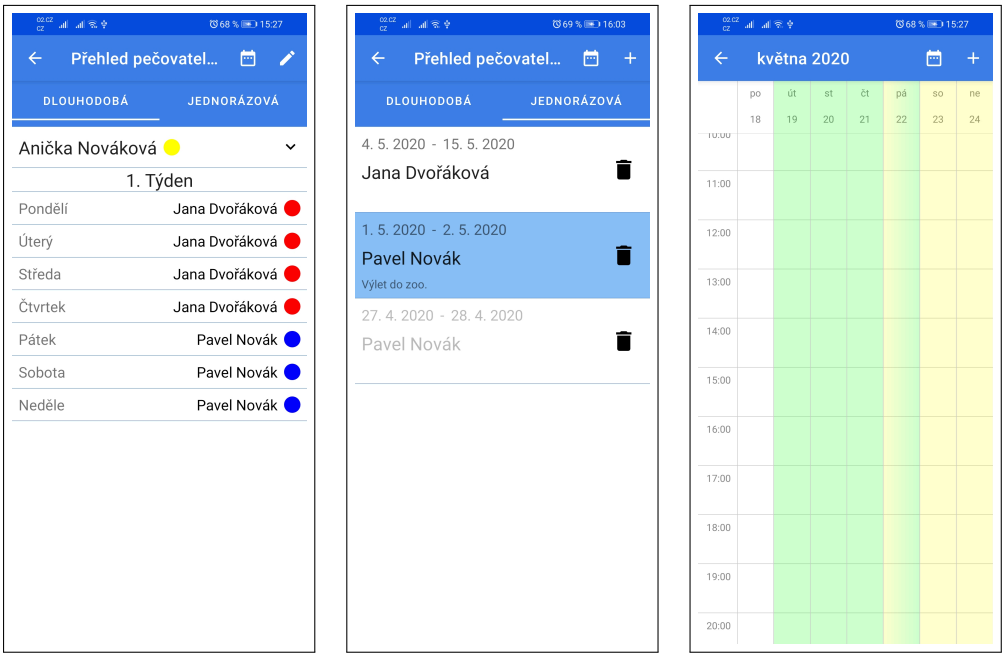
Při nastavení si uživatel vybere jestli chce jednotné a nebo podrobná pravidla. Pro jednotné vybere pečovatele pro dané dny. Tito pečovatelé mohou být pouze členové rodiny s rodičovskou rolí. A každý den v týdnu musí mít nastaveného svého pečovatele. U podrobných probíhá nastavení téměř stejně s jediným rozdílem, že si uživatel vybírá, pro jaké dítě v rodině nastavení definuje. Výběr dětí probíhá přes stejně dialogové okno jako nastavení pravidla, jak je vidět na obrázku 3.8. Pokud uživatel chce uprostřed editace přepnout nastavení pravidla, je upozorněn dialogem. Po přepnutí pravidla přijde o aktuální konfiguraci, kterou v průběhu úprav vytvořil.

Ve spodní části obrazovky nastavení je dialog, který po gestu tažením lze vytáhnout nahoru. Na tomto dialogu je startovní den, od kterého platí pravidla, a to jak dopředu tak dozadu. Dále si zde uživatel definuje, na kolik týdnů dopředu si přeje pravidla specifikovat v rozsahu 1-99 týdnů. Po uložení nastavení je uživatel vrácen zpět na obrazovku přehledu, kde již vidí aktuální nastavení.

Druhá položka na obrazovce přehledu pečovatelských dnů je výpis jednorázových nastavení pečovatelských dnů. Zde vidíme výjimky, které jsou nadřazené dlouhodobým pravidlům. Tyto výjimky mají platnost od-do. Dále je u nich potřeba specifikovat rodiče, který se v daném období stará, a dále děti, o



