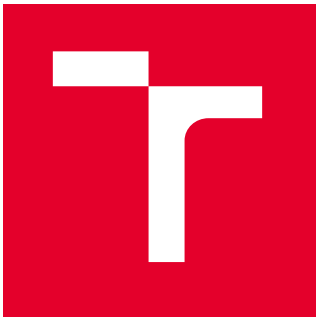


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

MOBILNÍ APLIKACE PRO INTELIGENTNÍ SYSTÉM CHOVU VČEL

MOBILE APPLICATION FOR AN INTELLIGENT BEEKEEPING SYSTEM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Martin Pecár

AUTHOR

VEDOUCÍ PRÁCE — Ing. Martin Kiac

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Martin Pecár ***ID:*** 211269

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Mobilní aplikace pro inteligentní systém chovu včel

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a realizovat aplikaci pro operační systém Android, která bude umožňovat komunikaci s navrženým inteligentním systémem chovu včel. Aplikace bude schopna uchovávat a spravovat jednotlivé informace o včelnici, úlech a samotných včelách. Výsledná Android aplikace by měla umožňovat kompletní správu včelstva a tak zefektivnit práci včelaře.

DOPORUČENÁ LITERATURA:

[1] LACKO, Ľuboslav. Vývoj aplikací pro Android. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

[2] GERSTMEIER, David, Tobias MILTENBERGER a Hannah GÖTTE. Jeden rok v živote včely: ako včely žijú, čo všetko robia a prečo je matka kráľovnou všetkých včiel. Přeložil Lenka FIFKOVÁ ŠKOLNÍKOVÁ. Brno: Kazda, 2020. ISBN 978-80-88316-80-0.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 31.5.2022

Vedoucí práce: Ing. Martin Kiac

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Cílem práce je navrhnout a vytvořit aplikaci, která umožní včelařům správu včelstva pomocí mobilního telefonu. Důvodem je centralizace a zpřehlednění všech nasbíraných dat z kontrol včelstva, přičemž tato data bude později možné použít při tvorbě statistik. Dále tato aplikace obsahuje možnosti, jak včelaře upozornit na potřebu zásahu do včelstva pomocí jím vytvořených upozornění a statistiky o vybraných sledovaných vlastnostech včelstva. Výstupem bude popsaná mobilní aplikace.

KLÍČOVÁ SLOVA

včelařství, Android, mobilní aplikace, Java, zápisník

ABSTRACT

The aim of this thesis is to design and create an appliaction, which will allow beekepers to manage their hives with a mobile phone. The reason for this is centralization and clarification of all collected data from visits to the hive, where this data could be later used to create statistics. Furthermore, this app contains ways to notify the beekeeper that there is a need of intervention with the hive using their own alerts and statistics of selected properties of a hive. The result of this work is the previously described application.

KEYWORDS

beekeeping, Android, mobile application, Java, diary

PECÁR, Martin. *Mobilní aplikace pro inteligentní systém chovu včel*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 52 s. Bakalářská práce. Vedoucí práce: Ing. Martin Kiac

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Martin Pecár
VUT ID autora:	211269
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	Mobilní aplikace pro inteligentní systém chovu včel

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Martinu Kiacovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěl poděkovat panu Bc. Ondřeji Babicovi za čas mi věnovaný při prohlídce jeho včelnice a za trpělivost při vysvětlování včelařského roku a paní Martě Stradějové za totéž a za možnost nahlédnutí do jejích záznamů o chovu včel.

Obsah

Seznam obrázků

Úvod

Tato práce si dává za cíl zjednodušit práci včelařům pomocí aplikace, díky které budou moci efektivně spravovat všechna svá včelstva. V současné době již existují způsoby, jak si vést informace o včelstvech pomocí mobilního telefonu, nicméně cílem této práce je tuto činnost zjednodušit a sjednotit. Uživatel by měl mít možnost zakládat lokality, včelstva, zapisovat návštěvy, záznamy medobraní a přidávat upozornění, díky kterým bude mít přehled o stavu včelstev a jejich současných problémech. Tento přehled bude realizován pomocí domovské obrazovky, která bude obsahovat shrnutí všech existujících problémů.

Po předchozím průzkumu již existujících aplikací pro včelaře na platformě Google Play bylo zjištěno, že neexistuje aplikace v češtině, která by plnila i jinou funkci, než pouze zápisníku. Proto výstupem této práce bude aplikace, která bude uživateli poskytovat statistiky na základě jím vložených dat.

První část této práce se zabývá problematikou chovu včel, vysvětluje používané pojmy a nastiňuje současné postupy při správě včelstev. Dále pak popisuje použité technologie při vývoji aplikace a přiblíží jejich použití v jednotlivých částech aplikace.

Druhá část pojednává o základním návrhu aplikace. Je možné se v ní dočíst o koncepci návrhu grafického uživatelského rozhraní a také o způsobu uložení a rozložení těchto dat.

Poslední část je věnována samotné aplikaci. Popisuje realizaci aplikace a její vnitřní chod.

1 Teoretická část práce

Tato kapitola se zaměřuje na uvedení do problematiky včelařství a chovu včel a na technologie, které budou při tvorbě aplikace využívány.

1.1 Základy chovu včel

Včela medonosná (lat. *Apis mellifera* L.) patří do řádu blanokřídlého hmyzu a jedná se o společenský druh hmyzu. Společenský druh znamená, že jedinci tohoto druhu tvoří sociální uskupení soustředěná na jednom místě, kde dochází ke vzájemné spolupráci všech jedinců při všech činnostech nutných ke správnému chodu celého společenství. Včely jsou zodpovědné za opylování přibližně 85% všech kvetoucích rostlin, což představuje přibližně 170 000 druhů rostlin. Pokud se zaměříme na ovocné stromy, včely opylují přibližně 90% všech jejich květů. Z tohoto důvodu jsou včely považovány za nepostradatelný živočišný druh, bez kterého by pravděpodobně došlo ke globální potravinové krizi [1, 2].

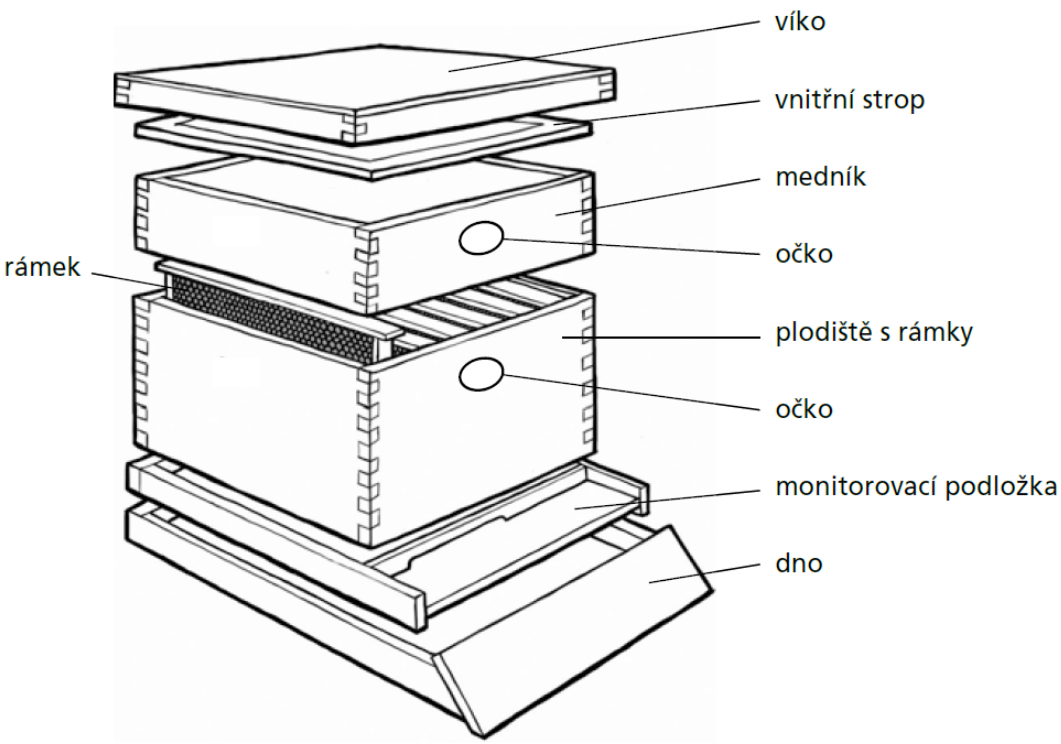
1.1.1 Základní pojmy

Aby v budoucnosti nedocházelo k nedorozumění, je třeba vysvětlit základní pojmy, se kterými včelaři běžně pracují.

Včelstvo je organizované společenství velkého množství včel, které může běžně čítat okolo 50 000 včel, často i více. Včelstvo má vždy jednu *matku* (někdy také *královnu*), několik tisíc *dělnic* a v letních měsících i malé množství *trubců*. Matek může být ve výjimečných případech více, avšak pouze po krátkou dobu před *rojením*, což je období obměny matek v úlu. Matka ve včelstvu plní funkci kladení vajíček a tím zajišťuje obnovu uhynulých dělnic, popřípadě trubců. K tomuto využívá spermie od trubců, se kterými se jednou za život spáří, a poté si je uchovává po zbytek života. Matka má okolo sebe neustále skupinu dělnic, které ji ochraňují, pečují o ni a krmí ji. Dělnice v úlu vykonávají všechny ostatní činnosti jako je čištění, péče o plod, krmení matky, stavba *voskového díla* (plástu) a kriticky také přinášení pylu, nektaru a vody do úlu. Trubci slouží ve včelstvu pouze k tomu, aby oplodnili mladou matku a popřípadě pomáhali se zahříváním plodů svým vlastním tělem. *Plod* je označení používající se pro larvu včely. Mezerovitost plodu označuje, jak moc jednolitě je rámek s buňkami osazen larvami [1].

Včelstva přírodně vytvářejí *úly*, nicméně včelaři jim v tomto pomáhají stavbou umělých prostor zvlášť navržených k lepší stavbě díla. Úl se dělí na několik částí podle jejich funkce. Do *plodiště* matka klade vajíčka a do *medníku* dělnice přinášejí nektar a medovici, které později zpracovávají v med (více v sekci o medobraní níže).

V současné době se používají takzvané *nástavkové úly*. Je s nimi jednoduchá práce, dají se snadno přenášet a jsou vertikálně škálovatelné [1].



Obr. 1.1: Nákres nástavkového úlu s popisem částí [1].

Lokalita, na které se nachází větší množství úlů, se obecně nazývá *včelnice*. Čtenář se mohl v minulosti setkat s pojmem *včelín*, který ovšem popisuje úly, které jsou blízko při sobě a pod jednou střechou. Včelnice by se neměly nacházet v blízkosti vedení vysokého napětí a silných zdrojů elektromagnetického záření, jako například vysílačů nebo radarů. Naopak je důležité, aby byly v oblastech s dostatečným prouděním vzduchu, které pomáhá včelám udržovat optimální teplotu v úlu, a aby měly dobrý přístup k vodě, který je nezbytný k přežití včelstva a produkci medu. [1].

Měl je souhrnné označení pro odpad, který napadá na úlovou podložku na dně úlu. Zejména se v něm nacházejí zbytky rozkousaných voskových víček od buněk s medem nebo cukerným roztokem a zbytky uhynulých včel [1].

Při *kontrole* včelstva včelař navštěvuje jednotlivé úly a sleduje různé faktory, které ovlivňují stav včelstva. Tyto faktory se mění zejména s ročním obdobím. Nicméně pro uvedení alespoň některých je třeba zmínit kontrolu váhy celého úlu, spad roztočů *Varroa destructor*, spad měli a jiných objektů, jako například cizí tělesa ve formě uhynulých vos, sršňů nebo mravenců, kontrola stavu zásob živin ve formě medu nebo cukerného roztoku, kontrola mezerovitosti plodu a kontrola rámků a nástavků [1].

Medobraní je činnost, při které včelař z úlu odebírá mezistěny naplněné medem a ten z nich vytáčí. Med vzniká přeměnou nektaru nebo medovice včelami. Medovice je na rozdíl od nektaru živočišný produkt, produkovaný zejména mšicemi. Kromě monosacharidů glukózy a fruktózy obsahuje také polysacharidy, například *melecitózu*. Ta ve větším množství způsobuje vznik takzvaného *cementového (melecitózního) medu*, který se vyznačuje velmi rychlou krystalizací, a tudíž je pro včely nestravitelný a pro včelaře velmi těžce zpracovatelný. Jednotlivé typy medů se dají rozlišit vizuálně pomocí barvy, jejíž změna je způsobena různým původem nektaru, ze kterého med vznikl. Zralost a kvalita medu se dá určit pomocí obsahu vody. Běžně se vytáčí med s obsahem vody okolo 18%. Když je obsah vody roven více než 20%, med má tendenci kvasit. [1, 2]

1.1.2 Vlastnosti včelstva

Včelstvo je možné popsat pomocí některých sjednocujících vlastností. Tyto vlastnosti jsou určeny zejména geneticky a včely jsou pro ně speciálně šlechtěny.

Síla včelstva je označení pro celkový stav včelstva. Velmi silné včelstvo může obsahovat i 120 000 dělnic. Takové včelstvo pracuje na všech poskytnutých rámcích a bývá zpravidla velmi výnosné. Síla včelstva se často mění společně se stářím matky. Čím je matka starší, tím méně jí zbývá spermatu, proto se rodí i méně dělnic a síla meziročně může upadat.

Agresivnost a bodavost jsou dvě velmi úzce provázané vlastnosti, které určují reakci včelstva na přítomnost útočníka a její sílu. Tento útočník může být jakéhokoli původu, od včel z jiných včelstev (viz slídivost), přes hlodavce až po včelaře. Jejich názvy mluví samy za sebe a není třeba je dále vysvětlovat.

Stavební pud určuje, jak aktivně včely budují své dílo. Ten se mění nárazově v průběhu roku a zejména při rojení, avšak lze jej sledovat i v průběhu roku.

Slídivost je nežádoucí vlastnost včelstva, při které se dělnice silného včelstva rozlétají do jiných, často slabších, včelstev. Tam páchají škody například vykrádáním zásob, v nejhorším případě mohou způsobit i uhynutí napadeného včelstva. Mimo nebezpečí úhynu včelstva hrozí také nebezpečí zavlečení nemocí nebo parazitů zpět do původního včelstva a tím jeho ohrožení.

Čisticí pud popisuje schopnost včelstva zbavovat se nežádoucích těles v úle. Za nežádoucí tělesa se považuje kupříkladu odpad produkovaný při běžné funkci úlu (měl) nebo těla vetřelců. Včely hynou z důvodu vyčerpání nebo kvůli onemocněním. Vynášením nakažených uhynulých včel se tedy včelstvo zbavuje původce nemoci.

Poslední důležitou vlastností je *pospolnost*, popřípadě *rozbíhavost* včelstva. Tyto vlastnosti jsou vzájemně provázané a mají opačný význam (tj. včelstvo s vysokou pospolností má nízkou rozbíhavost). Pospolnost a rozbíhavost určují, jak moc se včely

udržují pohromadě při kontrole úlu, popřípadě v zimních měsících, kdy semknuté udržují teplotu v úlu. Dělnice ve včelstvu s vysokou pospolností se budou při kontrole držet na rámcích a nebudou mít tendence odlétávat daleko od úlu. Tato vlastnost umožňuje včelstvu přežít zimu ve velké síle [1].

