FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	iOS aplikace Farmsoft
Student:	Bc. Martin Beran
Vedoucí:	Ing. Petr Smolík
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem této práce je realizace iOS aplikace pro systém Farmsoft. Aplikace nebude na systém přímo napojena, ale bude zde existovat rozhraní, které toto napojení v budoucnu umožní. Aplikaci bude možné plně nabízet zákazníkům systému Farmsoft.

Postupujte v těchto krocích:
1. Nastudujte systém Farmsoft a následně navrhněte aplikaci pro iOS.
2. Na základě analýzy implementujte aplikaci pro mobilní telefony se systémem iOS. Ve vlastním návrhu realizujte aplikaci bez napojení na server. V návrhu však zohledněte případné následné propojení na systém API Farmsoft.
3. Návrh otestujete a řešení konzultujte se zákazníkem. Případné výhrady opravte.
4. Prozkoumejte možnosti automatického nasazení na AppStore.
5. Zhodnoťte použitelnost výsledné aplikace a navrhněte vhodné budoucí směry vývoje (např. rozšíření, technologické směřování, apod.).

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 26. dubna 2022 v Praze.

Diplomová práce

IOS APLIKACE FARMSOFT

Bc. Martin Beran

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Petr Smolík
23. prosince 2022

Obsah

Poděkování vii

Prohlášení viii

Abstrakt ix

Seznam zkratek x

Úvod 1

1 Představení systému Farmsoft 3
1.1 Problematika . . . 3
1.2 Představení . . . 3
1.3 Historie . . . 4
1.4 Hlavní části systému Farmsoft . . . 4
1.4.1 Server . . . 4
1.4.2 Program pro PC . . . 6
1.4.3 Android . . . 6

2 Analýza 9
2.1 Fungování podniku . . . 9
2.1.1 Životní cyklus zvířete . . . 10
2.2 Doménový model . . . 11
2.2.1 Změny . . . 12
2.3 Návrh mobilní aplikace . . . 20
2.3.1 Přihlášení . . . 21
2.3.2 Synchronizace . . . 21
2.3.3 Změny . . . 21
2.3.4 Laktace . . . 22
2.3.5 Zvířata a filtrace . . . 22
2.3.6 Denní události . . . 22
2.3.7 Sestavy . . . 22
2.3.8 Oprávnění . . . 23
2.3.9 Bluetooth . . . 23
2.3.10 Rskot . . . 24

3 Průběh vytváření iOS aplikací 25
3.1 Programovací jazyky . . . 25
3.1.1 Objective-C . . . 25
3.1.2 Swift . . . 25
3.1.3 Dart . . . 26
3.2 Xcode . . . 26
3.2.1 Knihovny . . . 28
3.3 SwiftUI vs UIKit . . . 30

3.3.1 UIKit . . . 30
3.3.2 SwiftUI . . . 30
3.4 Architektura . . . 32
3.4.1 Model view viewmodel . . . 32
3.4.2 The Composable Architecture . . . 33
3.5 AppStore . . . 34

4 Vývoj **37**
4.1 Návrh . . . 37
4.2 Průběh vývoje . . . 37
4.3 Použité technologie a knihovny . . . 39
4.3.1 Architektura . . . 39
4.3.2 SwiftGen . . . 39
4.3.3 Alamofire . . . 40
4.3.4 PMJSON . . . 40
4.3.5 SwiftUI . . . 40
4.3.6 CoreData . . . 40
4.3.7 Firebase . . . 41
4.3.8 GoogleMLKit - BarcodeScanning . . . 41
4.4 Popis aplikace . . . 41
4.4.1 Synchronizace . . . 43
4.4.2 Přehled podniků a přihlášení . . . 47
4.4.3 Hlavní obrazovka . . . 49
4.4.4 Nastavení . . . 50
4.4.5 Stáje . . . 51
4.4.6 List zvířat . . . 53
4.4.7 Skenování . . . 54
4.4.8 Inseminační býci . . . 55
4.4.9 Detail zvířete . . . 57
4.4.10 Pořízení laktace . . . 59
4.4.11 Obrazovky změn . . . 60
4.4.12 Sestava inseminací . . . 64

5 Testování **71**
5.1 Testování návrhu . . . 71
5.2 Testování aplikace . . . 72
5.2.1 Aplikace v produkci . . . 73

6 Automatické nasazení na AppStore **75**
6.1 Automatizace sestavení a nahrání aplikace . . . 75
6.2 Automatizace spuštění procesu po nahrání na git . . . 76

7 Zhodnocení a budoucí vývoj aplikace **81**

Závěr **83**

Obsah přiloženého média **89**

Seznam obrázků

1.1 Ukázka FlameRobin připojeného k testovací databázi . . . 5
1.2 Počet záznamů v tabulce změny v testovací databázi . . . 5
1.3 Program Farmsoft hlavní obrazovka . . . 7
1.4 Program Farmsoft karta zvířete . . . 7
1.5 Android ukázka obrazovek . . . 8

2.1 Základní doménový model . . . 12
2.2 Doménový model změn léčení . . . 19
2.3 Ukázka zadávání léčení paznehtu v aplikaci Farmsoft . . . 20

3.1 Ukázka Xcode s kódem aplikace . . . 27
3.2 Ukázka editoru databáze v Xcode . . . 27
3.3 Cyklus UIViewControlleru [25] . . . 31
3.4 Schema MVVM architektury [28] . . . 32
3.5 Schema komunikace TCA architektury [29] . . . 33

4.1 Hlavní storyboard prvního pokusu o aplikaci . . . 38
4.2 Porovnání hlavní obrazovky Android (vlevo) iOS (vpravo) . . . 42
4.3 Porovnání světlého a tmavého módu aplikace . . . 44
4.4 Diagram synchronizace aplikace . . . 45
4.5 Obrazovka přihlášených podniků a obrazovka přihlášení podniku . . . 48
4.6 Ukázka hlavní obrazovky . . . 49
4.7 Obrazovka nastavení . . . 51
4.8 Obrazovka výběru aktivní stáje a nastavení polohy stáje . . . 52
4.9 Obrazovka přehledu stáda . . . 53
4.10 Obrazovka skenování čárových a QR kódů . . . 54
4.11 Obrazovka inseminačních býků . . . 55
4.12 Obrazovka detailu zvířete . . . 56
4.13 Obrazovky spojené s laktací . . . 59
4.14 Označení/Vážení/Poznámky . . . 65
4.15 Říje/Březost/Otelení . . . 65
4.16 Inseminace . . . 66
4.17 Zaprahnutí/Selekce/Umístění . . . 66
4.18 Paznehty . . . 67
4.19 Nemoc dojírna/Léčení . . . 68
4.20 Sestava inseminace . . . 69
4.21 Export sestavy inseminace . . . 70

6.1 Přehled všech spuštěných Pipline . . . 78
6.2 Ukázka detailu pipeliny . . . 79
6.3 Ukázka logů konkretního Jobu . . . 79

Seznam tabulek

2.1 Tabulka druhů změn a příslušných id 13

Seznam výpisů kódu

2.1 Část SQL scriptu pro založení tabulky změny 14
3.1 Podfile projektu Farmsoft 29
4.1 Ukázka použití dependency injection 39
6.1 Část souboru Fastfile pro sestavení aplikace 76
6.2 Soubor .gitlab-ci.yml pro konfiguraci Gitlab CI/CD 80

Chtěl bych poděkovat rodině, přátelům a spolužákům za podporu při studiu. Dále bych chtěl poděkovat panu Ing. Petru Smolíkovi za vedení této práce a za předání znalostí nutných pro implementaci této práce. Nakonec bych chtěl poděkovat společnosti Farmtec a.s, která je vlastník programu Farmsoft, za možnost použít tvorbu iOS aplikace Farmsoft jako moji diplomovou práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. 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 ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 23. prosince 2022 .....

Abstrakt

Famrosft je program pro správu a evidence dojných a chovných stád skotu. V této práci jsem implementoval mobilní aplikace systému Farmsoft pro mobilní telefony se systém iOS.

Klíčová slova Farmsoft, iOS, mobilní aplikace, Farmtec a. s.

Abstract

Farmsoft is a program for the management of dairy and breeding cattle farms. In this thesis, I implemented Farmsoft mobile application for mobile phones with an iOS system.

Keywords Farmsoft, iOS, mobile app, Farmtec a. s.

Seznam zkratek

BT	Bluetooth
EID	Elektronická identifikace
IDE	Vývojové prostředí
iOS	iPhone Operating System
MVVM	Model–view–viewmodel
ku	kontrola užitkovosti - laktace
sn	synchronizační číslo (serial number)
TCA	The Composable Architecture
UI	uživatelské rozhraní

Úvod

Jsem si jistý, že nikdo nebude namítat, pokud řeknu, že zemědělství je jedním z nejdůležitějších průmyslů. Jenom díky průmyslovému zemědělství jsme schopni nakrmit momentální populaci lidí na Zemi. Nemalý podíl v tomto odvětví zabírá chovatelství stád skotu.

Chovatelství skotu je obrovský průmysl, ve kterém je třeba řídit a kontrolovat velké množství věcí. Musíme mít přehled o našem stádu a stavu jednotlivých zvířat. Musíme hlídat finance, vytvářet výkazy a reporty pro ministerstvo atd...

Proto vznikl program Farmsoft. Farmsoft je program pro řízení dojných a mastných stád skotu. Tvorba programu je financována společností Farmtec a. s. Farmtec je firma specializující se na projektování a stavbu moderních stájí a technologií dojení. Společnost Farmtec je výhradním vlastníkem programu Farmsoft.

Program Farmsoft nabízí velké množství funkcí jako evidence stáda, tvorba sestav a reportů a evidence a hlídání léčiv. Je používaný zootechniky napříč celou Českou republikou. Je využíván i v zahraničí, převážně v Polsku ale také v Bělorusku či Rusku.

Hlavní část programu je implementovaná v PC verzi. Zde jsou dostupné všechny funkce, které Farmsoft nabízí. Další částí je mobilní aplikace pro Android. Aplikace slouží lidem v terénu. Zobrazuje informace o přehledu stáda a pořizování nových změn. Díky tomu mají zootechnici aplikaci Farmsoft kdykoliv po ruce a vždy mají přehled o svém stádu.

Někteří uživatelé si přáli mít Farmsoft i ve svém mobilním telefonu se systémem iOS. Vznikla proto potřeba vytvořit mobilní aplikaci Farmsoft i pro systém iOS.

Cílem této práce bylo vytvořit iOS aplikaci bez napojení na API Farmsoft. Tato aplikace by měla být v budoucnu napojena na API Farmsoft a předána zákazníkům. Aplikace by měla být schopná splnit základní požadavky uživatelů Farmsoftu. Nemusí implementovat všechny funkce jako verze pro Android, ale do budoucna je třeba počítat s implementací stejných funkcí jako jsou aktuálně dostupné v Android verzi.

Aplikace by měla být otestována a nalezené chyby by měly být opraveny. Následně bude prototyp použit pro implementaci ostré verze aplikace, která bude dostupná zákazníkům v AppStore. Další úkolem je zjistit možnosti automatického nahrávání aplikace do AppStore tak, aby při každém vydání nové verze proběhlo toto nahrání automaticky. Na závěr jsem měl zhodnotit použitelnost vzniklé aplikace a navrhnout kroky pro její vylepšení.

Kapitola 1

Představení systému Farmsoft

1.1 Problematika

Představte si, že jste majitel menšího zemědělského podniku. Řekněme, že vlastníte 100 kusů dobytka a chcete mít přehled o svém stádu. Potřebujete vědět identifikaci kusu, věk jednotlivých zvířat, jejich nemoci, místo, kde se dané zvíře nachází a v neposlední řadě laktační a reprodukční cyklus.

Laktační cyklus je důležitý kvůli dojení. Z mléka Váš podnik vydělává peníze a proto je dobré vědět, jak dobře které kráva dojí. Reprodukční cyklus nám zase řekne, kdy daná kráva může znovu zabřeznout, a tudíž kdy opět bude schopná dojit.

To už je celkem velké množství dat, ale schopný zootechnik s velkým množstvím papíru to zvládne. Co však dělat, když nemáme 100 kusů ale 1000. Navíc takhle velké stádo už bude nejspíš rozdělené přes několik středisek vzdálených od sebe i několik kilometrů, kde v každém bude jiný zootechnik a kusy se mohou mezi středisky různě pohybovat. V takovém případě už nám papír a tužka přestávají stačit. A to ani nemluvím o dalších problémech spojených s problematikou chovu dobytka jako je nutnost hlásit nově narozené kusy do centrální evidence, přehled o nemocných kusech, přehled o dostupných lécích na skladu a tak dále.

1.2 Představení

Systém Farmsoft vznikl za účelem ulehčení řešení těchto problémů. Farmsoft je soubor programů pro správu dojných a mastných stád skotu. Slouží jako evidenční program pro usnadnění managementu stáda a řízení podniku. Umí importovat a exportovat data v různých formátech a pro různé zdroje. Všechny informace jsou centrální, tudíž je snadné sdílet je mezi jednotlivými středisky.[1]

Systém umí zároveň pracovat s technologickými zařízeními jako jsou selekční branky, informační display, systém pro identifikaci krav atd. Všechny tyto zařízení dále usnadňují práci zootechniků přímo ve stáji. [2]

Za zmínku dále stojí Rskot. Rskot je Farmsfot upravený pro inseminační techniky. Inseminační technici na objednání objíždění jednotlivé podniky a inseminují krávy. Dále také vyšetřují, jestli se inseminace povedla a daná kráva zabřezla. Tito lidé nepotřebují plnou funkcionalitu Farmsoftu, a proto vznikla menší verze programu upravené pro jejich potřeby s názvem Rskot.

Hlavní části systému Farmsoft jsou server s databází pro ukládání dat, program Farmsoft pro PC, který tvoří hlavní část systému, webové verze aplikace umožnující základní přehled bez možnosti editace dat a nakonec mobilní verze aplikace. Mobilní verze jsou aktuálně dvě a to pro Android a pro iOS, která je předmětem této práce.

Jednotlivé komponenty systému si více rozebereme dále v textu, avšak jejich detailní představení by dalo na samostatnou práci. Proto u nich budu zmiňovat hlavně věci, které se týkají této práce. Avšak dříve než si představíme samotné části, tak bych rád ještě rozvedl celkovou historii projektu a mé působení v něm.

1.3 Historie

V této části bych rád rozvedl mé celkové působení v projektu a zmínil lidí, s kterými jsem na projektu spolupracoval. Analýza Farmsoftu začala v roce 2007 a následný vývoj o rok později, v roce 2008. Hlavní vývojář je Ing. Petr Smolík. S dalšími lidmi pomalu vytváří PC verzi programu společně s databází na serveru. Z lidí, kteří mu pomáhají, zmíním dále pana Jiřího Smolíka, kteří v počátku pomáhá s vývojem PC verze programu a později začíná programovat verzi pro Android. Vývoj Androidu začíná v roce 2015. [3]

Program si pomalu získává první zákazníky a začíná se pomalu dostávat do světa. Poté do projektu nastupuje Ing. Tomáš Dvořák, který má na starost serverovou část a tvorbu webových aplikací Farmsoftu a později API rozhraní.

Já jsem do projektu nastoupil právě na doporučení Tomáše Dvořáka. Nastoupil jsem do projektu jako Android vývojář. Nástup proběhl v roce 2017 a měl jsem po panu Jiřím převzít Android projekt. Aplikaci jsem tedy převzal a od té doby pracuji na jejím vývoji až do dnešních dnů. Vývoj Android verze mi velice pomohl pochopit fungování Farmsoftu a díky tomu jsem byl schopný navrhnout iOS verzi.

Prvotní pokusy o vytvoření iOS verze Farmsoftu začaly v roce 2019, avšak iOS verze nebyla velkou prioritou a proto po necelém roce bylo od záměru ustoupeno. K jeho znovuoživení došlo v roce 2021, kdy aplikace byla napsána znovu s použitím nejnovějších standardů a její vývoj pokračuje dodnes.

1.4 Hlavní části systému Farmsoft

1.4.1 Server

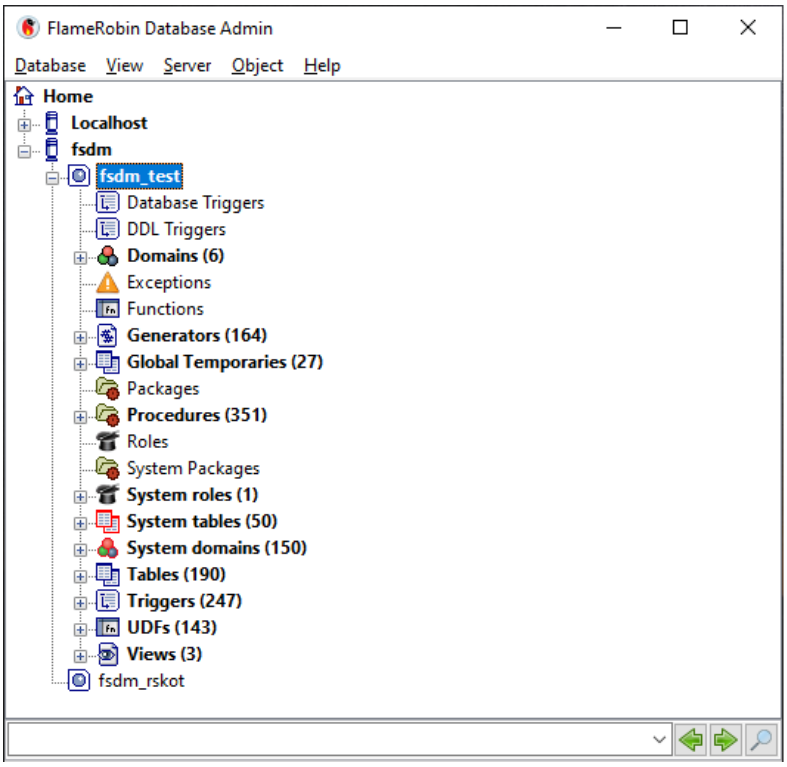
Farmsoft podporuje dva způsoby ukládání dat na server. Jsou to Cloud a FTP. Cloud je standardní ukládání do databáze Jako databázi se tvůrci Farmsoftu rozhodli použít open source databázi Firebird. Firebird je cross-platform relační databáze. Jedná se o fork databáze InterBase, která je nyní udržována komunitou pod záštitou Firebird Foundation. Pro připojení lze použít nástroj FlameRobin. [4]

Momentálně existují tří různé databáze - jedna testovací, jedna produkční pro Farmsoft a jedna produkční pro Rskot. Schéma databází je vždy stejné a databáze se liší pouze v datech.

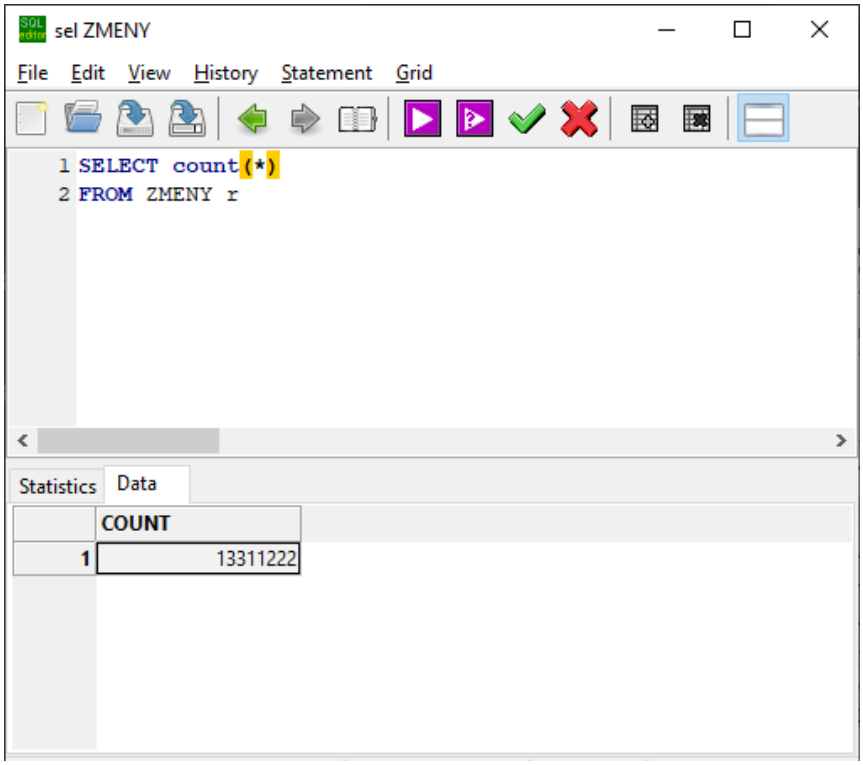
Jak vidíte na obrázku 1.1 databáze je poměrně rozsáhlá. Obsahuje 190 tabulek, 351 procedur, 164 generátorů a 247 triggerů. Navíc některé tabulky jsou opravdu velké a mají více než 50 sloupců. V databázi je i celkem velké množství dat. Na obrázku 1.2 můžeme vidět, že v testovací databázi v tabulce Změny je přes 13 milionů záznamů. Databáze tedy není úplně malá a je třeba s tím při návrhu počítat.

Aplikační logika probíhá také v databázi, což už je v moderním pojetí softwarového inženýrství špatně už kvůli vendor locku na dané databázi, špatnému škálování této architektury atd. [5]

Pro komunikaci s databází lze využít API Famrsoft. Toto API je webová služba, která mapuje vstupní json na sql dotazy přímo do databáze a do procesu nepřidává žádnou další logiku.



Obrázek 1.1 Ukázka FlameRobin připojeného k testovací databázi



Obrázek 1.2 Počet záznamů v tabulce změny v testovací databázi

1.4.2 Program pro PC

Program Farmsoft pro PC je bezpochyby nejdůležitějším stavebním kamenem systému Farmsoft. Program je velice komplexní. Nabízí možnost práce jak v online, tak i v offline módu bez přístupu k internetu. Program je napsaný v C++ s využitím QT frameworku. Program má svojí vlastní interní databázi Firebird schématem totožnou s verzí na serveru.

Po spuštění nás program přesměruje na hlavní obrazovku. Pohled na ní vidíme na obrázku 1.3. Program na hlavní obrazovce nabízí navigaci programem, základní informace o podniku a novinky. V navigaci programu nalezneme několik hlavních záložek. Jsou to Evidence, Zdraví, Sestavy, Číselníky a Ostatní.

Evidence nám dává přístup do přehledu stáda. Vidíme zde všechny kusy v našem stádu. U každého zvířete můžeme vidět jeho kartu se základními informacemi o daném zvířeti, viz obrázek 1.4. U každého zvířete vidíme jeho změny a můžeme tyto změny libovolně zadávat, popřípadě editovat. Změny reprezentují důležité události v životě zvířete, přesuny, nemoci, březost atd. Dále zde vidíme kartu laktace. Na té uživatel vidí výsledky kontrol užitkovosti. V evidenci také vidíme Denní události. Pomocí Denních událostí si můžeme šířit zprávy napříč podnikem. Ještě zde zmíním Hromadné změny. Pomocí Hromadných změn můžeme vybrat skupinu zvířat a na tu aplikovat změnu, například přesun dané skupiny na jinou stáj atd.

Záložka Zdraví nám dává možnost vidět definované diagnozy a léčiva. Máme možnost si definovat vlastní léčiva a diagnózy podle potřeby. Dále zde vidíme Sklad léčiv. Ten nám dává přehled o aktuálně vlastněných léčivech a díky němu víme, že lék dochází a je nutné jej dokoupit. Taky si můžeme zobrazit léčená zvířata a léky, jaké jim byly podány.

Sestavy nám slouží pro podporu rozhodování. Zobrazují různé informace, které by se mohly managmentu podniku hodit pro budoucí rozhodování, například dojivost jednotlivých kusů, statistiky narození atd. Uživatel má taktéž možnost definovat si vlastní sestavu pomocí Návrháře sestav. Všechny sestavy je možné tisknout a poté exportovat podle potřeby.

V záložce Číselníky máme možnost definovat si naší podnikovou strukturu - jednotlivé podniky, střediska, stáje a skupiny. Definují se zde taky technici a inseminační býci.

Poslední záložkou je Ostatní. Ta slouží k celkovému nastavování programu. Najdeme zde možnost nastavení připojení do databáze, celkové nastavení programu, nastavení jazyka, prostředí a konstant v programu. Program je aktuálně dostupný v české, anglické, polské a ruské variantě.

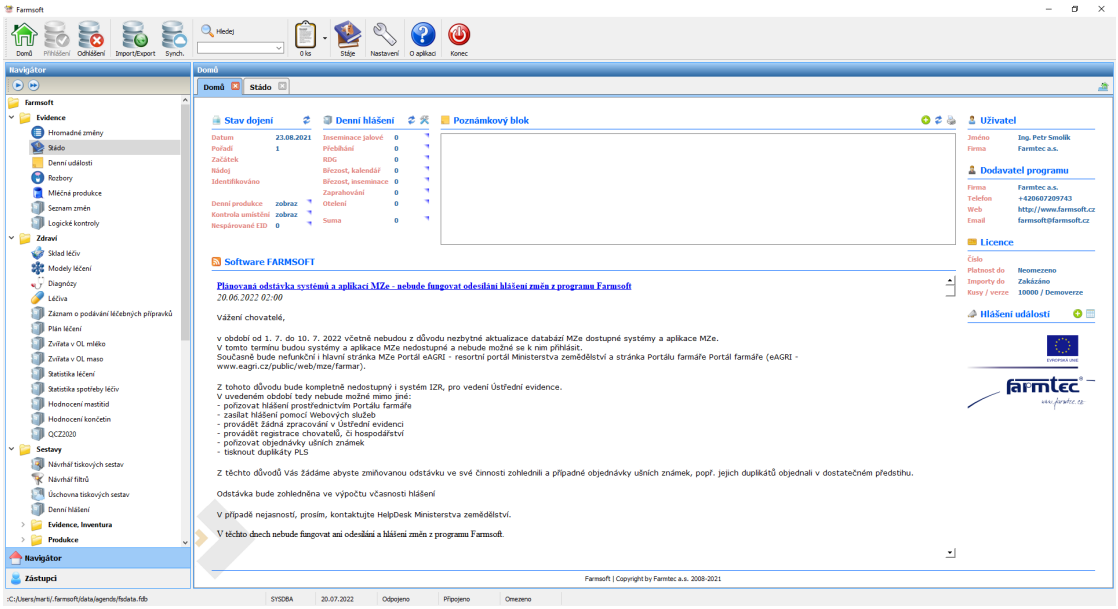
Program podporuje import a export do různých systémů. Rád bych zmínil systém Plemdat. Jedná se o národní registr všech chovatelů a jejich zvířat. U zvířat eviduje údaje o reprodukci, vzhledu, umístění atd... Plemdat umí generovat spoustu různých reportů, od užitkovostí po reprodukci. Tyto reporty potom následně umí Farmsoft zobrazit v podobě Změn.

1.4.3 Android

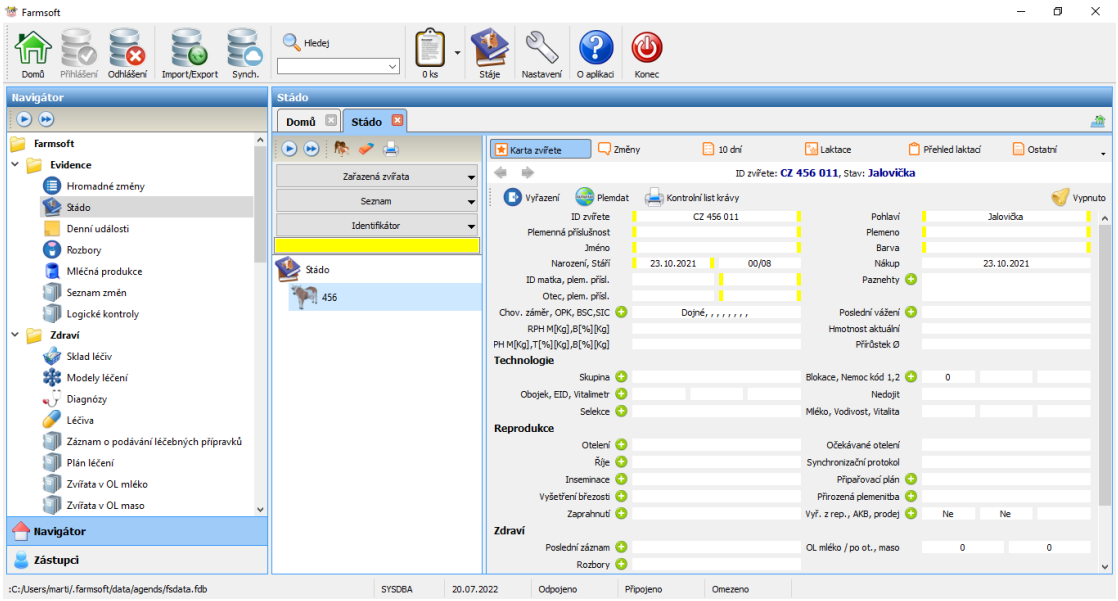
Mobilní verze Farmsoft pro systémy Android je určená pro zadávání změny přímo na stájí. Je dostupná jak pro mobilní telefony, tak pro tablety. Funguje online i offline. Jedná se o nativní android aplikaci psanou v Javě a Kotlinu. Na návrhu této aplikace jsem se aktivně podílel a zkušenosti při jejím vývoji jsem využil i při tvorbě iOS verze.