Obrázek 3.6: Obrazovky pro správu rodiny

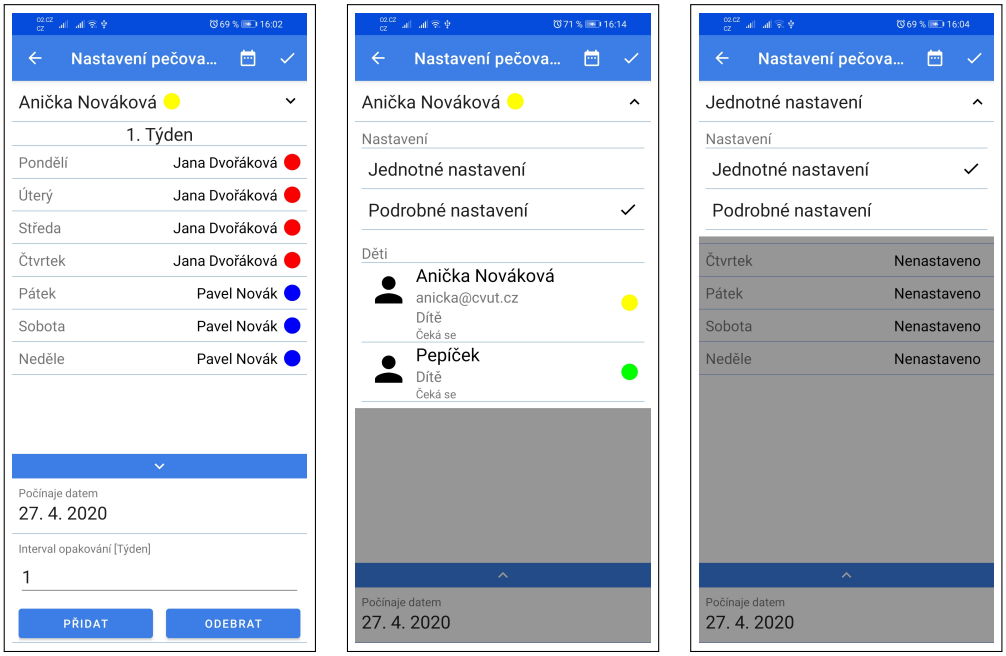


Obrázek 3.7: Obrazovky přehledu pečovatelských dnů

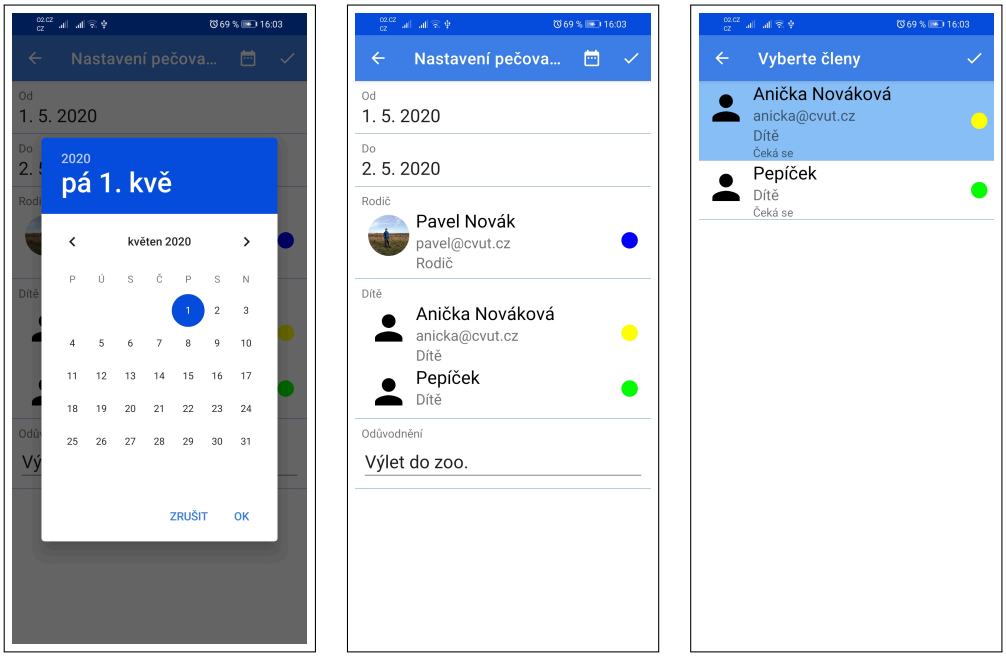
které se bude starat. Položka v tomto listu může mít různou barvu pozadí. Položky zašedlé jsou ty, které již proběhly. Podbarvené modrou barvou jsou aktuální a položky nijak nezvýrazněné teprve proběhnout. V seznamu jsou položky seřazené od nejstarších po nejnovější a kliknutí na ikonku Popelnice je lze smazat. Úprava probíhá kliknutí na položku a novou lze přidat kliknutím na ikonu plus v horní části aplikace. Po dohodě s klientem bylo zakázáno specifikovat jednorázová pravidla bez definice těch dlouhodobých.

Na obrazovce editace uživatel musí definovat platnost rodiče a dětí. Veškeré nastavení probíhá kliknutí na danou položku. Pro nastavení data se zobrazí dialog. Pro nastavení rodiče je uživatel přenesen na obrazovku, kde vidí všechny členy rodiny s rodičovskou rolí. Rodiče vybere kliknutím na položku v tomto listu. U dětí je postup podobný. Uživatel je přenesen na obrazovku, kde se mu zobrazí list dětí v rodině. Zde si může vybrat děti, o které bude pečováno. Dále uživatel může specifikovat důvod, proč v daném intervalu vytvořil jednorázové pravidlo.

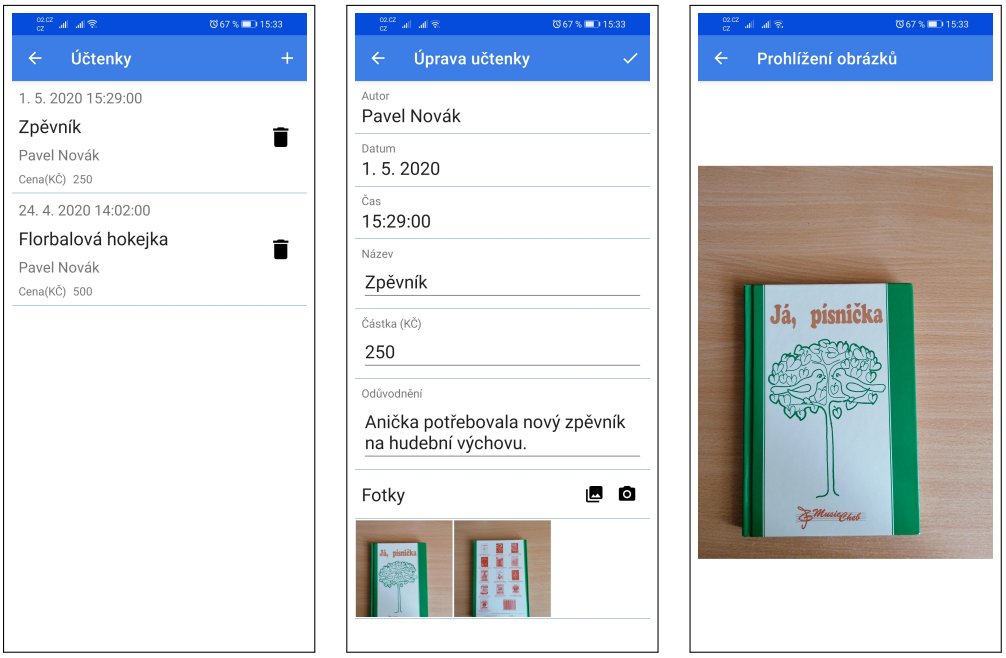
Uživatel si může překliknout do náhledu kalendáře. Zde vidí, jak jeho aktuální nastavení vypadá. Každý den je podbarven barvou člena rodiny, o kterého má pečovat, respektive který o něj pečuje. Rodič může samozřejmě pečovat i o více dětí. V takovém případě je zde průnik barev dětí. Pokud rodič o žádné dítě v daný den nepečuje, den nemá žádnou barvu.



Obrázek 3.8: Obrazovky nastavení dlouhodobých pečovatelských dnů



Obrázek 3.9: Obrazovky nastavení jednorázových pečovatelských dnů



Obrázek 3.10: Obrazovky pro správu účtenek

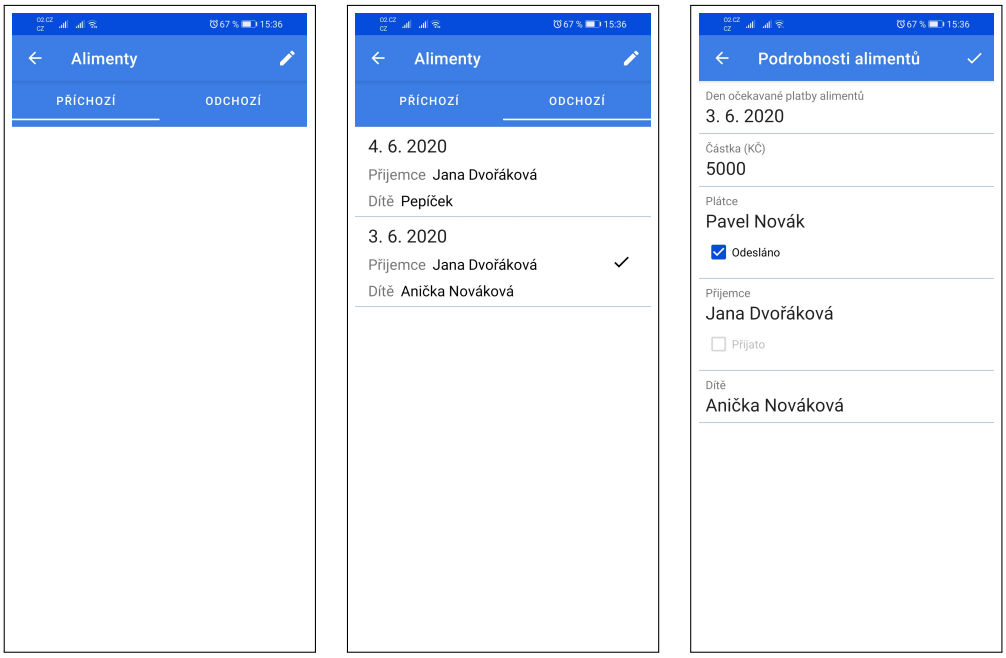
3.2.6 Správa účtenek

Aplikace podporuje uchovávání účtenek. Členové rodiny si zde mohou ukládat nákupy, které pro danou rodinu vykonali. Uživatel definuje datum a čas nákupu, cenu a důvod nákupu. Jako důkaz může připnout několik fotografií kupovaného objektu včetně účtenky. Fotografie si dále může procházet.