1.1.3 Nemoci včel

Korektní správou včelstva a podporou jejich přirozených pudů je možné do velké míry zamezit výskytům jakékoli nemoci, nicméně je třeba s jejich výskytem počítat. Níže jsou uvedené nejčastější nemoci, které mohou včelstvo postihnout. Žádná z nich není přenosná na člověka [1].

Varroáza

Varroáza je parazitické infekční onemocnění způsobené výše zmíněným roztočem *Varroa destructor*, česky kleštík včelí. Tento roztoč se původně endemicky vyskytoval v Indonésii, kde parazitoval na včele východní (lat. *Apis cerana* Fabricius). Ta je geneticky vybavena jistou mírou odolnosti vůči tomuto roztoči. Avšak po zavlečení včely medonosné do těchto oblastí se roztoč varroa začal postupně šířit i do jiných oblastí Asie a dále na západ. První případy varroázy se v Česku objevily až v 80. letech 20. století. Asi nejhorší katastrofa, která postihla české včelaře, byla epidemie varroázy v zimě mezi lety 2007 a 2008. Tato zima byla nebývale teplá, a proto se roztočům varroa velmi dobře dařilo i ve vyšších polohách, kde včelaři tuto nákazu neočekávali. Z tohoto důvodu zmíněnou zimu nepřežilo přibližně 119 tisíc včelstev, což v té době odpovídalo přibližně jedné třetině všech včelstev v České republice [1, 2, 3].

Naštěstí je varroáza poměrně úspěšně léčitelná a existují i formy neinvazivní preventivní léčby. Abychom byli schopni efektivně léčit celé včelstvo, používají se zejména dvě metody. První metoda se nazývá *fumigace*, který spočívá v nakapání léčivého přípravku na dýmající pásku, který později zadýmí celý prostor úlu, a tudíž se také rovnoměrně rozšíří účinná látka léčiva. Druhým způsobem je pak kontaktní léčba, která spočívá v přímém kontaktu včely s páskou, která je napuštěná léčivou látkou [1].

Mor včelího plodu

Mor včelího plodu je bakteriální onemocnění způsobované bakterií *Paenibacillus larvae*, které se projevuje u larev. Jeho příčinou je často špatné dbaní včel o čistotu úlu. Nakažené larvy v buňkách hynou, proto se ostatní včely snaží těchto larev zbavit, a tím se samy nakazí a šíří toto onemocnění dále. Spory této bakterie přežívají

Obr. 1.2: Roztoč varroa na včele

velmi dlouhou dobu v úlech nebo i ve volné přírodě, často jsou schopny v tomto stavu setrvat i okolo 30 let. Jasným příznakem moru včelího plodu je vysoká mezerovitost plodu. Plody hynou dříve, než se stíhají rodit a včelstvo takto postupně umírá.

Včelstvo se zvládne původce nákazy zbavit samo díky svému čistícímu pudu, nicméně pokud se tomu tak nestane, je třeba přistoupit k likvidaci včelstva a veškerý materiál, který se dostal do kontaktu s infikovaným včelstvem, je nutno spálit [1].

1.2 Technologie použité při vývoji

Kontrastně k předchozí části je ovšem také důležité se zabývat technologiemi, které budou použity k realizaci aplikace. Toto bude podrobněji popsáno v následujících podkapitolách.

1.2.1 Operační systém Android

Android je open source mobilní operační systém (*OS*) založený na monolitickém linuxovém jádře, který je téměř od svého počátku vyvíjen společností Google. Dle dostupných informací se v současné době celosvětově OS Android nachází na přibližně 70% mobilních zařízení. V Evropě je toto číslo o něco menší, pouhých 64%, nicméně to jsou stále téměř dvě třetiny všech zařízení a proto byla volba platformy poměrně jednoduchá [4, 5].

V současné době se OS Android nachází ve verzi 12, oficiálně vydána 4. října 2021. Ještě stále však existuje nezanedbatelné množství zařízení, na kterých běží Android verze 5 nebo dokonce 4. V dnešní době je stále ještě možné programovat i na tyto verze díky podpoře starších API. *Application Programming Interface* je prostředek, pomocí kterého mohou spolu dvě aplikace komunikovat. V tomto případě se jedná o komunikaci mezi operačním systémem a programovanou aplikací. API verze Androidu 12 má číslo 31 [6].

Jak už bylo řečeno, novější verze Androidu jsou schopné spouštět aplikace postavené na starších verzích API. V tomto projektu bude použita verze API 26, která odpovídá verzi OS Android 8 a vyšší. Tato verze byla zvolena z toho důvodu, že podporuje API *java.time* z *OpenJDK8*, které bylo využito při zpracovávání statistik [7].

1.2.2 Základy vývoje aplikace pro Android

Aby bylo možno v budoucnosti sjednotit terminologii, je nutné definovat některé základní pojmy, které se v tomto kontextu často vyskytují.

Activity

Activity, nebo také aktivita, je součást aplikace, se kterou uživatel přímo interaguje. Aktivita je odpovědná za vytvoření okna, které v sobě obsahuje komponenty uživatelského rozhraní a za definici funkcí těchto komponent. Aktivitu si tedy můžeme představit jako kontejner obsahující všechny položky uživatelského rozhraní [6].

Fragment

Ačkoli je možné postavit celou aplikaci jen na aktivitách, není tak možné vytvořit například vícezáložkovou obrazovku. K tomuto slouží fragment. Fragment může být součástí aktivity a představuje znovupoužitelnou součást uživatelského rozhraní. Znovupoužitelnost je možné využít jak v rámci stejné aplikace, tak při vytváření uživatelského rozhraní například pro tablet, který má jiný poměr stran a větší obrazovku. Dotykové plochy budou na větší obrazovce větší a je tak možné umístit na obrazovku více užitečných komponent.

Fragment si můžeme tedy představit jako takovou sub-aktivitu. Součástí aktivity může být několik fragmentů, které obsahují své vlastní prvky uživatelského rozhraní a tudíž i svou vlastní funkcionalitu. Další důležitý rozdíl mezi aktivitou a fragmentem je ten, že každá aktivita musí být zmíněna v *manifestu* aplikace, což je soubor, který obsahuje důležitá metadata týkající se aplikace (jméno aplikace, hlavní aktivita,

která se zobrazí při spuštění, názvy aktivit a jejich vlastnosti...). Fragment také nemůže existovat sám o sobě, nicméně musí být součástí aktivity [6].

View

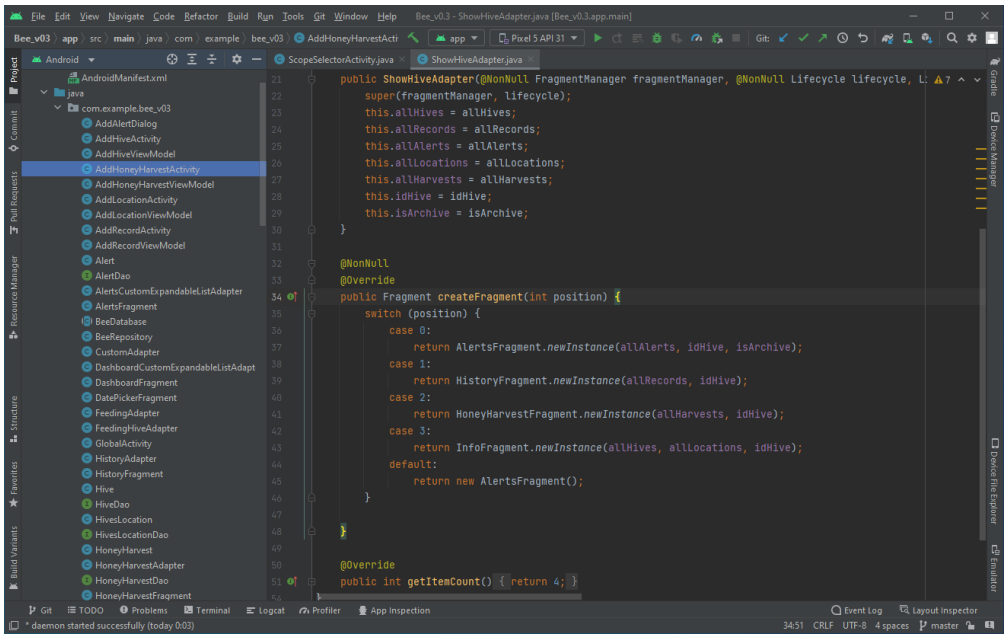
V předchozích odstavcích byly zmíněny komponenty uživatelského rozhraní. Tyto komponenty se označují souhrnným názvem *Views* a mezi ně patří například textová pole, editovatelná textová pole, tlačítka, seznamy, rozbalující se seznamy a velké množství dalších. View může být také například *ViewPager*, který je možno využít na zobrazování několika fragmentů vedle sebe. [6].

Layout

Layout je další důležitou součástí aplikace. Jak už intuitivně napovídá český překlad „rozložení“, jedná se o soubor, ve kterém je popsáno rozložení jednotlivých views na obrazovce, popřípadě v jiných views (například *ExpandableListView*). Layout soubory se nacházejí ve své vlastní složce a mají formát XML souborů (*eXtensible Markup Language*). Existuje několik typů layoutů, které mají odlišné vlastnosti a každý z nich je užitečný k jinému účelu. Použité typy layoutů v tomto projektu budou podrobněji vysvětleny v kapitole *Realizace aplikace* [6].

1.2.3 Android Studio

Jako *Integrated Development Environment* (*IDE*, vývojové prostředí) v tomto projektu bylo použito Android Studio. Jedná se o IDE vyvíjené společností JetBrains a poskytuje komplexní sadu nástrojů jak pro vývoj, tak pro testování a zlepšování výkonu aplikace. Pro uvedení některých důležitých a užitečných nástrojů je třeba zmínit například Android Virtual Device, což je virtualizovaný operační systém Android společně s emulovaným zařízením dle vlastního výběru. Toto umožňuje optimalizovat aplikaci pro různá rozlišení displeje, poměry stran a podobné. S rozlišením displeje nedílně souvisí design jednotlivých aktivit v aplikaci. Ten je možné vytvářet pomocí dalšího nástroje zvaného Visual Layout Editor. Tento editor v reálném čase zobrazuje grafickou podobu vytvářeného layoutu. Dále se zde také nachází nástroj Realtime Profiler na profilování výkonu aplikace, čili na její nároky na výpočetní zdroje jako například využití procesoru, operační paměti nebo síťové karty. Další užitečný nástroj se jmenuje Layout Inspector, který v reálném čase zobrazuje pozice jednotlivých layoutů a zdali byly správně načteny [6].



Obr. 1.3: Snímek obrazovky programu Android Studio

1.2.4 Java

Java je multiplatformí objektově orientovaný programovací jazyk, který byl dlouhou dobu znám jako oficiálně doporučovaný jazyk pro vývoj aplikací pro operační systém Android. V současné době je vyvíjena společností Oracle Corporation. Multiplatformnost jazyka Java je zaručena díky způsobu jeho spouštění. Java je kompilována do binárních souborů ve formátu *class*. Více těchto souborů je možné později zabalit do balíčků *.jar*. Tyto soubory obsahují tzv. *bytecode*, což je sada strojových instrukcí pro *Java Virtual Machine* (*JVM*). JVM si můžeme představit jako interpret jazyka Javy. Je možné jej implementovat pro širokou škálu zařízení od nositelné elektroniky, přes mobilní zařízení, osobní počítače až po serverový hardware. Specifikace podporuje implementaci přímo na silikon procesoru, nicméně v současné době je toto využíváno pouze experimentálně.

Díky nezávislosti na hardware, na kterém je JVM provozována, je používána jako interpret pro jiné jazyky jako jsou například Ruby, Python, Jython, nebo současný oficiálně doporučovaný jazyk pro vývoj aplikací pro Android, jazyk Kotlin. Java byla pro tento projekt zvolena z důvodu předchozích zkušeností autora s tímto jazykem [8, 9].

1.2.5 Room

Room je knihovna, která slouží jako abstrakční vrstva nad SQLite databází a umožňuje objektově orientovaný přístup k databázím a objektům v nich uložených. Jinými slovy, backend Room knihovny je stále SQLite databáze. Je možné přímo využívat SQLite API na přístup k databázím, nicméně toto není oficiálně doporučováno. Na rozdíl od přímého využití SQLite API má totiž Room několik výhod. Mezi ty největší patří, že již při kompilaci kontroluje syntaxi SQL dotazů a jejich aplikovatelnost na databázi a její tabulky a také, že omezuje výskyt repetetivního kódu díky zapouzdření SQL dotazů do metod.

V současné době je doporučována architektura aplikace *MVVM* (Model-View-ViewModel) využívající knihoven *Room*, *ViewModel* a *LiveData*. Tento bude hlouběji vysvětlen v následujících odstavcích společně s architekturou knihovny Room [6].

Architektura

V této sekci se nachází rychlý popis architektury knihovny Room. Existují paralely s tradiční SQL databází, které umožní lepší představu o struktuře této knihovny.

Základním prvkem každé databáze je tabulka. Ta je v knihovně Room reprezentována třídou, které se říká *Entity* (česky *Entita*). Každá tato entita popisuje jednu tabulku, definuje její název, sloupce, primární klíče a popřípadě cizí klíče. Jména sloupců jsou definována pomocí lokálních proměnných, stejně tak jejich datové typy. V každé entitě je třeba definovat konstruktor, pomocí kterého je později možné interagovat s objekty tabulky. Jeden objekt vytvořený tímto konstruktorem odpovídá jednomu řádku tabulky. Pro jednodušší práci s těmito objekty je také doporučeno vygenerovat si sadu metod pro nastavování hodnot vnitřních proměnných objektu (tzv. *setterů*, z anglického „set“ - stanovit, nastavit) a sadu metod pro získávání jejich hodnot (*getterů*, z anglického „get“ - získat).

Pro interakci s SQL databázemi se používají tzv. *SQL dotazy*. Tyto dotazy, jak už bylo dříve zmíněno, je možno volat na SQLite databázi přímo, ale díky knihovně Room je možné si toto opakované volání zjednodušit pomocí třídy nazvané *Database Access Object* (zkr. *DAO*, čti [dɑo]). Každé DAO se váže k jedné entitě, potažmo k jedné tabulce databáze. Mělo by implementovat základní SQL dotazy související s prací s touto tabulkou, mezi které patří *INSERT*, *UPDATE* a *DELETE*. Dále je pak možno přidávat další dotazy, jako například *SELECT* s různými parametry, které umožní efektivně získávat data z tabulek.

Poslední částí, která ještě nebyla zmíněna, je samotná *databáze*. Bavíme-li se v kontextu SQL databází, tabulky se vytvářejí až po založení databáze. Naproti tomu v knihovně Room se většinou vytvoří entity a teprve poté se tyto entity přiřadí k databázi. Instance databáze obsahuje DAO všech entit, které databázi tvoří [6].