Hlavní obrazovka obsahuje základní informace a slouží jako rozcestník pro zbytek aplikace. Máme zde možnost synchronizovat aplikaci se serverem, zobrazit aktuálně zvolenou stáj, kartu Přehledu stáda, která nám po kliknutí zobrazí všechny zvířata na dané stáji, Inseminační býky, které může uživatel použít při zadávání inseminace a Hromadné změny, které slouží stejně jako v PC verzi programu pro zadání hromadných změn u zvířat. Dále zde najdeme Pořízené změny, které zatím nebyly synchronizovány se serverem, Denní události a Sestavy.

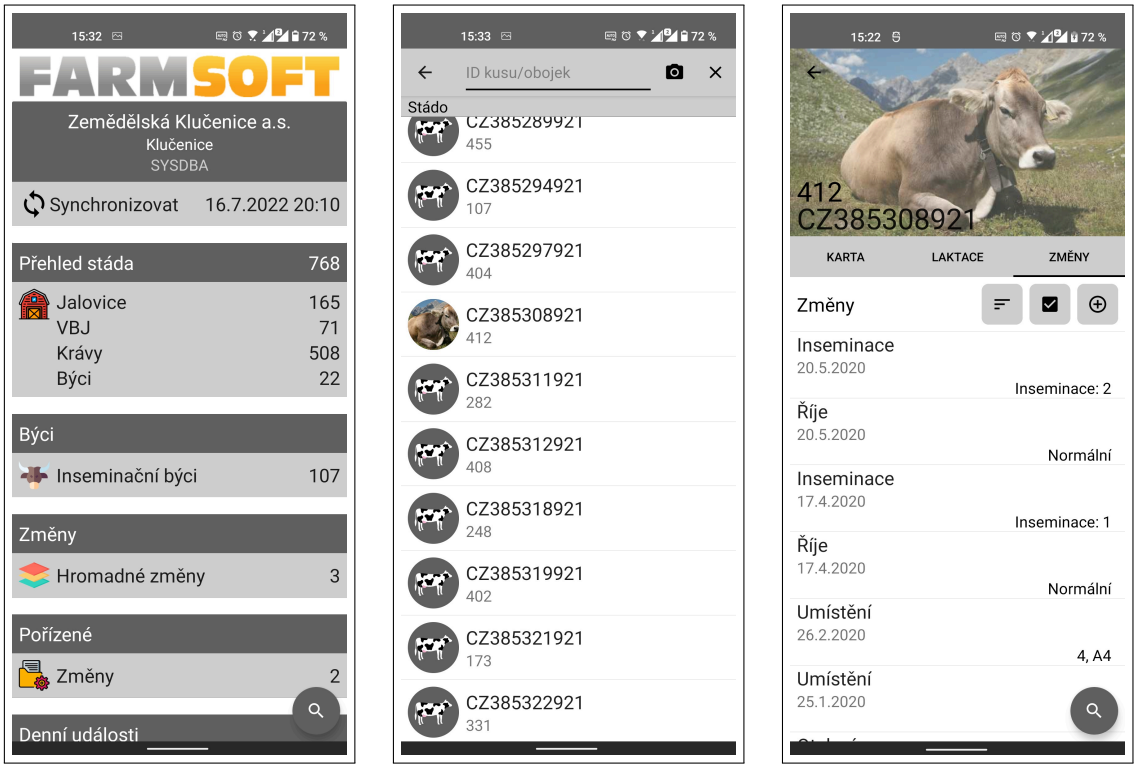
Sestavy jsou o dost omezenější než u PC verze a aktuálně se jich v android verzi nachází 7. Typicky jsou to sestavy plnící určitý případ užití, který lze splnit v mobilu, například selekce jistých kusů ze skupiny, nebo výkaz vykonaných inseminací.



Obrázek 1.3 Program Farmsoft hlavní obrazovka



Obrázek 1.4 Program Farmsoft karta zvířete



Obrázek 1.5 Android ukázka obrazovek

Dále se zde nachází Průvodce aplikací, který uživatele uvede do aplikace a nakonec karta s nastavení, kde si uživatel může nastavovat proměnné a konstanty v aplikaci.

Po kliknutí na Přehled stáda je uživateli zobrazena obrazovka, kde vidí všechny kusy v aktuální stáji. Má možnost si kusy uspořádat do skupiny. Po kliknutí na určité zvíře je uživatel přesměrován na detail tohoto zvířete. Detail zvířete obsahuje tři podsekce. První je Karta zvířete. Zde vidí základní údaje o zvířeti. Dále kartu s laktacemi a kartu se změnami. Obě tyto položky lze pořizovat, ale jakmile jsou nahrané na server, již je nelze z mobilu upravovat a jakékoliv další úpravy jsou možná pouze přes PC verzi programu.

Kapitola 2
Analýza

Již jsme si lehce představili základní prvky Farmsoftu. V této části vysvětlím podrobněji, jak celý systém funguje. Probereme si doménový model a jak spolu jednotlivé entity interagují. Dále zde uvedu, co by měla výsledná aplikace splňovat a co by měla umět. Nakonec si rozebereme technologie, které je možné použít pro vytvoření iOS aplikace.

2.1 Fungování podniku

Tato kapitola slouží čtenáři pro přiblížení fungování zemědělského podniku, či družstva. Budu se zde zaměřovat jenom na správu fungování kolem skotu. Nebudu rozebírat další aspekty podniku jako je pěstování plodin, popřípadě chov dalších zvířat. Začneme tedy představením možných podniků.

Podnik může být malý, kdy je tvořen jednou stájí s několika málo kusy dobytka. Potom může být středně velký, který může mít více stájí, které můžou být od sebe vzdáleny i několik kilometrů. Poslední kategorií jsou opravdu velké podniky, které skupují ostatní menší podniky a tvoří jeden obří podnik. [3]

Podnik vydělává na prodeji mléka a masa. Mastná produkce má za úkol zvíře pouze vykrmit a potom prodat na maso. Toto je však možné pouze tehdy, pokud kus netrpí žádnými chorobami. Mléčná produkce se soustředí na prodej mléka. Kráva dojí pouze, pokud má tele. Podnik se snaží, aby kráva za svůj život dojila co nejvíce a vydělala co nejvíce peněz. Kráva má tele jenom, pokud je oplodněna neboli inseminována. Tuto činnost typicky zařizují externí pracovníci, neboli inseminátoři. Ti krávu oplodní pomocí inseminační dávky. [3]

Každý podnik má člověka, jenž se stará o krávy. Tento člověk se nazývá zootechnik. Jeho úkolem je starat se o krávy, hlídat jejich zdravotní stav, kontrolovat mléko, rozhodovat zdali je kráva v říji, a tudíž připravená pro inseminaci. [3]

Podnik je úspěšný, pokud vydělává peníze. Každé zvíře je třeba krmit. Krávy pro mléčnou produkci je třeba oplodňovat. Podnik má i další výdaje za energie a platy zaměstnanců. Každé zvíře tedy stojí podnik určitý finanční obnos. Cílem podniku je najít ideálně cestu, jak za každé zvíře zaplatit co nejméně, ale ne však na úkor komfortu zvířete. Toho lze dosáhnout například ideálním dávkováním krmení, popřípadě maximalizování šance na oplodnění po první inseminační dávce, tak aby nebylo potřeba platit další inseminační dávky. Toto téma je velice rozsáhlé a nad rámec této práce. V následující části 2.1.1 jej rozeberu trochu důkladněji. [3]

2.1.1 Životní cyklus zvířete

Cyklus začíná narozením zvířete. Po narození je nutné zvíře označit ušní známkou. Označení musí proběhnout do 72 hodin od narození zvířete plastovou ušní známkou v levém uchu a kovovou v pravém. Obě ušní známky obsahují identifikační číslo zvířete. Identifikační číslo je složeno z kódu země a devítimístním číslem. Identifikační číslo může být například CZ123456789. Toto číslo zůstává zvířeti celý život a ani po jeho smrti nemůže být přiděleno jinému zvířeti. [6]

Zvíře vždy chováme za nějakým účelem. Zvíře můžeme chovat pro mastnou produkci či dojnou produkci. Existují ještě další účely chovu. Zvíře můžeme chovat i pro pracovní účely, či pro účely zábavy jako je například korida. Zde však proberu pouze chov za účelem mastné a dojné produkce. [6]

Pokud je zvíře určeno pro mastnou produkci, potom je vykrmeno a následně prodáno. V takovém případě se zvíře snažíme dostatečně vykrmit. Více vykrmené zvíře může být prodáno za vyšší částku při prodeji na maso. Zvíře je pravidelně váženo a je prodáno na maso pouze, pokud je dostatečné mohutné. Díky pravidelnému vážení můžeme odhadnout, kdy zvíře bude dostatečné hmotné pro následující prodej. [3]

Pro zvíře chované na dojnou produkci je situace složitější. Dojit mohou pouze samice a pouze, pokud mají tele. Samice má tele, pokud je oplodněna. K oplodnění může dojít přirozenou cestou, či uměle inseminační dávkou. Oplodnění je úspěšné pouze, pokud je zvíře v říji. Říje trvá 12 až 24 hodin a průměrně se opakuje po 21 dnech. Říje se u zvířete projevuje krom jiného zvýšenou pohybovou aktivitou a snížením nádoje mléka. Pokud jsou u zvířete pozorovány tyto příznaky, je ideální doba pro oplodnění daného zvířete. [6]

Pokud oplodnění bylo úspěšné, říkáme že zvíře zabřezlo. Zvíře tedy čeká nového potomka. Délka březosti je typicky 285 - 289 dní. Po této době se zvíře otelí (porodí) a dojde k narození telete. Zvíře, které se zatím neotelilo, se nazývá jalovice. Již otelené zvíře se nazývá kráva. Po otelení začíná období laktace. Prvních několik dnů laktace produkuje kráva mlezivo. Mlezivo se využívá jako potrava pro nově narozená telata. Poté následuji již samotná produkce mléka, které podnik prodává. Kráva mléko produkuje až do konce laktačního období. [6]

Cílem je, aby laktační období u zvířete bylo co nejdelší a zvíře tedy vyprodukovalo co nejvíce mléka. Proto je zvíře nutné co nejdříve oplodnit, aby období bez laktace bylo co nejmenší možné. U jalovic je optimální hmotnosti pro oplodnění 400 kg, kterou jalovice má typicky v 16. až 18. měsíci. U krav je potřeba nechat zregenerovat dělohu. ”Doporučuje se zapouštět krávy 60 až 80 dní po otelení, krávy s nižší mléčnou užitkovosti můžeme zapouštět o něco dříve.”[6] V této době již by děloho měla být připravena pro další oplodnění. [6]

Každá kráva se před otelením nechává zaprahnout, či zasušit. Jedná se o proces, kdy kráva přestane před otelením dojit, aby tělo mělo čas se uzdravit a připravit na další otelení. Zaprahnout se nechávají pouze krávy, u jalovic zaprahnutí nedává smysl, protože ještě nedojí. Po zaprahnutí následuje další otelení a celý postup se opakuje. [6]

Samci jsou určeny převáženě na mastnou produkci. Pouze ti nejlépe geneticky vybavení jedinci jsou určeny pro odběr spermatu, který je následně používám v inseminačních dávkách. Popřípadě jsou využíváni k oplodnění samic přirozenou cestou. [3]

Každé zvíře může trpět onemocněním. Zootechnik popřípadě lékař určí diagnózu a rozhodne, jaké léky se zvířeti mají podat. Pokud kráva onemocní, potom je zakázáno její mléko prodávat a mléko je zlikvidováno. Speciální druh léčení jsou paznehty. Paznehty jsou podobné lidským nehtům. Jsou umístěny na konci končetin u skotu. Podobně jako lidské nehty je třeba i o ně pečovat a zkracovat je. Navíc paznehty mohou trpět celou řadou onemocnění. [3]

Cyklus končí smrtí zvířete. Tím jsme si opravdu hodně zjednodušeně přiblížili problematiku chovu skotu. Celé téma je samozřejmě o dost složitější, ale věřím, že tento základní pohled pro pochopení Farmsoftu bude dostatečný.

2.2 Doménový model

V této kapitole bych rád představil doménový model. Nepředstavíme si všechny entity Farmsoftu, ale jenom podstatné entity pro tuto práci. Doménový model jsem se kvůli zjednodušení rozhodl rozdělit na dvě části. První je základní model. Ten představuje základní entity a ukazuje nám jejich provázanost. Druhá část bude zaměřená pouze na změny a entity, které jsou s nimi provázané.

Začneme tedy základním modelem. Ten vidíme na obrázků 2.1. Hierarchie entit má stromovou strukturu, kde se v kořenu nachází entita Cloud. Cloud je reprezentace zákazníka. Čili pokud přijde nový platící zákazník, založí se pro něj nový Cloud. K entitě Cloud se vztahují všechny entity v systému a tato vazba je povinná. Každá entita musí být součástí právě jednoho Cloudu. Tento fakt by do schématu na obrázku 2.1 přinášel zbytečnou složitost, proto jsem ho zde neuvedl a naznačil pouze ostatní vazby. [3]

Každý Cloud může mít několik uživatelů. Uživatel získá přihlašovací údaje. Ty následně použije pro přihlášení do systému Farmsoft. Uživatel má kromě standardních položek jako jméno a heslo ještě položky představující práva daného uživatele. Jsou jimi právo na pořizování, respektive mazání změn, právo používat mobilní aplikaci, právo mít aktivované všechny stáje a právo volby technika. [3]

Tím se dostáváme k technikovi. Technik se využívá při zadávání změny inseminace, kdy při zadávání této změny vybereme, který technik inseminaci provedl. Může to být i externí pracovník z jiné firmy. Uživatel buď má právo technika zvolit, nebo pokud toto právo nemá, je mu technik přidělen. Tento přidělený technik je automaticky zvolen při zadávání inseminace. [3]

Uživatel stáje slouží pro určení přístupu do jednotlivých stájí v daném Cloudu. Touto vazební entitou se tedy určí stáje, kam daný uživatel má, popřípadě nemá přístup. A tím se dostáváme k druhé části stromu. [3]

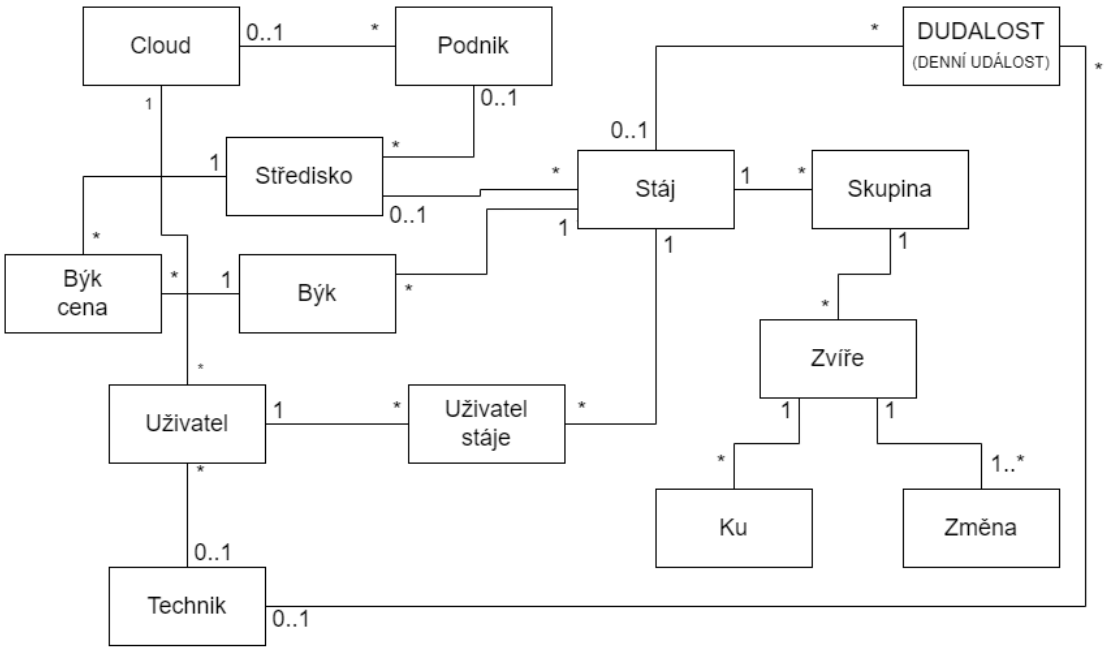
Zde máme sekvenci Podnik, Středisko a Stáj. Stáj z této trojice je nejmenší jednotka. Měla by reprezentovat určitou reálnou budovu. Středisko je kolekce těchto budov spojená společným atributem, například místem. Nad tím vším je Podnik, který je tvořen několika středisky. Podnik lze využít, pokud dojde k odkoupení jiného podniku a jeho následnému začlenění do systému Farmsoft. Jedinou povinnou položkou z této trojice je Stáj. Ostatní jsou volitelné. Takže může existovat Stáj, která nepatří do žádného Střediska a taktéž může existovat Středisko, které nepatří do Podniku. Hierarchii je tedy celkem volná a je na zákaznících, jak se rozhodnou jí poskládat. Hierarchie je i celkem složitá, ale pokryje potřeby velkých i malých podniků. [3]

Další část hierarchie tvoří Skupina a Zvíře. Skupina musí být součástí Stáje. Dále máme Zvíře, které musí být členem nějaké Skupiny. Možnost, kdy Zvíře není součástí Skupiny, respektive ve Skupině není nastavena Stáj, je sice možná a systém jí dovoluje, ale nepovažuje se za validní, a proto je kladen důraz na to, aby všechna zvířata byla ve skupině a každá Skupina patřila do Stáje. [3]

Podnik, Středisko, Stáj a Skupina mají vždy atribut jméno a číslo. Jméno není povinné, Číslo ano. Za zmínění dále stojí uchovávání GPS polohy u Stájí. Ta se používá pro automatické přepínání Stájí, které se vysvětlíme později.

Ke každému zvířeti se následně vztahují dvě entity změny a kontrola užitkovosti (ku), neboli laktace. Kontrola užitkovosti se využívá při kontrole mléka. Každý podnik musí jednou za čas odeslat vzorky nadojeného mléka ke kontrole do laboratoře. Laboratoř pošle podniku zpět výsledky, které následně vidí ve Farmsoftu. Obsahuje přes 20 různých položek. Jsou jimi například datum dojení, nádoj, tuk atd. Nepovažuji důležité zde vysvětlovat všechny položky laktace podrobně, když u některých jako třeba Nlátky, popřípadě Ketolátky si autor práce sám není jistý, co reprezentují. Ještě však zmíním položku Zdroj, která nám určuje zdroj dat. Může se o jednat o manuální zadání nebo nespecifikováno. Nespecifikováno je typicky použito právě při vyhodnocení z laboratoří. [3]

Druhou entitou je Změna. Každé zvíře by mělo mít vždy minimálně jednu změnu a to narození. Změna reprezentuje událost v životě zvířete, například přesun mezi stájemi, inseminace, březost



Obrázek 2.1 Základní doménový model

atd. Samotný systém změn si detailněji probereme v samostatné sekci 2.2.1.

Dále si představíme Denní událost (dudalost). Denní události slouží k posílání zpráv napříč podnikem. Je zde možnost přiřazení události na určitou stáj, popřípadě určitému technikovi. Uživatel si může definovat druh a samotnou zprávu. Ostatní uživatelé jí potom v systému vidí. Některé podniky jí dokonce umí promítat na velký display přímo na stáji. [3]

Zbývá nám představit ještě býci a býci cena. Býci reprezentují inseminační býky, kteří se používají pro inseminaci. Býk obsahuje jeho registrační číslo, jméno, cenu a stáj. Atribut ceny a stáje jsou dané historicky, kdy bylo třeba určit cenu daného býka na pro danou stáj. Nyní tuto úlohu přebírá Býk cena, který se již neváže ke stáji, ale ke středisku. Navíc umí určit cenu i od určitého data. [3]

2.2.1 Změny

Změny jsou nejdůležitější položkou Farmsoftu. Tvoří profil zvířete v systému a entita Zvíře je s nimi přímo provázaná. Zvíře jenom reprezentuje souhrn informací získaných na základě změn. Jinými slovy při zadávání změny typu narození nám zároveň vzniká entita typu zvíře. Tvorbou dalších změn je instance příslušného zvířete aktualizována podle hodnot příslušných změn. Pokud zvíře smažeme, jsme schopni ho celkově obnovit jenom na základě zadaných změn. Tento postup se však v praxi neuplatňuje. [3]

Změny se rozdělují do několika typů. Typ je určen položkou ID_DRUH_ZMENY. Kompletní seznam typů a jejich id vidíme v tabulce 2.1. Změna obsahuje další položky. Její částečný seznam vidíme v SQL scriptu pro vytvoření tabulky změny 2.1. Nejdůležitější položkou je Datum. Ten určuje datum provedení změny a jedná se o libovolný údaj jak v minulosti, tak v budoucnosti. DATUM_PORIZENI je datum, kdy byla změna vytvořena. POCET_OPRAV nám říká, kolikrát byla změna upravena. Položky DATUMCAS1-2, CISLO1-9, FLOAT1-11, DECIMA1-3, ZNAKY1-7 jsou generické a její význam se určuje až podle druhu dané změny. Položky jako ID_POHLAVI a ID_BYK jsou více zaměřené pro potřeby jednotlivých změn. Například ID_BYK je využíván při změně inseminace. Poslední položkou, kterou zmíním, jsou Data. Ta

slouží pro ukládání obrázků a její význam je opět určen druhem změny. [3]

Teď bych rád představil konkrétní typy změn. Řekneme si co konkrétní druhy znamenají a jaké položky obsahují. Konkrétní mapování změn na tabulku však rozebírat nebudu.

Tabulka 2.1 Tabulka druhů změn a příslušných id

ID_DRUH_ZMENY	Název
2	Vyřazení
6	Kastrace
9	Přesun
13	Narození
15	Označení
16	Nákup
17	Otelení
18	Říje
19	Inseminace
20	Březost
21	Zaprahnutí
22	Krmení
23	Plány
25	Užitkovost
27	Poznámka
28	Nemoc dojírna
30	Léčení
31	Vážení
32	Selekce
33	Připařovací plán
34	Hodnocení zevnějšku
35	Cena
36	Přirozená plemenitba

2.2.1.1 Narození

Narození je počáteční změna každého zvířete. Při zadání změny narození je zároveň s ní vytvořeno samotné zvíře. Je povinné zadat datum narození, zemi, číslo a pohlaví. Pohlaví jsou Jalovička pro samice a Býček pro samce. [3]

Nepovinný, ale v České republice využívaný atribut je kodex. V ČR je identifikační číslo zvířete tvořeno ze země, čísla kusu a kodexem. Kodex je poslední trojčíslí a používá se například pro rozlišení samců a samic. V jiných zemí však může být využívat odlišně či vůbec. [3]

Dále je možnost zadat číslo kusu matky, otce a otce matky a jejich plemenné příslušnosti. Plemenné příslušnosti je ekvivalence národnosti u lidí. Taktéž můžeme zadávat sourozence. Zadáním těchto atributů získáme potřebný rodokmen zvířete. [3]

Změna umožňuje vyplnit datum označení. To je proces, kdy je zvířeti po porodu dána fyzická identifikace ať už v podobě čipu či ušní známky. [3]

A konečně lze zadávat barvu, jméno a popřípadě fotografii zvířete. Fotografie nereprezentuje fotku při narození, ale aktuální fotku zvířete, a tudíž může být v průběhu jeho života několikrát změněna. [3]

Výpis kódu 2.1 Část SQL scriptu pro založení tabulky změny

```
CREATE TABLE ZMENY
(
  ID_ZMENY bigint NOT NULL,
  ID_KUSU integer NOT NULL,
  ID_DRUH_ZMENY integer NOT NULL,
  DATUM timestamp NOT NULL,
  USERNAME varchar(20),
  DATUM_PORIZENI timestamp,
  DATUM_POSLEDNI_OPRAVY timestamp,
  POCET_OPRAV integer DEFAULT 0,
  DATUMCAS1 timestamp,
  DATUMCAS2 timestamp,
  CISLO1 integer,
  CISLO2 bigint,
  CISLO3 bigint,
  CISLO4 integer,
  CISLO5 integer,
  CISLO6 integer,
  CISLO7 integer,
  CISLO8 integer,
  CISLO9 integer,
  FLOAT1 float,
  FLOAT2 float,
  FLOAT3 float,
  FLOAT4 float,
  FLOAT5 float,
  FLOAT6 float,
  FLOAT7 float,
  FLOAT8 float,
  FLOAT9 float,
  FLOAT10 float,
  FLOAT11 float,
  DECIMAL1 decimal(13,0),
  DECIMAL2 decimal(13,0),
  DECIMAL3 decimal(13,0),
  ZNAKY1 varchar(128),
  ZNAKY2 varchar(15),
  ZNAKY3 varchar(15),
  ZNAKY4 varchar(15),
  ZNAKY5 varchar(15),
  ZNAKY6 varchar(15),
  ZNAKY7 varchar(15),
  DATA1 blob sub_type 0,
  ID_POHLAVI integer,
  ID_POHLAVI_SOUROZENCI integer,
  ID_MRTVE_ROZENI_SOUROZENCI integer,
  ID_ZEME1 integer,
  ID_ZEME2 integer,
  ID_ZEME3 integer,
  ID_SKUPINA integer,
  ID_DODAVATEL_ODBERATEL integer,
  ID_TECHNIK integer,
  ID_BYK integer,
  ID_CHARAKTER_OTELENI integer,
  ID_CHARAKTER_RIJE integer,
  ID_CHARAKTER_INSEMINACE integer,
  ID_CHARAKTER_BREZOSTI integer,
  .....
);
```

2.2.1.2 Nákup

Nakup je pro stav, kdy zvíře bylo pořízeno od jiného chovatele. Při zadávání nákupu je samozřejmě nutné zadat i změnu narození zvířete, aby údaje byly kompletní. Pro nákup je nutné zadat pouze datum nákupu. Volitelné položky jsou odběratel, cena, faktura a poznámka. [3]

2.2.1.3 Vyřazení

Vyřazení je opak narození. Při zadáním této změny je zvíře odstraněno ze stáda. Povinné údaje jsou datum vyřazení a druh. Druh může být prodej, úhyn či odcizení. Dále můžeme zadat odběratele a příčinu vyřazení. Nakonec zadáváme hmotnost a procento srážek hmotnosti. Tyto atributy jsou využity hlavně při prodání kusu na jatka. [3]

2.2.1.4 Plány

Plány přímo souvisí s vyřazením. Reprezentují stav, kdy k určitému datu dáváme zvíře na prodej. Zadává se datum změny, datum plánovaného prodeje a vyřazení daného kusu z reprodukce. Pokud je zvíře vyřazeno z reprodukce, potom se s ním nepočítá pro další reprodukční cyklus. [3]

2.2.1.5 Kastrace

U změny typu kastrace je jenom jedna položka a to datum provedení. [3]

2.2.1.6 Poznámka

Poznámka dává možnost uživateli si ke zvířeti napsat poznámku podle libosti. Zadává se datum, samotná poznámka, druh poznámky a je možnost nahrát jednu fotografii. Druh je kladné celé číslo, které má pomoci pro lepší členění poznámek. Nabývá hodnot od 1 do 6. Fotografie může být jakýkoliv obrázek. Poznámka může být jakýkoliv text.

2.2.1.7 Označení

Změna reprezentuje přidání zvířeti další identifikační atribut. Tyto atributy se na rozdíl od čísla kusu mohou v průběhu života zvířete měnit. Zvíře je typicky opatřeno ušní známkou, kde lze najít jeho kompletní číslo. Dále mu však může být přidělen obojek s číslem, které nemusí být unikátní. Většinou je však unikátní v rámci skupiny. Číslo toho obojku je první atribut, který může uživatel při tvorbě této změny zadat. [3]

Další je EID. To je elektronická identifikace. Jde o položku, která se využívá při identifikaci například RFID čtečkou. Jedná se o ID čipu, který má zvíře na sobě. [3]

Poslední položkou je vitalimetr. Vitalimetr je výrobek společnosti Farmtec a.s, který funguje jako obojek, avšak zároveň přidává možnosti monitoringu zvířete a je schopný upozornit na říjí či zdravotní problémy. [7]

Všechny tyto položky jsou nepovinné. Jedinou povinnou položkou je datum označení.