Každý člen vidí všechny nákupy v rodině. Avšak upravovat a mazat může pouze účtenky, které on sám založil. Ostatní může jenom prohlížet. Dále účtenka lze připnout k potřebě jako důkaz naplnění dané potřeby. Když je účtenka připojena k potřebě, nejde upravovat ani mazat.

3.2.7 Alimenty

Alimenty taktéž k rozvodům patří, proto i aplikace by si nimi měla pomoci. Proto zde existuje sekce alimenty. Lze se do ní dostat přes boční menu na hlavní obrazovce. První obrazovka této sekce je rozdělená na dvě části, část příchozí a odchozí. V příchozí části najde uživatel očekávané příchozí peníze za alimenty a datum do kdy by peníze měly přijít. Dále také od jakého člena rodiny (většinou otec) a jakému členu rodiny (většinou matce). Tato data se vždy generují měsíc dopředu a jsou seřazeny od nejaktuálnějších. Pokud uživatel klikne na aliment, vidí podrobnější informace. Zde taky může potvrdit, že mu daná platba přišla, respektive že odeslal peníze. Vidí také, jestli druhá strana již peníze odeslala, respektive přijala. Tento stav ovšem měnit



Obrázek 3.11: Obrazovky přehledu alimentů

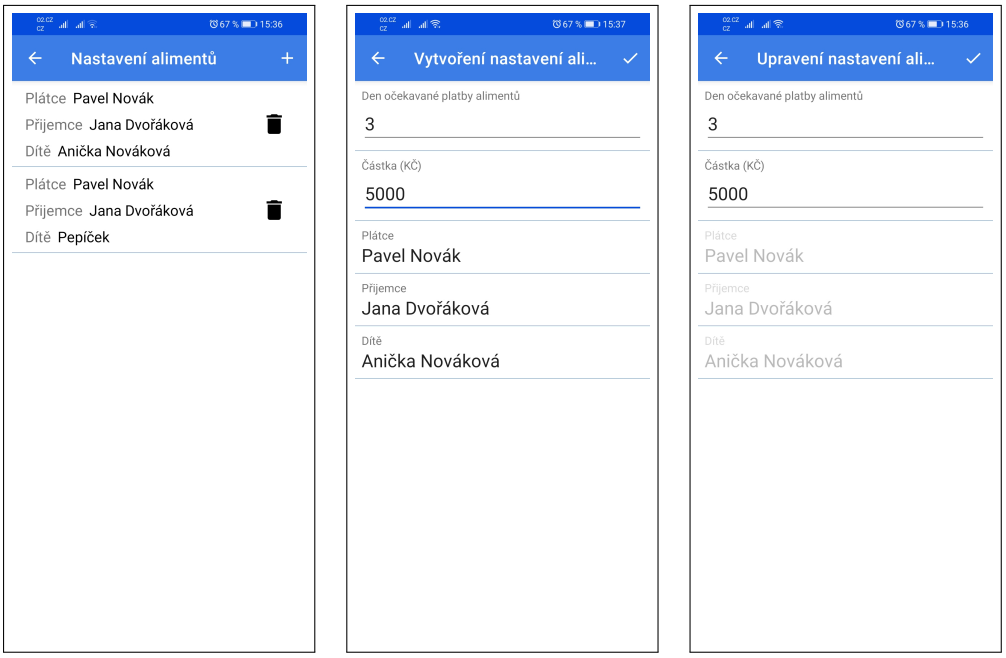
nemůže a políčko pro změnu je zašedlé. Viz obrázek 3.11. Uživatel může splnit vytvořený aliment. V přehledu potom vidí splněné alimenty odlišeně od nesplněných.

Aby systém generoval alimenty, je potřeba vytvořit nastavení jejich generování. Uživatel do něj vstoupí kliknutí na ikonku tužky na obrazovce alimentů. Na této obrazovce vidí aktuální nastavení alimentů pro jeho aktivní rodinu. Nastavení může vytvářet, upravovat a mazat. Při vytváření definuje den v měsíci, kdy očekává příchod peněz. Rozsah je od 1 do 28. Dále zadává částku, plátce, příjemce a dítě. Plátce a příjemce mohou být pouze osoby v rodině s rodičovskou rolí. Děti mohou být pouze osoby, co nemají rodičovskou roli.

Po vytvoření nastavení již nelze měnit plátce, příjemce ani dítě. Položky se stanou neaktivní a není je možné měnit, jak vidíme na třetí obrazovce obrázku 3.12. Uživatel také může mazat nastavení alimentů. Jakákoliv změna se ovšem promítne pouze do generování budoucích alimentů. Již vygenerované nejsou smazány a jsou jim ponechány staré hodnoty.

3.2.8 Kniha potřeb

Kniha potřeb má spravovat a uchovávat potřeby dětí. Dítě nebo jeden z rodičů zde nastaví určitou potřebu, kterou dítě potřebuje. Na obrazovku se lze dostat z hlavní obrazovky. Zde uživatel vidí přehled potřeb v aktuální rodině a jejich stav.



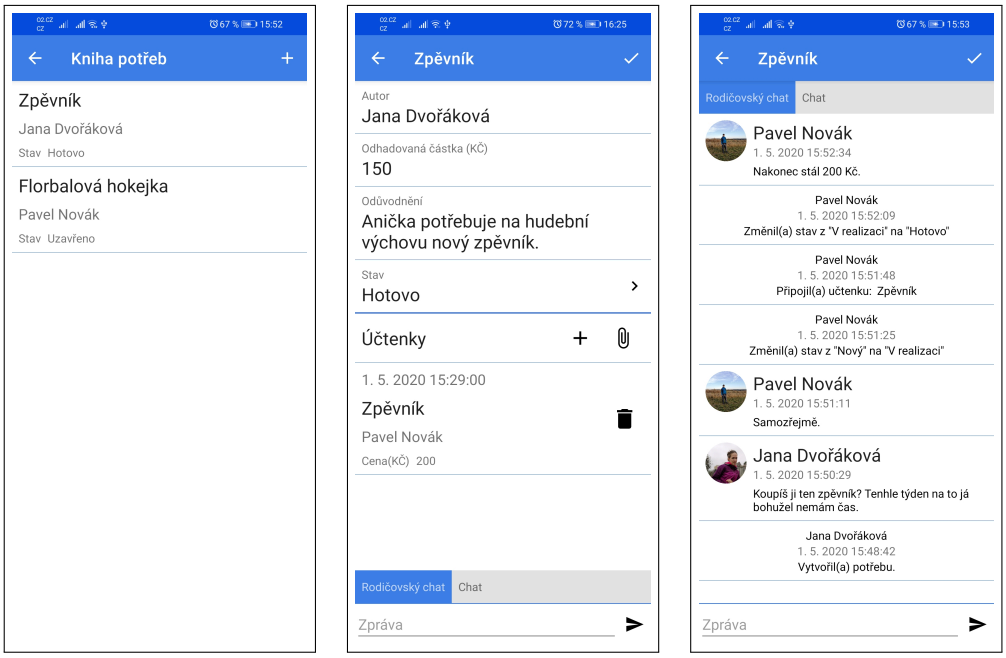
Obrázek 3.12: Obrazovky nastavení alimentů

Kliknutím na plus vytvoří novou potřebu. Nadefinuje název, například nové boty. Může vyplnit odhadovanou částku a důvod, proč dítě nové boty potřebuje. Po potvrzení aplikace přenese uživatele do detailu potřeby.