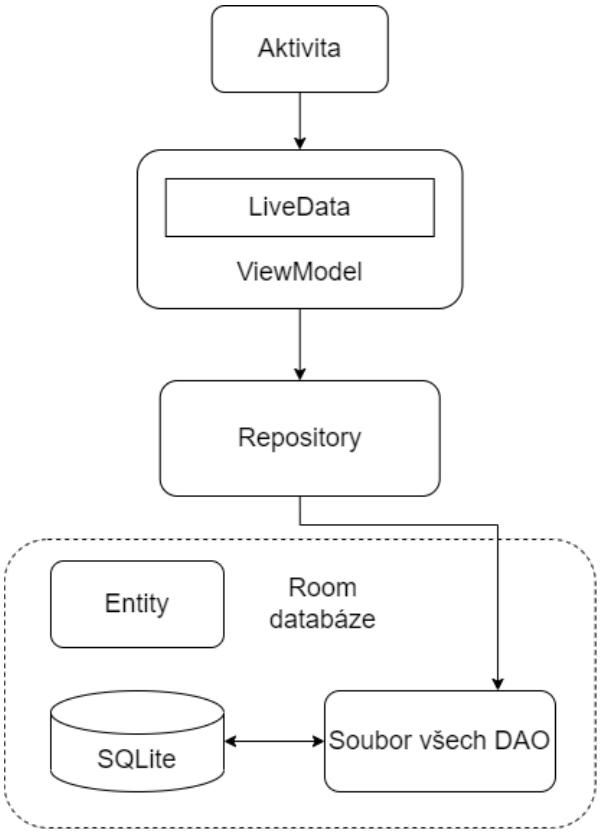
MVVM

Architektura MVVM spočívá v oddělení uživatelské části aplikace (*View*) a datové části aplikace (*Model*) a jejich vzájemné propojení pomocí komponenty *ViewModel*, která zajišťuje výměnu dat. Tato aplikace je postavená na architektuře MVVM.

MVVM je v této aplikaci implementovaná pomocí architektonických komponent Room, LiveData a ViewModel. Model v této aplikaci představuje samotná knihovna Room a v ní implementovaná databáze. View je logicky grafické uživatelské rozhraní aplikace. ViewModel je implementován pomocí architektonické komponenty ViewModel, ve které se nacházejí data ve formátu LiveData.

LiveData je prvek architektury Android a třída, která v sobě drží aktuální data. Tato data jsou pozorovatelná, tudíž jsme schopni jasně definovat, co se stane v aktivní části aplikace při změně těchto dat. LiveData jsou závislá na životním cyklu aktivity, a proto metody, které by se spustily při změně dat, se spouštějí pouze když je aktivita na popředí.

Mezi Roomem a ViewModelem ovšem stojí další komponenta, která umožňuje další stupeň abstrakce mezi databází a aplikací. Tato komponenta se nazývá *Repository*.



Obr. 1.4: Znázrnění architektury Model, View, ViewModel.

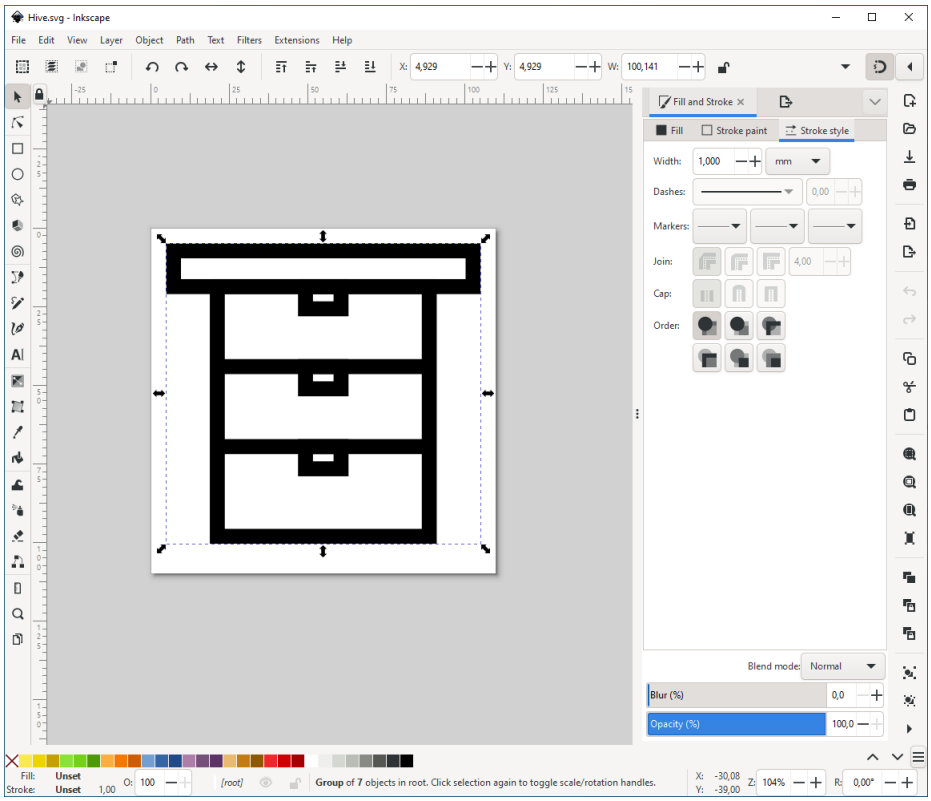
Room databáze se stará o lokální data uložená v SQLite databázi na zařízení pomocí objektů. Repository se stará o jednotný přístup k datům na lokální nebo vzdálené databázi. Ve ViewModelu se udržují veškerá data potřebná pro uživatelské rozhraní. Aktivita samotná poté sleduje LiveData ve ViewModelu [6].

1.2.6 GitHub

V tomto projektu bylo využito VCS Git. Zkratka VCS znamená v angličtině *Version Control System*, v češtině se používá název *verzovací software*. Git je integrovaný přímo do IDE Android Studio, takže synchronizace s online repozitářem byla prakticky bezbolestná. Repozitář tohoto projektu je veřejný a volně dostupný na platformě GitHub na adrese `https://github.com/pecarm/Bee_v0.3` [10, 11].

1.2.7 Inkscape

Inkscape je open-source software určený k vytváření vektorové grafiky. V tomto projektu byl využit k nakreslení ikon tlačítek použitých v aplikaci. (zdroj)



Obr. 1.5: Snímek obrazovky programu Inkscape

2 Koncepce aplikace

Tato kapitola se zaměřuje na myšlenkové postupy, které vedly ke konečné podobě aplikace. Bude popsána struktura grafického rozhraní a také základní rozložení dat.

Nejdůležitějším prvkem celé aplikace je bez debaty prvek zápisníku. Tento zápisník by měl být strukturovaný a měl by umožňovat rychlou navigaci mezi jednotlivými včelstvy. Dále při zobrazení včelstva by měly být hned dostupné nejdůležitější akce, jako zobrazení problémů s daným včelstvem, historie návštěv a základní informace o včelstvu, které byly zmíněny v předchozí kapitole.

I skutečný zápisník je vlastně jen médium, na které zapisujeme data. Proto i tento digitální zápisník bude potřebovat nějakou formu ukládání dat. Toto bude realizováno pomocí databáze.

Další funkcí aplikace je také schopnost zobrazovat statistiky, které se budou počítat z dat dříve zadaných uživatelem, nyní uložených ve výše zmíněné databázi. Zobrazované statistiky by měly obsahovat relevantní informace, ze kterých by mohl včelař chtít něco dedukovat a popřípadě změnit přístup v péči pro zajištění lepšího zdraví včelstva nebo vyšších výnosů včelích produktů.

2.1 Navigace v aplikaci

Níže popsané rozložení aplikace je inspirováno současnými populárními aplikacemi. Obsahuje hojně používané prvky a uživateli by mělo připadat používání této aplikace velmi intuitivní. Tuto podkapitolu je také možno využít jako uživatelský manuál.

2.1.1 Úvodní stránka

Při prvotních myšlenkách na návrh aplikace bylo hlavním cílem zobrazit důležitá data hned při otevření aplikace. Proto je hned při otevření aplikace vidět strukturovaný seznam upozornění a historie návštěv. Upozornění i návštěvy jsou každé zobrazeny ve vlastní záložce, mezi kterými je možné pohybovat se pomocí potáhnutí prstem.

Upozornění jsou zobrazena jako první v záložce o názvu *Upozornění*, aby měl uživatel okamžitě možnost bližší inspekce a případného plánování řešení. Jsou tříděna na základě závažnosti. U každého upozornění je napsáno datum, kdy bylo vytvořeno, název včelstva, ke kterému náleží, a také část samotného textu upozornění. Po kliknutí na položku v tomto seznamu je uživatel přesměrován na stránku, která zobrazuje informace o včelstvu, kterému toto upozornění náleží.

V záložce *Návštěvy* se zobrazují veškeré provedené návštěvy od počátku používání aplikace. Návštěvy jsou řazeny chronologicky od nejstarší po nejmladší. Kliknutí

na náhled návštěvy má stejný efekt, jako kliknutí na položku v seznamu upozornění, opět otevře stránku s informacemi o včelstvu.

Pro rychlou navigaci jsou na úvodní stránce také přítomná tlačítka, která umožňují vykonávání základních akcí. Jsou implementována pomocí prvku *floating action button* (FAB). Tato tlačítka umožňují přidávání lokalit včelnic, včelstev, záznamů návštěv a také zobrazování včelstev. FABy by byly velmi rušivé, pokud by byly stále přítomny na obrazovce. Proto jsou zastoupeny jedním tlačítkem, které všechna ostatní zobrazí. Toto tlačítko se nachází v pravém dolním rohu. Tlačítka nejsou vázána na záložky a jsou tedy přítomna neustále, což uživateli usnadňuje orientaci v aplikaci.

Dále je na úvodní stránce přítomen prvek *navigation drawer*, což je lišta, která se vysouvá z levé strany obrazovky. Obsahuje odkazy na základní aktivity aplikace. Tyto aktivity zahrnují aktivitu úvodní stránky, výběr včelstva, archiv a statistiky. Navigation drawer je implementován také ve všech aktivitách, které zobrazují data [6].

2.1.2 Výběr včelstva

Aktivita pro výběr včelstva je hojně využívána napříč celou aplikací. Na tuto aktivitu je možné se z úvodní stránky dostat pomocí stisknutí tlačítka *Zobrazit včelstvo*, stisknutím položky o stejném názvu v *navigation draweru* nebo stisknutím tlačítka *Přidat záznam návštěvy*, které v tomto případě použije tuto aktivitu na výběr včelstva, ke kterému se bude návštěva vázat. V ostatních dvou případech slouží tato aktivita k otevření aktivity zobrazující informace o včelstvu.

Nachází se zde seznam všech včelstev, která jsou seskupena na základě včelnice, ve které se nacházejí. Při klepnutí a podržení položky úlu se zobrazí vyskakovací menu s možnostmi *Archivovat* a *Odstranit*. Archivace včelstva znamená, že již nebude viditelné pod lokalitou včelnice, nicméně veškerá data včetně záznamů návštěv, medobraní a upozornění budou stále uložena v databázi aplikace a je možné je později zobrazit v archivu. Odstranění mluví samo za sebe, avšak je třeba dodat, že budou také odstraněny všechny informace náležící tomuto včelstvu.

Dále se zde nacházejí tlačítka na základní manipulaci se včelstvy, tj. přidávání nových lokalit včelnic a přidávání a odebírání včelstev. Opět jsou zpočátku skryta, odhalena jsou až po stisknutí tlačítka v pravém dolním rohu. Tato tlačítka jsou použitelná pouze ve výše zmíněných případech, tj. při zobrazení včelstva nebo po kliknutí tlačítka přidání návštěvy na hlavní obrazovce.

Jiné akce, které otevřou tuto aktivitu, zahrnují kliknutí tlačítka *Archiv* v postranní liště úvodní stránky nebo výběr včelstva nebo lokality pro zobrazení statistik.

2.1.3 Informace o včelstvu

Tato stránka slouží k zobrazování informací o vybraném včelstvu. Stránka je rozdělena do čtyř záložek, které každá zobrazují jiné informace týkající se včelstva.

V první záložce s názvem *Upozornění* se nachází všechna upozornění, opět seskupená dle závažnosti. U každého upozornění je zobrazeno datum a část textu upozornění. Na každou položku je možné kliknout a po kliknutí se zobrazí dialogové okno, ve kterém se zobrazí všechny informace včetně nezkráceného textu. V tomto dialogovém okně je možné upozornění archivovat. Archivace upozornění způsobí to, že již nebude vidět v seznamu upozornění, nicméně stále bude uloženo v databázi. Tlačítko *Ok* tento dialog zavře.

V záložce *Historie* se nacházejí veškeré záznamy návštěv náležící tomuto včelstvu. U každé návštěvy je opět uvedeno datum a část poznámky uvedené při zápisu návštěvy. Záznamy jsou seřazeny chronologicky od nejmladší po nejstarší. Po klepnutí na položku v seznamu se opět otevře dialog, ve kterém jsou uvedeny všechny informace pořízené při návštěvě.

Dále se zde nachází záložka *Medobraní* se seznamem medobraní. U každé položky je uvedeno datum, hmotnost v kilogramech a obsah vody v procentech. Klepnutí na záznam opět otevře dialog s kompletními informacemi.

V poslední záložce se nachází základní informace o včelstvu. Mezi informace patří agresivnost, síla včelstva, stavební pud, čisticí pud, bodavost a slídivost. Tyto jsou zobrazeny pomocí šestiúhelníkových ikon. Dále je zde zobrazeno číslo řady a rok narození matky společně s barevným čtvercem, který určuje barvu označení matky. Jako poslední se zde nachází tlačítko, po jehož stisknutí je otevřena mapová aplikace, která zobrazí polohu včelstva.

Nakonec se zde také nachází všudypřítomné tlačítko, které slouží k přidávání záznamu návštěvy a další podobné tlačítko k přidání medobraní. Stejně jako v předchozích případech se zobrazí až po stisknutí tlačítka v pravém dolním rohu.

2.1.4 Archiv

Archiv se vizuálně ani obsahem informací nijak neliší od stránky s informacemi o včelstvu. Jediný rozdíl mezi nimi je, že k archivovanému včelstvu nelze přidávat žádné záznamy. Interakce s prvky v seznamech funguje stále stejně.

Archivace upozornění jej pouze označí za splněné, v seznamu však zůstane stále.

2.1.5 Statistiky

Po kliknutí na položku *Statistiky* v postranní liště se otevře stránka, ve které uživatel vybírá rozsah zobrazovaných statistik.

Výběr rozsahu

V horní části okna se rozhoduje jaká včelstva do statistik zahrnout. Na výběr je zobrazení statistiky jednoho včelstva, celé lokality, nebo všech včelstev. Při výběru položky *Včelstvo* má možnost *Zahrnout archivovaná včelstva* vliv jen na to, z jakých včelstev si může uživatel vybrat. Po zaškrtnutí uvidí uživatel i včelstva, která dříve archivoval. Při volbě položky *Lokalitu* nebo *Všechna včelstva* aplikace automaticky zahrne do statistik archivovaná včelstva zvolené skupiny.

Ve spodní části pak uživatel vybírá časový rozsah. Jsou zde čtyři přednastavené možnosti, *Poslední měsíc*, *Poslední rok*, *Od začátku měsíce*, *Od začátku roku* a také možnost výběru přesného rozsahu uživatelem, a to pomocí dialogů zobrazených po stisknutí nyní aktivovaných tlačítek *Od* a *Do*. Aplikace automaticky kontroluje, jestli datum *Od* není časově po datu *Do* nebo naopak a uživatele upozorní.