2.2.1.8 Přesun/Cena/Vážení

Jelikož jsou tyto druhy změn velice podobné, tak je spojím do jedná sekce. U všech těchto změn je nutné zadat datum provedení a příslušný atribut. [3]

U přesunu to je skupina, kam bylo zvíře přesunuto. Ta vychází z organizační struktury podniku. U vážení je nutné zadat váhu v kilogramech. Tato změna reprezentuje váhu daného zvířete v určitém čase. Cena reprezentuje odhad ceny zvířete. Také se používá pro odhad aktiv podniku a pro následné ekonomické účely. [3]

U všech těchto položek je taktéž možnost zadat poznámku k provedené změně a taky její druh. Druh představuje, jestli dané měření bylo provedeno v rámci většího skupinového měření a nebo bylo provedeno individuálně. Popřípadě druh určit nemusíme. [3]

2.2.1.9 Otelení

Otelení reprezentuje porod. Je nutné zadat datum otelení a pořadí laktace. Tato hodnota reprezentuje, kolik porodů dané zvíře absolvovalo. Potom je možné zadat charakter otelení. Charakter nám naznačuje, jak daný porod probíhal. Možnosti jsou normální, těžší, s komplikacemi, císařský řez a neurčeno. Dále je možné zadat čas porodu, poznámku a počet porozených mrtvých zvířat. Dále je možné zadat čísla kusů nově narozených zvířat. [3]

2.2.1.10 Říje

Změna říje reprezentuje stav, kdy je zvíře v říji, respektive kdy říje byla zjištěna. Zadává se datum říje a její charakter. Charakter má 5 hodnot. Jsou jimi normální, tichá, vyvolaná, vitalimetr a pozorovaná. Existuje možnost zadat technika. [3]

2.2.1.11 Inseminace

Inseminace by měla být zadávána vždy, když se zvíře inseminuje. Zadává se datum provedení, pořadí inseminace, inseminační býk a charakter. Pořadí je kladné celé číslo. Toto číslo říká, kolikrát byla kráva inseminovaná od posledního otelení. Inseminační býk reprezentuje dávku podanou při inseminaci. Je to jedna položka z entity býk, viz obrázek 2.1. [3]

Poslední povinnou položkou je charakter. Jedná se o flag, který může nabývat 4 hodnot - inseminace, přirozená plemenitba, reinseminace a embryotransfer. Přirozená plemenitba je historický flag a nově byla nahrazena samostatnou změnou. Flag inseminace reprezentuje standardní inseminaci. Reinseminace nastává, pokud kráva byla inseminováva nebo reinseminována v posledních několika dnech (typicky v 15 dnech). V takovém případě se pořadí nezadává a zůstává pořadí z předchozí inseminace. Přirozená plemenitba popíšu v samostatné sekci 2.2.1.12 při popisu změny. Embryotransfer je proces, kdy jsou vajíčka z jiné krávy přemístěny do zvolené krávy. V takovém případě se ještě navíc zadává matka, neboli kráva, která svá vajíčka poskytla. [3]

Další nepovinné položky jsou technik, dávka, sexované sperma, počet hodin od začátku říje, poznámka, pedometr a odečíst ze skladu. Technik reprezentuje interního nebo externího člověka, který inseminaci provedl. Dávka reprezentuje množství spermatu podaného při inseminaci. Sexované sperma, pedometr a odečíst ze skladu jsou typu bool a výchozí hodnota je false. [3]

2.2.1.12 Přirozená plemenitba

Je druhý možný proces oplodnění krávy, kdy proces probíhá naprosto přirozeně. Vlastně se pustí býk mezi stádo krav a vše se nechá na přírodě. [3]

Při zadávání této změny je nutné vložit stav a býka. Býk reprezentuje použitého býka a stav nabývá dvou hodnot, zařazeno a vyřazeno. Stav nám tedy určí období, kdy daná kráva byla ve stádu s daným býkem. Dále je možné zadat nepovinnou poznámku. [3]

2.2.1.13 Březost

Březost je změna reprezentující, že daná kráva zabřezla a čeká potomky. Zadává se datum a charakter. Datum je typicky datum, kdy se daná březost odhalila vyšetřením. Charakter je kladné celé číslo, které nabývá 4 hodnot. Jsou to březí, jalová, otevřené a zmetání. [3]

2.2.1.14 Zaprahnutí

Kráva po otelení začne produkovat mléko. Pokud však chceme krávu znovu co nejrychleji otelit, aby mohla produkovat další mléko, je lepší jí nechat zregenerovat a nedojit. Zaprahnutí je proces, kdy jsou krávě v laktaci podány přípravky, které laktaci zabrání a ona může rychleji zregenerovat. Ve Farmsoftu se pro tuto změnu zadává jenom datum provedení zaprahnutí. [3]

2.2.1.15 Krmení

Při krmení je zadáváno datum krmení. Je možné zadat typ vypočtu dávky, užitkovost pro výpočet, krmná křivka, manuální dávku 1 a 2 a nakonec výživové kategorie. Typ vypočtu dávky, nabývá tří hodnot. Jsou to automatická, krmná křivka a manuální. Užitkovost pro výpočet může být jedna ze tří hodnot a to plánovaná užitkovost, milkmetr a ku. Krmná křivka ta reprezentuje uživatelem definovanou krmící křivku. Zbylé položky jsou textová pole pro kladná celá čísla reprezentující danou hodnotu. [3]

2.2.1.16 Selekce

Selekce je změna, pokud je potřeba dané zvíře selektovat ze stáda a provést na něm nějaké další úkony, například vyšetření. Selekce se provádí manuálně nebo automaticky. Manuální je, když zootechnik prochází stádo a hledá příslušné kusy, které následně odvede. Automatická je za pomocí selekčních branek, které zvíře separují od stáda pokud je zvíře v selekci. [3]

Při zadávání selekce je nutný pouze datum, další položky jsou volitelné. Zadává se důvod selekce. Jde o kladné celé číslo reprezentující důvod. Jedná se o hodnoty v rozmezí 1-11, kde každé číslo reprezentují určitý důvod. Například 1 je inseminace, 2 je zjišťování březosti atd. Je možnost zadat skupinu, do které zvíře bylo selektováno. Dále je možnost zadat pořadí. Opět se jedná o kladné celé číslo a záleží na uživateli, co tato položka reprezentuje. Obvykle se jedná o denní doby provedení selekce. Předposlední položkou je jednotka. Jedná se o kladné celé číslo, které se používá k filtraci selekce. Pokud má stáj více selekční branek, tak právě touto hodnoto nastavuje, na které brance chceme selekci aplikovat. Pokud není vyplněno, je aplikována všude. Poslední položka reprezentuje stav provedení, konkrétně jestli daná selekce již byla provedena. Nabývá hodnot ano či ne. [3]

2.2.1.17 Užitkovost

Užitkovost slouží pro získání vlastností zvířete. Změna užitkovosti je typicky generována programem Plemdat. Farmsoft si umí tento report z plemdatu importovat a následně ho zobrazit jako samostatnou změnu typu užitkovost. Veškeré výpočty jsou prováděny programem Plemdat a Farmsoft je pouze zobrazí.

2.2.1.18 Hodnocení zevnějšku

Hodnocení zevnějšku se používá při vizuálním pohledu na dané zvíře. Zootechnik zde určuje skóre pro jednotlivé body ve formuláři. Formulář se může lišit na základě plemenné příslušnosti hodnoceného zvířete. Hodnocené věci jsou například: kondice, kvalita kostí, vemeno, sklon zádě atd... Na základě provedeného hodnocení následně vzniká graf popisující vlastnosti tohoto zvířete. [3]

Tento graf se využívá při následném šlechtění. Pro příklad z tohoto měření zjistíme, že daná kráva má určité vlastnosti horší. Proto do připařovacího plánu této krávy vložíme býka, o kterém je známo, že tuto vlastnost vylepšuje. Potom je velká šance, že jejich mláďata tuto vlastnosti již budou mít v pořádku. [3]

2.2.1.19 Připařovací plán

Připařovací plán neboli SireMatch je změna, která dané krávě vybere ideálního býka pro inseminaci. Tito býci jsou potom při zadávání změny inseminace nabízení přednostně. Je možné zadat až tří býky. Zadává se registrační číslo býka, jméno býka a zdali je sperma sexované. Registrační číslo se zadává jako prostý text a nejedná se o odkaz do aktuální kolekce býků. [3]

2.2.1.20 Nemoc dojírna

Nemoc dojírna je změna, která nám říká, že v mléku dané krávy byly nalezeny látky, díky kterým je mléko neprodejné a musí se zlikvidovat. Zadává se u ní datum, od kdy je daná kráva neschopná dojit a počet dojení, který přeskočí. Následně je možné zadat dva kódy nemocí, kvůli kterým nemůže dojit. [3]

2.2.1.21 Léčení

Léčení, občas v systému nazýváno zdraví, je jednou z nejsložitější změn v Farmsoftu. Reprezentuje jakékoliv lékařské zákroky na daném zvířeti. Zadává se datum stanovení léčení a kdo léčení stanovali. Lze vybrat chovatel, veterinář, paznehtář, plemenář a ostatní. Dále se zadává druh, což je pouze číselný příznak používaný pro členění léčení v podniku. Uživatel má možnost zadat náklady, jedná se o číslo určující náklady na léčbu a nakonec může zadat textovou poznámku k léčbě. [3]

Nejdůležitější je však zadávání diagnóz a léčiv. Diagnózy reprezentují nalezené choroby a nemoci. Uživatel má k dipozici několik předefinovaných diagnóz a další si může dodefinovat. Diagnóza obsahuje kód a název. Uživatel má vždy možnost vybrat několik diagnóz k jednomu léčení. [3]

Zadávání léčiv funguje podobně jako u diagnóz. Zadávání léčiv reprezentuje podání léků na potlačení diagnostikované nemoci. Léky jsou definovány v entitě léčiva. Uživatel si opět má možnost tato léčiva definovat dle libosti. Léčiva obsahují velké množství položek. Pro tuto práci jsou důležité jenom dvě a to jméno léčiva a jednotka, ve které je léčivo podáváno. [3]

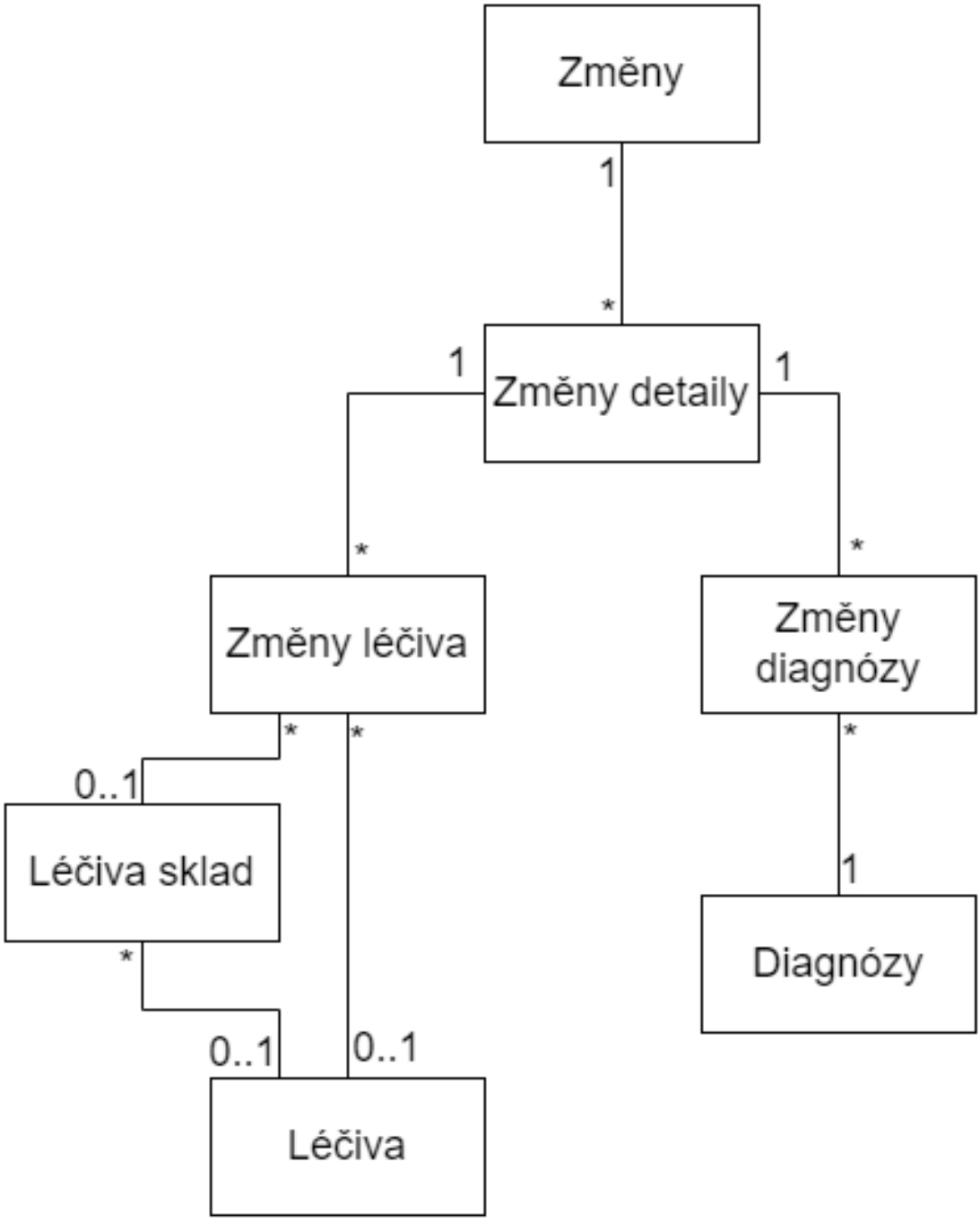
U léčení dále definujeme sklad léčiv. Ten nám pomáhá monitorovat, kolik léčiv aktuálně má podnik k dispozici. Vlastně reprezentuje reálné zásoby daných léků. Zadává se zde nakoupené množství, šarže, datum expirace léků atd. [3]

Kompletní provázání entit vidíme na obrázku 2.2. Zde jsou vidět taktéž vazby mezi jednotlivými entitami. Změna typu léčení má vazbu na N entit typu změny detail. Ta se u normálního léčení nepoužívá. Používá se při zadávání paznehtů popsaných v sekci 2.2.1.22. U standardního léčení se tedy jedná jenom o vazební entitu. Potom zde máme změny léčiva a diagnózy. To jsou vazební entity propojující změnu a použitá léčiva, respektive stanovené diagnózy. Jak je na obrázku vidět, tak změny léčiva mají vazbu buď na léčiva sklad, tak je tomu u standardního léčení, nebo přímo na léčivo. Druhá možnost nastává pouze u zadávání paznehtů. [3]

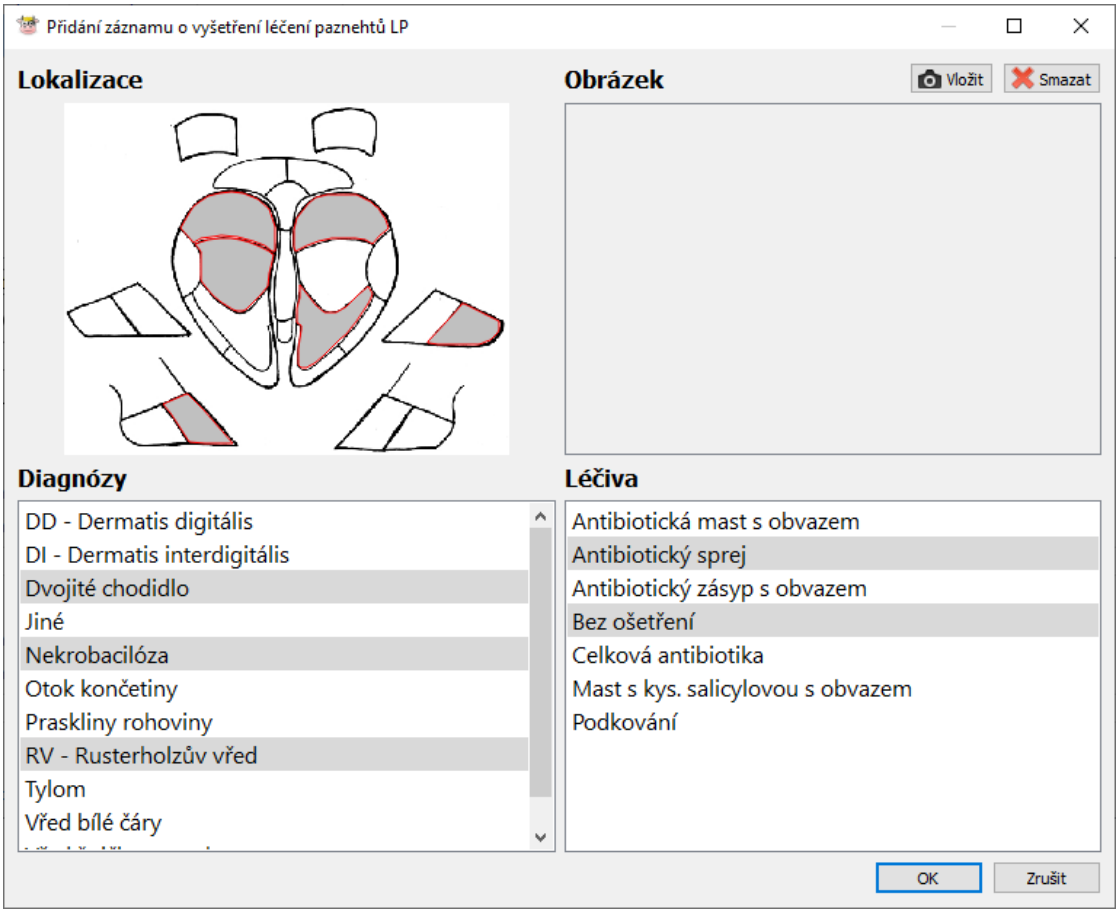
Nakonec si uživatel může naplánovat kontroly popřípadě podávání léků a to i pro určitý počet opakování. Těchto plánů uživatel opět může mít několik. Maximální počet je však 7 plánů na jedno léčení. Tyto plány lze dále řetězit, kdy po skončení prvního plánu se aktivuje další plán. [3]

2.2.1.22 Léčení - paznehty

Léčení paznehty je speciální druh změny typu léčení. Jedná se o proces, kdy dochází k léčení respektive vyšetření paznehtů zvířete. zadává se datum vyšetření a jeho typ. Typ může být FÚP a kontrola. Dále se zadává LCS což je hodnotící škála. Hodnocení je celé číslo od 1 do 5, popřípadě nehodnoceno pokud k hodnocení nedošlo. Potom je možnost zadat poznámku v textové podobě k danému vyšetření. [3]



Obrázek 2.2 Doménový model změn léčení



Obrázek 2.3 Ukázka zadávání léčení paznehtu v aplikaci Farmsoft

Poté se přejde k samotnému vyšetření jednotlivých končetin. Uživatel má možnost pro každou končetinu zadat, zdali byla vyšetřena. Závažnost nalezených problému s indexy celých od 1 do 4 včetně nehodnoceno a ohodnotit DDScore od 1 do 5 včetně neohodnoceno, opět v celých číslech. Zda však existuje zvláštní přemapování způsobené potřebou zákazníků, kde pro hodnotu 5 v systému uživatel vidí hodnotu 4.1. U každé končetiny je navíc možnost zadat záznamy léčby. Těch může být libovolné množství. [3]

Pro každý záznam je možnost zadat libovolné množství diagnóz a léčiv podobně jako u léčení. Lze však vybrat jenom léčiva a diagnózy se speciálním příznakem, že se jedná o léčiva respektive diagnózy určená pro paznehty. Je možnost zadat lokalizaci léčení určitého místa paznehtu a nahrát obrázek. Zadávací formulář záznamu léčení vidíme na obrázku 2.3. [3]

Nakonec má uživatel možnost si naplánovat další vyšetření paznehtů. Avšak ne tak komplexně jako u léčení. Zde si pouze nastaví počet dní za kolik chce být upozorněn. [3]

2.3 Návrh mobilní aplikace

V době mého příchodu do projektu už existovala základní Android verze aplikace, a tudíž i analýza, co by mobilní aplikace měla umět. Na této analýze jsem se dále podílel a vylepšoval ji. A právě dle této původní analýzy jsem i programoval iOS aplikaci.

Dle původního návrhu měla aplikace sloužit především k vytváření změn. Nemusí pořizovat všechny změny, které systém podporuje, ale pouze ty, u kterých má smysl, aby je uživatel zadávat

v terénu a tyto změny následně nahrát na server, kde jsou uloženy a zpracovány. Po zpracování změn serverem již není možné tyto změny v mobilu jakkoliv upravovat ani mazat. Všechny tyto operace jsou dostupné pouze přes PC verzi Farmsoftu. Aplikace musí být schopná zadávat změny v offline módu a jednou za čas tyto změny v podobě balíku nahrát na server. Tomuto procesu říkáme synchronizace.

Postupně se však požadavky na aplikaci navyšovaly, například nutnost komunikace aplikace s Bluetooth zařízeními. Pokročila filtrace stáda včetně automatického určení stáje na základě polohy atd. Nakonec přišla nutnost udělat speciální verzi aplikace pro Rskot. Každé mobilní aplikace by měla jít vytvořit ve dvou variantách pro Farmsoft a pro Rskot.

Design aplikace by měl odpovídat PC verzi aplikace Farmsoft. Po přihlášení uživatel uvidí úvodní obrazovku. Na té by měl vidět hlavní informace o přihlášeném podniku. Bude mít možnost synchronizovat aplikaci a uvidí čas poslední synchronizace. Dále se přes hlavní obrazovku může dostat do ostatních částí aplikace jako je nastavení, stádo, list inseminačních býků atd.

2.3.1 Přihlášení

Aplikace by měla umět komunikovat s Cloudem i s FTP. Z obou těchto datových zdrojů si musí umět stáhnout informace, s kterými pracuje. Pro přihlášení pomocí cloudu je zadáváno číslo podniku, následně jméno a heslo. Přes FTP se k přihlášení používá pouze číslo podniku a heslo. Po přihlášení dojde k založení podniku v mobilu. Mobil bude mít možnost založit si několik paralelně fungujících podniků.

2.3.2 Synchronizace

Po přihlášení podniku dojde k jeho synchronizaci. Během synchronizace si systém stáhne veškeré dostupné záznamy ze serveru, které mobil potřebuje pro fungování aplikace v offline módu. Dále během synchronizace jsou odeslány pořízené změny.

Kvůli fungovaní aplikace v offline módu je třeba vyřešit možné konflikty a zároveň aktualizaci již stažených dat v mobilu. Kvůli tomu má každá tabulka v databázi synchronizační číslu (SN). Při každé úpravě dané řádky je zároveň zvýšeno globální sn a toto nové sn je upraveno v záznamu daného řádku. Mobil si tedy pamatuje poslední sn, které si stáhl. Při nové synchronizaci nemusí stahovat všechny data znova, ale stáhne jenom poslední změny, které mají větší sn než má mobil z poslední synchronizace.

Problém, který sn nevyřeší, je smazání záznamů. Pro vyřešení tohoto problému vznikly takzvané tabulky Z. Z tabulka je kopie normální tabulky. Obsahuje však jenom 3 sloupečky. Jsou to ID položky, která byla smazána. SN a odkaz na podnik. Takže například pro změny se tato z tabulka bude jmenovat změny_z a ID položky bude ID smazané změny. Mobil si opět stáhne záznamy s vyšším sn než je jeho poslední uložené a ví, že tyto záznamy na serveru byly smazány, a tudíž se je odmaže i u sebe. Jelikož mobil nemůže upravovat již založené změny, tak není nutné řešit konflikty při úpravách těchto změn z mobilu.

Aplikace by se měla periodicky synchronizovat ideálně bez přičinění uživatele. V minulosti tato synchronizace byla vždy naplánovaná na ranní hodinu, aby uživatel při vstupu do nového pracovního dne měl nejnovější data. Od tohoto řešení bylo upuštěno kvůli technické náročnosti a nyní stačí pouze synchronizace, které se spustí jednou za určitý časový úsek.

2.3.3 Změny

V aplikaci bude možnost zobrazit si seznam zvířat a pro jednotlivá zvířata pořizovat změny. Tyto změny lze editovat a mazat do doby, než jsou nahrány na server. Změny nemusí vždy obsahovat stejné položky jako PC verze, ale pouze položky, které uživatel potřebuje při práci v terénu.

Aplikace by měla umožnit pořizovat následující změny: změnu označení s možností zadávat obojek eid a vitalimetr, změnu vážení s možností zadat váhu zvířete, poznámku s typem samotnou poznámku a fotkou k poznámce, změnu říje s jejím charakterem, změnu inseminace kde bude možné zadat pořadí, charakter a položky s definicí charakteru spojené, apříklad matku u embryotransferu. Dále inseminačního býka, technika, dávku a zdali sperma bylo sexováno. Býci v přípařovacím plánu zvířete budou při zadávání inseminace zvýraznění a na prvních pozicích. Dále změnu březosti s charakterem březosti, změnu otelení pouze s položkou pořadí laktace, změnu zaprahnutí, změnu selekce s možností zadat důvod selekce, skupinu, pořadí, jednotku a splnění dané selekce, změnu umístění při které bude možné zadávat novou skupinu, změnu nemoc dojírna s možností zadat počtu blokace dojení a kódů obou nemocí. A konečně paznehty s možností zadávání v plné míře jako umí PC verze. Změnu léčení bude mobilní aplikace umět zobrazovat, ale nebude umět jí pořizovat.

Dále je možnost pořizovat hromadné změny. V hromadných změnách si uživatel vybere více zvířat, na které poté aplikuje jednu změnu. Změny, které jdou aplikovat v hromadných změnách jsou poznámka, selekce a umístění.

2.3.4 Laktace

Aplikace by měla podporovat zobrazení a pořizování laktací. Jelikož je položek pro laktaci hodně a uživatelé vždy využívají pouze některé z nich, je nutné, aby si uživatelé položky mohli definovat. Uživatelé si tedy budou moci v aplikaci nastavit, jaké položky chtějí mít k dispozici při práci s laktací. Tyto položky se budou lišit pro zobrazení a pro zadávání. Laktace půjde filtrovat podle pořadí a to buď na konkrétní pořadí, popřípadě zobrazení pouze laktací z posledního cyklu.

2.3.5 Zvířata a filtrace

Aplikace umožní zobrazení zvířat. Mezi zvířaty půjde vyhledávat podle obojku a čísla kusu. Zvíře půjde vyhledat naskenováním čárového respektive QR kódu na kartě zvířete. Zvířata lze taktéž filtrovat podle pohlaví a zařazení respektive vyřazení z reprodukce. Dále pokud dojde k výběru stáje, budou zobrazena pouze zvířata na dané stáji.

Uživatel bude mít možnost zapnout automatické přepínání stájí podle polohy. Pokud je automatické přepnutí stáje aktivní a uživatel je 200 m od stáje, mobil ho upozorní na možnost přepnutí stáje. Uživatel si polohu stájí může nastavit přímo v aplikaci. Tato poloha je následně nahrána na server jako standardní změna.

Po kliknutí na dané zvíře se zobrazí karta zvířete. Na té uživatel vidí základní informace o zvířeti podobně jako v programu Farmsoft.

2.3.6 Denní události

Aplikace by měla mít možnost zobrazovat a pořizovat denní události v podniku. Události si půjde filtrovat podle data a půjde je i zadávat. Při zadávání lze zadat stáj, ve které se událost odehrává a technika. Dále je možné zadat druh události, což je kladné celé číslo sloužící pro rozlišení událostí. Dále dva celočíselné atributy, kterými jsou dojení a březost. Nakonec je možné zadat samotný textový popis události. V případě aktivní stáje jsou zobrazeny jenom události v dané stáji.

2.3.7 Sestavy

Sestavy v mobilní aplikaci byli vytvořeny na základě požadavků uživatelů mobilní aplikace. První sestava je pohybová aktivita. Ta zobrazí zvířata, která systém vyhodnotí jako potenciální kandidáty pro říji. Druhá je říjový deník. Ten zobrazuje zvířata, u kterých je daný den změna

typu říje. Tato sestava má sloužit pro zootechniky, kdy přehledně vidí, která zvířata jsou v říji a mohou s nimi dále pracovat. Třetí sestava je sestava paznehtů. Ta zobrazí všechny změny typu paznehty v definovaném období. S touto sestavou přímo souvisí sestava plány léčení paznehtů, kde uživatel vidí naplánované paznehty na dané období. Tyto plány může i splnit.