V detailu vidí dříve zadané údaje. Ty se již nedají měnit. Dále zde nalezne stav potřeby a účtenky, které jsou připnuté k potřebě jako důkaz jejího splnění. Pro připnutí lze kliknout na ikonu sponky pro výběr již evidované účtenky. Pro vytvoření je třeba kliknout na plus. Potom aplikace přenese uživatele na obrazovku vytváření účtenky. Po jejím vytvoření dojde k jejímu automatickému připojení a zanesení do seznamu účtenek.

U potřeby lze měnit stavy potřeby. To má indikovat, v jakém stavu se dané potřeba nachází. Jestli už jí někdo vykonává a nebo je teprve návrh nějakého řešení. Momentální dostupné stavy jsou: nový, schválený, odmítnuto, v realizace, hotovo a uzavřeno. Mezi stavy lze libovolně přepínat, avšak v budoucích verzích aplikace bude přesně definován postup průchodu.

Se spodní části se nachází chat, kde se členové rodiny mohou bavit o dané potřebě. Chat je rozdělen na dvě části, a to na dětskou a rodičovskou. V dětském chatu mohou psát všichni členové a všichni vidí zprávy všech. Rodičovský chat nevidí děti. Dále v rodičovském chatu jsou vidět i změny, které za životnosti potřeby proběhly jako připojení účtenek, změna stavů potřeby atd. U těchto změn je vždy napsán čas, autor změny a z jakého stavu do jakého stavu potřeby změnil.



Obrázek 3.13: Obrazovky knihy potřeb

3.2.9 Backend aplikace

Backend aplikace je implementování pomocí architektury popsané v 2.2. Nebudu zde popisovat fungování jednotlivých modulů backendu, spíše zde popíšu princip a porovnám, kde se od architektury moje implementace vzdaluje a kde jí dodržuje.

Každá obrazovka v aplikace má vlastní ViewModel. Tento ViewModel obstará pro obrazovku potřebná data. Data získá z Services. Service je komponenta ekvivalentní Repository, která je popsaná na obrázku 2.3. Service nabízí data a ViewModely si tyto data stahují. Data buď stáhnou ze serveru, a nebo si je načtou z lokální databáze. Servici nabízí skutečně surová data. ViewModel získaná data musí zpracovat podle potřeby dané obrazovky, aby na obrazovce již nedocházelo k žádnému zpracování dat a obrazovka data pouze zobrazovala.

Hlavní úlohou Servici je poskytnout aktuální data k zobrazení. Nejlépe aktuální data přímo ze serveru, popřípadě z lokální databáze v případě nedostupného serveru. Všechny Servici májí po celou dobu aplikace pouze jednotu instanci, na kterou se dotazují všechny ViewModely. Každá service má na starost určitou část domény. Například UserService nabízí všechny dostupné uživatele. UserFamily nabízí všechny rodiny. Servica se tedy rozhoduje, z jakého zdroje data poskytne. Data ze serveru ukládá do lokální databáze.

Do databáze přistupuje přes Repository. Na obrázku 2.3 tuto komponentu nalezneme jako Model. Repository je rozhraní mezi aplikací a lokální databází.

Zde využívám knihovnu Room Database popsanou v 2.1.1.4. Opět platí že každá Repository řeší určitou část domény a jedná se o vztah 1:1 k Service. Čili k UserService vždy existuje UserRepository. Programátor se nestará přímo o SQL dotazy. To za něj řeší Room Database. Jeho jediným úkolem je popsat potřebné rozhraní. Příklad najdeme ve výpisu kódu 3.2.

Již jsme si popsali, jak komunikuje Service s lokální databází. Nyní popíšu komunikaci se serverem. Pro tento účel jsou v aplikaci vytvořené dvě třídy, WpService a WpAdapter. WpService je třída, která zařizuje transformaci datových tříd uvnitř aplikace na DTO objekty (Data transfer object). Dále odchytává výjimky při komunikaci se serverem a ty transformuje.

Veškeré volání je poté směrováno na WpAdapter. Ten už pracuje pouze s DTO objekty a jeho úkolem je směřovat volání na jednotlivé endpointy serveru. Dále se také bude starat o autorizaci požadavků na server. WpAdapter bude směrovat požadavky na vstupní body server. Avšak momentálně je směřuje na mocky, které simulují práci serveru.

Mocky jsou obyčejné třídy, které uchovávají list objektů podobně jako tabulka v databázi. Dále při každé změně listu dochází k jeho uložení do souboru pomocí ObjectOutputStream. Tento stream je schopný uložit a načíst instance objektů ze souboru. Opět platí, že každý mock se zabývá specifickou částí domény. Všechny mocky dědí od WpMockUp, který se stará o ukládání dat a jejich načítání. Postupem vývoje se objevila i potřeba toto rozhraní verzovat, aby v případě změny DTO tříd nedocházelo k chybám, kdy se aplikace pokoušela načíst soubor se instancí staré třídy. Tento problém byl vyřešen přidáním verze do názvu souboru. Celkové ukládání instance vidíme ve výpisu kódu 3.3.

Jelikož se jedná o blokující volání, tak veškerá komunikace na backendu aplikace musí probíhat mimo hlavní vlákno aplikace. Jinak by docházelo k sekání aplikace. K tomu je využit framework Coroutines 2.3.2. Jelikož volání je asynchronní, je třeba ViewModelu oznámit změnu dat. K tomuto účelu využívám LiveData popsaná v 2.1.2.7. Díky tomuto návrhu má uživatel vždy aktuální data a to i napříč všemi fragmenty, což by jinak byl vcelku problém.

3.2.10 Api rozhraní

Serverové rozhraní, na které by se aplikace měla napojit, vytváří Iaroslav Kolodka ve své bakalářské práci. Toto rozhraní bylo během vývoje s panem Kolodkou neustále konzultováno a upravováno.

Jako základ rozhraní posloužil class diagram z předmětu BI-SP1 vytvořený Iaroslavem Kolodkou a Maximem Shchukinem. Dále také přímo návrh REST API rozhraní od Richarda Schäfera, taktéž vytvořený v BI-SP1.

Díky těmto návrhům se mnou vyžadované rozhraní moc neliší od rozhraní poskytnuté serverem pana Kolodky. Server poskytuje REST rozhraní a ke komunikaci využívá formát JSON. K autorizace je využíván OAuth 2.0. Server podporuje veškeré aplikací vyžadované enpointy.

Ovšem data, která server poskytuje na těchto endpointech, se mírně liší. Například data dostupné pro potřeby a pečovatelské dny se liší od aplikací požadované struktury.