Po stisknutí tlačítka *Pokračovat* má uživatel možnost vybrat včelstvo nebo včelnici na stránce pro výběr včelstva (viz výše), potažmo při výběru možnosti *Všechna včelstva* se rovnou otevře stránka se statistikami. Existují dvě varianty stránky se statistikami, pro včelstvo a pro skupiny včelstev. Každá z nich zobrazuje trochu rozdílné informace.

Stránka pro včelstva

Tato stránka je rozdělena do čtyř záložek a vzdáleně nám může připomínat stránku s informacemi o včelstvu. Podobně jako na té se i zde nacházejí 4 záložky: *Upozornění*, *Záznamy*, *Medobraní* a *Info*.

Záložka *Upozornění* zobrazuje statistiky o celkovém počtu upozornění, počtu vyřešených upozornění a stejné informace i pro jednotlivé závažnosti upozornění. Pod nimi se nachází rozbalovací seznam, kde jsou všechna upozornění vypsána, tříděna podle závažnosti. Opět jsou položky v seznamu klikatelné, po poklepání se zobrazí dialog s kompletním obsahem upozornění.

Na záložce *Záznamy* můžeme najít informace o celkovém počtu provedených návštěv, dále také dva lineární grafy, na kterých je zobrazen časový vývoj stavu zásob a mezerovitosti plodu. Jako poslední se na této záložce nachází chronologicky seřazený seznam všech provedených návštěv od nejmladší po nejstarší, které měly vyplněnou položku *Krmení/přikrmování*. Je to z toho důvodu, že včely lze krmit jak cukerným roztokem, tak vložením rámků s medem, který byl včelstvu odebrán dříve nebo byl vzat jinému včelstvu. Pro přehlednější počítání nákladů na krmení je tak asi nejlepší možnost zobrazit všechny záznamy a přenechat kalkulaci na včelaři samotném. Poklepání na položku opět otevře dialogové okno, tentokrát zobrazující veškeré detaily návštěvy, kdy bylo včelstvo krmeno.

V záložce *Medobraní* se nachází souhrnné informace o celkovém množství vytočeného medu a jeho průměrném obsahu vody. Pod nimi jsou pak dva grafy, první sloupcový zobrazuje množství vytočeného medu a druhý průměrný obsah vody.

V poslední záložce jsou uvedeny základní informace o včelstvu. Tato záložka je totožná se záložkou *Info* na stránce *Informace o včelstvu*.

Stránka pro skupiny

Na rozdíl od stránky pro včelstvo se zde nachází pouze tři záložky, *Upozornění*, *Záznamy* a *Medobraní*.

Záložka statistiky upozornění je naprosto shodná s odpovídající záložkou stránky pro včelstva. Seznam upozornění na poklepání reaguje také stejně.

Oproti tomu záložka se záznamy prošla několika změnami. Obsahuje celkový počet návštěv, navíc je zde uveden celkový počet návštěvních dní a opět je zde chronologicky řazený klikatelný výpis všech záznamů o krmení. Protože je očekáván poměrně malý počet včelstev, v řádu jednotek nebo maximálně desítek, byly vypuštěny grafy zobrazující stav zásob a mezerovitost plodu. Jednak jsou tato hodnocení subjektivní, např. slabší včelstvo může mít co do množství méně zásob, nicméně k jeho síle je to naprosto dostačující, a dále kvůli malému počtu včelstev nevznikne dostatečně zprůměrovaná hodnota, ze které by se dal usuzovat například vliv počasí na tyto ukazatele.

Poslední záložka, záložka medobraní, je opět stejná se stejnojmennou záložkou stránky statistik včelstev.

2.1.6 Aktivity pro zapisování dat

Samozřejmě je nutné, aby aplikace obsahovala prostředky, které umožní zapisování dat do databáze a jinou podobnou manipulaci s nimi. To je realizováno pomocí jednoúčelových aktivit, k nimž je možno přistoupit na očekávatelných místech. Kupříkladu je možné provést zápis nové návštěvy jak z úvodní obrazovky, tak z aktivity, která zobrazuje informace o včelstvu, nicméně není to možné provést z aktivity výběru včelstva, kde by tato možnost byla pouze rušivým prvkem.

V současné době existují aktivity na přidávání lokality včelnice, včelstva, záznamu návštěvy a medobraní. Upozornění je možné přidat v aktivitě záznamu návštěvy.

Jediný formulář, který je třeba osvětlit více, je formulář pro přidání lokality včelnice. Ten využívá službu GPS na to, aby získal aktuální polohu zařízení. Vizuálně se v této aktivitě nacházejí tři editovatelná textová pole. První odpovídá názvu lokality, následující zeměpisnou šířku (souřadnice sever/jih) a poslední zeměpisnou délku (souřadnice východ/západ). Aplikace určí polohu zařízení na žádost, a to

po stisku tlačítka *Určit současnou polohu GPS*. Při prvním použití aplikace bude žádat o přístup k datům polohy. Poté, co aplikace zjistí polohu, sama ji vypíše do odpovídajících textových polí. Kliknutí na tlačítko *Zobrazit na mapě* vyzve zařízení k otevření mapové aplikace na daných souřadnicích. Pokud nejsou žádné zadány, otevře se mapa na souřadnicích odpovídající budově T12 FEKT VUT. Vyhledávání místa pomocí názvu přímo z aplikace by bylo možné pouze za použití Places API společnosti Google, za jehož použití je nutné platit. Proto bylo od tohoto upuštěno. V případě, že by uživatel potřeboval vytvořit lokalitu, která by neodpovídala jeho GPS poloze, je možné toto obejít přečtením souřadnic z jiného mapového zdroje a následně je přepsat do aplikace [13].

2.2 Databáze

Už při zrodu aplikace byl plán ukládat uživatelská data do relační databáze. Jak již bylo zmíněno, aplikace využívá architekturu *Room, ViewModel & LiveData*, kterého konkrétní implementace bude více rozebrána v následujících kapitolách. Nicméně nyní je možné ukázat návrh, jak tato databáze vypadá, jaké obsahuje tabulky a co uchovává za informace. Současná databáze sestává z pěti tabulek. Tabulka pro ukládání včelnic, tabulka na včelstva, na záznamy návštěv, upozornění a medobraní.

Diagram databáze se nachází níže včetně reálného názvu všech sloupců. Primární klíče u všech tabulek jsou vedeny jako automaticky se inkrementující kladná celá čísla. Tabulku *hives_location_table* si můžeme představit jako nejvyšší v hierarchii. Na ni navazuje tabulka *hive_table*, jsou provázány pomocí cizího klíče. Nakonec na tabulku se včelstvy navazuje zbytek tabulek, jmenovitě *record_table*, *alert_table*, a *honey_harvest_table*, které jsou s ní opět provázány cizími klíči.

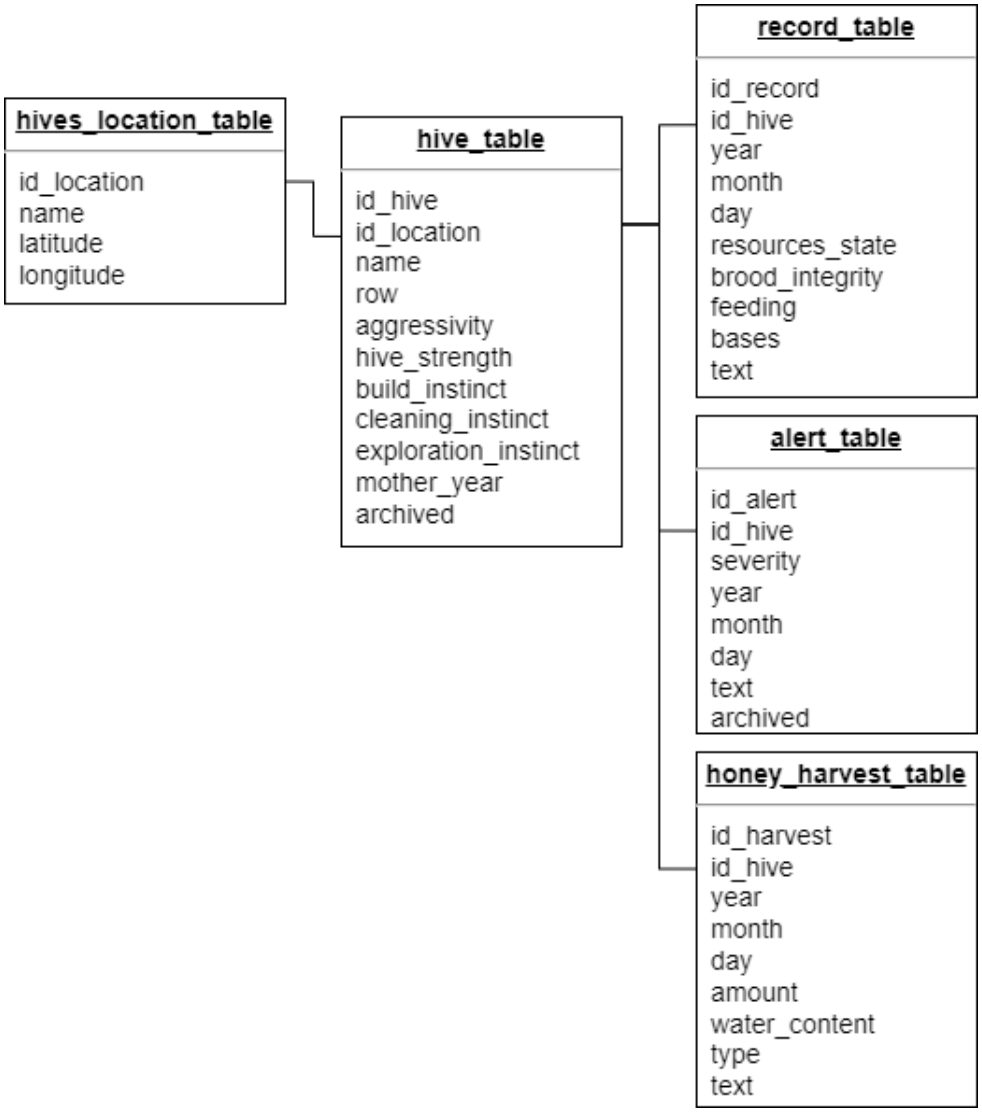
Tabulka včelnic *hives_location_table* uchovává pouze nezbytná data, a to název a souřadnice.

Dále bylo důležité se rozhodnout, které vlastnosti včelstva jsou prakticky stálé nebo neměnné v průběhu roku, a které stojí za to zaznamenávat pravidelně. Po konzultaci se včelaři bylo učiněno rozhodnutí, že v tabulce včelstev *hive_table* se budou uchovávat vlastnosti agresivnost, síla včelstva, stavební pud, bodavost, slídivost, čisticí pud, rok narození matky a v tabulce *record_table* pak stav zásob, mezerovitost plodu, informace o krmení nebo přikrmování a počet přidaných nebo odebraných rámků. U záznamu návštěvy se také uvádí datum, kdy byla návštěva provedena a také text poznámky pro nečekaná zjištění. Pro interní účely aplikace je v tabulce včelstev také sloupec určující, zda se nachází v archivu nebo ne.

Tabulka upozornění *alert_table* obsahuje informace o závažnosti problému, datum pořízení a samotný text upozornění. Závažnost je ukládána jako celé číslo,

hodnota 1 představující nejvyšší, 2 střední a 3 nejnižší závažnost. Tato tabulka opět obsahuje sloupec o archivaci.

Poslední tabulka *honey_harvest_table* slouží na zápis medobraní. Ukládá informace o množství, obsahu vody, typu, datu pořízení a text pro poznámky.



Obr. 2.1: Znázrnění architektury Model, View, ViewModel.

3 Realizace aplikace

V sekci *Koncepce aplikace* byl již poměrně dopodrobna popsán návrh aplikace, ať už po grafické stránce, tak částečně i po stránce backendu a ukládání dat. Toto uživatelské rozhraní je trochu odlišné od toho, které bylo původně navrženo vedoucím práce při konzultacích. Změny byly provedeny z důvodu zjednodušení a zmenšení počtu interaktivních prvků na obrazovce a zlepšení jejich přístupnosti. Pro zopakování z dřívějších kapitol, veškerý kód tohoto projektu je dostupný online na repozitáři `https://github.com/pecarm/Bee_v0.3`.

3.1 Databáze

Nejprve je nutno představit práci s daty, aby později bylo možno lépe vysvětlit jejich zobrazování. Struktura databáze samotné již byla popsána dříve, nicméně je nutné také ukázat, jak je možné toto naprogramovat.

3.1.1 Room databáze

Jak již bylo popsáno v předchozích kapitolách, Room databáze obsahuje SQLite databázi, definici databáze, entity a DAO. Každý z těchto prvků je třeba popsat a uvést, že jsou součástí Room databáze. K tomu slouží *anotace*. Anotace popisují prvky tříd a třídy samotné. Třídu lze anotovat jako databázi (`@Database`), entitu (`@Entity`), nebo DAO (`@Dao`).

V anotaci databáze je třeba uvést i entity, které tuto databázi tvoří a verzi, ve které databáze je. Pokud dojde k jakékoli změně ve struktuře databáze, jako například přidání nebo odebrání sloupců v entitách, přidání nového dotazu do DAO, je nutné změnit číslo verze, aby byla databáze překompilována. Změna verze databáze znamená ztrátu všech uložených dat.

U entity není třeba nic více označovat, avšak pokud chceme přidat cizí klíče, nebo zajistit, aby název tabulky nebyl shodný s názvem třídy entity, můžeme přidat do anotace také položku `tableName` respektive `foreignKeys` s anotací `@ForeignKey`. V rámci entity je nutné označit jednu vnitřní proměnnou třídy jako `@PrimaryKey`, aby bylo jasno, který sloupec odpovídá primárnímu klíči tabulky, popřípadě přidat do anotace položku `primaryKeys`, pomocí které můžeme označit více sloupců jako primární klíč. SQLite databáze nemají nativní datový typ, pomocí kterého lze zaznamenávat datum, proto jsou u některých tabulek přítomny sloupce s hodnotou roku, měsíce a dne. Pro jednodušší práci s objekty, které v sobě nesou informaci o datu, byly implementovány dodatečné *get* metody v podobě `getDate()`. Nacházejí se v tabulkách *record_table*, *honey_harvest_table* a *alert_table*. Ty vracejí instanci třídy

LocalTime, která implementuje rozhraní *Comparable*, a je proto možné seznamy těchto objektů jednoduše řadit na základě data.

DAO stačí označit pouze anotací `@Dao`. Při deklaraci metod pro budoucí SQL dotazy je však nutné nejprve anotovat tyto metody pomocí `@Insert`, `@Update` a `@Delete` pro vytvoření příslušného SQL dotazu. V případě, že chceme vytvořit jiný dotaz, je nutno tuto metodu anotovat pomocí `@Query` a do parametrů zadat SQL dotaz ve formě řetězce [6].

3.1.2 Repository

V repozitáři se v současné době vyskytuje pouze přístup k lokální databázi a definuje tak jednotné rozhraní, jak přistupovat ke všem datům nezávisle na tabulce, ze které tato data pochází. V lokálních proměnných jsou uloženy instance DAO ke všem tabulkám a LiveData obsahující všechny položky každé tabulky.