Prozatím nejdůležitější sestava je sestava inseminací. Ta zobrazí všechny změny typu inseminace. U každé změny bude vidět zvíře, datum a stáj, na které bylo zvíře umístěno v době inseminace. Sestava půjde filtrovat časem, technikem a množinou stájí. Uživatel může vybrat všechny stáje a nebo výčet jemu dostupných stájí. Dále sestava půjde seskupit podle data a stáje. Sestava půjde exportovat do pdf formátu v podobě tabulky a do formátu pro Plemdat podle specifikace zde: zde. Export je důležitý pro uživatele zvláště aplikace Rskot, kdy uživatelé nemají přístup k PC verzi. Tudíž jediná možnost, jak tyto informace exportovat, je pomocí mobilního zařízení.

Poslední sestavou je sestava selekce. Ta zobrazuje zvířata, která mají v daný den pořízenou změnu typu selekce. V nastavení je možnost definovat jednotku selekce. Pokud je jednotka definovaná, potom se uživateli zobrazují pouze selekce odpovídající dané jednotce. Uživatel dostane zvukové upozornění pří vstupu na kartu zvířete, které je v selekci a jeho jednotka odpovídá jednotce definované uživatelem. Toto zvukové upozornění uživatel taktéž může vypnout v nastavení.

2.3.8 Oprávnění

Uživatelé mají svá oprávnění. Mezi to základní oprávnění patří možnost používat mobilní aplikaci a možnost pořizovat změny. Pokud uživatel nemá dovoleno používat mobilní aplikaci, potom se mu přes mobil nepodaří přihlásit do daného podniku. Druhé základní oprávnění je možnost pořizovat změny. Když uživatel nemá možnost pořizovat změny, potom je aplikace pouze v módu pro čtení. Uživatel vidí všechna data, ale nemůže při synchronizaci odesílat jakékoliv změny. UI aplikace se v takovém případě přizpůsobí a skryje možnost zadávat nové změny, laktace a upravovat pozice stájí.

Uživateli může být nastaven výchozí technik. V takovém případě uživatel nemá právo přepínat mezi techniky a všechny změny inseminace jsou prováděny příslušným technikem bez možnost změny.

Dalším oprávněním je přístup do stájí. Uživateli je definován přístup do množiny stájí. Do této množiny má uživatel přístup a vidí všechna zvířata a změny v těchto stájích, respektive skupinách, které jsou na těchto stájích. Mezi těmito stájemi si taktéž může uživatel vybírat aktivní stáj. Stáje, do kterých nemá přístup, se uživateli v aplikaci vůbec nezobrazí a taktéž ani zvířata a změny na těchto stájích. Se stájemi přímo souvisí další oprávnění a to možnost výběru všech stájích. Pokud je uživateli toto právo odebráno, pak musí mít vybranou vždy jednu stáj a nikdy nemá možnost mít vybrané všechny jeho dostupné stáje pro filtrování.

2.3.9 Bluetooth

Zařízení by mělo být schopné komunikovat se čtečkami a váhami společnosti Tru-Test. Čtečky společnosti jsou schopné vyčíst EID zvířete z čipu na zvířeti. Toto EID by aplikace měla dostat pomocí BT komunikace a pokusit se vyhledat zvíře ve své databázi. Pokud zvíře nalezne, zobrazí jeho kartu.

Váhy odesílají proud dat o aktuálně naměřené váze. Mobil by měl být schopen zjistit z váhy aktuální naměřenou hmotnost a tu použít pro založení změny vážení, kde váha bude naměřená hmotnost na Tru-Test váze.

2.3.10 Rskot

Mobilní verze by měla být k dispozici ve speciální variantě pro Rskot. Tato verze by měla být na první pohled odlišná od verze pro Farmsoft. Verze Rskotu bude sloužit především pro inseminační techniky. V aplikace nepůjdou pořizovat všechny změny. Půjdou pořídit pouze změny typu poznámka, inseminace, březost, otelení a umístění. Bluetooth zařízení v Rskotu nemusí být podporovány. Taktéž Rskotu nemusí zobrazovat všechny sestavy. Zvláště sestavy týkající paznehtů v Rskotu nedávají smysl. Pořizování a zobrazování laktace (Ku) nebude podporováno.

Oproti Farmsoft verzi Rskot musí mít možnost pořídit změnu narození, a tudíž i vytvoření nového zvířete. Při zadávání změny se bude zadávat datum narození, pohlaví, číslo kusu, plemenná příslušnost, matka, otec a skupina pro nově narozené zvíře. Pokud při skenování QR kódu nedojde k nalezení zvířete, aplikace nabídne uživateli možnost založit toto zvíře a údaje předvyplní z dat získaných čtením QR kódu.

Kapitola 3

Průběh vytváření iOS aplikací

V této kapitole bych rád popsal, jak probíhá vytváření iOS aplikací. Řekneme si, jaké jsou podmínky pro vývoj iOS aplikací. Ukážeme si jazyky používané pro vývoj. Představíme si vývojové prostředí Xcode a jeho případné alternativy. Ukážeme si práci s knihovnama v iOS Xcode projektu. Povíme si něco o SwiftUI a UIkit. Projdeme si architektury, které se pro iOS vývoj používají. Nakonec si představíme AppStore a Apple Store connect, které se používají pro distribuci aplikace mezi koncové zákazníky.

3.1 Programovací jazyky

V této sekci bych rád představil programovací jazyky, které se používají pro iOS aplikaci. Na trhu momentálně existuje velké množství programovacích jazyků, které dokáží vytvořit iOS aplikaci a každý má své výhody a nevýhody. Zde si některé jazyky představíme a něco si o nich povíme. I když je možné vyvíjet aplikaci na jiných systémech a v jiných jazycích než od Applu, k samotnému sestavení aplikace je vždy nutný Xcode a jeho knihovny. [8]. Xcode lze spustit jenom na operačních systémech macOS od Applu. [9]. Tudíž pro sestavení iOS aplikace budeme potřebovat macOS. Můžeme použít i virtualizaci, ideální je avšak mít zařízení s tímto systémem.

3.1.1 Objective-C

Jedná se o starý jazyk, který byl představen v roce 1984. Tento jazyk byl zvolen jako primární programovací jazyk pro iOS a macOS. [10] Jazyk vychází z C a C++ a je s těmito jazyky kompatibilní. Takže je možné vložit část C a C++ kódu do kódu jazyka Objective-C. Jazyk byl používán dlouhá léta pro vývoj iOS aplikací a díky tomu má ohromnou komunitní základu a velké množství knihoven. Nyní je však tento jazyk nahrazen jazykem Swift přímo od společnosti Apple. [11]

3.1.2 Swift

Swift patří mezi nejpoužívanější jazyky pro vývoj softwaru běžící na systémech od společnosti Apple. Byl představen v roce 2014 společností Apple jako náhrada Objective-C. Jedná se o moderní programovací jazyk, který je velice rychlí. Jazyk navíc hlídá null hodnoty (ve Swiftu je null přejmenováno na nil) v době kompilace a implementuje automatickou správu paměti. Díky velkému množství návodů ať už od samotného Applu a nebo od komunity je jazyk celkem jednoduchý na naučení. Navíc je do jisté míry kompatibilní s Objective-C. [11]

3.1.3 Dart

Dart je programovací jazyk použitý jako jazyk pro Flutter framework. Flutter je framework od společnosti Google pro vývoj mobilních aplikací pro iOS a Android. Myšlenka je, že aplikace mají společný kód a až v době kompilace se vytvoří build pro iOS a pro Android. [12] Flutter pro vykreslení UI nepoužívá nativní komponenty platformy, ale grafického enginu, který tyto komponenty vykresluje za běhu aplikace. [13]. Díky společnému kódu si programátor ušetří hromadu práce, avšak je to za cenu zmenšené optimalizace, které u Flutteru nedosahuje rychlostí nativních aplikací. [14].

3.2 Xcode

Nyní bych Vás rád seznámil s vývojovým prostředím od Applu Xcode. Xcode, respektive knihovny, které nabízí, jsou nutné pro sestavení aplikace. [8] Xcode je momentálně dostupný pouze pro systémy macOS od společnosti Apple, která zároveň Xcode vyvíjí. Xcode umožňuje vytvářet aplikace pro všechny systémy od společnosti Apple, od mobilů přes hodiny až po chytré televize. Podporuje programovací jazyky Swift a Objective-C. [9]

Jak je u Applu typické, Xcode je uživatelsky velice přívětivý. Velké množství věcí lze naklikat přímo v IDE. Xcode navíc programátora celkem elegantně odstíní od samotného procesu sestavování. Tohle funguje tak dobře, že programátor ani nepotřebuje znát příkazy, které aplikaci sestavují a vše se děje tak nějak na pozadí a to včetně distribuce do AppStore. Samozřejmě v případě nutnosti umožní proces sestavení do velké míry upravovat.

Kromě diagnostických nástrojů jako debuger a zobrazení grafů o aktuální vytíženosti testovacího zařízení, nabízí i pokročilí editor databáze aplikace pomocí CoreData a to včetně editoru pro migrování. Pokud aplikaci píšeme v UIKitu, můžeme použít storyboardy k navržení prakticky celého UI. Storyboard je pokročilý editor, kdy prakticky každou obrazovku jsme schopní navrhnout jenom za pomoci myši a interakcí se samotným Xcode bez napsání čárky kódu. V kódu následně dodefinováváme pouze chování na akce UI, například kliknutí na tlačítko.

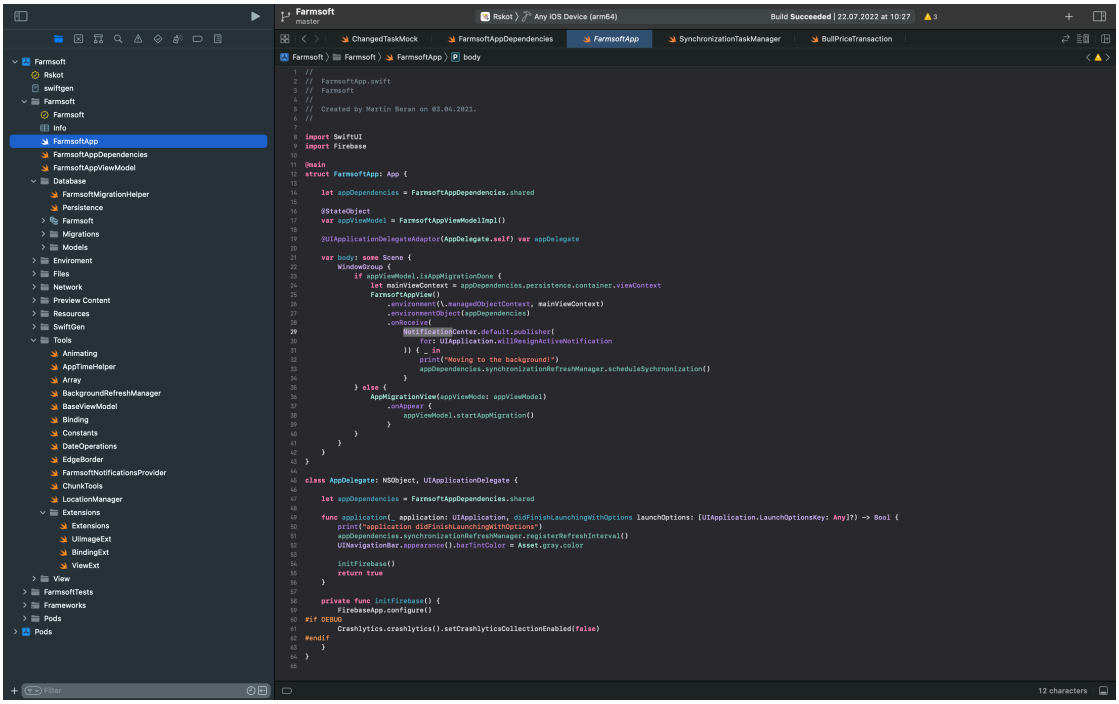
Nyní bych rád probral strukturu projektu. Xcode je jediný schopný sestavit iOS aplikaci. [8] Proto je nutné, aby strukturu projektu Xcode dodržovali i frameworky, které přímo nejsou od společnosti Apple jako již zmíněný Flutter. Mezi nejdůležitější soubory projektu patří soubory typu *.xcodeproj*. Tento soubor není ani tolik soubor jako složka s dalšími podsoubor. Podstatný je *project.pbxroj*.

Jedná se o textový soubor ve kterém jsou uloženy všechny důležité informace o projektu. Jsou zde cesty k souborům se zdrojovými kódy, cesty k frameworků atd. Toto je důležité, jelikož Xcode nepracuje se soubory projektu jako ostatní IDE. Struktura projektových souborů nemusí vůbec odpovídat struktuře souborů na filesystému počítače. Xcode si nad filesystémem systému staví vlastní filesystém. Díky tomu můžeme mít složky v projektu přeuspořádané jinak než v systému. Všechny tyto informace jsou uloženy právě v souboru *project.pbxroj*. [15]

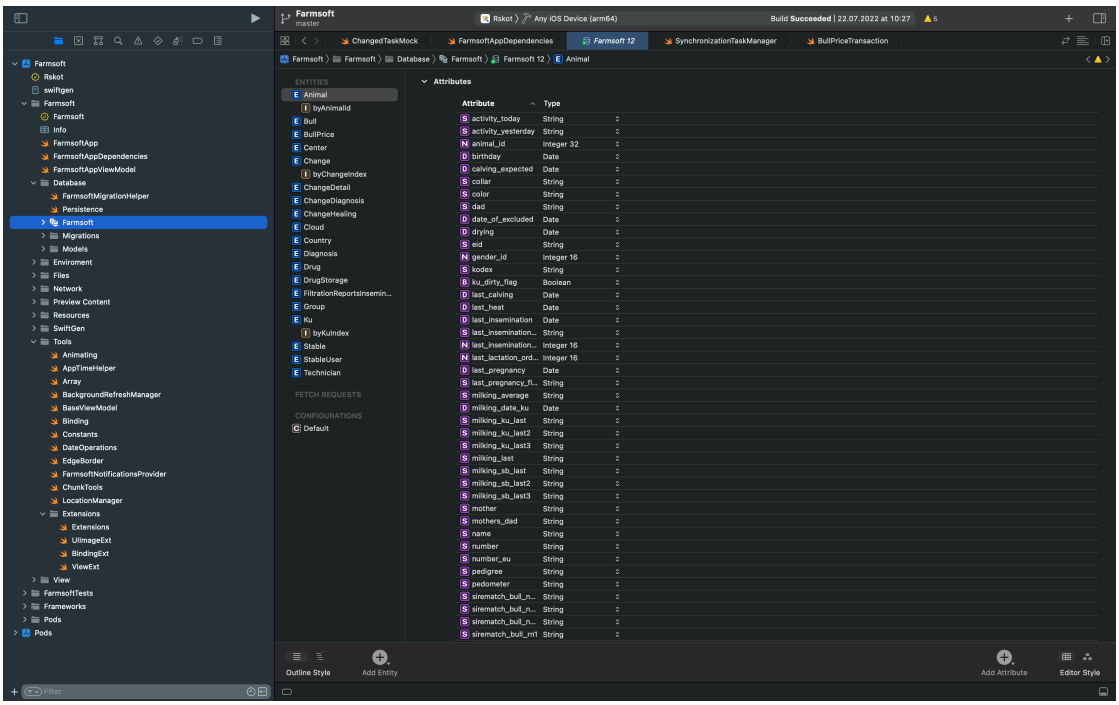
Kromě toho jsou zde uloženy další informace o projektu, jako jsou targety. Target v Xcode si lze představit jako modul aplikace. Můžeme mít jeden target pro iOS aplikaci a target pro watchOS aplikaci v jednom projektu. A pokud dodržíme kompatibilitu, můžeme použít ten samý kód pro watchOS i pro iOS. Target v sobě tedy uchovává informace o platformě, na kterou daný target cílí, a další důležité informace jako je název aplikace, verze atd.

Podobnou úlohu jako *.xcodeproj* má soubor typu *.xcworkspace*. Workspace nám umožní otevřít projekt s více podprojekty. To je výhodné, pokud chceme pracovat s knihovnami, které máme uložené v jiném projektu. [16]

Poslední důležitý soubor, který zde zmíním, je info.plist. Jedná se o xml soubor, který v sobě zahrnuje základní vlastnosti výsledné aplikace. Definuje se zde verze aplikace, id aplikace, práva vyžadována pro běh aplikace, orientace, načítací obrazovka atd. [17]



Obrázek 3.1 Ukázka Xcode s kódem aplikace



Obrázek 3.2 Ukázka editoru databáze v Xcode

3.2.1 Knihovny

V této části bych rád popsal práci s knihovnami v Xcode, konkrétně import knihoven do projektu. v Xcode existují dvě různé entity pro import cizího kódu. Jsou to knihovny a frameworky. Knihovna jsou standardní knihovny jako v ostatních jazycích. Obsahují spustitelný kód poskytující jistou funkcionalitu. Frameworky jsou složky, které taktéž obsahují spustitelný kód. Navíc však mohou obsahovat obrázky, soubory s lokalizací atd. Frameworky lze taky sdílet mezi jednotlivými iOS aplikacemi, a tudíž nedochází k zbytečnému plnění paměti. Frameworky tedy funguje jako lepší knihovna. Pro zjednodušení budeme knihovnu a framework považovat za ekvivalentní. [18]

V Xcode jsou 3 základní způsoby hlídání závislostí. Jsou to Cocoapods, Carthage a Swift Package Manager. Samozřejmě můžeme vzít kód přímo a vložit ho do projektu. Obecně je však lepší použít jakýkoliv management závislostí pro zpřehlednění projektu. Proto si nyní jednotlivé způsoby představíme.

3.2.1.1 Cocoapods

Cocoapods je velice oblíbený dependency manager pro Swift a Objective-C. Poskytuje více jak 91 tisíc knihoven a je používán ve více jak 3 milionech aplikací. Program je nejdříve nutné nainstalovat příkazem `sudo gem install cocoapods`. [19]

Po instalaci programu je nutné cocoapods inicializovat v projektu. Pro inicializaci použije příkaz `pod init` ve složce s projektem. Příkaz nám vytvoří Podfile a workspace. Podfile slouží pro samotné nastavení závislostí. Pro každý target si programátor může nastavit knihovny, které chce v daném targetu používat (takzvané pody). Po definici závislostí spustíme příkaz `pod install`, kterým příslušné knihovny stáhneme. [19].

Po stažení knihoven je nutné otevřít *.xcworkspace* soubor. Tím otevřeme příslušný workspace, ve kterém budou dva projekty. Projekt aplikace a projekt s názvem Pods. Zde jsou k dispozici stažené knihovny, které od této chvíle můžeme libovolně používat.

3.2.1.2 Carthage

Carthage je velice jednoduchý manager. Samotný Carthage je nutné nejdříve nainstalovat. Po jeho instalaci vytvoříme ve složce s projektem soubor s názvem Cartfile. Cartfile je textový soubor, který nám definuje potřebné závislosti. Závislosti se definují jako url adresy gitu. Následně spuštěním příkazu `carthage update --platform iOS` dojde ke stažení příslušných frameworků. Frameworky najdeme ve složce *Carthage/Build*. Frameworky je následně nutné přetáhnout do projektu pomocí drag and drop. [20]

3.2.1.3 Swift Package Manager

Swift Package Manager je dependency manager přímo od společnosti Apple. Je součástí přímo aplikace Xcode. Pro práci s ním není potřeba znát žádné příkazy a vše můžeme obsloužit pomocí uživatelského rozhraní Xcode. V Xcode existuje přímo wizard, který nám s vložením knihovny pomůže. [21]

Pro vložení knihovny nám stačí znát url adresy gitu, kde se daná knihovna nachází. Stačí vybrat verzi dané knihovny, popřípadě můžeme zvolit vždy verzi poslední. Počkáme, než se knihovna importuje a tím je vše hotovo. Podle mého názoru se tedy jedná o nejednoduší způsob práce s knihovnami. Avšak pokud se rozhodneme používat Swift Package Manager, potom se nedoporučuje jej kombinovat s ostatními dependency managery. Tyto kombinace by mohly vést k neočekávanému chování. [21]

Výpis kódu 3.1 Podfile projektu Farmsoft

```
# Uncomment the next line to define a global platform for your project
platform :ios, '14.0'
use_frameworks!

project 'Farmsoft.xcodeproj'

pod 'FloatingButton', '~> 0.0.3'
pod 'SwiftGen', '~> 6.4.0'
pod 'Alamofire', '~> 5.4'
# Stream Json parser
pod 'PMJSON', '~> 3.0'
pod 'ZIPFoundation', '~> 0.9'

# Verze 2.3.0 mi z nejakeho neznameho duvodu
# nesla archivovat a poslat na App Store
pod 'GoogleMLKit/BarcodeScanning', '2.2.0'

pod 'Firebase/Core', '8.6.0'
pod 'Firebase/Crashlytics', '8.6.0'


target 'Farmsoft' do
end

target 'Rskot' do
end

target 'Farmsoft-Testing' do
end

target 'FarmsoftTests' do
end
```

3.2.1.4 Alternativy pro Xcode

Nakonec bych rád ještě zmínil alternativní IDE pro Xcode. Konkrétně tedy pro nativní iOS projekty, tedy projekty psané ve Swiftu popřípadě Objective-C. Jediné IDE, které se mi podařilo najít je AppCode od JetBrains. Hlavní výhodou AppCode je právě jeho vazba na JetBrains. Pokud člověk pracuje s jakýmkoliv IDE od JetBrains, potom si na AppCode velice rychle zvykne. A ohledně práce s kódem jako formátování, napovídání a zvýrazňování bych osobně řekl, že funguje lépe než Xcody samotné. [22]

AppCode však výrazně pokulhává v ostatních aspektech, např. v práci se Storyboardy, kterou vůbec neumožňuje. Samotné nastavování projektu, jako platforma popřípadě úpravy Info.plist, taktéž nejsou příjemné na používání jako v Xcode. Navíc AppCode je placený a Xcode je zdarma. Proto pro mě jsou Xcode momentálně lepší IDE pro vývoj a rozhodl jsem se používat právě ty. [22]

3.3 SwiftUI vs UIKit

Nyní si představíme dva frameworky pro tvorbu UI v nativních iOS aplikací. Oba tyto frameworky slouží pro práci a návrh UI iOS aplikace. Xcode podporuje oba frameworky.

3.3.1 UIKit

UIKit byl vydán v roce 2008 společností Apple. Jedná se o framework pro návrh a práci s UI iOS aplikace. Je napsan v jazyce Objective-C. Díky kompatibilitě mezi Swift a Objective-C lze použít i ve Swiftu. [23]

Každá obrazovka v UIKitu je tvořena takzvaným UIViewControllerem. UIViewController je základní třída. UIViewController má vlastní cyklus, který vidíme na obrázku 3.3. Programátor přetíží funkce, které potřebuje pro svoji práci a do těchto přetížených funkcí vloží svojí implementaci. [23]

Samotné view se typicky definuje ve funkci viewDidLoad. V této funkci si programátor vytváří vizuál obrazovky za pomoci definování různých view a jejich přidáváním do controlleru. Toto však není jediná možnost. Druhou možnost je definice view pomocí storyboardů a grafického návrháře, který nám Xcode nabízí. V takovém případě si ve svém hlavním storyboardu nadefinujeme obrazovku, tu následně propojíme s UIViewControllerem, ve kterém doplníme hlavní programovou logiku.

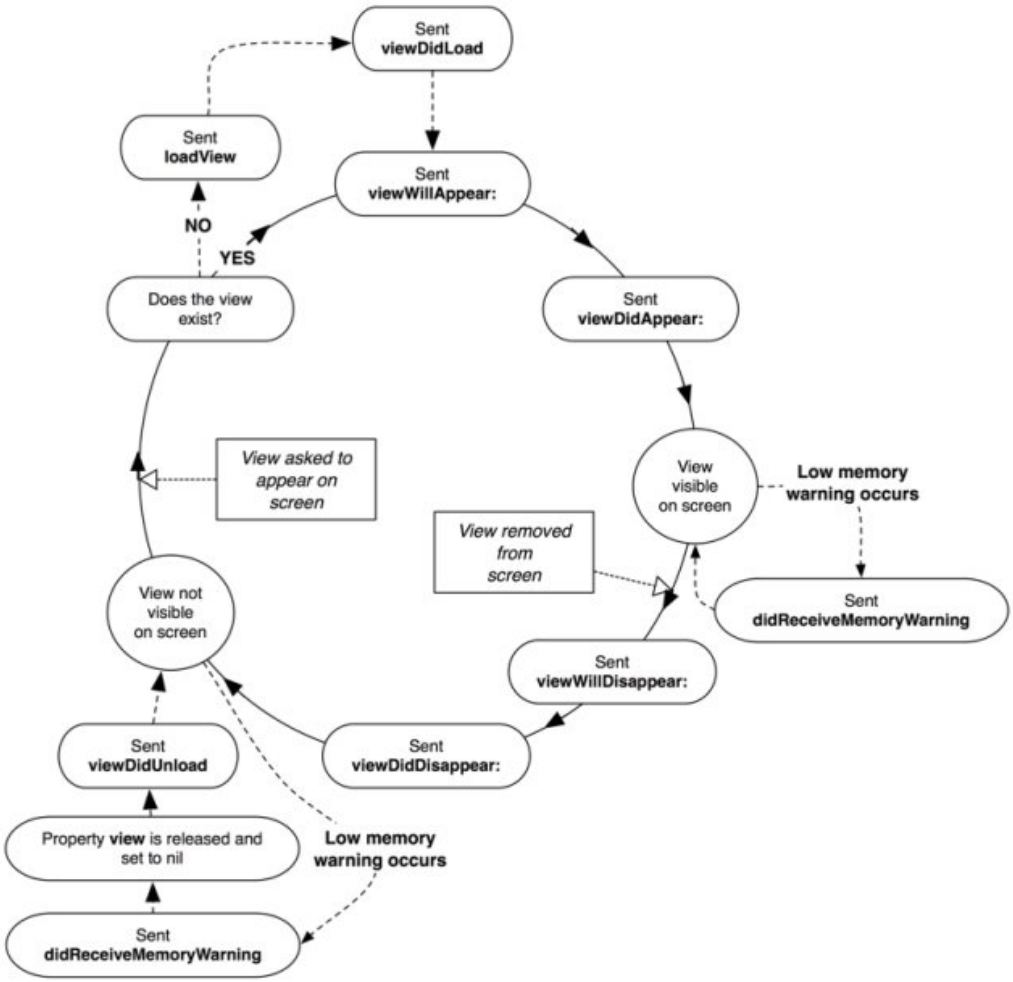
Pokud chceme složitější navigaci aplikací nebo tabulku, potom nám UIKit nabízí řešení v podobě UITableViewController a UINavigationController, kteří oba vychází z UIViewControlleru. Komunikace s nimi probíhá pomocí delegátů. Vlastně velké část UIKitu využívá právě tohoto návrhového vzoru. [24]

Programátor má tedy plnou kontrolu nad tím, co se na obrazovce děje. To se dá považovat za výhodu i nevýhodu zároveň. Za výhodu bych považoval možnost vytvořit opravdu cokoliv a framework nás nijak nelimituje. Na druhou stranu právě díky ohromné kontrole musí programátor hlídat velké množství věcí a vzniká prostor pro chyby.

Díky svému stáří nám UIKit nabízí ohromné množství obsahu a knihoven. Nemusíme tedy vše vytvářet znovu. Pokud se objeví nějaký problém, je velká šance, že daný problém už někdo řešil a najít řešení na internetu nebude náročné.