Dalším problémem bylo posílání obrázků. V prvotní verzi posílal server obrázky přímo v DTO objektech jako pole bytů. Toto později upravil na URL odkaz. URL odkaz je pro Android klienta lepší. Díky tomuto řešení lze použít knihovnu, která obrázek stáhne a načte. Mnou použitá knihovna Picasso popsaná v 3.1.2.1 tuto funkcionalitu taktéž podporuje. Pokud by server posílal obrázky v DTO objektech, tuto knihovnu by nešlo použít a bylo by třeba vytvořit vlastní řešení této problematiky.

```
@Dao
interface UserRepository {

    @Insert
    fun insert(vararg items: User)

    @Update
    fun update(vararg items: User)

    @Delete
    fun delete(item: User)

    @Query("SELECT * FROM user")
    fun findAll(): List<User>

    @Query("SELECT * FROM user WHERE id == :userId")
    fun findById(userId: Long): User?

    @Query("SELECT * FROM user WHERE email == :userEmail")
    fun findByEmail(userEmail: String): User?

    @Transaction
    fun insertOrUpdate(user: User) {
        val exists = findById(user.id) != null
        if (exists) {
            update(user)
        } else {
            insert(user)
        }
    }
}
```

Výpis kódu 3.2: Rozhraní UserRepository pro komunikaci s lokální databází.

```
fun loadData() {
    try {
        val fileName = getFullFileName()
        val fis = context.openFileInput(fileName)
        val `is` = ObjectInputStream(fis)
        val result = `is`.readObject()
        `is`.close()
        fis.close()
        onDataLoaded(result)
    } catch (ex: FileNotFoundException) {
        onDataError()
        ex.printStackTrace()
    }
}

protected fun getNewId(): Long {
    sleep(2) //Uspavám, abych měl jistotu unikátnosti
    return Calendar.getInstance().time.time
}

fun saveData(objToSave: Any) {
    check(Looper.myLooper() != Looper.getMainLooper())
    val fileName = getFullFileName()
    val fos = context.openFileOutput(fileName, MODE_PRIVATE)
    val os = ObjectOutputStream(fos)
    os.writeObject(objToSave)
    os.close()
    fos.close()
}

private fun getFullFileName() = "${getFileName()}_$MOCK_VERSION"
```

Výpis kódu 3.3: Ukládání a načítání objektů ve třídě WpMockUp.

Kapitola 4

Testování

Pokud chceme dodat funkční aplikaci, kterou budou lidé používat, určitě není na místě podcenit testování aplikace. Proto i já jsem se ve své práci testování věnoval. V této kapitole popíši, jak probíhalo testování aplikace, které postupy jsem pro testování využil a jaké chyby jsem odhalil.

4.1 Unit testy

Unit testy patří mezi nejdůležitější testy v programování. Jedná se o malé testy. Obvykle testují jednu komponenty a každý unit test by měl otestovat určité vstupy a výstupy komponenty. Každý test by se měl specializovat na jeden určitý scénář, který simuluje, například testování na záporná čísla, nulu, null hodnoty atd.

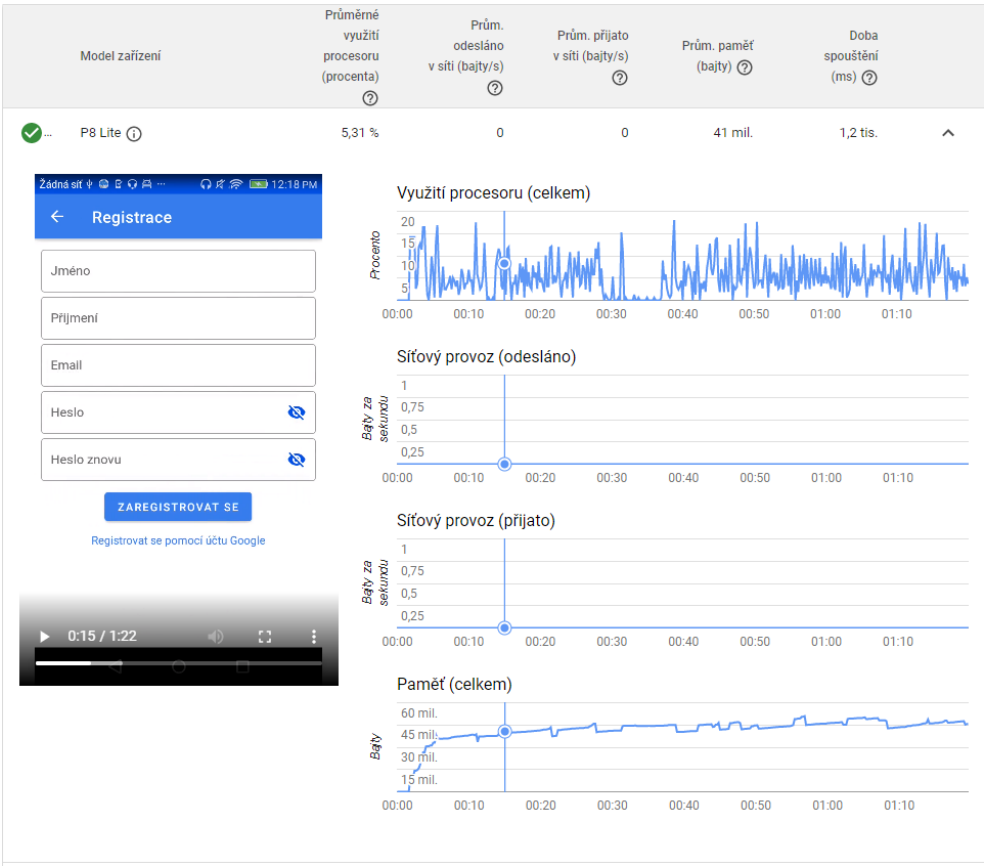
Bohužel tvorba těchto testů je časově velice náročná a jejich udržování je snad ještě náročnější. Proto jsem je využíval jenom na komponenty, které mají různé množství stavů a vstupů.

Unit testy jsem v mojí práci využil na testování generování alimentů. Komponenta dostane nastavení alimentů a musí z nich ve správný čas vygenerovat požadavek na zaplacení alimentů. Nicméně tato komponenta by se ve finální aplikace neměla vyskytnout a měla by být umístěna přímo v BE aplikaci.

4.2 Automatické testy aplikace v Google Play

Android aplikace běží na velké spoustě různých zařízení s různou architekturou a různým rozlišením displaye. Velice často se může stát, že na našem mobilu aplikace funguje a na jiném se ani nespustí a nebo se chová jinak, než čekáme.

Vyzkoušet aplikaci na různých zařízeních s různou verzí Androidu není v lidských silách. Naštěstí Google Play přináší možnost automatického testování, kde za nás aplikaci může otestovat umělá inteligence. Popřípadě můžeme vytvořit testovací skript.



Obrázek 4.1: Výstup testování pro zařízení Huawei P8 Lite [27]

Pří každém nasazení se automaticky spustí v Google Play test, který prokliká aplikaci podle skriptu. Pokud skript selže, nebo není definován, začne AI klikat náhodně po aplikaci. Po dokončení testů lze stáhnout video a sledovat jak aplikace vypadá na jednotlivých zařízeních. Z těchto pozorování vidíme jednotlivé části naší aplikace na různých zařízeních. Lze zde tedy odhalit vizuální chyby, které se na našem zařízení vůbec nemusí projevit. Výstup testů vidíme na obrázků 4.1.