Zároveň jsou zde pro každou tabulku definovány metody pro interakci s databází. Tyto metody by při běžném spuštění pravděpodobně způsobily zamrznutí uživatelského rozhraní, a tudíž snížení pohodlnosti uživatele. Proto bylo využito *asynchronních metod*, které mají za úkol provést operaci s databází v jiném než hlavním vlákně [6].

3.1.3 ViewModely

Co se týče ViewModelů, těch je implementováno více v závislosti na aktivitě, ke které se vážou. Cíl byl takový, aby měly aktivity přístup pouze k těm metodám, které jsou nutné pro její funkci a tímto zvýšit bezpečnost aplikace [6].

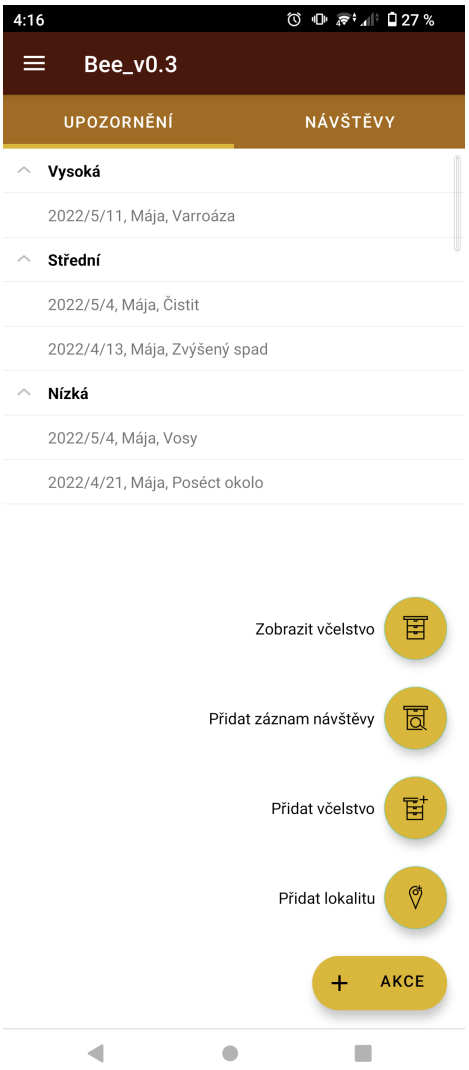
3.2 Uživatelské rozhraní

V této části bude podrobněji popsáno rozložení prvků v jednotlivých aktivitách a fragmentech. Budou také blíže vysvětleny jednotlivé typy *layoutů*, které byly nastíněny v kapitole *Teoretická část práce*. Také budou vysvětleny funkce a interaktivita jednotlivých *views* v aktivitách a fragmentech. Pořadí aktivit v textu bude stejné, jako pořadí aktivit v navigation draweru, za nimi budou aktivity statistik a nakonec aktivity sloužící k přidávání prvků do databáze.

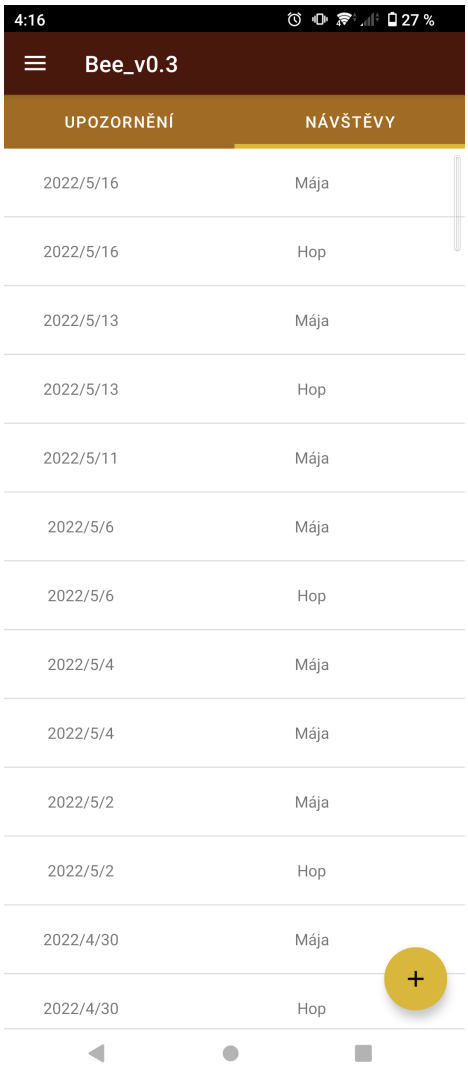
3.2.1 MainActivity

MainActivity je aktivita, která se zobrazí při spuštění aplikace. Její layout je popsán souborem *activity_main.xml*. Bylo potřeba využít poměrně velké množství views, abychom se dostali na vzhled, který byl popsán dříve v textu. Rodičovský layout,

který zobrazuje všechny ostatní views na této obrazovce musí být typu *DrawerLayout*. Tento layout je navržený k tomu účelu, aby umožňoval vysouvání bočního interaktivního panelu známého jako *navigation drawer*. Ten často obsahuje důležité položky jako nastavení, často používané funkce a podobné [6].



Obr. 3.1: Fragment DashboardFragment v aktivitě MainActivity se zobrazenými FABy (vlevo)

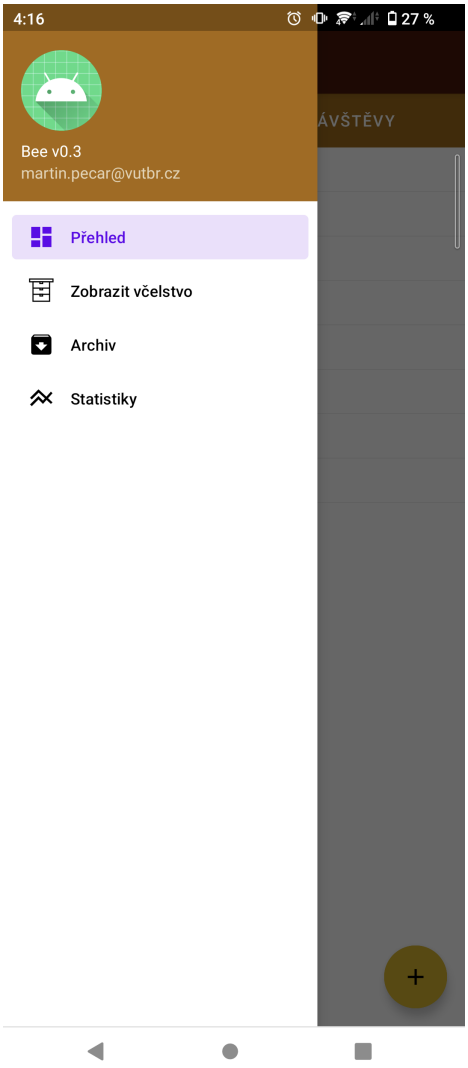


Obr. 3.2: Fragment TimelineFragment v aktivitě MainActivity (vpravo)

Navigaion drawer

NavigationView musí existovat na stejné úrovni XML souboru DrawerLayoutu, jako ostatní prvky na obrazovce. DrawerLayout umožňuje vysouvání bočního panelu

z jednoho nebo obou stran obrazovky. Proto je v NavigationView třeba definovat vlastnost `android:layout_gravity`, která určuje, kde se toto view bude nacházet. Při nastavení hodnoty `start` se NavigationView bude nacházet na levé straně obrazovky.



Obr. 3.3: Navigation drawer v aktivitě MainActivity

NavigationView obvykle obsahuje hlavičku a menu interaktivních položek. Tyto se udávají pomocí vlastností `app:headerLayout` pro hlavičku a `app:menu` pro menu. Tyto jsou poté nastaveny na názvy layoutů, které jsou za ně zodpovědné.

Kvůli existenci navigation draweru musí třída MainActivity implementovat rozhraní `OnNavigationItemSelectedListener` rozhraní `NavigationView`. Rozhraní se poté implementuje metodou `onNavigationItemSelected()`. V této metodě se implementuje akce, která proběhne po stisknutí položky v menu navigation draweru.

Implementace je provedena pomocí spuštění nové aktivity. Aktivita se spouští pomocí instance třídy `Intent`. Intent je abstraktní popis operace, která má být provedena. Aktivita předá intent operačnímu systému a ten podle něj zareaguje. Intent může být využit ke spouštění lokálních aktivit nebo i ke spouštění externích aplikací (například otevření prohlížeče). V tomto případě dojde k vytvoření příslušných aktivit, které budou popsány dále v textu. Do intentu se mohou přidávat další informace ve formě páru klíče a hodnoty. Toto je možné pomocí metody `putExtra()`. Je možné předávat široké množství různých objektů od primitivních datových typů po seznamy. Za předpokladu, že je třeba předat větší množství informací, je doporučováno používat instanci třídy *Bundle*, do které je možné vkládat objekty ve stejném formátu klíče a hodnoty jako při použití `putExtra()`. Bundle se poté k intentu přiřadí pomocí metody `putExtras()` [6].

Hlavní obrazovka

Zde se zobrazuje zbytek interaktivních prvků main activity. Ty jsou uloženy v rozložení *ConstraintLayout*. Toto rozložení umožňuje efektivně upravovat rozložení a velikost views na obrazovce nezávisle na velikosti a poměru stran výsledné obrazovky. V současné době je toto rozložení velmi populární. Constraint layout například umožňuje vázat okraje jednotlivých views k sobě nebo ke krajům obrazovky nebo nadřazeného view a také nastavovat velikost okrajů, čili pevnou vzdálenost daného view od kraje obrazovky nebo nadřazeného view.

Jak již bylo popsáno, vpravo dole v ConstraintLayoutu se nachází FAB, který po rozkliknutí zobrazí další FABy zodpovědné za akce. Nalevo od tlačítek se při zobrazení objeví i popisky funkcí, které tato tlačítka vykonávají. Popisky jsou v celé aplikaci realizovány pomocí prvku *TextView*, který představuje jednoduché textové pole.

Tlačítka podle popisku spouštějí jim příslušné aktivity. Výjimkami z tohoto jsou hlavní FAB, který je zodpovědný za zobrazování a skrývání ostatních FABů a jejich popisků. U popisků je toto provedeno pomocí změny vlastnosti `visibility` na `VISIBLE`, respektive `GONE` a u FABů se využívají metody `show()` a `hide()`. Dále pak FABy zodpovědné za zobrazení včelstva a přidání kontroly spouštějí aktivitu *SelectActivity*, která umožňuje uživateli výběr včelstva. Aby SelectActivity poznala, co se má stát po výběru včelstva, předává jí každé z těchto dvou tlačítek jinou hodnotu s klíčem `"TARGET"`. Pro zobrazení je předána hodnota `"view"`, pro přidání záznamu `"record"`.

Dále se uvnitř ConstraintLayoutu nachází *LinearLayout*. Ten slouží k zobrazování views v jednom směru. V pořadí shora dolů se zde nacházejí views *Toolbar*, který je zodpovědný za zobrazování horní lišty aplikace, na které je v tomto oka-

mžiku zobrazeno jméno aplikace a ukazatel existence navigation draweru. Ta je značena standardně pomocí tří krátkých horizontálních čar. Pod toolbarem se nachází *TabLayout*, což je ukazatel existence záložek a také určuje současnou polohu uživatele v těchto záložkách. Jako poslední se zde nachází *ViewPager2* (dále jen ViewPager), evoluce původního *ViewPager*. Ten v tomto případě slouží k zobrazování fragmentů, které jsou zodpovědné za jednotlivé záložky. ViewPager umožňuje přechod mezi těmito fragmenty pomocí horizontálního potáhnutí prstem. O plnění ViewPageru se stará `CustomAdapter`

Každá aktivita musí implementovat metodu `onCreate()`. Jedná se o metodu zděděnou ze třídy *AppCompatActivity*, proto je třeba implementaci anotovat pomocí `@Override`, jinými slovy metoda se předefinuje. V metodě `onCreate()` se běžně inicializují views, které se nachází v layoutu aktivity a přiřazují se jim funkce. Inicializace view v kódu se provádí metodou `findViewById()`. Tato metoda vrací instanci třídy dceřinné třídě *View*, je tedy třeba ji přetypovat na instanci třídy odpovídající požadovanému View. Až po inicializaci je možné programově přidávat views funkce. Jsou zde také definováni pozorovatelé (observers) jednotlivých tabulek. Při změně dat se uloží aktuální data z tabulek do seznamů hlavní metody jako lokální proměnné ve formě seznamů objektů `List<>`, jeden pro každou tabulku databáze.

Dále je zde předefinována dříve zmíněná metoda `onNavigationItemSelected()`. Je implementována tak, že spouštějí aktivity dle názvu, tj. SelectActivity pro *Zobrazit včelstvo* a *Archiv* a ScopeSelectActivity pro *Statistiky*. Další implementace navigation draweru v aplikaci spouštějí také aktivitu MainActivity pro *Přehled* a předávají intentu označení `Intent.FLAG_ACTIVITY_CLEAR_TOP`, který před spuštěním volané aktivity dokončí veškeré aktivity běžící nad úrovní této aktivity. Například uživatel může ze ShowHiveActivity spustit SelectActivity stisknutím tlačítka *Zobrazit včelstvo*. Bez zmíněného označení by se po stisku systémového tlačítka zpět dostal na ShowHiveActivity, takto bude vrácen na MainActivity.

Jako poslední je předefinována i metoda `onBackPressed()`. Ta nyní reaguje na stav navigation draweru a FABů. Za předpokladu, že je navigation drawer otevřen, klepnutí na systémové tlačítko *Zpět* jej zavře a obdobná akce proběhne i v případě, že jsou zobrazeny FABy, tj. tlačítko zpět je skryje [6].

CustomAdapter

Adaptér obecně zajišťuje plnění views daty. Ať už se jedná o vložení fragmentu do ViewPageru nebo o naplnění seznamu lokálně uloženými položkami. `CustomAdapter` takto přidává do aktivity *MainActivity* dva fragmenty, *DashboardFragment* a *HistoryFragment*. Při vytvoření instance adaptéru jsou kromě nezbytných parametrů předány i výpisy tabulek databáze (viz pozorovatelé výše). Adaptér pak přiřadí zá-

ložkám ViewPageru instance výše zmíněných fragmentů. Práce s daty ve fragmentu je popsána v následujících podkapitolách.

DashobardFragment

První záložka s názvem *Dashboard* zobrazuje veškerá upozornění a obsahuje instanci DashboardFragmentu. V něm se nachází *ExpandableListView*, ve kterém jsou tato upozornění rozdělena podle závažnosti na *High*, *Medium* a *Low*. Tento ExpandableListView je plněn pomocí adaptéru `DashboardCustomExpandableListAdapter` (viz níže). Data potřebná k plnění tohoto view jsou fragmentu předána CustomAdapterem (viz výše).

Instanci fragmentu získáváme pomocí námi definované statické metody pojmenované `newInstance()`, které jako argumenty předáváme data k plnění views. V té je vytvořena prázdná instance fragmentu. K fragmentu je možné připojit argumenty metodou `setArguments()` ve formě instance třídy Bundle.