3.3.2 SwiftUI

SwiftUI byl poprvé představen v roce 2019 společností Apple. Podobně jako UIKit nám nabízí možnosti designu UI aplikace, však s naprosto odlišným konceptem. SwiftUI využívá stromové



Obrázek 3.3 Cyklus UIViewControlleru [25]

struktury jednotlivých view. Neexistuje žádný specifický objekt pro obrazovku, jako byl UIViewController u UIKitu. Ve SwiftUI je všechno View, které obsahuje další View. SwiftUI můžeme použit na iOS zařízení s iOS 14 nebo vyšší. [23]

SwiftUI je reaktivní, tudíž aktualizace obrazovky proběhne ihned po aktualizace příslušné proměnné a nemusí být manuálně vynucena. Programátor se tedy více může zaměřit na stylování obrazovky zatímco samotné fungování se děje na pozadí bez jeho přičinění. SwiftUI není určené jenom pro iOS. Dá se použít na všech platformách od společnosti Apple od macOS až po wathcOS. [23]

Je kompatibilní s UIKitem, kdy můžeme jednotlivá view napsaná v UIkit importovat do SwiftUI. A pokud vše provedeme správně, neztratíme tak ani schopnost reaktivní aktualizace jednotlivých částí view. [23]

Na rozdíl od UIKit zde nejsou žádné storyboard. Vše je psáno pouze v kódu, avšak Xcode nám nabízí náhled, jak naše view bude v mobilu vypadat. [26]

Ve SwiftUI je tedy všechno view. Programátor skládá tyto view dohromady a tím tvoří složitější struktury. Jednotlivá view se dají krásně znovu používat a to velice usnadňuje práci. Celkově se dá říct, že samotná tvorba bude ve SwiftUI probíhat rychleji. [23]

SwiftUI nám tím tedy velice ušetří práci. Bohužel tím však ztrácíme kontrolu a spoléháme

na framework, že vše bude fungovat bezchybně a to se občas neděje. Jelikož je SwiftUI celkem nové, tak zatím neexistuje takové množství knihoven a občas se člověk setká s vážně zvláštními chybami. [23]

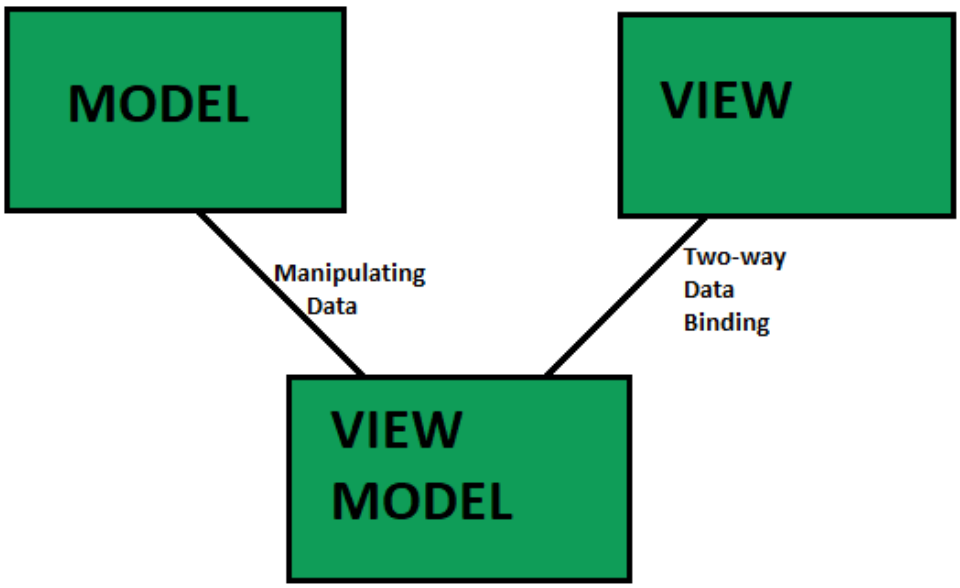
Avšak s SwiftUI se obecně dle mého názoru pracuje lépe a do budoucna se z něj stane hlavní framework pro návrh UI aplikace.

3.4 Architektura

Dobrá architektura je důležitou součástí dobrého programu. Volba správné architektury nám dokáže usnadnit spousty trápení. Aplikaci jsem se rozhodl napsat ve SwiftUI. Proto jsem hledal vhodnou architekturu, která by si se SwiftUI rozuměla. Bohužel Apple při vydání SwiftUI neuvedl žádnou doporučenou architekturu a na začátku psání iOS aplikace se mi ani nepodařilo najít architekturu doporučovanou komunitou. [27]

V době psaní práce se mi zamlouvala jenom jedna architektura a to Model view viewmodel (MVVM), kterou jsem již dobře znal z Android projektů. Bohužel až po napsání práce jsem našel The Composable Architecture (TCA). Trochu předběhnu, ale později jsem zjistil, že pro SwiftUI aplikace je alespoň z mého úhlu pohledu TCA lepší než MVVM. Nyní si však obě tyto architektury představíme.

3.4.1 Model view viewmodel



Obrázek 3.4 Schema MVVM architektury [28]

Model view viewmode (MVVM) architektura má tři základní komponenty. Jsou to Model, View a Viewmodel. Model reprezentuje byznys logiku a obsahuje veškerou práci s daty. Stará se o stahování, parsování, ukládání dat do databáze atd. [28]

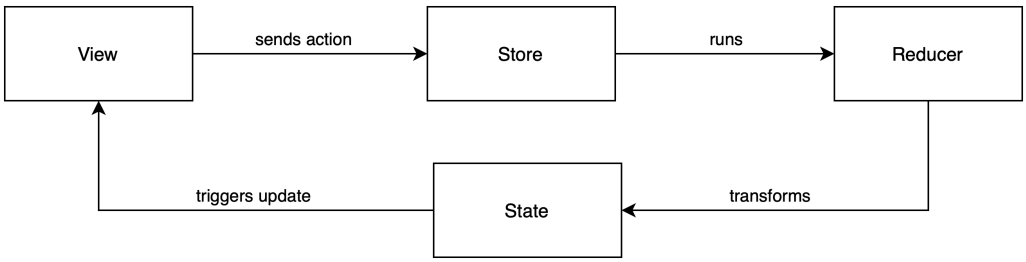
Viewmodel stojí mezi view a modelem. Jeho úkolem je získat data z modelu a ty následně připravit pro view. Na druhou stranu validuje vstupy z view a po jejích validaci tyto vstupy předává dále do modelu. [28]

Poslední komponenta je view. Ta se stará pouze o vykreslení a zobrazení dat. Není zde žádná validace vstupů ani datová úprava. View by tedy mělo získat data už připravená, aby je nemuselo dále upravovat. Aktualizace view obvykle probíhá za pomoci observer patternu. View observeruje viewmodel a při změně dat v viewmodelu je view automaticky aktualizováno. [28]

Komunikace mezi view a viewmodelem je tedy obousměrná. Viewmodel lze taktéž teoreticky použít pro různá view. To se však z mé dosavadní praxe téměř nedá zařídit a obvykle platí, že pro jedno view je jeden viewmodel. Komunikace mezi modelem a viewmodelem může být jenom jednosměrná. V takovém případě viewmodel inicializuje veškerou komunikaci, typicky stahování dat z modelu a jeho případné změny. V případě obousměrné komunikace je viewmodelu zaslána notifikace, pokud je model změněn například jiným viewmodelem. Kompletní schéma komunikace vidíme na obrázku 3.4. [28]

Podle mého názoru se jedná o velice jednoduchou a přímočarou architekturu vhodnou pro malé i velké projekty. Její nespornou výhodou je, že nezáleží na technologii. Architektura funguje stejně dobře pro iOS i Android, pro UIKit i SwiftUI.

3.4.2 The Composable Architecture



Obrázek 3.5 Schema komunikace TCA architektury [29]

The Composable Architecture (TCA) je architektura především pro SwiftUI. Nejedná se ani tolik o architekturu jako spíše o knihovnu, která nás při vývoji vede. Při dodržení jejích postupů dostaneme velice robustní kód, který je velice dobře testovatelný a navíc modulární. Architektura jako taková je však celkem složitá na pochopení a její naučení a zvláště pro nezkušené programátory je docela náročné. Knihovna je napsaná ve Swiftu, a tudíž je použitelná pouze pro zařízení od Applu. [29]

Já osobně mám s touto architekturou zkušenosti z jiného projektu, kde mě velice oslovila. Proto jsem se rozhodl jí zde popsat, i když jsem jí ve své práci nepoužil.

Má 6 komponentů. Jsou to View, Store, State, Action, Reducer a Enviroment. State, jak název napovídá, nám uchovává aktuální stav. Měly by zde být všechny položky, které view bude potřebovat pro své fungování. Action je typu enum. Jedná se o výčet všech dostupných událostí, které mohou nastat, jako například kliknutí na tlačítko, zavření formuláře atd. Enviroment se stará o propojení reducere se zbytkem aplikace jako napojení na serverové api, popřípadě databázi. Jedná se o jakousi mezivrstvu, jejíž hlavní výhoda spočívá v mockování dat při testování. Reducer se stará o logiku. Implementuje obsluhu všech událostí, které jsou definované v action. Pokud reducer obdrží akci, potom provede příslušný kód odpovídající dané akci a typicky změní state na základě dat získaných přes enviroment. View slouží pro interakci s uživatelem. View ví

pouze o store, přes který si může získat aktuální state. Přes store taky odesílá akce do reduceru. Store se stará o uchovávání aktuálního stavu a při změně tohoto stavu aktualizuje view. Taky se stará o posílání akcí reduceru vyvolaný view a nebo reducerem samotným. Komunikační cyklus vidíme na obrázku 3.5.

Architektura používá stromovou hierarchie. V kořenu je takzvaný AppState, AppReduce a AppAction. Hlavní view aplikace má tedy odkaz na AppState pomocí store. Vždy platí, že každé další view v hierarchi má vlastní state, reducer a action. Volání každé akce je tedy směřováno do kořenu a z kořenu zpět k požadovanému reducerovi. Všechny prvky výše v hierarchie dostávají zprávy a ví, co se děje v jejích následovnících. Mohou tedy podle toho upravit svůj state v případě potřeby. Dále každý reducer se stará o správné přeposlání akcí o úroveň níže, tak aby zbytečně zprávy nedocházely k reducerům, kterých se netýkají.

Tento postup je velice složitý a navíc poměrně pracný. Pro každé view musíme definovat tři další položky a navíc vždy musíme směřovat volání akcí. Všechno tohle však začne dávat smysl, když zjistíme jak konkrétní view psát, aby byl znovupoužitelný. Architektura nabízí ohromnou modularizaci, kdy view z jedné části aplikace můžeme krásně použít v jiné bez nutnosti upravovat logiku a jediné, co stačí, je přidat směřování akcí do příslušného nového reduceru.

Navíc pokud člověk správně pracuje s reducerem, je i celkem těžké udělat chybu. Velké množství kódu je zkontrolováno už při kompilaci a pokud ta projde, aplikace nejspíše bude fungovat a kdyby náhodou ne, k opravě typicky dojde velice lehce. Protože programátor typicky hned ví, kde je chyba. Chyby záludnějšího charakteru, se při použití této architektury téměř nevyskytují.

Architektura se tedy dle mého názoru hodí na náročnější projekty. Pro malé projekty je zbytečně složité a člověk zde musí napsat opravdu velké množství vcelku zbytečného kódu. Pro velké projekty však nabízí nesporné výhody, které převažují nad nevýhodami. Jak jsem již avizoval v úvodu, bohužel jsem při psaní této práce o této architektuře ještě nevěděl. Nyní bych zcela určitě použil právě tuto architekturu.

3.5 AppStore

AppStore slouží pro distribuci aplikací pro zařízení od společnosti Apple. Aplikace prochází přísnou kontrolou, a to jak automatizovanou kontrolou pomocí počítače, tak manuální lidmi. Pokud chceme dostat aplikaci mezi zákazníky, musíme aplikaci nahrát na AppStore. Aplikace tedy musí splňovat požadavky Applu, jinak bude zamítnuta. [30]

Aplikace se nahrávání pomocí App Store Connect. Pro přístup do App Store Connect je potřeba vývojářský účet. Vývojářský účet stojí 99 dolarů ročně. Po jeho pořízení dostane uživatel přístup do App Store Connect a je schopný nahrávat aplikace na AppStore. [31]

App Store Connect je přímo propojený s Xcode. Nahrání aplikace do App Store Connect pro následnou distribuci na AppStore lze udělat pomocí průvodce přímo v Xcode. Správa certifikátů nutných pro nahrání aplikace do App Store Connect je prováděna automaticky Xcody. Tuto možnost může programátor vypnout. V takovém případě je správa certifikátu jen a pouze na programátorovi.

Certifikáty jsou velice důležitou součástí procesu. Apple velice dbá na zabezpečení a to se odráží na nutnosti prakticky vše podepisovat certifikáty. Pokud uživatel má vývojářský účet, potom získává možnost tvořit tyto certifikáty. U mobilních zařízeních jsou tyto certifikáty nutné pro přístup k službám Applu. Tyto služby jsou například mapy, počasí či posílání push notifikací do zařízení. Tuto práci s certifikáty je nutné pochopit, pokud chceme aplikaci distribuovat automaticky mimo prostředí Xcode.

Dále bych se rád zmínil o TestFlight. TestFlight je součástí App Store Connect a slouží pro testování aplikací. Aplikaci můžeme sdílet mezi interní testery, anebo mezi externí testery. V případě externích testů je aplikace před testováním opět zkontrolována pracovníkem Applu, aby se potvrdila její nezávadnost.

Pokud chce uživatel testovat, musí si stáhnout aplikaci TestFlight do svého zařízení. Poté

musí být pozván do programu. V pozvánce je přiložen kód, který zadá do TestFlight a TestFlight mu nabídne aplikaci ke stažení. Postup je vcelku jednoduchý a zvládne ho i nezkušený uživatel.

Kapitola 4
Vývoj

Nyní bych rád představil jak probíhal vývoj aplikace. Prvně popíšu, které funkce jsem se rozhodl implementovat a proč. Potom již přejdeme k tomu, jak probíhal vývoj. Vývoj aplikace byl trochu složitější. První pokus o iOS aplikaci začal v létě roku 2019. Ten byl však později zahozen a na zelené louce vznikla nová aplikace. První pokus si představíme jenom lehce a shrneme si, které funkce zhruba obsahoval. Druhý pokus si představíme kompletně. Představíme si použité postupy a knihovny. Aplikaci si celou podrobně projdeme a popíšeme si jednotlivé obrazovky a jejich funkce.

4.1 Návrh

V sekci 2.3 jsme si představili, co by mobilní aplikace Farmsoft měla obsahovat. V iOS aplikaci jsem se rozhodl z časových důvodů implementovat jenom části důležité pro fungování aplikace. Jedná se o přihlášení, synchronizace, změny, laktace, zobrazení zvířat a jejich filtrace, některé sestavy a oprávnění. Aplikace by měla podporovat Farmsoft i Rskot verzi programu.

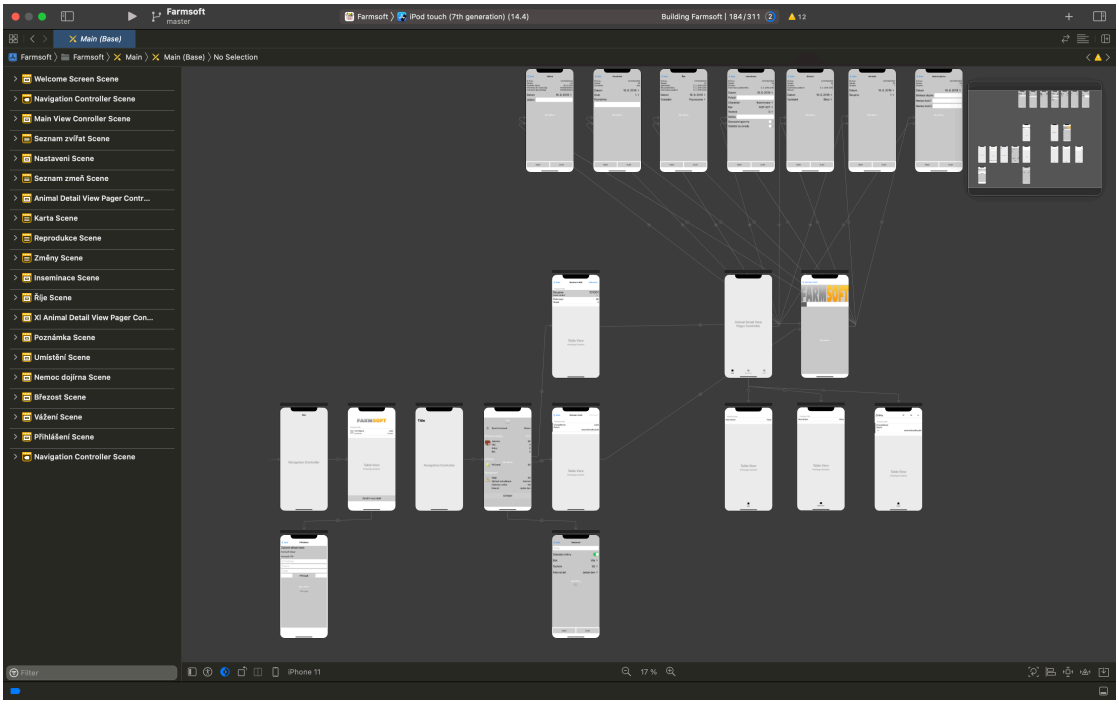
Přihlášení by mělo být dostupné jak pro Cloud, tak pro FTP. Synchronizace by měla být vyřešená v plné míře, včetně práce s sn. V rámci této práce bude strana serveru mockována. Pro ostrý provoz však bude aplikace již na server plně napojena. Práce se zvířaty, laktací a změnami bude implementována v plné míře podle původního návrhu. Všechny tyto prvky jsou nutné pro základní fungování aplikace. Systém oprávnění a sestava inseminací by taktéž měly být v aplikaci implementovány.

4.2 Průběh vývoje

Nyní si povíme, jak vlastně samotný vývoj probíhal. Rád bych zde shrnul 4 roky vývoje, kterým si aplikace prošla. Jak jsem již avizoval, aplikace byla vyvíjena nadvakrát.

První pokus o vytvoření iOS aplikace Farmsoft začal v červnu roku 2019. První verze bylo implementována za použití UIKitu, zvláště za pomocí storyboardu. Navržené obrazovky v storyboardu můžeme vidět na obrázku 4.1. UIkit byl použit, jelikož SwiftUI v této době bylo teprve "v plenkách". Pro verzování kódu jsem používal firemní Gitlab.

Tato aplikace se uměla přihlásit, zobrazit zvířata ve stájí a pořídit některé změny. Synchronizace nebyla implementována a všechny data byly mockovány. Pořizování změn nebylo implementované v plné míře a některé položky chyběly. Některé změny jako paznehty nebyly implementovány vůbec. Sestavy ani oprávnění taktéž implementovány nebyly. Systém oprávnění v této době ještě vůbec nebyl definován, stejně tak jako myšlenka implementace aplikace i pro Rskot.



Obrázek 4.1 Hlavní storyboard prvního pokusu o aplikaci

Pro správu knihoven jsem použil Cocoapods. Tento nástroj se mi velice osvědčil a rozhodl jsem se ho použit i ve finální aplikaci. Věc, se kterou jsem při implementaci této verze nebyl spokojen, byly použité knihovny, zvláště databáze. Použitá knihovna s názvem SQLite.swift nebyla špatná a svůj účel plnila dobře, ale pro finální aplikaci jsem se rozhodl použít CoreData především z důvodů jejich integrace v Xcode. Jako architekturu jsem použil standardní MVVM.

Poslední commit do této verze byl přidán 31. října 2019. Vývoj byl ukončen převážně z časových důvodů, kdy jsem nestíhal psát zároveň Android a iOS aplikaci společně se školou. IOS aplikace z těchto věcí měla nejnižší prioritu, a proto byl projekt pozastaven.

Obnova projektu přišla 3. dubna 2021. Bylo rozhodnuto aplikaci napsat od základů s využitím SwiftUI a dalších nových technologií. Toto rozhodnutí bylo alespoň z mého pohledu správně, jelikož jsme se tím vyhnuli technickému dluhu. Navíc já byl o dva roky zkušenější a věděl jsem, že novou verzi dokážu napsat lépe.

Prvně vznikla verze bez napojení na server, která je součástí této práce. Později se k tomu připojila verze již s napojením. První verze v1.3 byla vydána na AppStore 13. srpna 2021. Verze bez napojení od té doby existují paralelně. Aktuální verze je v1.43. Nutno však podotknou, že ne každá verze byla nahrána na AppStore. Některé byly určeny pouze pro testování.

Verzování probíhá ve formátu v1.xx. Při sestavení nové verze byl vždy přidán commit, který měl ve zprávě Build XXX, kde XXX je název verze. Zároveň byl v Gitlabu vytvořen tag s názvem verze. Do budoucna však toto verzování plánuji sjednotit s verzování používaným na Androidu, které je ve formátu yy.mm.xxx, kde yy je poslední dvojčíslí roku sestavení, mm je měsíc a xxx je verze sestavení.

Vývoj vždy probíhal v iteracích, kdy jsem postupně přidával nové funkce. Tyto funkce následně prošel zákazník. Ten zhodnotil, zdali vyhovují a případné výhrady byly opraveny. Postup testování si probereme detailněji dále v textu. Nyní si však představíme samotnou aplikaci a technologie, které používá.

Výpis kódu 4.1 Ukázka použití dependency injection

```
typealias Dependencies =
HasCloudRepository &
HasBullBackgroundRepository &
HasBullRegistrationIdFormatter

init(_ dependencies: Dependencies) {
    bullRepository = dependencies.bullBackgroundRepository
    bullRegistrationIdFormatter = dependencies.bullRegistrationIdFormatter
}
```

4.3 Použité technologie a knihovny

Teď bych rád představil technologie a knihovny, které jsem využil při programování aplikace. Povíme si důvod použití daných knihoven. Pro správu knihoven jsem použil Cocoapods. S tímto nástrojem jsem byl velice spokojen a za celou dobu vývoje jsem nenarazil na žádný závažnější problém způsobený tímto nástrojem. Později si jednotlivé použité knihovny představíme. Začneme však architekturou.

4.3.1 Architektura

Architekturou jsem zvolil MVVM. Hlavní důvod zvolení bylo, že jsem s ní měl zkušenosti již z jiných projektů. Taky jsem jí použil při psaní prvního pokusu o aplikaci, kde jsem s ní neměl žádný problém.

Při implementaci jsem taktéž pamatoval na testování. Všechny třídy tedy komunikují přes interface (ve Swiftu protocol). Tím se sníží provázanost a je lehčí psát automatické testování. Druhou podmínkou pro možné testování je takzvané dependency injection. Jedná se o postup, kdy všechny závislosti třídy jsou předloženy při její inicializaci a třída si je tedy nevytváří sama. Díky tomu potom při testování můžeme třídě podstrčit mockované závislosti. I když tuto problematiku umí řešit frameworky, rozhodl jsem se jí implementovat manuálně. Bylo tomu tak po doporučeních, které jsem dostal na kurzech programování pro iOS, které jsme měli ve škole. Postup implementace je taktéž inspirovaný ukázkou dependency injection, kterou jsme měli v tomto předmětu.

Každá třída má tedy v konstruktoru jeden vstupní argument typu Dependencies. Dependencies je alias, který je tvořen protokoly obsahující potřebné závislosti. Konkrétní použití vidíme v ukázce kódu 4.1. Tím získáme schopnost vkládat do tříd závislosti dle libosti. Je výhodné si tyto závislosti udržovat na jednom místě. K tomu slouží třída *FarmsfostAppDependencies*. Tato třída má v sobě všechny závislosti napříč aplikací. Pro inicializaci stačí předat v konstruktoru instanci *FarmsfostAppDependencies*. Toto nám dává kontrolu nad jednotlivými komponenty aplikaci. Taktéž to zařídí modularitu, kdy stačí ve třídě *FarmsfostAppDependencies* změnit implementaci jisté závislosti a tato změna se automaticky propíše do celé aplikace.

4.3.2 SwiftGen

SwiftGen je knihovna pro automatickou správu zdrojů aplikace jako jsou obrázky, barvy a texty. Pro získání zdrojů aplikace v kódu se využívá název těchto zdrojů. Název je textový řetězec. To je však problém, jelikož programátor se při psaní názvu může překlepnou a nedojde k načtení daného zdroje, například obrázku. Co je však horší, že jediný způsob, jak si toho všimnout, je až ve výsledné aplikaci. Právě tomuto problému se SwiftGen snaží předcházet.

SwiftGen při každé kompilaci vygeneruje třídy, přes které lze ke zdrojům přistupovat. Název třídy je složen ze dvou částí, jsou to typ a samotné jméno zdroje, pro příklad *Image.Logo*. Výhodou je, že chyba se odhalí podstatně dříve a to už při samotné kompilaci. Jelikož jsou třídy ve swiftu, tak název třídy nám dokáží napovědět samotné Xcody.

Tyto vlastnosti jsou užitečné hlavně v případech, kdy aplikaci chceme mít vícejazyčnou. Všechny texty jsou potom uloženy v *Localizable.strings* souborech. SwiftGen nám umí generovat třídy pro jednotlivé texty v těchto souborech. Tyto třídy se postarají i o získání správného textu podle považovaného jazyka. SwiftGen bohužel neumí poznat, pokud klíče textů v jednom lokalizačním souboru neodpovídají ostatním. To se stává, pokud například zapomeneme implementovat překlad nějakého slova v jednom jazyce, zatímco v ostatních ho implementujeme. Toto je však jediná věc, kterou je při práci potřeba hlídat. Ostatní problémy při práci se zdroji za nás SwiftGen vyřeší.

4.3.3 Alamofire

Alamofire je knihovna, která slouží pro HTTP dotazy. Pro tuto práci je tato knihovna nepotřebná, ale pro finální produkt bude nutná. Dotazy na api probíhají pomoci HTTP dotazů. Můžeme sice použít standardní URLSession, která je součásti standardní knihovny od Applu. Alamofire nám však http volání umí zabalit do hezkého api a práce s ním je příjemnější. Alamofire nám umí zařídit všechny potřebné funkce, které při komunikace se serverem programátor potřebuje, jako např. stahování, nahrávání, ověřování odpovědí, odchytávání výjimek atd.

4.3.4 PMJSON

PMJSON je knihovna pro parsování a zpracování JSON souborů. Hlavní důvod výběru této knihovny je podpora parsování JSONu postupně. Většina parsovacích knihoven nahraje celý JSON do paměti a ten poté převede na objekty. Toto řešení jsem aplikovat nemohl. Aplikace musí fungovat offline a při synchronizaci je staženo opravdu velké množství dat, které je třeba zpracovat. Při tvorbě android aplikace při synchronizaci docházela paměť a aplikace padala, pokud byl celý soubor nahrán do paměti. Tomuto se předešlo právě postupný zpracováním JSONu. Při iOS implementaci jsem očekával podobný problém a použitím této knihovny jsem se mu snažil předejít.

4.3.5 SwiftUI

SwiftUI jsme si již popsali v textu výše. Pro připomenu, jedná se o framework pro desing a návrh UI aplikace. SwiftUI se mi zdála jako dobrá volba hlavně díky rychlosti, se kterou programátor dokáže tvořit jednotlivé obrazovky. Taktéž SwiftUI velice usnadní práci. Programátor totiž nemusí tolik hlídat správně přepisování obrazovky a může se soustředit pouze na samotnou tvorbu UI. Správné přepisování obrazovky na základě dat za něj SwiftUI vyřeší samo.

4.3.6 CoreData

CoreData je knihovna od Applu pro podporu a obsluhu databáze. Neposkytuje pouze databázi, ale nabízí nám i mapování dat v databázi na instance objektů. Knihovna podporuje i jistou formu cachování a díky tomu je velice rychlá.

Jelikož je od společnosti Apple je zde velice dobrá provázanost s Xcody. Existuje zde vizuální editor. Přes tento editor si můžeme datový model databáze pouze naklikat. Po definování datového modelu jsou pro něj vytvořeny odpovídající struktury. Tyto struktury si můžeme rozšířit o libovolné funkce pomocí takzvaných *extension functions*. Mapování umí zachytit i vztahy mezi

entitami. To dokáže velice usnadnit práci, jelikož není nutné tvořit složité sql dotazy a nebo sekvence dotazů.

Samotné dotazy jsou tvořeny pomocí NSFetchRequest. Nastavením atributů této třídy sestavujeme požadovaný dotaz, který je potom poslán do databáze. Dotazy lze i řetězit a tím tvořit složitější dotazy s podporou AND a OR operací. Díky tomu můžeme sestavit prakticky jakýkoliv dotaz, jako bychom byli schopní v SQL. Vnořené dotazování není dovoleno. CoreData umí provádění dotazů spouštět na různých vláknech. V případě jednoduchých dotazů je normální pouštět dotazy přímo na hlavním vláknu. [32] Uživatel nemusí dlouho čekat na odpověď a jelikož jsou CoraData rychlá, tak nedojde ani k zamrznutí UI. V případě potřeby je však možné spustit dotaz na pozadí, aby nedošlo k zaseknutí obrazovky.

CoreData taktéž podporují migrace. Migrace jsou v tomto projektu velice důležité, jelikož aplikace musí fungovat i v režimu offline. Může se stát, že uživatel má lokálně pořízené nějaké změny, o které nesmí při migraci přijít. CoreData jsou ve většině případech schopná migraci zařídit automaticky bez zásahu programátora. Umožňují i takzvané těžké migrace, kdy si proces migrace může definovat sám programátor. To probíhám pomocí programováním ve Swiftu.

4.3.7 Firebase

Firebase je soUbur knihoven od Googlu. Obsahuje spoustu věcÍ jako push notifikací, analytické nástroje, databázi, popřípadě vzdálenou konfiguraci aplikace. Já jsem z toho balíku využil pouze Firebase-Crashlytics, které se používají pro sbírání a odesílání logů při pádu aplikace. Logy jsou odesílány na firebase server. Zde si následně může programátor procházet pády aplikace, které se uživatelům dějí a ty opravit. Tuto technologii jsem zvolil z důvodu, že stejné trasování pádů používám i na Androidu a jsem na tuto platformu již zvyklý.