Díky těmto testům jsem v aplikaci odhalil hlavně problémy, které by se projevily na menších zařízeních. Jednalo se o špatné zobrazení prvků při vysunutí klávesnice, překrytí a nebo úplně zakrytí tlačítka jiným prvkem aktuální obrazovky. Dokonce jsem tímto postupem odhalil i neošetřený vstup, kdy aplikace padala.

4.3 Konzultace s klientem

Kromě automatického testování také probíhaly konzultace s klientem. Zde docházelo k ukázce aplikace a představení funkcí, které se od minulého setkání změnily. Tato sezení probíhala buď v budově školy, a nebo pomocí online video konference. Verze aplikace z daného sezení byla vždy k dispozici na Google Play. Klient si jí mohl stáhnout a proklikat ještě před samotnou schůzkou.

Vždy na začátku sezení jsem prezentoval aplikaci. Při této prezentaci jsem se klienta ptal na jeho názor na danou obrazovku. Jaké vlastnost obrazovky se mu líbí a jaká mu naopak chybí. Jeho výtky jsem do dalšího sezení opravil. Takto jsme se postupně propracovávali k uspokojivější verzi aplikace.

Například jsme zde definovali nutnost přidávat jednotlivým členům rodiny jakési aliasy, které by viděl pouze aktuální uživatel. Díky tomu vznikla obrazovka přizpůsobení člena, kterou lze najít na obrázku 3.6. Dále zde vznikl nápad automatické aktivace rodiny, pokud existuje jenom jedna, aby jsme ušetřili uživatele od nutnosti rodinu aktivovat po přihlášení.

Další velice probíranou položkou na našich konzultacích se stala obrazovka pro správu pečovatelských dnů. Tyto obrazovky prošly vcelku velkou sérii změn. V první verzi na původní obrazovce nebyl vůbec vidět náhled aktuálního nastavení dlouhodobých pravidel. Po prezentaci toto bylo změněno a v dalších verzí napraveno. Při další konzultaci bylo zjištěno, že aplikace nepodporuje nastavení pečovatelských dnů pro rodiny s více dětmi. Toto bylo následně také upraveno. Nakonec při poslední konzultaci byla zjištěna nutnost evidovat v aplikaci i hodinu s předáním dítěte. Tato funkce zatím nebyla implementována, ale bude se s ní počítat v budoucím vývoji aplikace.

Díky konzultacím prošla změnou i kniha potřeb. Konkrétně chaty na táto obrazovce. V původní verzi zde existoval chat General, který byl dostupný všem, a rodičovský chat. Do rodičovského chatu neměly přístup děti. Chat General obsahoval změnové řízení. Bylo rozhodnuto, aby se toto změnové řízení dané potřeby přesunulo do rodičovského chatu. Stalo se tak z důvodů, aby děti neměly přístup k informaci, který rodič zamítl danou potřebu a nedocházelo tak k nepříjemným situacím. Chat General byl poté přejmenován pouze na Chat. Dále zde i přišla myšlenka na přesně definovanou posloupnost stavů potřeby. Díky ní budou mít o potřebě přehled oba rodiče a nebude se moci stávat, že jeden z rodičů přeskočí všechny stavy potřeby bez přičinění druhého rodiče. Tuto funkcionalitu jsem ve finální verzi implementovat nestihl.

Díky těmto konzultacím byly z aplikace odstraněny výše zmíněné konceptuální chyby, které by znepříjemňovaly uživateli používání aplikace. Díky jejich odstranění došlo zajisté ke zkvalitnění aplikace.

4.4 Uživatelské testování

Dnes 6.5 2020 proběhlo uživatelské testování aplikace. Aplikace testovala paní Kateřina Malečková, kterou mi doporučila paní Mgr. Bohuslava Janků. Paní Malečková je po rozvodu a stará se o dvě děti. Díky těmto faktům byla ideální kandidátkou na testování.

Testování probíhalo přes videohovor v aplikaci Google Hangouts. Pro přenos obrazovky zařízení jsem použil aplikaci Anydesk. Přenos obrazovky a zvukový záznam naší konverzace jsem nahrával pomocí aplikace Xbox Game bar. Bohužel vzniklý zvukový záznam videohovoru byl nepoužitelný.

Testování probíhalo podle mnou vytvořeného scénáře, který zahrnoval základní úkoly jako zaregistrovat se do aplikace, dále v ní vytvořit rodinu a nadefinovat pečovatelské dny a platby alimentů.

První přihlášení a registrace proběhl bez problémů. Po přihlášení byla uživatelka lehce zmatená. Po prvotním rozkoukání jsme přešli k založení rodiny. Zde přišly první výtky a návrhy. Role v aplikaci byly podle uživatelky nedostatečné a v budoucnu by se měly rozdělit mnohem více. Ideálně například otec, matka a dítě.

Po založení rodiny jsme přešli k nastavení pečovatelských dnů. Zde byl jasný požadavek od uživatelky, a to možnost nastavovat i hodiny předání. Prý podle jejích zkušeností je toto veliký problém při péči o děti. Dále by zde ocenila předdefinované šablony pečovatelských dnů. Třeba taková typická šablona by byla péče u otce každý lichý víkend. Nakonec možnost předat dítě například tetě a nebo babičce. Toto nastavení by však druhý rodič neměl vidět.

Při nastavování alimentů uživatelka chtěla zadat datum místo dne, z čehož jsem usoudil, že tuto obrazovku by chtělo lépe textově popsat, aby zde bylo na první pohled jasné, co uživatel musí nastavit. K nastavení alimentů žádné výtky uživatelka neměla.

Následně jsme přešli k nastavení účtenek. Zde byl jasný požadavek od uživatelky mít možnost nastavení přístupu k účtenkám. Dále by ocenila nějaký měsíční přehled výdajů a možnost sdílet výdaje se členy rodiny. Doporučila mi aplikaci Splitwise, která toto řeší.

Poslední obrazovku, kterou jsme prošli byla kniha potřeb. Jelikož tato obrazovka je primárně určena pro komunikaci s ostatními členy, tak zde testování nebylo příliš přínosné.

V souhrnu je podle uživatelky aplikace na správné cestě, ale je třeba zpříjemnit používání uživatelského rozhraní a celkově grafický dojem aplikace. Dále vylepšit pečovatelské dny a přehled účtenek. Potom bude aplikace ve stavu, kdy by jí chtěla začít používat a reálně by jí pomohla s životní situací.

Kapitola 5

Budoucí možnosti rozšíření aplikace

5.1 Vylepšení stávajících funkcí

Pro začátek by se v aplikaci měly dodělat rozdělené věci. Určitě je třeba dokončit nastavení pečovatelských dnů. A to tak, aby zde uživatel mohl nadefinovat čas předání a implementovat možnost předat děti jinému pečovateli, například tetě či babičce.