Metoda `onCreateView()` má za úkol k fragmentu přiřadit požadovaný layout a přečíst data předaná v argumentech. Na rozdíl od metody `onCreate()` aktivity, zde se neinicializují views fragmentu a to z toho důvodu, že v průběhu této metody ještě neexistují a vzniknou až po přiřazení layoutu.

O inicializaci a plnění daty se stará metoda `onViewCreated()`, ve které můžeme s fragmentem pracovat podobně jako s aktivitou [6]. Je zde také nadefinována metoda, která určuje reakci na poklepání na prvek upozorněné v ExpandableListView. Po poklepání se otevře ShowHiveActivity zobrazující přehled celého včelstva, kterému toto upozornění náleží [6].

DashboardCustomExpandableListAdapter

DashboardCustomExpandableListAdapter je založen na třídě *BaseExpandableListAdapter* a využívá data včelstev a záznamů. Před plněním daty je třeba definovat seznam skupin, v tomto případě textový seznam závažností a seznam zobrazovaných objektů, čili seznam instancí Alert. Kvůli zobrazení relevantních dat se seznam upozornění filtruje tak, aby obsahoval pouze nearchivovaná upozornění. K tomuto filtrování je využita funkce `stream().filter()`, kde pomocí lambda výrazu bylo možné vyfiltrovat ze seznamu všech upozornění pouze ta, která obsahují požadované `id_hive`. Tato metoda filtrování je využita všude v projektu. Aby bylo možné získat zpět data opět ve formě seznamu, byla využita metoda `.collect(Collectors.toList())`.

Je zde také nutné předefinovat velké množství metod, nicméně se zaměříme pouze na ty nejdůležitější. Metody `getGroup()` a `getChild()` vracejí instance objektů,

které odpovídají pozici v ExpandableListView. O plnění položek se pak starají metody `getGroupView()` a `getChildView()`. V těch je nadefinováno, jaký layout se použije pro zobrazení. Layout skupiny bývá většinou definován tak, aby byl vizuálně výraznější. V těchto layoutech se nachází TextView, které využijeme k zobrazení dat. K inicializaci opět použijeme metodu `findViewById()` a následně nastavíme text tak, aby zobrazoval buď závažnost, nebo náhled upozornění [6].

TimelineFragment

Ve druhé záložce se nachází seznam všech provedených kontrol. Chronologické seřazení prvků je provedeno pomocí metody `Collections.sort()`. Tento výpis je zobrazován pomocí ListView, které je plněné pomocí adaptéru `TimelineAdapter`. Po klepnutí na položku v seznamu je uživatel stejně jako na DashboardFragmentu přesměrován na ShowHiveActivity, kterému záznam náleží [6].

TimelineAdapter

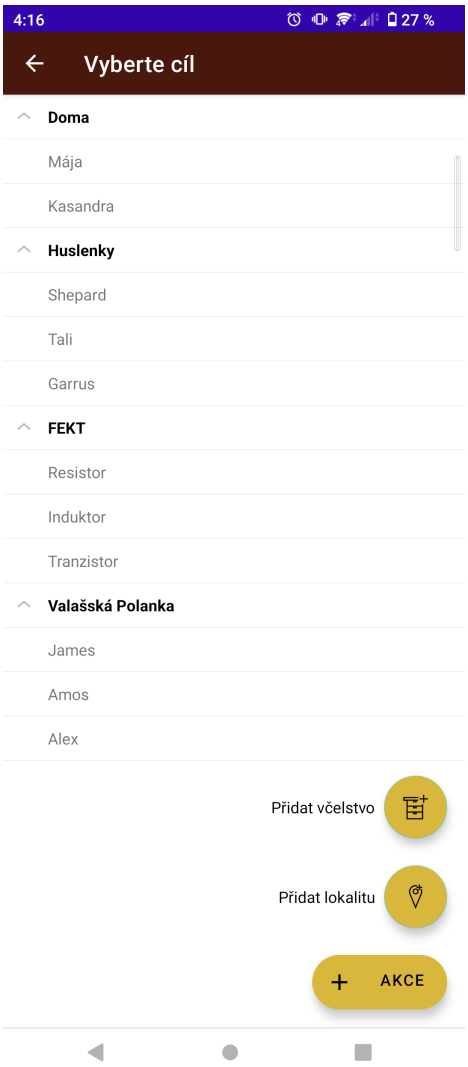
TimelineAdapter rozšiřuje třídu `SimpleAdapter` a jako data přebírá seznam párů klíč-hodnota, datový typ *List<HashMap<»*. Takto je možné v ListView zobrazit velké množství informací z různých zdrojů. V tomto případě je to jméno úlu a záznam. ListView slouží k zobrazování seznamu velkého množství položek se stejnými vlastnostmi. Při jeho plnění je třeba definovat vzhled položek pomocí layoutu, zde se vychází z `adapter_view_dashboard_timeline`, ve kterém jsou dvě TextView, první slouží pro zobrazení data, druhé pro název včelstva [6].

3.2.2 SelectActivity

Aktivita sloužící k výběru včelstva, respektive včelnice, je v rámci této aplikace nejčastěji odkazovanou aktivitou. Vizuálně tato aktivita obsahuje pouze *ExpandableListView* s existujícími včelstvy a také dva FABy, které slouží k přidání včelstva a včelnice.

ExpandableListView je opět plněno podobně jako v DashboardFragmentu pomocí adaptéru `SelectCustomExpandableListAdapter`, který se liší tím, že jako skupiny zobrazuje lokality včelnic a jako položky včelstva a že umí reagovat na účel spuštění aplikace.

V metodě `onCreate()` se opět nacházejí inicializace views, observery LiveData repozitáře a definice různých metod typu *onClick*, které jsou popsány dále. Metoda `onBackPressed` opět kontroluje, zdali jsou zobrazené FABy a případně je po stisku tlačítka *Zpět* skryje. Funkce FABů je stejná jako u odpovídajících FABů v MainActivity.



Obr. 3.4: Lokality a včelstva zobrazené v aktivitě SelectActivity

Při vytvoření instance této aktivity je jí pokaždé předána informace o účelu použití. Tyto cíle jsou definovány řetězci *„view“*, *„record“*, *„archive“*, *„stats_hive“*, *„stats_hive_all“*, *„stats_location“* a *„stats_location_all“*. Na jejich základě poté aktivita mění své chování. V případě „view“ jsou FABy klikatelné a po klepnutí na položku včelstva je uživatel přesměrován na ShowHiveActivity. V případě „record“ jsou FABy stále klikatelné a klepnutí na položku včelstva spustí AddRecordActivity pro zvolené včelstvo. Ve všech zbývajících účelech jsou FABy deaktivované. V případě „archive“ otevře klepnutí na včelstvo ShowHiveActivity s příznakem `"IS_ARCHIVE"` nastaveným na `true` (viz ShowHiveActivity) a v rozbalovacím seznamu budou zobrazena pouze archivovaná včelstva. V případech „stats_hive“ a „stats_hive_all“ klepnutí na včelstvo otevře StatsHiveActivity s tím, že při druhé možnosti lze vybírat ze všech včelstev včetně archivovaných. V posledních dvou pří-

padech aktivita reaguje na poklepání na název včelstva a otevře StatsActivity s tím, že možnost „stats_location_all“ zahrne do výpočtu statistik i archivovaná včelstva dané lokace.

Poslední implementovaná funkce je klepnutí a přidržení položky včelstva. Tato akce je dostupná jen, když účel odpovídá *„view“* nebo *„record“*. Funkce je vysvětlena v kapitole *Navigace v aplikaci*. Archivace se provádí tak, že na zvolené včelstvo se zavolá metoda `setArchived(true)` a poté se odpovídající řádek databáze updatuje metodou `selectViewHiveModel.update(hive)` [6].

3.2.3 ShowHiveActivity

ShowHiveActivity slouží k zobrazování včelstva. Obsahuje několik záložek, které jsou stejně jako v MainActivity tvořené párem TabLayout a ViewPager2. Opět existuje speciální adaptér `ShowHiveAdapter`, který toto rozložení plní fragmenty, jejichž jména jsou `AlertsFragment`, `HistoryFragment`, `HoneyHarvestFragment` a `InfoFragment`. ShowHiveAdapter pracuje na stejném principu jako CustomAdapter. Jako poslední jsou zde všudypřítomné FABy na záznam návštěvy a medobraní.

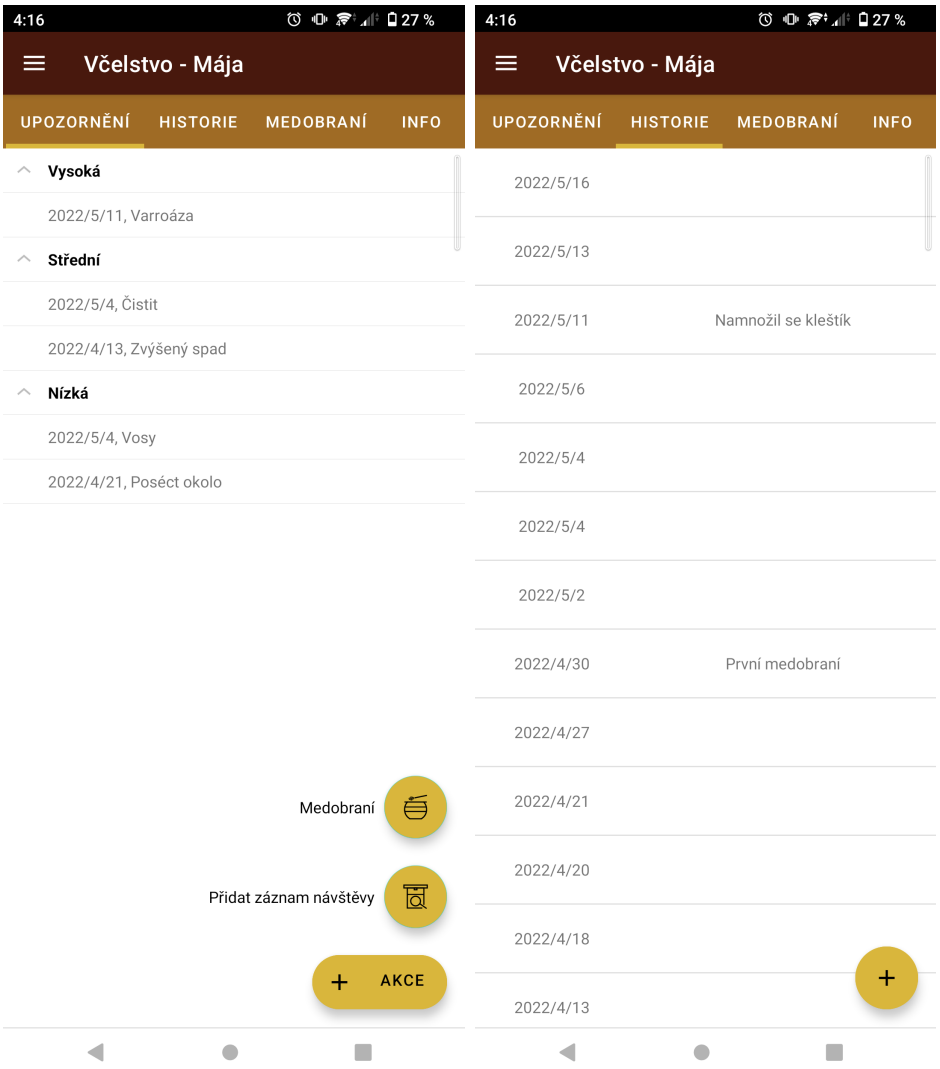
Tato aktivita čte příchozí parametry `"HIVE_ID"` a `"IS_ARCHIVE"`. První zmíněný určuje včelstvo, jehož informace budou zobrazeny a druhý zaručuje jejich správnou filtraci. Jestliže je `"IS_ARCHIVE"` předán jako *„true“*, pak se budou zobrazovat všechna upozornění včetně archivovaných a FABy budou deaktivovány.

AlertsFragment

AlertsFragment je vzhledově a implementací velmi podobný DashboardFragmentu. Záložka *Upozornění* obsahuje upozornění, které se týkají pouze zvoleného včelstva. Zobrazování archivovaných upozornění určuje příznak `"IS_ARCHIVE"`, jehož hodnota se předává *AlertsCustomExpandableListAdapteru*. Ten pracuje podobně jako DashboardCustomExpandableListAdapter s tím rozdílem, že zvládá dle potřeby filtrovat archivovaná upozornění.

HistoryFragment

Záložka *Historie* je opět velmi podobný TimelineFragmentu z MainActivity, je vizuálně identická s tím rozdílem, že obsahuje pouze záznamy týkající se vybraného včelstva. ListView tedy plní `HistoryAdapter`, který zobrazuje mírně odlišné informace oproti těm zobrazeným v TimelineFragmentu, ale stejným způsobem jako TimelineAdapter.



Obr. 3.5: Fragment AlertFragment v aktivitě ShowHiveActivity (vlevo)

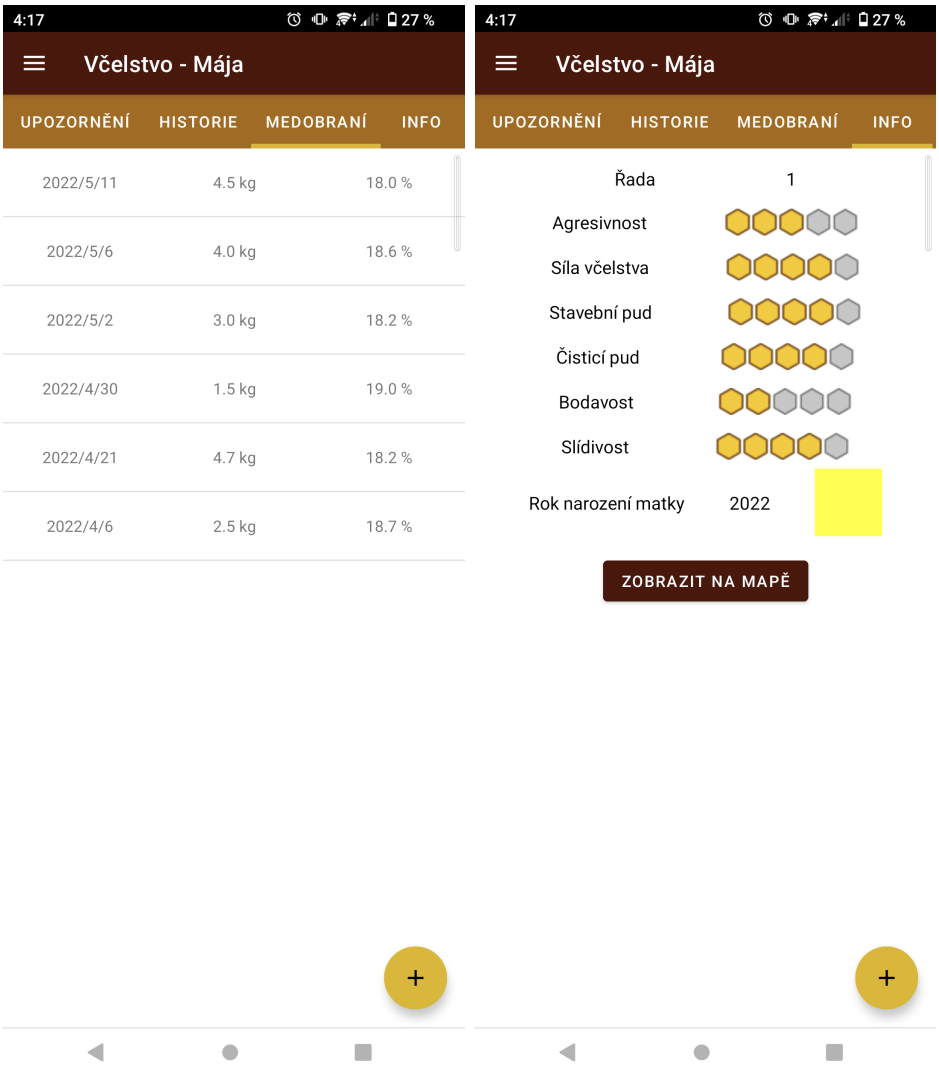
Obr. 3.6: Fragment HistoryFragment v aktivitě ShowHiveActivity (vpravo)

HoneyHarvestFragment

HoneyHarvestFragment pracuje prakticky stejně jako HistoryFragment jen s využitím *HoneyHarvestAdapteru* na správné plnění daty.