4.3.8 GoogleMLKit - BarcodeScanning

Tuto knihovnu používám pro skenování QR a čárových kódů při vyhledávání zvířat v systému. S knihovnou jsem opět měl zkušenosti již z Androidu, kde fungovala perfektně. Knihovna podporuje detekci různých formátů čárových kódů. Pro Farmsoft je důležitý formát QR a Code128, protože právě v těchto formátech jsou čárové kódy na kartách zvířat. Oba tyto formáty tato knihovna podporuje.

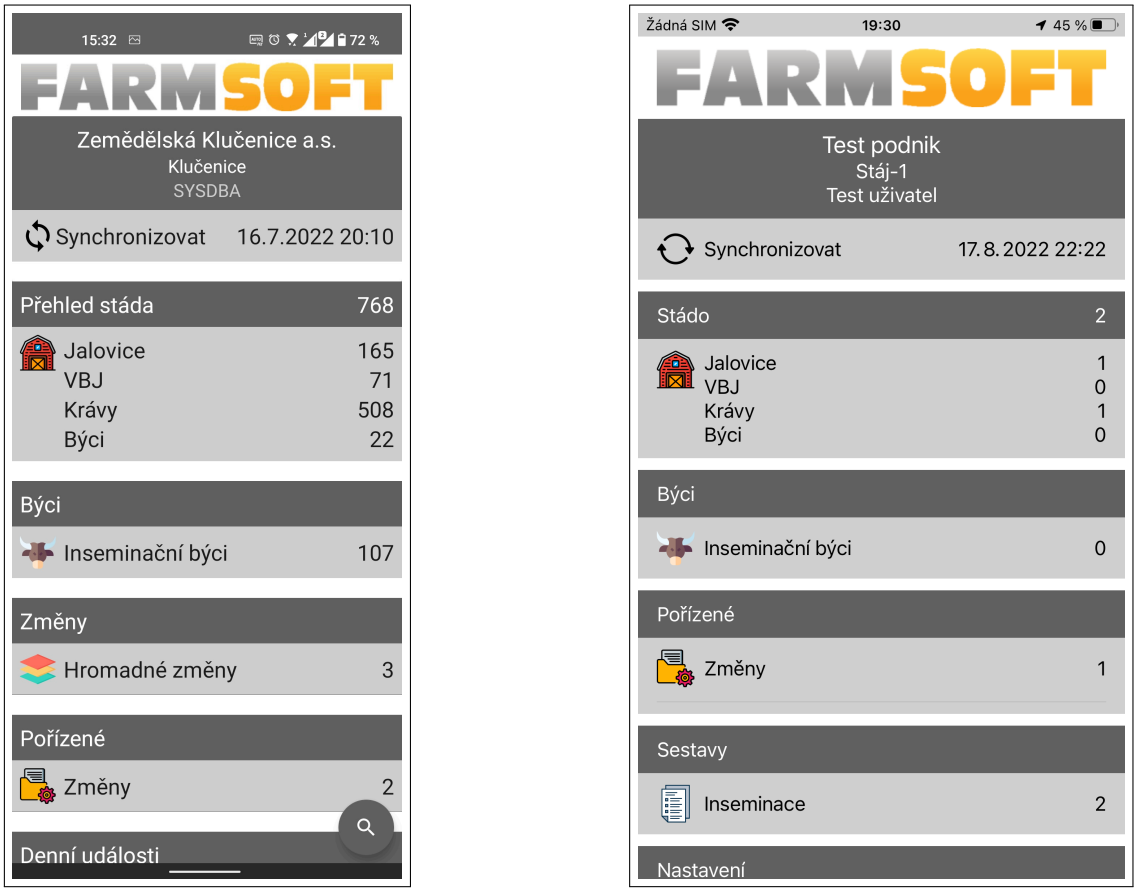
Čtení knihovny je velice rychlé. Kód není potřeba vůbec zarovnávat doprostřed obrazovky. Stačí, aby se kód pouze zobrazil na obrazovce v čitelné podobě. Dokonce nezáleží ani na natočení kódu. Knihovna funguji i offline, což byl další důležitý požadavek.

Jako alternativu šlo použít AVMetadataObject. Jedná se o třídu standardní knihovny přímo od Applu. Ta funguje podobně dobře jako GoogleMLKit - BarcodeScanning. Je však designovaná pro jednoduché použití bez možnosti načítat kódy třeba z již pořízených obrázků či naskenovat několik kódů naráz. Všechny tyto vlastnosti GoogleMLKit - BarcodeScanning obsahuje. Jediná nevýhoda GoogleMLKit - BarcodeScanning je složitější použití této knihovny.

4.4 Popis aplikace

Aplikace je vytvořená na operační systém iOS. Aplikace podporuje mobilní zařízení s operačním systémem iOS 14 nebo novější. Je dostupná pro iPhony i iPady. V AppStoreConnect byl vytvořen vývojářský účet společnosti Farmtec a.s. Do toho účtu mám přístup. Účet jsem použil při distribuci aplikaci a generování podpisových klíčů aplikace.

Design aplikace vychází primárně z aktuální podoby android verze aplikace Farmsoft. Chtěl jsem, aby si výsledné aplikace byly co nejvíce podobné a uživatel na první pohled poznal, že se jedná o stejnou aplikaci na jiné platformě. Taktéž jsem však chtěl ponechat specifické ovládací prvky pro danou platformu, na které jsou uživatelé dané platformy zvyklí. Mám na mysli



Obrázek 4.2 Porovnání hlavní obrazovky Android (vlevo) iOS (vpravo)

konkrétně třeba vyhledávání, které na každé platformě vypadá specificky pro danou platformu. Design je z dnešního pohledu již zastaralý, ale naši uživatelé jsou na něj zvyklí a bylo by velice náročné jej předělat pro obě platformy.

Drobné nesrovnalosti jsou v ikonách v aplikaci, kdy zvláště při použití systémových ikonek jsem byl nucen na každé platformě použit jiné ikony. Toto se však týká nevýrazných ikon, které celkově neničí celkový desing. Systémové ikony jsou ikony dostupné přímo na konkrétní platformě, které nám nabízí samotné IDE. Tyto ikony jsem musel použít, protože v našem týmu není žádný grafik, který by nám navrhl společné ikony pro všechny platformy v rozumné kvalitě.

Výjimkou jsou ikonky změn. Ikonky změn jsem nechával udělat na zakázku od externího grafika přímo pro tuto aplikaci. Obrazovky jednotlivých platforem jsou až na drobné výjimky velice podobné. Pro příklad porovnání hlavních obrazovek vidíme na obrázku 4.2.

Aplikace podporuje lokalizaci. Aktuálně je aplikace lokalizovaná do dvou jazyků. Jsou to čeština a angličtina. Do budoucna se plánuje přidání i ostatních jazyků, které Android momentálně podporuje jako je polština a ruština. Lokalizace se určuje podle systémového nastavení mobilu. Do aplikace jsou při sestavení vloženy slovníky. Jsou to textové soubory s názvem *Localizable.strings*. Tyto soubory jsou ve složkách a podle názvu složky systém pozná, pro který jazyk je daný slovník. Například český slovní bude ve složce cs, anglický ve složce en atd. Pro práci s texty v kódu jsem použil knihovnu SwiftGen, která sama zařídí zvolení správného textu ze slovníku podle systémového nastavení. Pro budoucí přidání jazyka stačí pouze přidat novou složku s *Localizable.strings* souborem, který v sobě bude obsahovat texty nového jazyka.

Oproti Android verzi iOS verze podporuje tmavý mód. Pro tmavý mód jsem využil Xcode a

SwiftGenu. Xcode nabízí definovat barvu pro světlí a tmavý mód. K těmto barvám ve zdrojích jsem poté přistupoval pomocí reference vygenerované SwiftGenem. Tmavý a světlý mód se volí podle aktuálního nastavení v systému.

Implementace tmavého módu tímto způsobem vyžaduje striktní práci s barvami. Není dobrý nápad definovat barvy červená, zelená atd. Každá barva vždy musí být definovaná za nějakým účelem, například pozadí, barva textu atd. Já jsem pro implementaci použil 4 barvené definice.

AccentColor a SubAccentColor jsou barvy kontrastní s pozadí pro zvýraznění prvků a odlišení je od pozadí. Sub varianta je trochu méně výrazná. Používá se pro barvu neaktivních prvků atd. Druhou půlku tvoří gray a gray_dark. Tyto barvy slouží pro pozadí prvků, například pozadí tlačítka či dlaždic na hlavní obrazovce. Toto pojmenování může být trochu matoucí. Je převzato z Androidu, kde tyto barvy reprezentovaly pouze šedivou. Barvu pozadí celé aplikace nenastavuji explicitně, ale nechávám výchozí barvu nastavenou SwiftUI.

Aplikace je k dispozici ve dvou variantách. Jedná se o Farmsoft a Rskot. Pro implementaci různých variant jsem využil targetů. V projektu aktuálně existují tři. Jsou to Farmsoft, Rskot a Farmsoft-testing. Farmsfot vytvoří aplikaci pro Farmsoft. Target Rskot vytvoří aplikaci pro Rskot. Farmsoft-testing je target určený pro testování. Obsahuje všechny funkce Farmsoftu i Rskotu.

Targety využívají společný kód. Pokud si přeji nějakou funkci pro specifický target, využívám preprocesoru, který Xcody nabízí. V kódu stačí napsat `#if RSKOT` a daný kus je kompilátorem zpracován pouze při kompilaci Rskotu. Pro ostatní varianty je zápis obdobný.

Rskot je ořezaná verze Farmsoftu. Veškeré budoucí popisy budu provádět právě na Farmsoft verzi. Nyní bych však uvedl rozdíly mezi verzemi. Hlavní rozdíl, kterého si uživatelé všimnou, je logo na hlavní obrazovce. Pro Farmsoft je zde napsáno FARMSOFT. Pro Rskot je nápis v logu RSKOT. Rskot umožňuje přihlášení pouze přes Cloud. Přihlášení přes FTP v Rskotu není podporováno. V Rskotu jdou pořizovat pouze některé změny. Jsou to poznámky, říje, inseminace, březost, otelení a přesun. Ostatní změny nelze ve variantě pro Rskot pořizovat. Rskot navíc umožňuje pořizovat změnu narození a tím vytvářet nová zvířata. V Rsktu nejdou pořizovat ani zobrazovat laktační záznamy. Tyto záznamy jsou pro uživatele mobilní aplikace Rskotu nepodstatné a jenom by zdržovaly synchronizaci.

4.4.1 Synchronizace

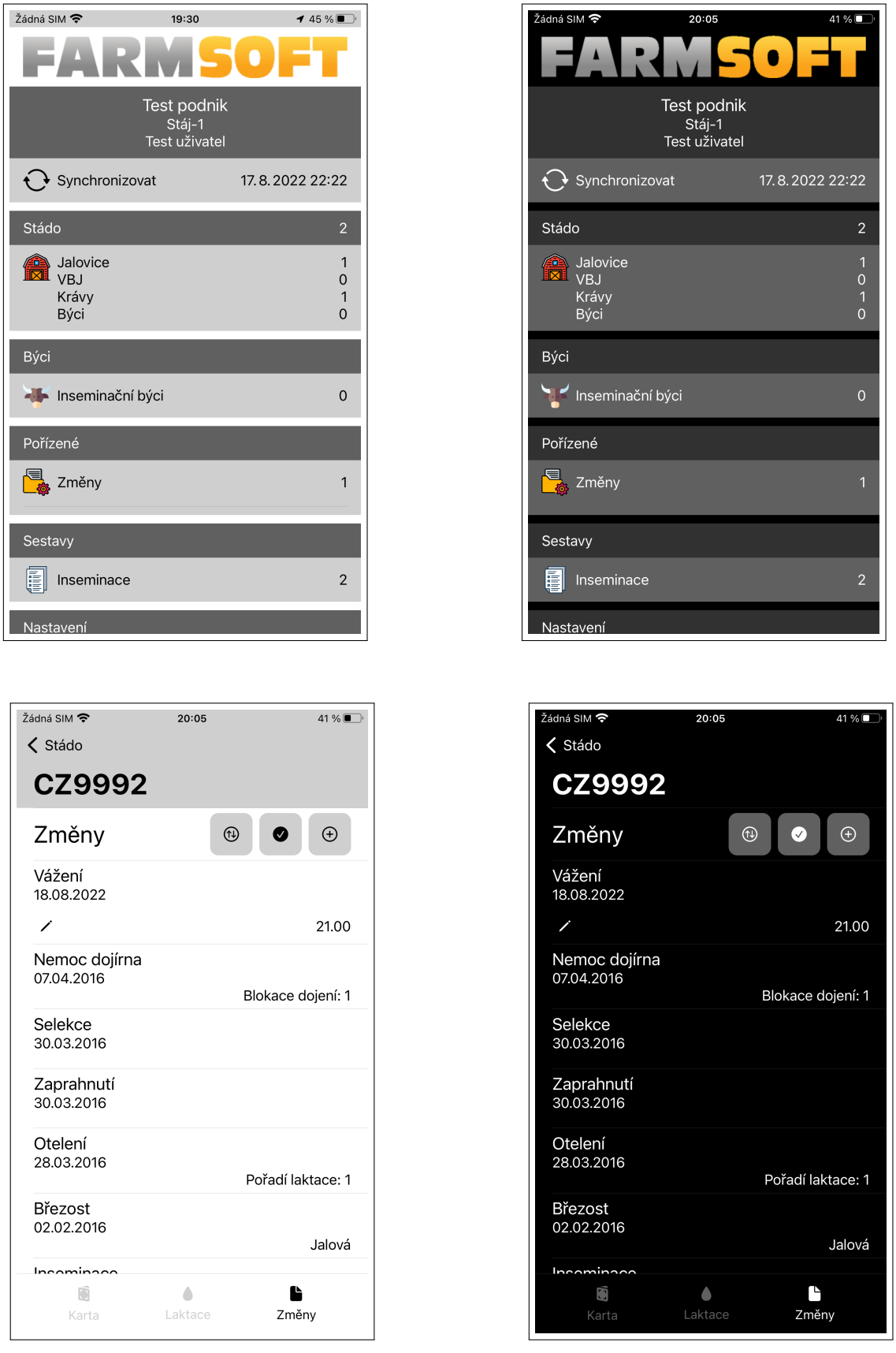
Komunikace se serverem je jednou z nejdůležitějších části aplikace. I když aplikace nemusí přímo komunikovat se serverem a komunikace může být simulována, snažil jsem se o nejlepší návrh této části, aby při ostrém napojení jí nebylo potřeba předělávat. K tomu jsem využil své zkušenosti z Android aplikace a proces jsem okopíroval téměř 1:1 až na finální napojení na server.

Jelikož aplikace musí fungovat i v offline módu a uživatel pro práci potřebuje všechny data, dochází při synchronizaci se serverem k opravdu velkým přenosům dat. Mobil si stáhne téměř celou databázi podniku a to může být řádově až miliony záznamů. Kvůli velkému množství dat je zde potřeba i správná práce s pamětí. Není správné řešení nahrát vše do paměti a doufat, že nám paměť nedojde.

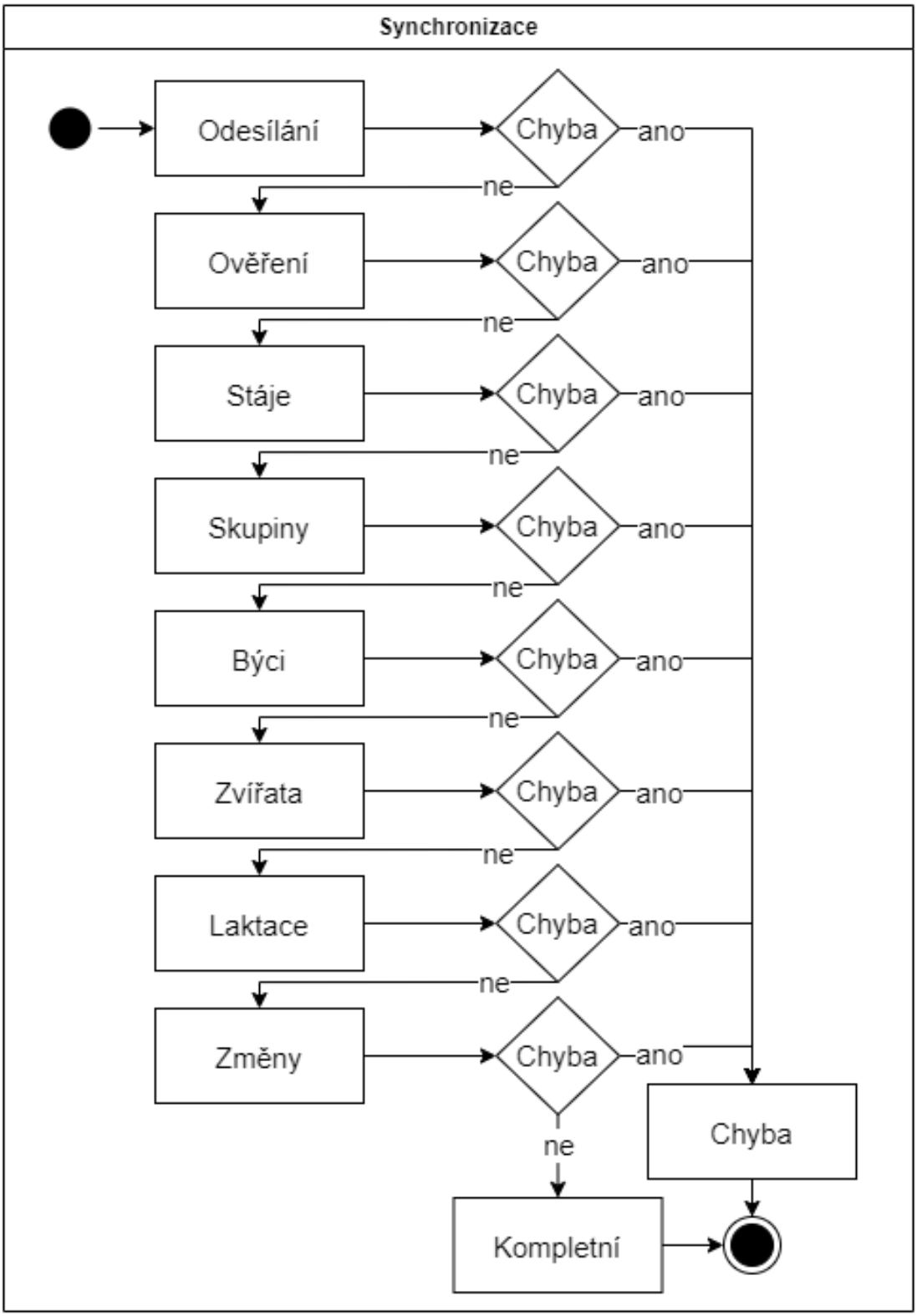
Proto jsem proces synchronizace rozdělil na dvě částí a to stahování a zpracování. Při stahování dojde k stažení souboru z API Farmsoftu. Pří stahování záleží, jestli aktuální podnik je umístěn v Cloudu a nebo v FTP. Pří Cloudu dojde k dotazu na API s posledním dostupným SN a API nám vygeneruje soubor s novými záznamy. Při FTP je vždy vygenerován kompletní obraz databáze. To je dáno z podstaty fungování FTP serverové části.

Vygenerovaný soubor je ve formátu JSON. Ten však nelze nahrát do paměti celý, a proto musí dojít k jeho postupnému zpracování. Pří zpracování je třeba vždy kontrolovat, zdali záznam již není k dispozici v aktuálním podniku. Tato kontrola probíhá porovnáním serverové id položky. Pokud položka existuje, je aktualizovaná, v opačném případě je vložena.

Pro odstranění starých záznamů se používají Z tabulky. Princip je zde stejný jako u zbytku aplikace. Pokud dojde k odstranění záznamu, je přidáno ID odstraněné položky do Z tabulky. S



Obrázek 4.3 Porovnání světlého a tmavého módu aplikace



Obrázek 4.4 Diagram synchronizace aplikace

tím je zároveň zvýšeno příslušné SN. Při dotazu na tuto tabulku jsou taktéž vygenerovány pouze řádky s vyšším SN než je v mobilu aktuálně k dispozici. Při zpracování je však nutné uvést, že zpracováváme Z tabulku a v takovém případě jsou položky rovnou mazány z lokální databáze.

Ještě bych rád zmínil rozdíl mezi serverovým ID a lokálním ID. Lokální id je primární klíč tabulky a je unikátní. Serverové ID je ID, které je vygenerováno serverem a na serveru je unikátní. Jelikož však aplikace podporuje možnost přihlášení stejných podniků, může dojít ke zdvojení těchto záznamů a proto je nežádoucí využívat tuto položku jako identifikátor. Jako identifikátor lze použít kombinaci server ID a lokální ID podniku. Tuto možnost jsem však nepoužil, protože osobně nerad pracuji se složenými klíči. Takže pro identifikaci využívám lokální id.

Nyní jsme si popsali základní koncept synchronizace. API vygeneruje soubor s daty a aplikace je zpracuje. To by však pro všechna data trvalo opravdu dlouho, a proto si aplikace rozdělí tuto činnost na menší části. Tím získáme několik výhod. Za prvé pokud dojde k chybě při komunikaci s Cloudem je díky SN záznamům možné pokračovat od místa chyby a není nutné stahovat vše znovu. Za druhé a to je podstatnější, získáváme tím možnost paralelního zpracování dat. Ze zkušenosti z Android aplikace nejdéle trvá samotné zpracování souboru, potom aktualizace dat v databázi a nakonec stahování. Pokud činnost rozdělíme, můžeme zpracovávat několik souborů najednou na různých vláknech a tím dojde k velkému urychlení synchronizace. Na Androidu se mi dokonce úspěšně povedlo paralelní synchronizaci implementovat a opravdu došlo k velkému urychlení.

Synchronizace je rozdělena do několika menších podúloh. Každá podúloha zpracovává jenom příslušnou doménu aplikace, například změny, zvířata, býky atd. V případě cloudu dojde v rámci jedné podúlohy jak ke stažení, tak k zpracování dat. V případě FTP jsou všechny soubory staženy v dedikovaném bloku pro stahování a v ostatních blocích dojde už jenom k zpracování.

V iOS aplikaci jsou podúlohy zpracovávány sekvenčně. A pokud dojde k chybě při zpracování podúloh, dojde automaticky k chybě celé synchronizace. Jednotlivé podúlohy jsou Odesílání, Ověření, Stáje, Skupiny, Býci, Zvířata, Laktace, Změny. Odesílání se stará o odeslání pořízených změn, laktací a aktualizací GPS pozic jednotlivých stájí. Ověření má za úkol aktualizovat oprávnění v aplikaci a kontrolovat kompatibilitu dat. Stáje zapracuje všechny stáje a uživatel stáje. Podúlohy Skupiny, Zvířata, Laktace a Změny vždy zpracují příslušné doménové entity odpovídající jejich názvu. U změn jsou zpracovány všechny entity, které se týkají změn. Kromě samotných změn jsou to změny detail, změny diagnózy a změny léčení. Všechny tyto podúlohy zároveň zpracovávají i Z tabulky dané části domény a tím odstraňují staré záznamy z databáze.

Pozorný čtenář si jistě všimne, že jsme zatím nemluvili o podúloze býci. Jak název napovídá, ta se stará o zpracování inseminačních býků. Kromě toho však i zpracovává ostatní věci jako diagnózy, léčiva, země atd. Tato podivná implementace vznikla z historických důvodů už na platformě Android. Tyto věci v aplikaci nebyly potřebné, a tudíž je nestahovala. V okamžiku, kdy jsem je začal potřebovat, jsem je z časových důvodu začal zpracovávat v této úloze. Bylo méně pracné je implementovat v již existující podúloze než vytvářet podúlohu novou. Při implementaci iOS jsem chtěl, aby se proces synchronizace tvářil stejně na obou platformách, a proto jsem tuto podivnost převedl i na iOS. V budoucnu však plánuji založení nové úlohy pro stahování těchto entit, které nemají jasné začlenění.

Kompletní diagram synchronizace v iOS vidíme na obrázku 4.4. Diagram obsahuje stavy, která reprezentují příslušné podúlohy. Kromě toho má navíc dva stavy Chyba a Kompletní.

Pokud při synchronizaci dojde k chybě, je synchronizace zastavena a uživateli je chyba zobrazena. Aktuálně aplikace rozděluje dvě chyby a to chyba kompatibility a obecná chyba. Chyba kompatibility znamená, že server poskytuje data, která mobil neumí zpracovat. V takovém případě je nutné aktualizovat aplikaci v mobilu. Obecná chyba je jakákoliv jiná chyba.

Pokud se synchronizace dostane do stavu Kompletní, potom je v databázi aktualizován čas poslední synchronizace, synchronizace končí a uživateli je zobrazen tento čas. Tento čas poslední synchronizace vidí uživatel na hlavní obrazovce vedle tlačítka pro synchronizaci. Díky tomu má přehled o staří dat v mobilu a sám může usoudit, zdali je nutné synchronizovat či nikoliv.

Jelikož synchronizace běží dlouho, byl jsem pro její implementaci nucen využít background

task. Tato funkce nám umožní spustit jistou operaci na pozadí. Díky tomu nedojde k zamrznutí UI aplikace. Veškerá synchronizace tedy běží v background tasku. Tento background task však může být kdykoliv zastaven. Toto zastavení inicializuje samotný systém, typicky pokud uživatel aplikaci shodí na pozadí nebo zhasne display mobilu. Systém však pár sekund před ukončením tasku pošle aplikaci upozornění o plánovaném zrušení daného tasku. Pokud toto upozornění přijde, aplikace přeruší synchronizaci a označí ji jako zrušenou. Zrušení synchronizace taktéž může být inicializováno samotným uživatelem.

Protože dojde k zrušení synchronizace při odchodu z aplikace a nebo při zhasnutí obrazovky, je nutné zařízení udržet vzhůru podobu synchronizace. Aplikace tedy zabrání automaticky zařízení usnout, pokud synchronizace probíhá. To neplatí v případě zásahu uživatele, kdy klikne na tlačítko uzamčení mobilu. V takovém případě zařízení usne nehledě na spuštěnou aplikaci. Toto řešení není zrovna šťastné, protože během synchronizace je nutné mít neustále zapnutý display mobilu a zároveň spuštěnou aplikaci. Jiné možné řešení tohoto problému jsem však nenalezl.

Background task je spuštěn v případě, že uživatel synchronizace spustí manuálně. Synchronizace však může být spuštěna i systémem pomocí takzvaného background processingu. Background processing je systémová úloha, kterou spouští systém v případě, že jsou splněny všechny nutné požadavky definované úlohou. Systém může spustit tuto úlohu, i když je aplikace vypnutá. Když jí spustí, je však pouze na systému. Programátor může definovat pouze podmínky nutné pro spuštění úlohy. Podmínky nutné pro spuštění synchronizace pomocí background processingu jsou přístup zařízení k internetu a zároveň zařízení musí být nabíjeno. Poslední podmínkou je čas. Od poslední úspěšné synchronizace musí uběhnout aspoň 24 hodin k tomu, aby bylo možné spustit tuto automatickou synchronizaci. I když dojde k naplnění všech těchto podmínek, systém nemusí aktualizaci spustit. Záleží pouze na systému, zdali nechá aplikaci synchronizovat.

Synchronizace je připravená, pouze jednotlivé podúlohy synchronizace jsou mockovány. Data jsou definovaná přímo v samotných mockách podúloh. Při synchronizaci nedochází k propsání uživatelem pořízených změn do aplikace. Vytvořené změny jsou tedy zapomenuty. Veškerá komunikace je realizovaná pomocí interface. Pro finální implementaci tedy stačí pouze změnit finální implementaci jednotlivých podúloh.