Dále u knihy potřeb je nutné dodělat možnost výběru dítěte, ke kterému se daná potřeba vztahuje. Tato funkce v první verzi byla opomíjena. Stalo se tak primárně z důvodů zaměření aplikace, který je zacílen převážně na rodiny s jedním dítětem. Dále by je potřeba přesně definovat návaznost stavů potřeby a možnosti změn v určitém stavu potřeby. Nakonec by se aplikace měla rozšířit o možnost vytváření událostí v dané rodině, které by se zobrazovaly v kalendáři.

Účtenky jako takové by se daly vylepšit také. Určitě by uživatel měl mít možnost nastavit si formu sdílení účtenky. Například sdílet účtenku s celou rodinu, a nebo jí uchovávat pouze pro svoje účely. Dále přehled generovaný z účtenek, který by zobrazoval finanční stav pro udržení dobré ekonomiky rodiny.

Kalendář jako takový je v tuto chvíli nevyužitý. V dalších verzích by se v něm mohly zobrazovat časy pořízení účtenek. Dále by určitě zasloužilo předělat zobrazení pečovatelských dnů. V budoucích verzích by se zde měly graficky rozdělit pečovatelské dny, které vyplývají z dlouhodobých pravidel, a dny, které jsou nastaveny pomocí jednorázových pravidel.

Další věcí, která by si zasloužila pozornost v aplikaci, je uživatelské rozhraní a design aplikace. Momentálně aplikace nevypadá příliš esteticky a určitě nezaujme na první pohled. Celkově nejlepší možnost by bylo vzít aplikaci jako celek a všechny obrazovky ucelit do jednotného grafického formátu, který by vypadal lépe než aktuální podoba.

A konečně aplikace by do budoucna měla směřovat směrem, který splní požadavky zadavatele v plné míře. Čili bude existovat systém práv v rodině, kdy nebude možnost editovat rodinu v případě nedostatečných práv. Toto platí i pro ostatní položky například pečovatelské dny či alimenty. Základní systém práv je v aplikaci již zanesen, ale jeho rozšíření je určitě možné. Od jednoduchého systému, kdy pro role v rodině úplně zakážeme některé operace, kdy by například děti nemohly editovat pečovatelské dny, bychom se mohli dostat až k opravdu pokročilému systému, kdy by samotní uživatelé mohli dynamicky měnit práva ostatním členům rodiny.

Kromě práv bude muset aplikace do budoucna uchovávat veškerou historii změn. Pravda je, že toto je úloha spíše pro BE aplikace. Nicméně Android aplikace by tuto historii mohla ukazovat a uživatel by viděl, v jakém stavu byla aktuální položka v daném čase. Tato schopnost je důležitá pro následné dokazování u soudů. Dále by aplikace mohla exportovat celou historii rodiny do souboru. Soubor by sloužil jako souhrn všeho, co se v rodině událo, a taktéž by sloužil jako důkazní materiál pro případný soudní spor.

I když aplikace je funkční, k její použitelnosti pro zákazníky je potřeba jí napojit na BE. To by měl být další krok, kterým by se budoucí vývoj aplikace měl ubírat. Ideální by bylo napojení aplikace na rozhraní, které poskytuje Iaroslav Kolodka ve své práci.

5.2 Firebase

Avšak toto není jediná možnosti. Velice zajímavou cestou, kterou by se aplikace mohla v budoucnu vydat, je využití možnosti od Firebase. Jedná se o soubor funkcí, které mají pomoci při tvorbě webových a mobilních aplikací.

Může poskytnout vlastní rozhraní pro ukládání dat do databáze na serveru. Toto úložiště dat běží na serverech Googlu. Díky tomu máme jako programátoři jistotu jejich prakticky trvalé dostupnosti a spolehlivosti. Další velice zajímavou vlastností této služby je synchronizace v reálném čase. To znamená, že všichni uživatelé mobilní aplikace by viděli okamžitě aktuální data, která jsou k dispozici na serveru. Samozřejmě v případě, že by byl jejich mobilní telefon připojen k internetu. Avšak knihovna za programátora řeší i synchronizaci a přechody mezi tím kdy je mobilní telefon připojen a kdy ne. Avšak hlavní výhoda toho řešení je zároveň i jeho nevýhoda. Uživatelé by svá citlivá data měli na serverech Googlu a to v případě naší aplikace není žádoucí. [28]

Další použitelnou funkcí této služby pro aplikaci je Firebase Cloud Messaging. Tato služba poskytuje možnost poslat na mobilní zařízení s operačním systémem Android i IOS notifikaci. Je potom na samotném zařízení, jestli zobrazí notifikaci, a nebo vykoná jinou akci, například spuštění synchronizace. Služba zaručí, že zpráva dorazí i na mobilní telefon, který v čase odesílání zprávy nebyl připojen k internetu a zajistí její poslání na mobil později. Respektive jí mobilní telefon obdrží po připojení k internetu. Tato služba oproti

předchozí zmíněné je pro aplikaci ideální. Daly by se díky ní posílat pozvánky do rodin, upozornění na zaplacení alimentů atd. Služba funguje na bázi tokenů. Při první spuštění mobil obdrží od Firebase serveru token. Token je unikátní pro každé zařízení. Tento token se následně pošle na BE server aplikace. BE si token uloží. Potom v případě, že BE chce poslat zprávu, zařízení pošle požadavek na Firebase server s příslušným tokenem. Služba dále podporuje i posílání zpráv určité skupině zařízení. Čili pokud by v zařízení byla aktivní rodina například Novákovi, mobilní telefon pošle požadavek na registraci do skupiny Novákovi. Poté, až server pošle zprávu do dané skupiny, jí obdrží všechna zařízení, ve kterých je aktivní rodina Novákovi. Tento scénář je ideální pro upozornění uživatelů v celé rodině. Například při přijetí pozvánky novým členem rodiny by všem v rodině přišlo upozornění na nového člena v dané rodině. [29]

Závěr

Cílem práce bylo vytvořit Android aplikaci, která by měla pomoci rodinám ulehčit rozvod a období po rozvodu. V práci jsem měl navázat na výsledky z předmětů BI-SP1 a BI-SP2.1, analyzovat požadavky klienta a na jejich základě implementovat aplikaci, tu následně otestovat a navrhnout kroky pro její zlepšení.

V mojí práci jsem úspěšně implementoval prototyp Android aplikace. Aplikace je plně funkční. Implementovaná aplikace pracuje pouze offline a není napojená na žádné serverové rozhraní. Přesto je připravená k napojení na rozhraní serveru, které je implementované Iaroslavem Kolodkou v jeho práci.

Z požadovaných funkcionalit se mi podařilo implementovat správu alimentů, uchovávání účtenek, knihu potřeb a správu pečovatelských dnů. Společné události v rámci rodiny se mi v této verzi implementovat nepodařilo z časových důvodů.

Aplikace používá architekturu, která je popsaná v sekci 2.2. Tato architektura je podporovaná již v samotném Android SDK, které jsem použil pro naprogramování aplikace. Celá aplikace je napsaná v programovacím jazyce Kotlin. Pro práci na pozadí jsem využil Coroutines.

Aplikace byla pravidelně konzultována s klientem v průběhu celého vývoje. Díky tomu aplikace splňuje jeho požadavky ve zjednodušené formě.