InfoFragment

Záložka *Stats* slouží k zobrazování neměnných vlastností včelstva. Ty jsou zobrazovány pomocí páru TextView a view jménem *RatingBar*, který představuje hodnocení pomocí šestiúhelníkových ikon. Dále se zde nachází informace o řadě, ve které se včelstvo nachází [6].



Obr. 3.7: Fragment HoneyHarvestFragment v aktivitě ShowHiveActivity (vlevo)

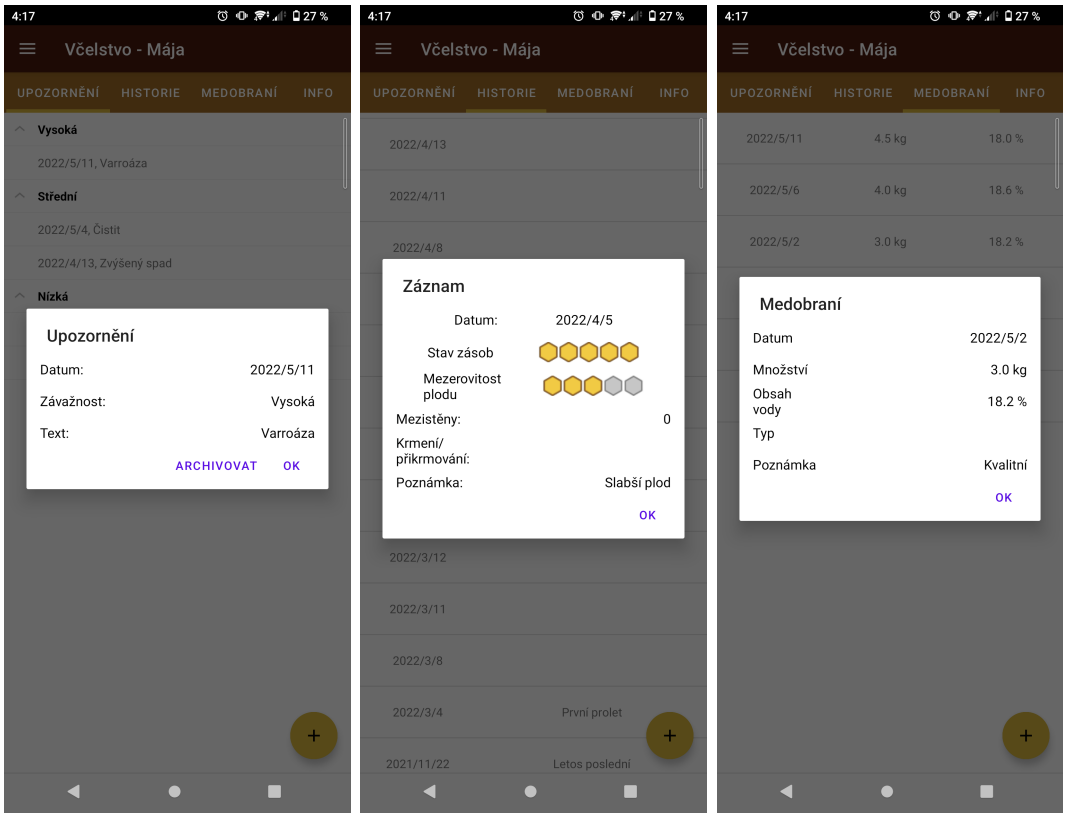
Obr. 3.8: Fragment InfoFragment v aktivitě ShowHiveActivity (vpravo)

Po stisknutí tlačítka *Zobrazit na mapě* se vytvoří nový intent akce, který přeloží *Uri* lokality. Uri se vytváří metodou `Uri.parse()`, které se jako parametr předává řetězec ve formátu „geo:0,0?q=latititude,longitude(name)“, kde latitude je zeměpisná šířka, longitude zeměpisná délka a name je text zobrazovaný u špendlíku na mapě [6].

Dialogy

Po klepnutí na položky v seznamech v prvních třech fragmentech se zobrazí informativní dialogy s kompletním obsahem záznamu. Pouze jsou do nich předány data

a s nimi se následně pracuje. Jsou tak odlišné od dialogu *AddAlertDialog*, který i vrací data nazpět rodičovské aktivitě.



Obr. 3.9: Dialog ShowAlertDialog v aktivitě ShowHiveActivity (vlevo)

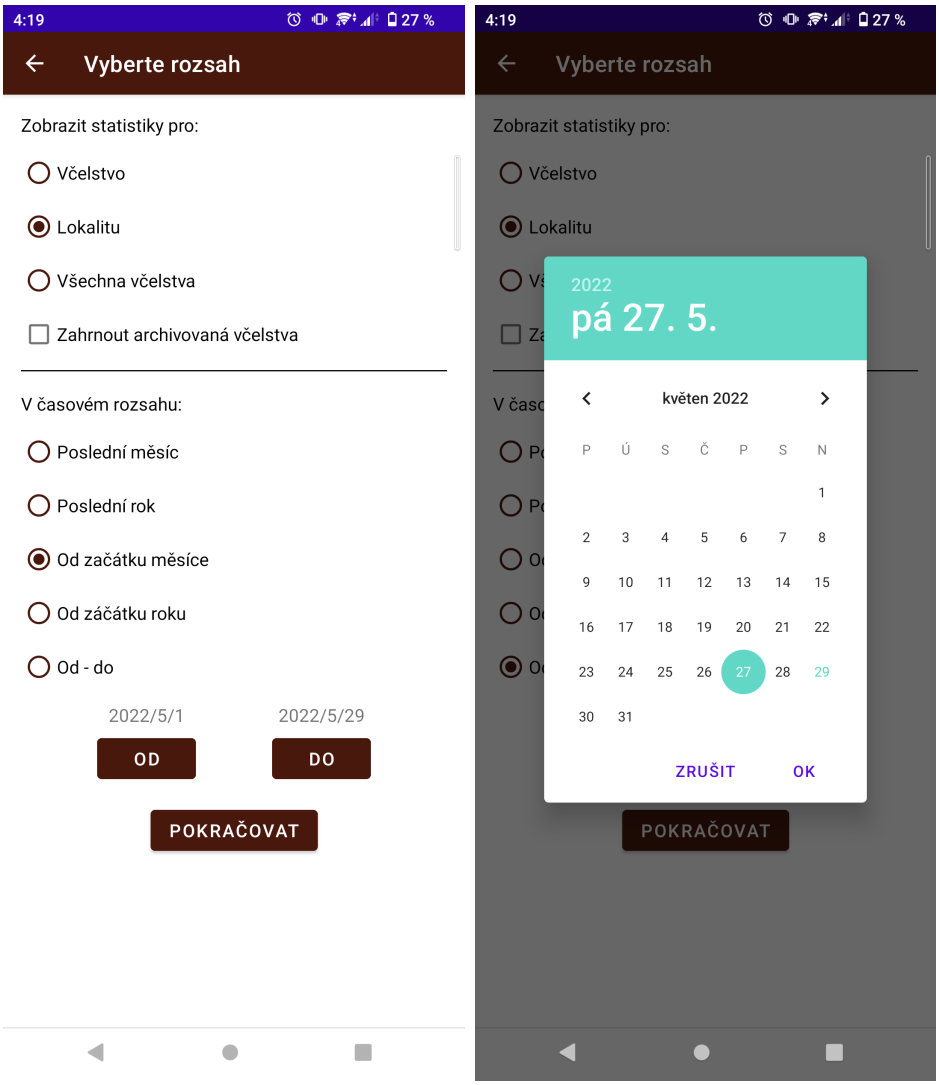
Obr. 3.10: Dialog ShowRecordDialog v aktivitě ShowHiveActivity (uprostřed)

Obr. 3.11: Dialog ShowHarvestDialog v aktivitě ShowHiveActivity (vpravo)

3.2.4 ScopeSelectorActivity

Zde si uživatel vybírá, z jakého zdroje se budou počítat statistiky. Pro výběr jednoho prvku z mnoha je zde využit *RadioButton*. Ty jsou v rámci layoutu umístěny v *RadioGroup*, která je uzavírá do skupiny. RadioGroups umožňují existenci více vzájemně nezávislých skupin RadioButtonů v jednom layoutu.

Protože se v této aktivitě vyskytuje mnoho prvků pod sebou, bylo nutné zajistit, aby byly všechny přístupné i na zařízeních s menší obrazovkou. *ScrollView* je rodičovský layout, který zajišťuje možnost posouvat se v aktivitě i za předpokladu, že se celá na obrazovku nevejde [6].



Obr. 3.12: Aktivita ScopeSelectorActivity (vlevo)

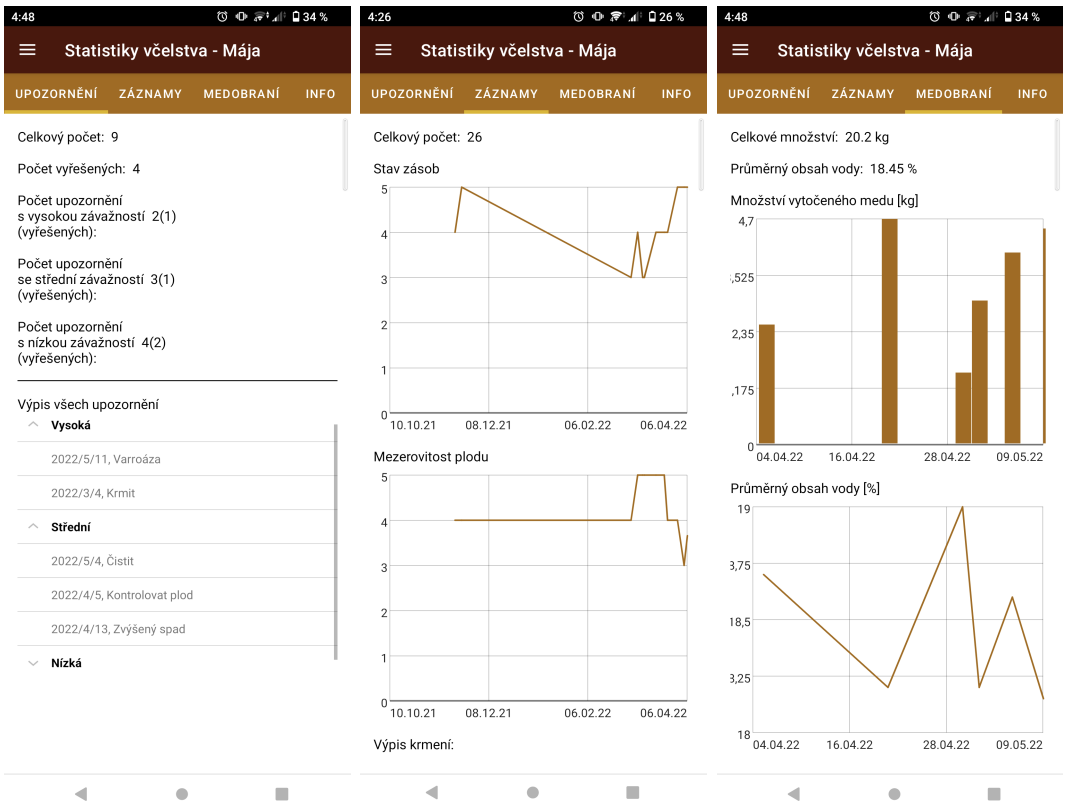
Obr. 3.13: DatePickerDialog v aktivitě ScopeSelectorActivity (vpravo)

Při volbě časového rozsahu *„Od - do“* se aktivují stejnojmenná tlačítka, opět metodou `setEnabled()`. Po jejich stisknutí se zobrazí *DatePickerDialog* implementovaný v *DatePickerFragmentu* umožňující interaktivní výběr data. O sběr dat z DatePickerDialogu se stará metoda `onDateSet()` a přepisuje je do TextView nad příslušnými tlačítky.

Stisknutí tlačítka *Pokračovat* zkontroluje, zda je zvolen rozsah včelstev i časový rozsah. Podle výběru v horní části obrazovky se pak volá buď přímo aktivita *StatsActivity* pro všechna včelstva, nebo *SelectActivity* pro jednotlivá včelstva a lokality, kde se pak volá příslušná aktivita na zobrazení statistik [6].

3.2.5 StatsHiveActivity

Tato aktivita slouží k zobrazování statistik jednotlivých včelstev a je vizuálně velmi podobná ShowHiveActivity, jen nejsou přítomny žádné FABy. Její záložky jsou plněny *StatsHiveAdapterem* odvozeným od ShowHiveAdapteru.



Obr. 3.14: StatsAlertHiveFragment v StatsHiveActivity (vlevo)

Obr. 3.15: StatsRecordHiveFragment v StatsHiveActivity (uprostřed)

Obr. 3.16: StatsHarvestHiveFragment v StatsHiveActivity (vpravo)

Fragmenty

Na první záložce se nachází *StatsAlertHiveFragment*. Mimo informací o upozorněních obsahuje také jejich kompletní seznam. Protože je celý layout vertikálně dlouhý, bylo použito ExpandableListView o pevné vertikální velikosti, ve kterém bude moci uživatel podle potřeby rolovat. Na malých obrazovkách by ale toto ExpandableListView bylo za spodní hranou, proto bylo nutné použít layout typu *NestedScrollView*, který podporuje vnořená rolování. Stejný typ layoutu byl použit ze stejného důvodu

i ve fragmentech *StatsRecordHiveFragment*, *StatsAlertFragment* a *StatsRecordFragment* a v aktivitě AddRecordActivity [6].

GraphView

Pro zobrazení grafů byla použita knihovna GraphView. Jedná se o open-source knihovnu v jazyce Java umožňující zobrazení několika různých druhů grafů a úpravu jejich vzhledu. V této aplikaci jsou použity pro zobrazení trendů vývoje některých sledovaných položek v čase. Typy využitých grafů jsou sloupcový a čárový.

Instanci GraphView se datová řada nastavuje metodou `addSeries()`. Datový typ argumentu předaného metodě určuje typ zobrazeného grafu, např. `BarGraphSeries` pro sloupcový graf nebo `LineGraphSeries` pro čárový graf. Protože zobrazujeme časovou závislost, je třeba mít osu x v časovém formátu. K tomu je potřeba předávat do datové řady body formátu `DataPoint` s hodnotou x ve formátu Date seřazené chronologicky [14].