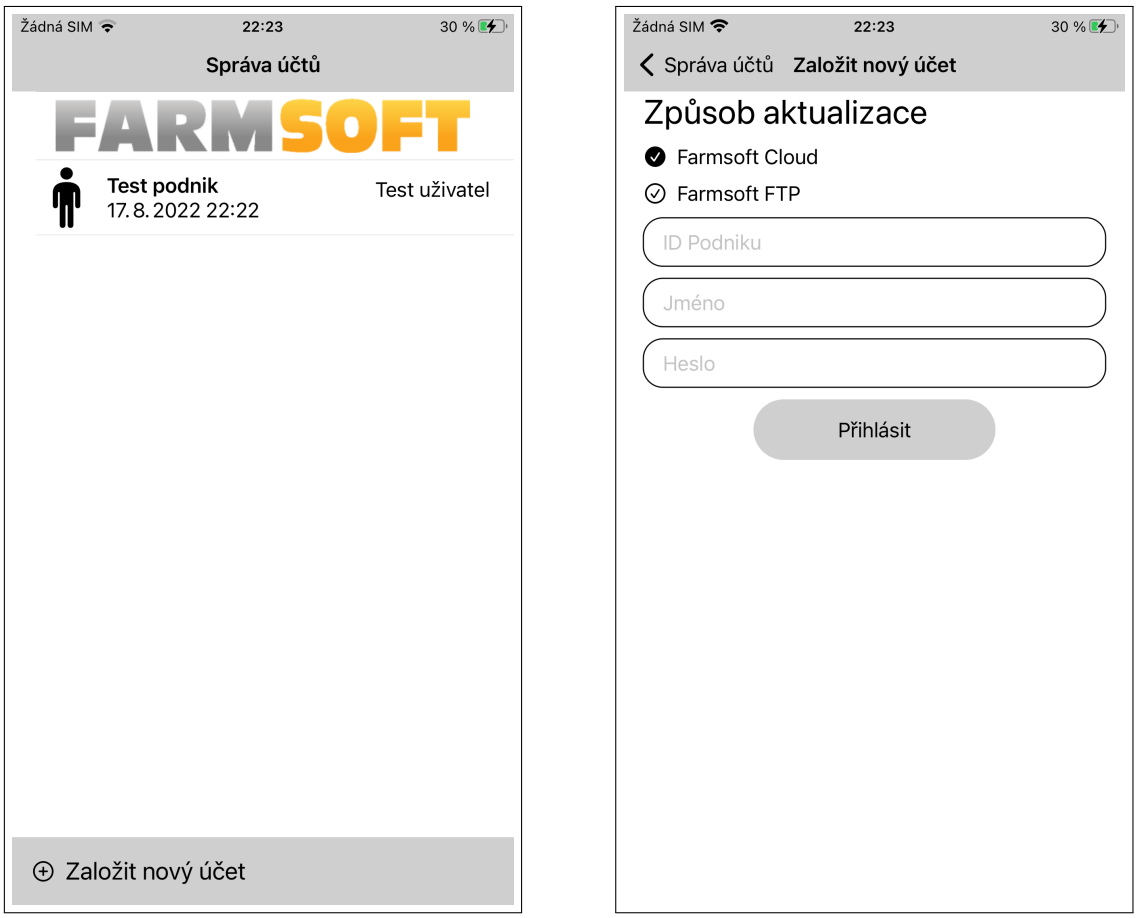
4.4.2 Přehled podniků a přihlášení

Po prvním spuštění aplikace je uživatel přesměrován na obrazovku přihlášených podniků. Ta je prázdná a má jedinou možnost, kliknout na tlačítko pro založení podniku. Tím je uživatel přesměrován na obrazovku pro přihlášení. Má možnost se přihlásit pomocí Cloudu a pomocí FTP. Při přihlášení přes Cloud musí uživatel zadat ID podniku, uživatelské jméno a heslo. Při přihlášení přes FTP je nutné zadat pouze ID podniku a heslo. Založení podniku na serveru aplikace neřeší.

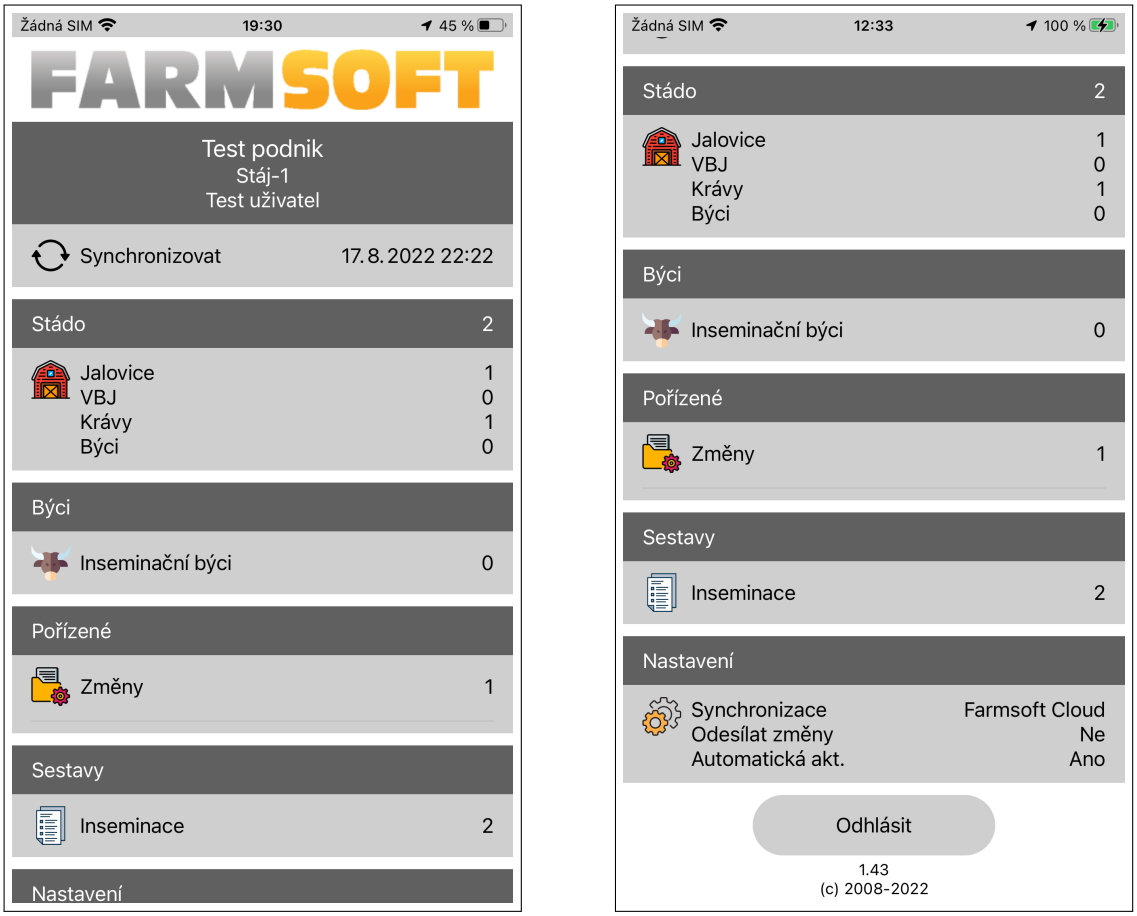
Po úspěšném přihlášení je uživatel automaticky přesměrován zpět na úvodní obrazovku s přihlášenými podniky. Zde vidí jeho nově přihlášený podnik. U podniku vidí jméno podniku, jméno přihlášeného uživatele a čas poslední synchronizace, pokud již nějaká proběhla. Uživatel může mít v jednu chvíli přihlášených několik podniků. Počet není nijak omezen.

Po kliknutí na podnik dojde ke změně jeho stavu na aktivní a uživatel je přesměrován na hlavní obrazovku tohoto aktivního podniku. Vždy může existovat pouze jeden aktivní podnik. Pokud existuje aktivní podnik, tak při otevření aplikace je uživatel automaticky přesměrován na hlavní obrazovku aktivního podniku. Status aktivního podniku zůstává u daného podniku i po ukončení aplikace.

Pro smazání podniku uživatel dlouze klikne na řádku podniku, který chce smazat. Objeví se dialogové okno s upozorněním, zdali si uživatel opravdu přeje smazat podnik. Mazání podniku může trvat i několik sekund, proto je uživateli zobrazena načítací obrazovka. Při smazání podniku jsou z databáze odstraněny veškeré záznamy spojené se smazaným podnikem. Po smazání podniku je uživatel opět přesměrován na úvodní obrazovku se seznamem podniků.



Obrázek 4.5 Obrazovka přihlášených podniků a obrazovka přihlášení podniku



Obrázek 4.6 Ukázka hlavní obrazovky

4.4.3 Hlavní obrazovka

Hlavní obrazovka aktivního podniku slouží jako rozcestník pro celou aplikaci. První položka je logo. Pro Farmsoft je zde logo FARMSOFT, pro Rskot je zde logo RSKOT. Do budoucna se zvažuje možnost zde umístit logo aktivního podniku. Pod logem se nachází několik karet. Každá karta má vždy na starost specifickou část aplikace, o které zobrazuje informace a po kliknutím na kartu je uživatel přesměrován dále v aplikaci. Kompletní hlavní obrazovku vidíme na obrázku 4.6.

Na první kartě pod logem nalezneme na tmavě šedivém pozadí jméno aktivního podniku a jméno uživatele aktivního podniku. Kliknutím na tento banner je uživatel přesměrován na list stájí dostupných uživateli. Pokud má uživatel vybranou aktivní stáj, je název této stáje zobrazen mezi názvem podniku a jménem přihlášeného uživatele. Ještě zde může být text ”žádná aktivní stáj”. To nastává, pokud uživatel nemá oprávnění mít vybrané všechny stáje a zatím žádnou stáj nevybral. Typicky se tak děje po první synchronizaci.

Druhá část této karty má na starost synchronizaci a všechny věci spojené se synchronizací. Uživatel na tomto řádku vidí ikonu synchronizace. Tato ikona se otáčí, pokud synchronizace probíhá. Vedle ikony je nápis synchronizovat. Pokud synchronizace probíhá, je zda název aktuálně probíhající podúlohy synchronizace. Na konci řádku je čas poslední úspěšné synchronizace. Pokud synchronizace probíhá, je zde tlačítko s ikonou křížku. Po kliknutí na toto tlačítko je synchronizace přerušena.

Druhou kartou je karta přehledu stáda. Po kliknutí na tuto kartu je uživatel přesměrován na obrazovku se seznamem zvířat. Na kartě vidí statistické údaje o stádu. Do statistik jsou vždy zahrnuta zvířata, která splňují aktuálně nastavené filtrování. Čili pokud má uživatel vybranou aktivní stáj, tak do statistik jsou zahrnuta pouze zvířata na této stáji. Další aspekty filtrování si probereme při popisu obrazovky se zvířaty.

Dále bych rád prošel jednotlivé položky statistik. Jalovice jsou všechny samice, které zatím neporodily. Tedy všechny samice, kde pořadí laktace je rovno 0. VBJ je podmnožina jalovic. Jedná se o březí jalovice, kterým uběhla od poslední inseminace časová konstanta. Konstanta je momentálně 150 dní a je definovaná přímo v kódu aplikace. Do budoucna však bude možné tuto definici přenechat na uživateli. Krávy jsou samice, které se alespoň jednou otelily (porodily). Jinými slovy samice, které mají pořadí laktace větší než 0. Býci jsou samci ve stádu.

Další karta je věnována býkům. Konkrétně inseminačním býkům používající při pořízení změny inseminace. Podobně jako u zvířat, počet inseminačních býků odpovídá aktuálně nastavenému filtru. Tento filter je však jiný než u zvířat a opět jej podrobně probereme při popisu obrazovky inseminačních býků. Na tuto obrazovku se lze dostat kliknutím na kartu.

Po kartě býků následuje karta pořízených položek. Tato karta je zobrazena pouze, pokud existuje pořízená položka, která zatím nebyla odeslána na server. Aktuálně v aplikaci lze pořídit tři položky a to změny, laktaci a aktualizace GPS souřadnic stáje. U každé položky se zobrazuje počet pořízených záznamů a pří kliknutí na položku se dostaneme na detailní list pořízených položek. Takže například když klikneme na změny, dostaneme se do listu pořízených změn. Podobně je tomu u laktací i aktualizací GPS pozic.

Následuje karta sestav. Karta aktuálně podporuje jenom jednu sestavu inseminací. Do budoucna však plánuji přidání i ostatních sestav aktuálně implementovaných v Android verzi. Číslo u sestavy inseminací reprezentuje počet nalezených inseminací podle aktuálně nastaveného filtru, který se nastavuje na obrazovce sestavy inseminací. Kliknutím na danou sestavu se přesměrujeme do jejího detailu.

Poslední kartou je karta nastavení. Na kartě nastavení uživatel vidí proti jaké části serveru probíhá synchronizace. Může to být Cloud či FTP. Dále jestli má zapnuté odesílání pořízených změn a automatické aktualizace. Kliknutím na kartu je uživatel přesměrován na obrazovku nastavení.

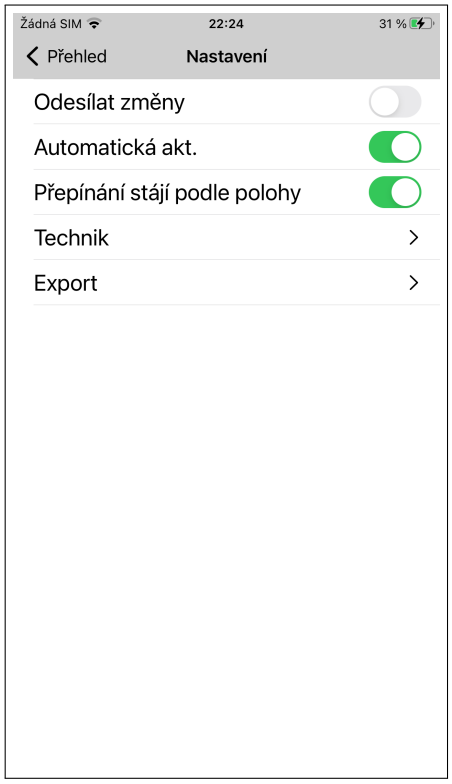
Poslední položkou na obrazovce je tlačítko pro odhlášení. Tlačítko deaktivuje aktivní podnik a uživatel je přesměrován na úvodní obrazovku s přihlášenými podniky. Zároveň je přerušena synchronizace, pokud nějak běží. Pod tlačítkem nalezne uživatel verzi aplikace a copyright.

4.4.4 Nastavení

Obrazovka nastavení slouží pro nastavení různých aspektů aplikace. Uživatel zde má možnost nastavit odesílání změn. Pokud je tato položka aktivní, potom jsou uživatelem pořízené položky odesílány na server. V opačném případě jsou pouze lokálně v zařízení. Další nastavení je nastavení automatické aktualizace. Pokud je aktivní, potom aplikace provádí automatické synchronizace pomocí background processingu, jak je popsáno v sekci synchronizace 4.4.1. Tato synchronizace běží na pozadí, i když aplikace není zapnutá. Kdy bude synchronizace spuštěna, však záleží pouze na systému.

Následuje přepínač pro automatické ”přepínání stájí podle polohy”. Pokud je aktivní, aplikace zažádá uživatele o práva k poloze telefonu. Tento přepínač je aktivní, pouze pokud aplikace tyto práva má. Pokud je ”přepínání stájí podle polohy”aktivní, aplikace při vstupu na hlavní obrazovku zjistí polohu uživatele. Pokud najde v dosahu 200 m od polohy uživatele stáj, je tato stáj nabídnuta uživateli k přepnutí jako aktivní stáj.

Následují dvě položky, které nejsou přepínače, ale navigují na další obrazovky. První je technik. Ten nás přesměruje na obrazovku výběru technika. Po výběru technika je číslo vybraného technika vidět vedle šipky. Pokud je vybrán výchozí technik, tak je tento technik automaticky nastaven při zadávání změny synchronizace. Uživatel však nemusí mít právo technika zvolit a



Obrázek 4.7 Obrazovka nastavení

technik mu může být přidělen správcem podniku. V takovém případě bude vedle šipky číslo přiřazeného technika a celý řádek bude zašedlý a nebude reagovat na interakci od uživatele. Ještě existuje možnost, kdy správce podniku přiřadí uživateli dva techniky. V takovém případě se na obrazovce objeví pod řádkem "Technik"nový řádek "Technik 2". Na tomto řádku bude zobrazeno číslo druhého nastaveného technika. Oba řádky budou zašedlé a nepůjde s nimi interagovat.

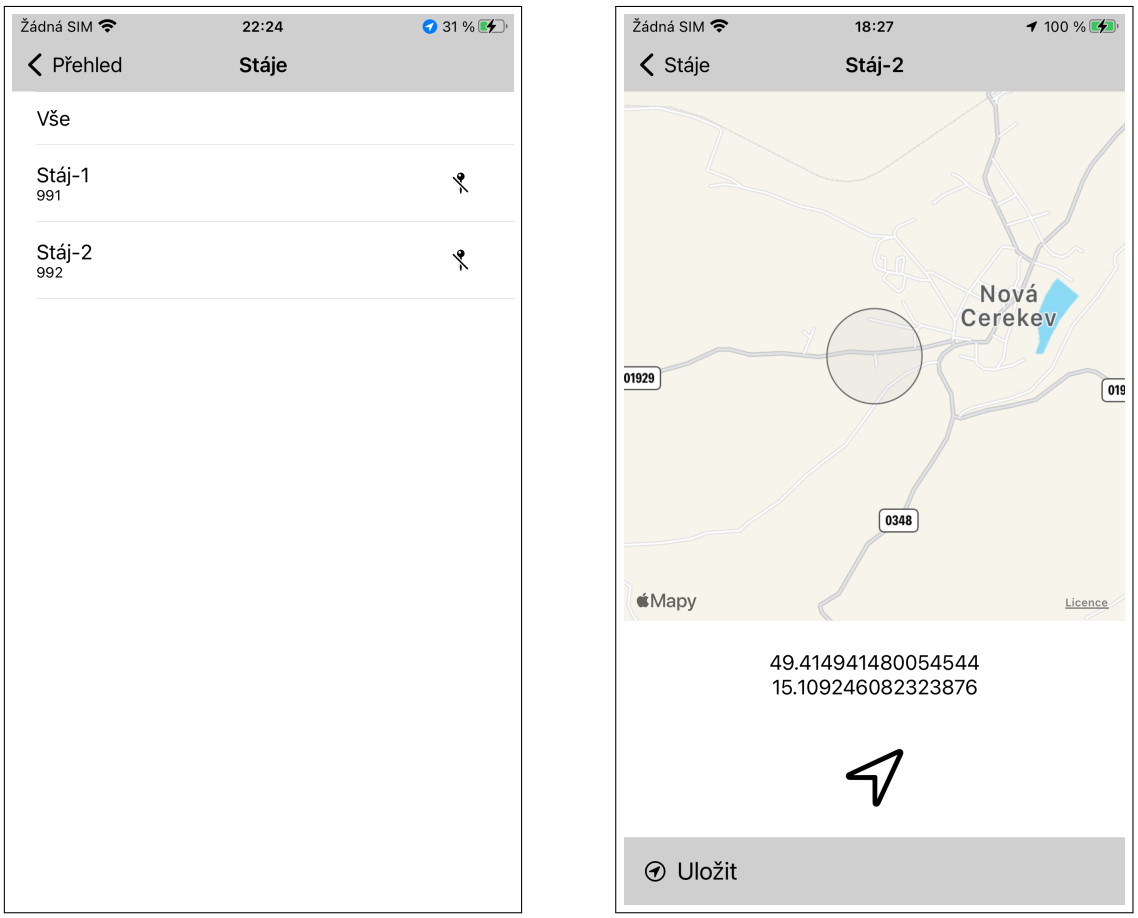
Poslední položkou je export. Ten nás přesměruje na obrazovku s aktuálně podporovanými systémy pro export. Momentálně je zde pouze systém Plemdat. Do budoucna však mohou přijít další systémy, a proto je navigace vymyšlena tímto způsobem. Po zvolením systému Plemdat je uživatel přesměrován na obrazovku nastavení exportu pro Plemdat. Momentálně zde může nastavit pouze "Identifikátor organizace". Jedná se o šestimístné celé číslo. Toto číslo se používá při generování inseminačních výkazů pro Plemdat. Toto generování si detailně popíšeme při popisu sestavy inseminací.

4.4.5 Stáje

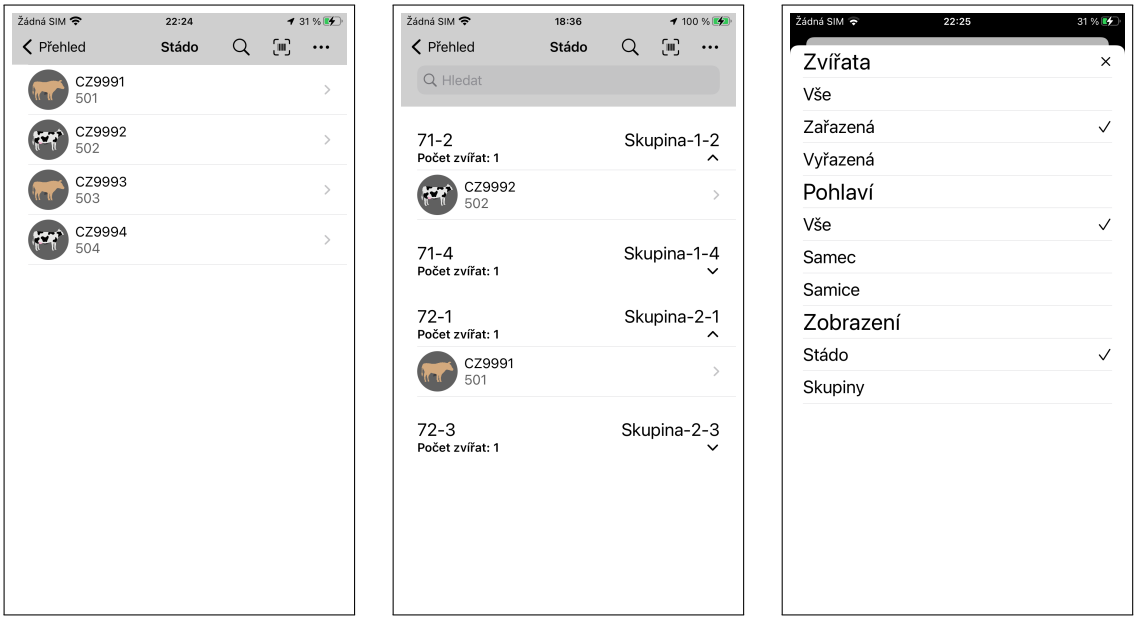
Obrazovka pro výběr aktivní stáje je celkem jednoduchá. Zobrazuje stáje, kam má uživatel přístup. Pokud má uživatel oprávnění, je mu nabídnuta možnost "vše". V takovém případě není vybraná žádná aktivní stáj a uživatel vidí všechna zvířata v podniku. Respektive všechna zvířata na stájích, kam má přístup.

Dostupné stáje jsou zobrazeny v listu. Každý řádek reprezentuje jistou stáj. Stáje může mít jméno a vždy musí mít číslo. Pokud je definováno jméno i číslo, je text v řádku dvouřádkový. První je jméno a druhý řádek textu je číslo dané stáje. Pokud má stáj pouze číslo, je zobrazeno pouze toto číslo.

Ikona špendlíku na pravé straně řádku je vidět pouze pokud je aktivní automatické přepínání



Obrázek 4.8 Obrazovka výběru aktivní stáje a nastavení polohy stáje



Obrázek 4.9 Obrazovka přehledu stáda

stájí podle polohy. Pokud je špendlík přeškrtnutý, potom není nastavena poloha dané stáje. Kliknutím na špendlík je uživatel přesměrován na novou obrazovku.

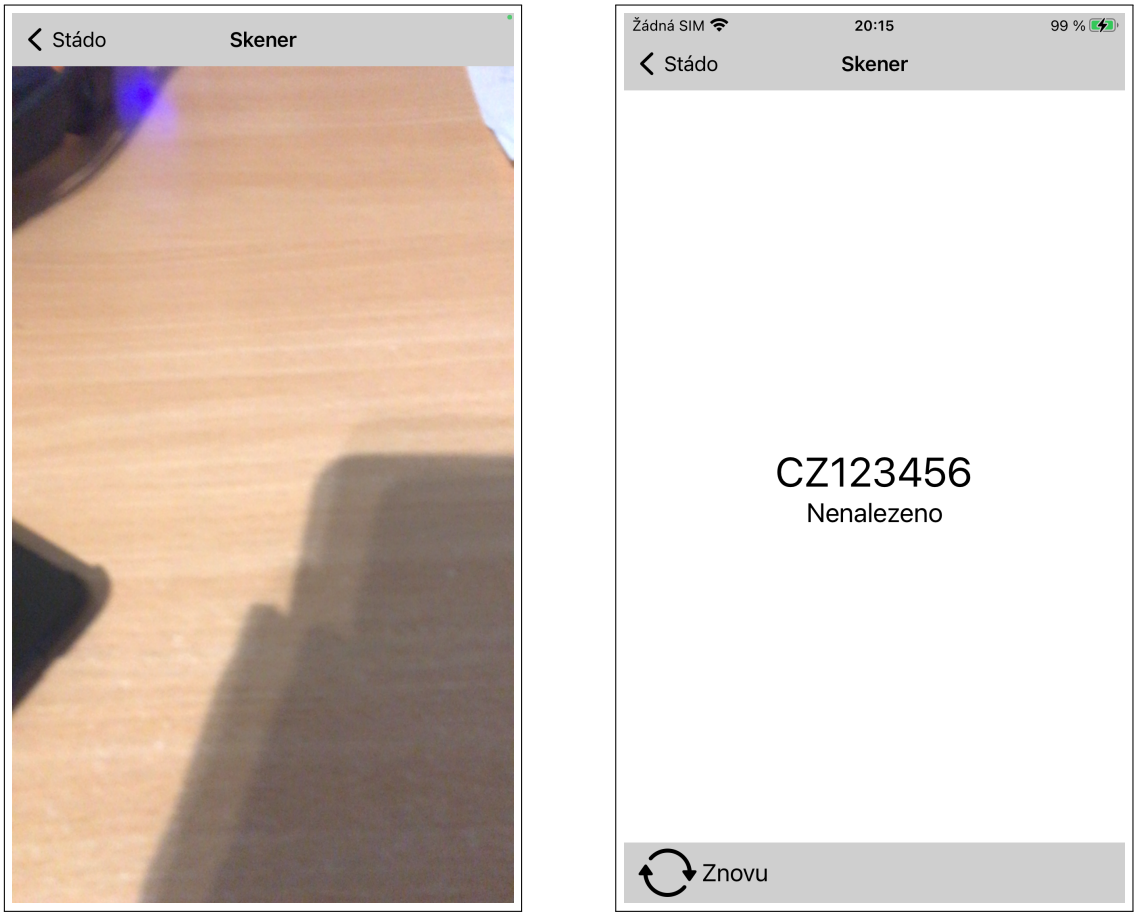
Na této obrazovce vidí mapu s kolečkem uprostřed. Toto kolečko reprezentuje rádius 200 metrů. Jedná se tedy o oblast, kde aplikace vyhodnotí danou stáj jako v dosahu. Pokud oblast je definována, potom je mapa automaticky nastavena na aktuální oblast vybrané stáje. Pokud oblast stáje není definována, potom je vybrána momentální poloha uživatele, jeli k dispozici. Samotná vybraná poloha je potom napsaná dole pomocí GPS souřadnic. Výběr polohy probíhá kliknutím na libovolný bod v mapě.

4.4.6 List zvířat

Obrazovka zvířat ukazuje všechna zvířata ve stádu. List má dva mody zobrazení, zobrazení stáda a zobrazení skupin. U zobrazení stáda jsou zvířata zobrazena v jednotlivých řádkách. Každý řádek obsahuje fotografii zvířete. Pokud fotografie existuje, je zobrazena, jinak je zobrazena ikona krávy, respektive býka podle pohlaví zvířete. Vedle fotografie je na první řádce kompletní číslo zvířete. Na druhá řádce je číslo obojku. Číslo zvířete je unikátní a povinný údaje, číslo obojku nemusí být unikátní a není povinný. Kliknutím na řádek je uživatel přesměrován na detail zvířete. Zvířata jsou seřazeny podle čísla kusu. Při zobrazení stáda je načteno vždy maximálně několik desítek záznamů, kvůli zrychlení načítání.

Druhý mód zobrazení je zobrazení skupin. V takovém případě jsou zobrazeny všechny skupiny v podniku, respektive všechny skupiny na aktivní stáji. Každá skupina zobrazuje v prvním řádku číslo skupiny vlevo a název skupiny vpravo. Další řádek popisuje, kolik zvířat je v dané skupině a indikaci rozbalení skupiny. Po rozbalení skupiny jsou zobrazena všechna zvířata v dané skupině. Porovnání zobrazení můžeme vidět na prvních dvou obrazovkách obrázku 4.9.

Třetí obrazovka ukazuje filtrační obrazovku. Filtrační obrazovka vyjede ze spodní strany obrazovky, pokud uživatel klikne na tří tečky v horním navigačním baru. Na filtrační obrazovce má možnost změnit mód zobrazení a dále nastavovat filtr zvířat. Filtr aktuálně podporuje dvě nastavení. Prvním je filtrace podle pohlaví. Druhým je filtrace podle zařazení do reprodukce. Zvířata vyřazená z reprodukce se typicky neinseminují. Uživatel tuto obrazovku zavře křížkem



Obrázek 4.10 Obrazovka skenování čárových a QR kódů

a nebo tažením prstu dolů. Ihned po uzavření vidí list podle nastavení filtrace, které učinil. Nastavení filtru je perzistentní a není resetováno při ukončení aplikace.