Testování aplikace probíhalo formou automatických testů a jednoho uživatelského testu aplikace. K automatickému testování jsem využil unit testy a testy v Google Play. Uživatelské testování proběhlo podle mnou vytvořeného scénáře, kterým se uživatelka řídila. Její připomínky budou zohledněny v budoucím vývoji aplikace.

Bibliografie

1. TALPA Lukáš, Bc; JANKŮ Bohuslava, Mgr. *Problematika rozvodů*.
2. *Documentation : Android Developers* [online] [cit. 2020-03-26]. Dostupné z: `https://developer.android.com/docs`.
3. *Android Platform: Android Developers* [online] [cit. 2020-05-09]. Dostupné z: `https://developer.android.com/reference/packages`.
4. *Android Support Library: Android Developers* [online] [cit. 2020-05-09]. Dostupné z: `https://developer.android.com/reference/android/support/packages`.
5. *AndroidX: Android Developers* [online] [cit. 2020-05-09]. Dostupné z: `https://developer.android.com/reference/androidx/packages`.
6. *Architecture Components: Android Developers* [online] [cit. 2020-05-09]. Dostupné z: `https://developer.android.com/reference/android/arch/packages`.
7. *App Manifest Overview : Android Developers* [online] [cit. 2020-05-09]. Dostupné z: `https://developer.android.com/guide/topics/manifest/manifest-intro`.
8. *Enable multidex for apps with over 64K methods : Android Developers* [online] [cit. 2020-03-26]. Dostupné z: `https://developer.android.com/studio/build/multidex`.
9. *Understand the Activity Lifecycle : Android Developers* [online] [cit. 2020-03-26]. Dostupné z: `https://developer.android.com/guide/components/activities/activity-lifecycle`.
10. *Fragments : Android Developers* [online] [cit. 2020-03-26]. Dostupné z: `https://developer.android.com/guide/components/fragments`.
11. *Services overview : Android Developers* [online] [cit. 2020-05-06]. Dostupné z: `https://developer.android.com/guide/components/services`.

12. *Slide between fragments using ViewPager : Android Developers* [online] [cit. 2020-03-30]. Dostupné z: https://developer.android.com/training/animation/screen-slide.

13. *ViewModel Overview : Android Developers* [online] [cit. 2020-05-06]. Dostupné z: https://developer.android.com/topic/libraries/architecture/viewmodel.

14. *Guide to app architecture : Android Developers* [online] [cit. 2020-03-26]. Dostupné z: https://developer.android.com/jetpack/docs/guide.

15. *Most Popular Programming Languages 1965 - 2019* [online]. Data Is Beautiful, 2019 [cit. 2020-03-26]. Dostupné z: https://www.youtube.com/watch?v=Og847HVwRSI.

16. *Kotlin vs Java Which one is Better for your Business* [online]. 2020 [cit. 2020-03-26]. Dostupné z: https://www.folio3.com/mobile/blog/kotlin-vs-java/.

17. *Kotlin Programming Language* [online] [cit. 2020-03-26]. Dostupné z: https://kotlinlang.org/.

18. *Coroutines for asynchronous programming and more* [online] [cit. 2020-03-26]. Dostupné z: https://kotlinlang.org/docs/reference/coroutines-overview.html.

19. MILLER, Paul. *Google is adding Kotlin as an official programming language for Android development* [online]. The Verge, 2017 [cit. 2020-03-26]. Dostupné z: https://www.theverge.com/2017/5/17/15654988/google-jet-brains-kotlin-programming-language-android-development-io-2017.

20. *React Native · A framework for building native apps using React* [online] [cit. 2020-03-26]. Dostupné z: https://reactnative.dev/.

21. FACEBOOK. *facebook/stetho* [online]. 2019 [cit. 2020-03-30]. Dostupné z: https://github.com/facebook/stetho.

22. *Strange out of memory issue while loading an image to a Bitmap object* [online]. 1958 [cit. 2020-03-30]. Dostupné z: https://stackoverflow.com/questions/477572/strange-out-of-memory-issue-while-loading-an-image-to-a-bitmap-object.

23. *Picasso* [online] [cit. 2020-03-30]. Dostupné z: https://square.github.io/picasso/.

24. MULTIDOTS. *Glide vs. Picasso* [online]. Medium, 2017 [cit. 2020-03-30]. Dostupné z: https://medium.com/@multidots/glide-vs-picasso-930eed42b81d.

25. IGREENWOOD. *SimpleCropView* [online]. 2018 [cit. 2020-05-06]. Dostupné z: https://github.com/igreenwood/SimpleCropView.

26. TOBIASSCHUERG. *tobiasschuerg/android-week-view* [online]. 2020 [cit. 2020-03-30]. Dostupné z: `https://github.com/tobiasschuerg/android-week-view`.

27. *Přehled testů před vydáním aplikace : Google Play Console* [online]. Google Play Console [cit. 2020-03-26]. Dostupné z: `https://play.google.com/apps/publish/?account=6509064164411815818#PreLaunchReportPlace:p=cz.cvut.fit.sp.rozvody.android&appid=4976219327654757330&plrvc=9&plrtab=PERFORMANCE`.

28. *Firebase Realtime Database | Store and sync data in real time* [online]. Google [cit. 2020-03-26]. Dostupné z: `https://firebase.google.com/products/realtime-database`.

29. *Firebase Cloud Messaging* [online]. Google [cit. 2020-03-26]. Dostupné z: `https://firebase.google.com/docs/cloud-messaging`.

Příloha A

Seznam použitých zkratek

AI Umělá inteligence

API Application Programming Interface

BE Backend - část aplikace pro výpočetní logiku

B182 Letní semestr bakalářské studia školního roku 2018/2019

BI-SP1 Softwarový týmový projekt 1

BI-SP2.1 Softwarový týmový projekt 2

DTO Objekt pro datové přenosy

FE Frontend - část aplikace, se kterou uživatel interaguje

HW Hardware

IDE Vývojové prostředí

IOS Operační systém mobilních telefonů Apple

JWM Java Virtual Machine

ORM Objektově relační mapování

REST Representational State Transfer

SDK Sada vývojových nástrojů

SQL Strukturovaný dotazovací jazyk

URL Jednotná adresa zdroje

Příloha **B**

Obsah přiloženého media

- `release.apk` ..... Instalační soubor aplikace
- `src`
  - `impl` ..... Zdrojové kódy implementace
    - `WeekView` ..... Modul vykreslení pečovatelských dnů
      - ...
    - `UserRepository.kt`
    - `WpMockUp.kt`
  - `thesis` ..... Zdrojová forma práce ve formátu LaTeX
    - `literatura` ..... Složka databázemi literárních zdrojů
      - `ref.bib`
    - `obrazky` ..... Obrázky v textu
      - `screenshots` ..... Obrázky obrazovek aplikace
        - ...
      - ...
    - `prohlaseni` ..... Všechna dostupná prohlášení
      - ...
    - `FITthesis.cls` ..... Šablona práce v LaTeX
    - `text.tex` ..... Zdrojový kód textu v LaTeX
    - `zadani.pdf` ..... Zadání práce
- `text` ..... Text práce
  - `text.pdf` ..... Text práce ve formátu PDF