3.2.6 StatsActivity

StatsActivity je implementována stejně jako StatsHiveActivity, jen postrádá čtvrtou záložku a neobsahuje některé grafy.

3.2.7 Aktivity pro zapisování dat

AddLocationActivity

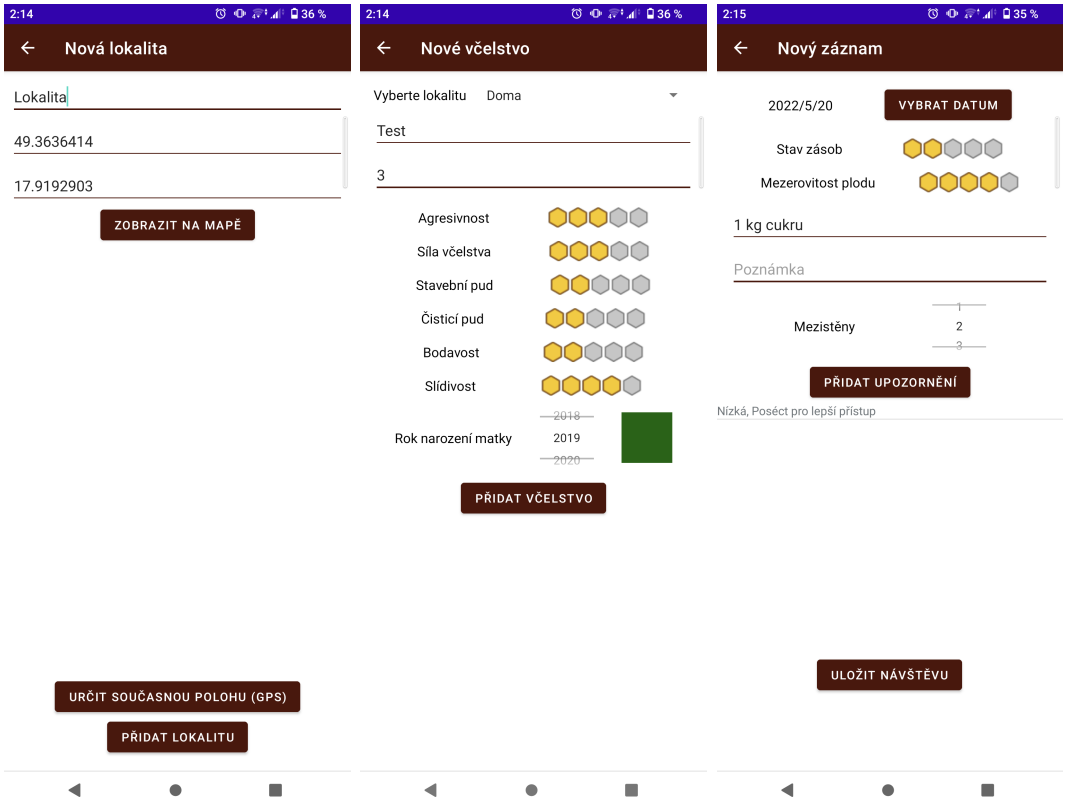
Tato aktivita slouží k přidání lokality včelnice. Jak již bylo zmíněno, je zde využito aktuální polohy zařízení přečtené z GPS.

Nejprve se vytvoří globální instance *LocationRequest*. Po stisknutí tlačítka *Určit současnou polohu (GPS)* aplikace vyzve uživatele k povolení využití služeb polohy a následně se dotáže systému, jestli jsou zapnuty služby určování polohy. Následně se předá LocationRequest jako argument metodě `requestLocationUpdates()` volané na instanci třídy `FusedLocationProvider` společně s anonymní implementací metody `onLocationResult()`. V té je následně definováno, co se se získanou instancí *LocationResult* stane. V tomto případě se souřadnice zapíší do textových polí [6].

AddHiveActivity

V této aktivitě se vyskytují podobné prvky jako ve StatsFragmentu, které slouží na určení statických neměnných včelstva, to jest páry TextView a RatingBaru. Společně s nimi se zde nachází i *Spinner*, který obsahuje seznam všech lokalit. Spinner

je view, které představuje rozbalovací menu. Tento spinner je plněn jednoduchým *ArrayAdapterem*, kterému je předán seznam názvů lokalit [6].



Obr. 3.17: Aktivita AddLocationActivity

Obr. 3.18: Aktivita AddHiveActivity

Obr. 3.19: Aktivita AddRecordActivity

AddRecordActivity

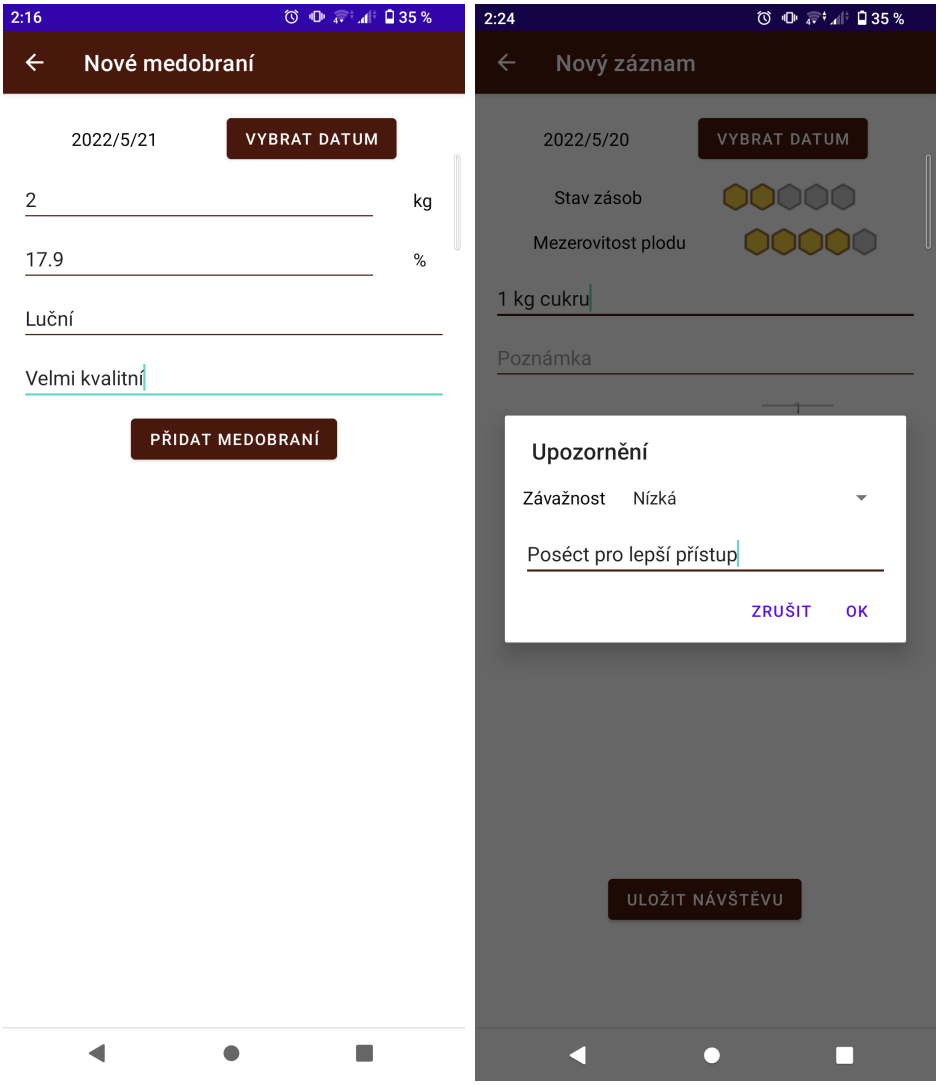
AddRecordActivity slouží k přidání záznamu kontroly. Logicky proto obsahuje prvek, pomocí kterého lze vybrat datum. Ten je realizován pomocí tlačítka, které zobrazí *DatePickerFragment*. Dále se zde také nachází tlačítko na přidávání upozornění (viz níže) a pod ním seznam vytvořených upozornění [6].

AddHoneyHarvestActivity

Aktivita *AddHoneyHarvestActivity* je velmi podobná AddRecordActivity. Datum je taktéž zadáváno pomocí DatePickerFragmentu a zbytek ovládacích prvků tvoří dříve popsaná textová pole RatingBary.

AddAlertDialog

Jak je již vidno z názvu, nejedná se o aktivitu, nýbrž o *dialog*. Dialogy jsou malá okna, která umožňují uživateli vykonat rozhodnutí nebo zadat informace. Tento dialog se spouští z výše zmíněné AddRecordActivity. Upozornění se nepřidá za předpokladu, že nejdříve nedošlo ke zvolení data [6].



Obr. 3.20: Aktivita AddHoneyHarvestActivity

Obr. 3.21: AddAlertDialog v aktivitě AddRecordActivity

3.3 Vlastní designové prvky

Views umožňují již v základu velké množství nastavitelných parametrů, jako velikost, průhlednost nebo například barva podkladu. Je ale také možné kompletně změnit vzhled některých prvků pomocí stylů.

V aplikaci je použito trojbarevné schéma, tmavá barva (#4F1308), středně světlá barva (#A86807) a světlá barva (#E0B400). Tmavá barva byla použita na horní lištu a na interaktivní prvky, středně světlá barva na TabLayout a na barvu datové řady grafů a světlá barva byla použita na FABy a na určení pozice v TabLayoutu.

FABy obsahují autorem vytvořené ikony. Ty byly vytvořeny v programu *Inkscape* a následně exportovány ve vektorovém formátu *.svg*. Při importu těchto ikon do projektu z nich Android studio vytvoří *.xml* soubory, které je poté možné použít v rámci aplikace [6].

Nicméně největší změnou určitě prošel *RatingBar*. Tomu byl vytvořen kompletně nový styl inspirovaný vzhledem buněk včelího díla. Návrh vzhledu jednoho prvku proběhl opět v programu Inkscape. Pro správnou funkci je třeba vytvořit dva vzhledy, přičemž jeden odpovídá vyplněnému políčku a druhý nevyplněnému. Pro každý z těchto vzhledů bylo třeba vytvořit XML soubor přiřazující ikonu ke každému stavu. Nakonec jsou tyto sjednoceny v souboru *custom_rb.xml*, který určuje, jaký vzhled se má použít pro pozadí, pro postupné vyplňování a pro vyplněný prvek RatingBaru.

Nakonec byl pro jednodušší použití přidán nový styl do souboru *themes.xml* s rodičovským view `Widget.Appcompat.RatingBar`. Zde bylo určeno, že se pro zobrazení RatingBaru má využít popis ze souboru *custom_rb.xml*. V layoutu je použití stylu následně definováno atributem `style="@style/CustomRatingBarStyle"` [6].

Závěr

Cílem bakalářské práce bylo navrhnout a vytvořit mobilní aplikaci na platformě Android, která bude včelařům usnadňovat práci se včelstvy a bude jim schopna poskytovat užitečné informace na základě jimi nasbíraných dat.

V první části práce byla nastíněna problematika chovu včel a problémů, které přitom mohou nastat. Dále byly shrnuty základní informace o operačním systému Android, programovacím jazyce Java, knihovně Room, architektuře MVVM a programech Android Studio a Inkscape.

Ve druhé části byl popsán koncepční návrh aplikace, dle kterého byla aplikace následně realizována. Byla popsána jak struktura aplikace, tak struktura databáze, se kterou aplikace při svém běhu interaguje.

Ve třetí části je podrobně popsána implementace jednotlivých částí aplikace, jak na sebe navazují a jakým způsobem je zajištěna požadovaná funkcionalita.

V tento moment aplikace splňuje základní požadavky, nicméně bylo by možné ji vylepšit několika způsoby. Přidání funkce exportu a importu databáze by umožnilo spolupráci více včelařů, popřípadě bezproblémový přenos dat při přechodu uživatele na nové zařízení. Dalším krokem v podobném směru by mohla být synchronizace s databázovým serverem, což by umožnilo paralelní práci více včelařů na stejných včelstvech bez nutnosti záznamy dlouze přepisovat.

Literatura

[1] *ŠEFČÍK, Jozef. Začínáme včelařit* [PDF]. Praha: Grada Publishing, 2014 [cit. 2021-12-13]. ISBN 978-80-247-9278-1. Dostupné z: <https://www.palmknihy.cz/ekniha/zaciname-vcelarit-151152>

[2] *URBAN, Miroslav. Včelaření od jara do zimy.* Praha: Grada Publishing, 2018. ISBN 978-80-271-0365-2.

[3] *Třetina včelstev přes zimu uhynula, škody jsou přes 100 miliónů.* In: Novinky.cz [online]. Praha, 2008 [cit. 2021-12-13]. Dostupné z: <https://www.novinky.cz/ekonomika/clanek/tretina-vcelstev-pres-zimu-uhynula-skody-jsou-pres-100-milionu-40199587>

[4] *Mobile Operating System Market Share Worldwide - November 2021.* Gs.statcounter.com: GlobalStats [online]. 2021 [cit. 2021-12-13]. Dostupné z: <https://gs.statcounter.com/os-market-share/mobile/worldwide>

[5] *Mobile Android Version Market Share Worldwide - November 2021.* Gs.statcounter.com: GlobalStats [online]. 2021 [cit. 2021-12-13]. Dostupné z: <https://gs.statcounter.com/android-version-market-share/mobile/worldwide/>

[6] *Oficiální stránka pro vývoj na platformě android* [online]. Dostupné z URL: <https://developer.android.com>

[7] *Android 8.0 Features and APIs* [online]. Dostupné z URL: <https://developer.android.com/about/versions/oreo/android-8.0#rt>

[8] *Oficiální dokumentace jazyka Java* [online]. Dostupné z URL: <https://docs.oracle.com/en/java/javase/18/index.html<

[9] *Oficiální specifikace Java Virtual Machine* [online]. Dostupné z URL: <https://docs.oracle.com/javase/specs/jvms/se17/jvms17.pdf<

[10] *Oficiální GitHub dokumentace* [online]. Dostupné z URL: <https://docs.github.com/en/get-started/quickstart/set-up-git>

[11] *Oficiální dokumentace Android Studio* [online]. Dostupné z URL: <https://developer.android.com/studio/intro#version_control_basics>

[12] *Oficiální stránky projektu Inkscape* [online]. Dostupné z URL: <https://inkscape.org/>

[13] *Places API Overview* [online]. Dostupné z URL: <https://developers.google.com/maps/documentation/places/web-service/overview>

[14] *jjoe64/GraphView repozitář* [online]. Dostupné z URL: <https://github.com/jjoe64/GraphView>

Seznam symbolů a zkratek

API Appliaction Programming Interface

DAO Database Access Object

FAB Floating Action Button

GPS Global Positioning System

IDE Integrated Development Environment

JVM Java Virtual Machine

MVVM Model, View, ViewModel

OS Operating System – Operační systém

SQL Structured Query Language

VCS Version Control Software

XML eXtensible Markup Language

A Obsah elektronické přílohy

```
/
├── Bee_v0.3_build_2022_05_30.apk
├── Bee_v0.3_source_code_2022_05_30.zip
├── pecar_bakalarska_prace.pdf
├── README.txt
└── testovaci_databaze.zip
```