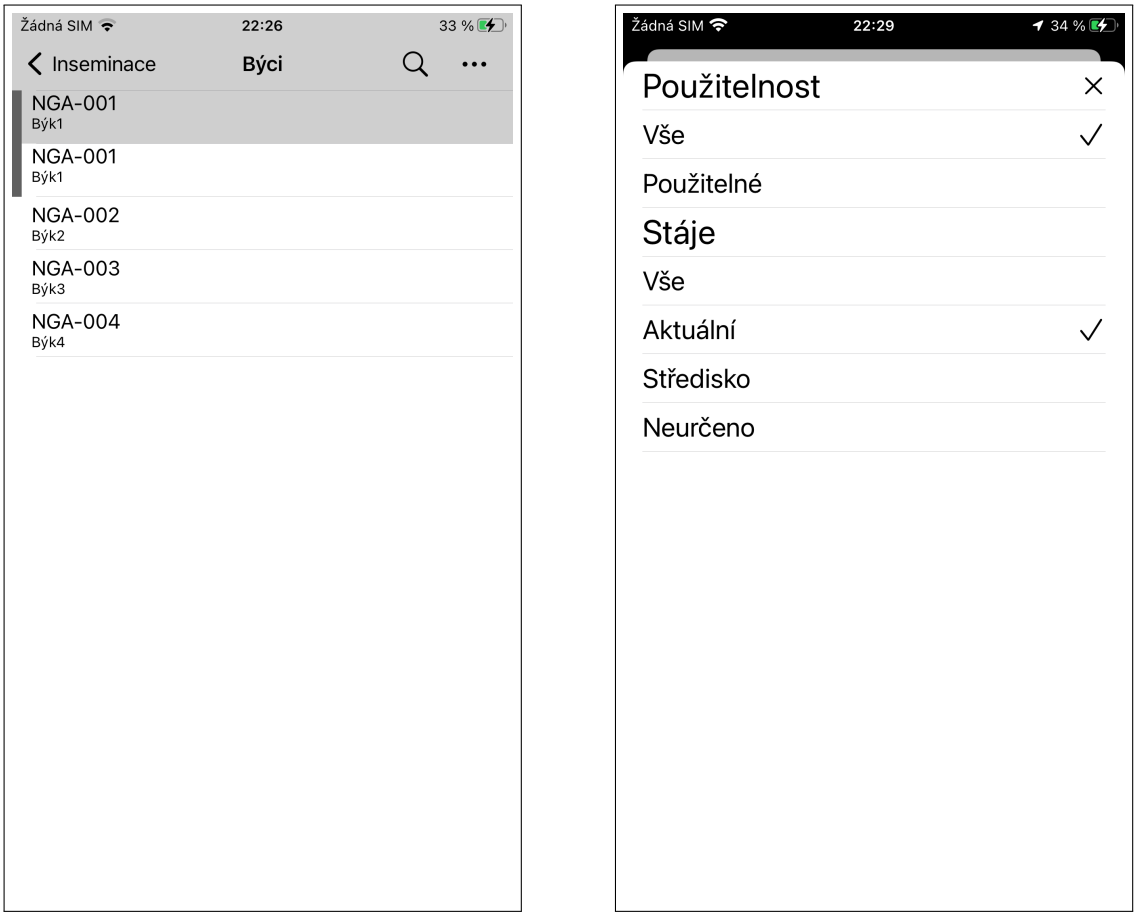
Ikona vedle tří teček spustí skenování. Lupa přepne uživatele do vyhledávacího režimu. Ten lze taky aktivovat tažením prstu dolů, pokud je uživatel na první položce v listu. Vyhledávat lze pouze čísly. Zvířata jsou vyhledávány podle čísla kusu i podle obojku. Výsledek vyhledávání je vždy zobrazen v módu zobrazení stáda. V případě Rskotu se na horní liště nachází ještě ikona plusu. Ta uživatele přesměruje na obrazovku založení změny narození.

4.4.7 Skenování

Obrazovku pro skenování využívají uživatelé pro načtení QR kódů a čárových kódů z karet zvířat. Aplikace podporuje skenování pouze ze zadního fotoaparátu. Pro naskenování stačí namířit kameru na QR, respektive čárový kód. Skenování funguje opravdu rychle a i za špatných světelných podmínek. Aplikace skenuje QR kódy a čárové kódy ve formátu Code128. O skenování se stará knihovna GoogleMLKit popsaná v sekci 4.3.8.

Pří naskenování kódu je uživatel přesměrován na další obrazovku, kde uprostřed vidí naskenované hodnoty. Aplikace se pokusí podle naskenovaných hodnot najít požadované zvíře. Pokud zvíře najde, přesměruje uživatele na detail tohoto zvířete. Pokud nic nenajde, zobrazí uživateli chybovou hlášku a ten má možnost skenování opakovat.

Pro Rskot je zde ještě možnost založit nové zvíře. Ta je k dispozici pouz,e pokud uživatel



Obrázek 4.11 Obrazovka inseminačních býků

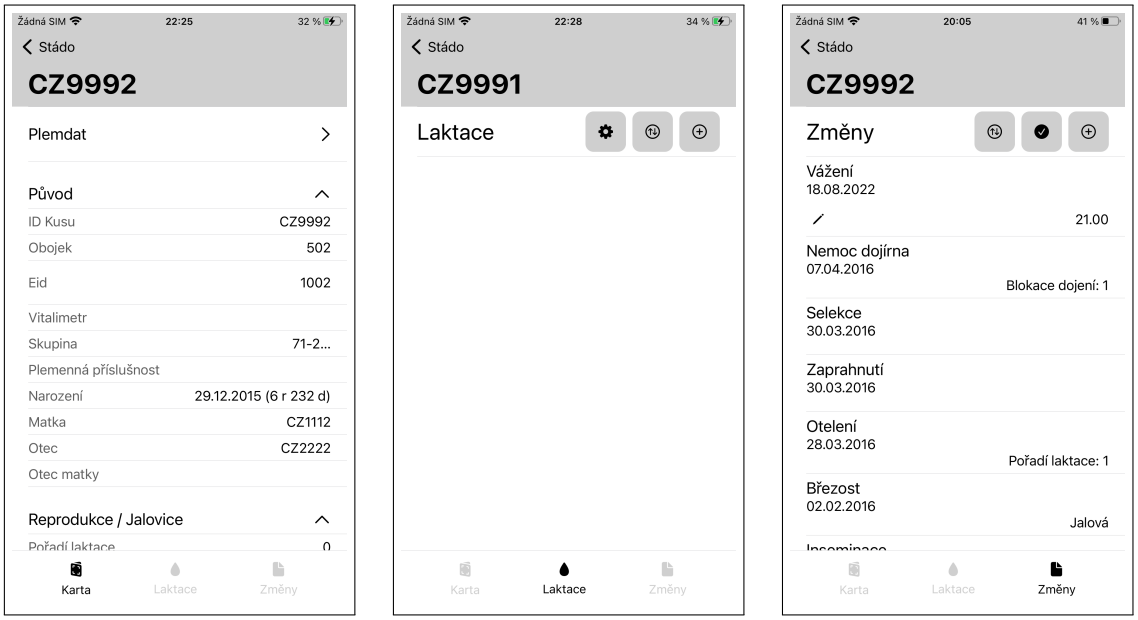
použil pro skenování QR kód a jsou v něm nalezeny všechny požadované informace. QR kód totiž obsahuje všechny informace nutné pro založení nového zvířete v aplikaci. Po kliknutí na tlačítko založit nové zvíře je uživatel přesměrován na změnu narození, kde má již předvyplněné hodnoty naskenované z QR kódu na kartě zvířete.

4.4.8 Inseminační býci

Obrazovka inseminačních býků slouží pro výběr býka při zadávání změny inseminace. Zobrazuje jednotlivé býky v řádkách. U jednotlivých býku se může vyskytnou v pravé části řádku šedivý proužek. Ten značí takzvané sexované sperma. Každý řádek s býkem může mít až tří řádky textu popisem tohoto býka. Na prvním řádku je vždy registrační číslo býka. Registrační číslo je jediná povinná položka. Po registračním čísle následuje název býka. Za názvem může být umístění býka. Býk může být umístěn v rámci stáje, nebo v rámci střediska. Umístění v rámci stáje je historické a do budoucna bude odstraněno.

Pokud býk není nikde umístěn, potom je to obecný býk. U obecných býků může být zobrazena cena býka, která se platí za inseminaci tímto býkem. Tato cena je samozřejmě i u neobecných býků. Ta se však v aplikaci nezobrazuje. Není zobrazena, protože tato cena je typicky nasmlouvaná a inseminátoři by ji neměli znát.

Obrazovku inseminačních býků lze zobrazit z hlavní obrazovky a z obrazovky změny inseminace. Pokud je zobrazena z hlavní obrazovky potom slouží pouze pro přehled a při kliknutí



Obrázek 4.12 Obrazovka detailu zvířete

na řádek s býkem se nestane žádná akce. Při zobrazení ze změny inseminace bude při kliknutí tento býk vybrán pro změnu inseminace. Dále na prvních třech pozicích mohou být býci se šedivým pozadím. Tyto býci patří do připařovacího plánu zvířete, na kterém je změna inseminace pořizována. Tyto býci by měli být ideální pro inseminaci tohoto zvířete.

Podobně jako u zvířat i u býků je možné vyhledávat. Do vyhledávání je možné zadávat jakýkoliv znak. Vyhledávání uživatel aktivuje kliknutím na lupu a nebo tažením prstu dolů, pokud se nachází na prvním záznamu v listu. Aplikace vyhledává podle registračního čísla a jména býka.

Tři tečky slouží pro filtraci býků. Filtrace je možná podle použitelnosti a podle stáje. Při filtraci podle použitelnost má uživatel právo zvolit vše a nebo pouze použitelné. Správce systému může uživateli odebrat právo tuto položku nastavovat. V takovém případě sekci použitelnost ve filtru vůbec neuvidí a vidí pouze použitelné býky.

Druhou možností je filtrace podle stáje. Pokud je zvoleno vše, potom jsou zobrazeny všichni býci. Pro neurčeno jsou zobrazeni pouze obecní býci. Při zvolení aktuální jsou zobrazení pouze býci, kteří jsou na aktuálně zvolené stáji. Při zvolení středisko jsou zobrazeni všichni býci, kteří jsou na středisku, na kterém se v podnikové hierarchii nachází aktuální stáj. Správce systému může některé položky uživateli odebrat a ten je nebude moci nastavovat. Jedná se o položky vše a aktuální stáj. Pokud je uživateli právo odebráno, tyto položky jsou ze seznamu odstraněny.

Filtrace podle stájí je důležitá hlavně u Rskotu. Zde jednotliví technici objíždí jednotlivé podniky. V podnikové hierarchii Rskotu jsou tyto podniky vedeny jako střediska, kde každé středisko má svoji stanovenou cenu pro inseminačního býka. Pokud technik chce inseminovat, musí zvolit správného býka na správné stáji a aplikace mu toto zvolení musí co nejvíce usnadnit. Při zvolení špatného záznamu může dojít ke špatnému ocenění provedené inseminace. Proto je vždy při vstupu na tuto obrazovku vybrána volba středisko, popřípadě aktuální. Uživatel tedy vždy vidí pouze býky na aktuální stájí a tím se eliminuje možnost chyby při zadávání.

4.4.9 Detail zvířete

Obrazovka detailu zvířete slouží jako detailní zdroj všech dostupných informací o daném zvířeti. Číslo zvířete je vidět v navigační liště. Díky tomu má uživatel přehled, na které zvíře se zrovna dívá. Obrazovka je rozdělena na tři části. Jsou jimi karta zvířete, laktace a změny.

Karta zvířete je zdroj všech informací o zvířeti. Uživatel zde vidí souhrn všech dostupných informací, které o zvířeti aplikace může poskytnou. Jedná se například o pohlaví, reprodukční stav, pořadí laktace, nádoj atd. Druhou částí jsou laktace. Zde jsou všechny pořízené laktační záznamy o zvířeti. Třetí je obrazovka se všemi provedenými změny zvířete.

Uživatele Rskotu laktační záznamy nevyužívají, a tudíž pro Rskot verzi je tato část odstraněna a uživatelé do ní nemají přístup. Jednotlivé podobrazovky si nyní popíšeme detailněji.

4.4.9.1 Karta

Karta zvířete zobrazuje souhrn aktuálních informací o zvířeti. Představuje stav, v jakém by se zvíře momentálně mělo nacházet. Karta je členěna do sekcí. Každá sekce jde zabalit a rozbalit. Stav sekcí zůstává uložen. Uživatel si tedy může nechat rozbalené pouze sekce, které ho zajímají.

První položkou na kartě je odkaz do systému Plemdat. Po kliknutí na tlačítko je uživateli otevřen webový prohlížeč a je přesměrován na dané zvíře v systému Plemdat.

Následuje sekce původ. V sekci vidí nejdůležitější informaci o aktuálním zvířeti. Jsou zde údaje k identifikace. To je ID Kusu, Obojek, Eid, Vitalimetr a skupina, v jaké se zvíře momentálně nachází. Poté následují informace Plemenná příslušnost, datum narození, číslo matky, otce a otce matky. U data narození je zobrazené i celkové stáří kusu. Sekce původu je jediná, která se při zabalení nezabalí úplně, ale informace nutné k identifikaci jsou vždy viditelné.

Sekce reprodukce zobrazuje veškeré informace o reprodukci zvířete. V názvu sekce je za lomítkem umístěn aktuální reprodukční status zvířete. Reprodukční stavy jsou například Jalovička, Březí, Otelená atd. První informací v této sekci je pořadí laktace, neboli počet otelení. Následuje datum posledního otelení, které má v závorce uvedený počet dní od posledního otelení. Další je datum poslední říje. V závorce je počet dní od poslední říje. Poté následují inseminační údaje jako datum poslední inseminace, pořadí této inseminace a býk, který byl při ní použit. Dalšími položkami jsou datum březosti, datum zaprahnutí a očekávané otelení. U březosti může být za datumem znak + respektive -. Ten značí výsledek březosti. Datum zaprahnutí a očekávané otelení mají v závorce počet dnů.

Aplikace zobrazuje vždy pouze relevantní data v aktuálním reprodukčním cyklu, který začíná otelením. Následuje říje, inseminace, březost a zaprahnutí. Aplikace zobrazí tyto údaje pouze, pokud jsou po datu posledního otelení. Inseminaci, březost a zaprahnutí zobrazí pouze, pokud je jejich datum po datu poslední říje. Březost a zaprahnutí se zobrazí pouze, pokud je po poslední inseminaci. Aplikace tedy vždy zobrazuje pouze data k aktuálním reprodukčnímu cyklu. Data z minulých cyklů jsou skryta. Lze je však najít ve změnách.

Další sekce je sekce věnovaná dojení. Uživatel zde vidí poslední nádoj, průměrný nádoj. Dále datum poslední kontroly užitkovost, nádoj při poslední kontrole užitkovosti a Sb údaje z této kontroly. Po dojení je sekce aktivita. Ta zobrazuje data z vitalimetru. Zobrazuje data maximální aktivity na dnešní a včerejší den. Po aktivitě je připařovací plán.

V sekci připařovacího plánu vidí uživatel býky ideální pro inseminaci. Tito býci budou při zadávání změny inseminace zobrazeni na prvních pozicích a mají barevně odlišné pozadí. Uživatel může zadat až tři býky do připařovacího plánu. V sekci vidí registrační čísla všech třech býků.

Poslední sekcí je vážení. V sekci vážení vidí uživatel datum posledního vážení a počet dní od tohoto data. Dále vidí zaznamenanou hmotnost při posledním vážení.

4.4.9.2 Laktace

Obrazovka laktace zobrazuje seznam pořízených laktačních záznamů u daného zvířete. První řádek obsahuje nadpis laktace a tři tlačítka. Tlačítko s ozubeným kolečkem slouží pro nastavení

zobrazených položek u laktace. Po kliknutí je uživatel přesměrován na obrazovku výběru položek, které si přeje zobrazovat u detailu zvířete. Druhé tlačítko je pro filtrování. Uživatel může zobrazit všechny záznamy, popřípadě záznamy k určitému laktačnímu cyklu. Laktační cyklus je doba mezi oteleními (porody). Poslední možnost filtrovaní je filtrovat záznamy pouze k poslední laktaci. Třetí tlačítko je tlačítko s ikonou plus. Tou uživatel založí nový laktační záznam.

Poté již následují laktační záznamy odpovídající aktuálnímu filtru. Každý záznam má na prvním řádku jeho datum. Druhý řádek je počet dní za laktační období. To je nastaveno filtrováním. Například pokud uživatel zvolí filtrovat vše, tak se tento počet dní počítá od narození zvířete. Pokud zvolí například druhé laktační období, potom se počet dní počítá od druhého otelení. Třetí řádek je rozdělen na dvě poloviny. Vpravo jsou umístěny položky laktačního záznamu, které si uživatel natavil pomocí ozubeného kolečka. Vlevo je ikona tužky, pokud je záznam pořízen lokálně a ještě nebyl odeslán na server.

Záznamy s tužkou jsou tedy lokální a uživatel je může mazat. Pokud uživatel na nějakém lokálním záznamu dlouze podrží prst, přepne se obrazovka do módu pro výběr. Uživatel si pak může jednoduchým kliknutím označit položky k smazání. Označit lze pouze lokální záznamy, tudíž záznamy s ikonou tužky. V módu výběru se v horní navigační liště objeví ikona popelnice. Po kliknutí na popelnici uživatel smaže označené záznamy. Označené záznamy mají šedivé pozadí a počet označených záznamu je napsán v navigační liště. Pokud uživatel klikne na již označený záznam, zruší tím jeho označení. Uživatel se přepne zpět do normálního módu kliknutí na ikonu popelnice, nebo zrušením označení všech záznamů. Pokud je uživatel v normálním módu a klikne na záznam, je přesměrován na jeho detail.

4.4.9.3 Změny

Část změny zobrazuje přehled všech změn u daného zvířete. První řádek obsahuje nadpis změny a tři tlačítka. První tlačítko slouží k řazení. Po jeho kliknutí je uživateli zobrazena obrazovka, kde může zvolit styl řazení. Aplikace umí řadit změny podle data a nebo podle typu. Při řazení podle data jsou změny seřazeny vzestupně. Poslední záznamy jsou tedy nahoře a uživatel je má hned k dispozici. Starší záznamy jsou v listu níže. Řazení podle typu změny všechny změny seskupí podle typů. Například pokud jsou v listu změny typu inseminace, potom aplikace v listu zobrazí všechny inseminace u sebe. Takto seskupené inseminace dále řadí podle data vzestupně.

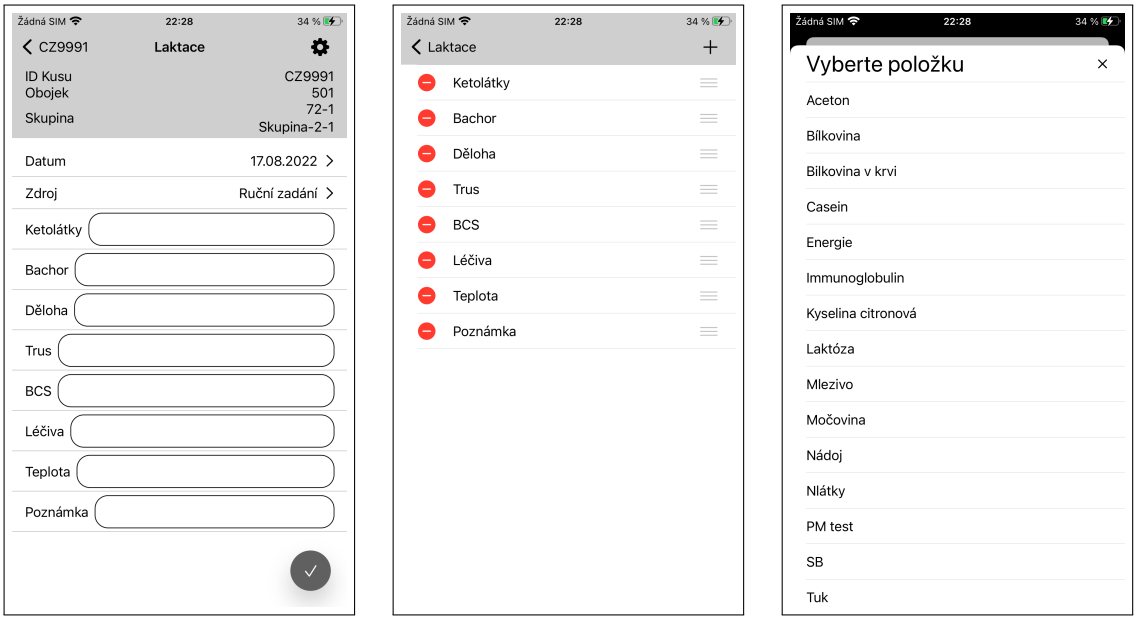
Druhé tlačítko filtruje změny podle typu. Po kliknutí na něj je uživateli zobrazena obrazovka všech dostupných typů změn, které aplikace podporuje. Uživatel si může zvolit, které typy ho zajímají a ty následně uvidí. Pro pohodlí uživatele je zde speciální tlačítko vše. To označí všechny změny. Pokud jsou všechny změny označené, tak naopak zruší označení u všech změn.

Posledním tlačítkem je tlačítko pro pořízení změny. Po jeho kliknutí je uživateli zobrazen dialog. Zde uživatel vidí všechny typy změn, které aplikace umí založit. Každé tlačítko obsahuje název změny a ikonu změny. Po kliknutí je uživatel přesměrován na obrazovku dané změny.

Po první řádce s tlačítky již následují jednotlivé změny, které splňují filtrování. Každá změna má na prvním řádku její typ. Druhý řádek je datum změny. Třetí řádek je rozdělen. Vlevo je indikace tužky, pokud je změna pořízena pouze lokálně. Pro změny již uložené na serveru zde žádná ikona není. Vpravo jsou potom důležité informace k jednotlivým změnám. Informace závisí na typu změny. Pro označení se jedná o číslo obojku. U váhy je zobrazena zaznamenaná váha, u poznámky zapsaná poznámka atd.

Lokální změny s tužkou lze dlouhým podržením označit. Po označení první změny se obrazovka přepne do módu označení a v navigační liště je zobrazena popelnice, která smaže označené záznamy. Mód označení se chová stejně, jako jsme si ho popsali u laktací 4.4.9.2. Pokud aplikace není v módu označení a uživatel klikne na změnu, potom je přesměrován do detailu změny.

Aplikace umí pracovat pouze s některými typy změn. Jsou to typ označení, vážení, poznámka, říje, inseminace, březost, otelení, selekce, zaprahnutí, umístění, paznehty, léčení, nemoc dojírna a narození. Změnu typu léčení umí aplikace pouze zobrazit a neumí jí pořídit. Změnu typu narození lze pořídit pouze v Rskot verzi. Farmsoft verze jí neumí pořídit, ale zobrazit jí dovede. Ostatní



Obrázek 4.13 Obrazovky spojené s laktací

podporované změny umí Farmsoft pořídit i zobrazit.

Pokud je uživatel na kartě býka, potom nedává smysl, aby mohl pořizovat změny spojené s reprodukcí. Proto u samců lze pořizovat jenom změny typu označení, vážení, poznámky, selekce, umístění a paznehty. Pro Rskot taktéž nedává smysl pořizovat některé změny. Proto v Rskot verzi lze pořídit změny typu narození, poznámka, inseminace, březost, otelení a umístění.

4.4.10 Pořízení laktace

Obrazovka pořízení laktace je rozdělena na dvě části. První část zobrazuje informace o zvířeti, pro které je laktace pořizována. Je zde číslo kusu, obojek a skupina, na které se zvíře aktuálně nachází. Druhá část slouží k samotné definici pořízené laktace. První položkou je datum pořízení. Druhou položkou je zdroj. Zdroje mohou být dvě hodnoty nespecifikováno a ruční zadání. Uživatel typicky zvolí ruční zadání, proto je toto nastaveno jako výchozí hodnota. Nespecifikováno je typicky použito při vytvoření záznamu pomocí automatického zpracování.

Poté jíž následují jednotlivé položky laktace. Uživatel si může definovat položky pomocí ikony v navigační liště. Ta ho přesměruje na obrazovku, kde vidí názvy jednotlivých položek. U položek může měnit jejich pořadí. Pořadí změní kliknutím na tří čárky v pravé části řádku a tažením na požadovanou pozici. Smazat položku může přes tlačítko mínus a následné potvrzení. Položku může přidat přes ikonu plusu v navigační liště. Nastavení je automaticky uloženo po opuštění obrazovky.

Na stejnou obrazovku nastavení položek je uživatel přesměrován i z karty zvířete. Zobrazované položky se liší pro zadávání a pro zobrazení na kartě. Vždy záleží, z jaké obrazovky zobrazované položky uživatel nastavuje. Na kartě si tedy může definovat pouze pár důležitých položek, které chce vidět. Na detailu potom může toto nastavení rozšířit o další položky. Nastavení zobrazení položek zůstává i po ukončení aplikace a to pro detail i pro kartu laktace.

Na obrazovce detailu laktace tedy uživatel vidí jím nadefinované položky. Všechny položky mají vpravo jejich název a vlevo textové pole, kam uživatel píše hodnotu položky. Některé položky můžou být textové, některé položky jsou pouze čísla. Všechny položky jsou volitelné, a tudíž některé uživatel vyplnit nemusí. Pro dokončení editace uživatel uloží změny kliknutím na fajfku

vpravo dole.

4.4.11 Obrazovky změn

Nyní bych rád popsal jednotlivé obrazovky pro změny. Obrazovky pro změny mají několik atributů, které jsou společné pro všechny obrazovky. Každá obrazovka změn je rozdělena na dvě části. Jsou to hlavička a tělo.

Hlavička má šedivé pozadí a jsou zde umístěny různé relevantní informace k zvířeti, ke kterému je změna pořízena. Obsah hlavičky záleží na typu změny, ale vždy jsou zde umístěny identifikační údaje zvířete, tedy číslo kusu a obojek. Hlavička je vždy rozdělena na dvě poloviny. Vpravo jsou názvy jednotlivých položek a vlevo jejich reálné hodnoty.

Při popisu změn budu vždy popisovat pouze rozdílné položky v hlavičce. Čili pro kompletní seznam položek v hlavičce dané změny si čtenář vždy musí domyslet i položky identifikace, tedy položky čísla kusu a obojku.

Tělo je vždy specifické pro každou změnu. Typicky je rozděleno na dvě částí. Vlevo jsou názvy příslušných atributů a vpravo zadané hodnoty. Uživatel hodnotu buď zadává přímo do textového pole, nebo hodnotu volí z předdefinovaných možností typicky kliknutím na řádek s atributem. Pokud je atributy výběrový, potom je na konci řádku vždy šipka. Vlevo vedle šipky je potom aktuální hodnota.

Společné pro všechny změny je zadávání data změny. Datum je vždy v těle úplně nahoře. Při zadávání nové změny je výchozí datum vždy aktuální datum. Pokud si uživatel přeje datum změnit, klikne na datum. Uživateli se objeví obrazovka pro výběr data. Pro potvrzení data klikne na tlačítko Ok a tím je změně nastaven nový datum. Podobně jako u hlavičky i zde budu popisovat pouze rozdílné položky a o položce datu se při detailním popisu již nebudu zmiňovat.

Název typu změny je vždy jako titulek v horní navigační liště. Pro uložení zadaných dat musí uživatel kliknout na tlačítko uložit. To je umístěno vpravo dole a má ikonu fajfky na šedivém pozadí.

Obrazovka změn má dva módy. Pokud se jedná o změnu, která je již uložena na serveru, potom je obrazovka v módu pouze pro čtení. Uživatel vidíš všechny atributy změny, ale nemůže je editovat a tlačítko pro uložení změn je schováno. Druhý mód je, pokud uživatel chce pořídit změnu a nebo upravuje lokální změnu, která zatím nebyla odeslána. V takovém případě má přístup k úpravám všech položek a má možnost kliknout na tlačítko pro uložení.

4.4.11.1 Narození

Změna narození je jediná změna, která neobsahuje žádnou hlavičku. V těle je datum, pohlaví, číslo kusu, plemenná příslušnost, matka a otec. Pohlaví je výběrový atribut. Po kliknutí je uživateli zobrazen dialog s dvěma hodnotami. Jsou to jalovice a býček.

Číslo kusu reprezentuje číslo nového kusu, to je složeno ze tří částí, při čemž první je tlačítko země. Po kliknutí je uživatel přesměrován na obrazovku výběru země. Po vybrání země je její zkratka napsaná v textu tlačítka. Výchozí hodnota je CZ. Další dvě části jsou textové. První část je samotné id kusu, druhá je kodex. Do obou textových polí lze psát pouze celá čísla.

Následuje plemenná příslušnost. Tu si uživatel může vyplnit libovolným textem. Po příslušnosti je zde matka. Položka matka je stejná jako položka číslo kusu. Uživatel by měl zde vyplnit číslo kusu matky nově narozeného zvířete.

Poslední položka je otec. Do položky otec, by uživatel měl vyplnit registrační číslo býka použitého k oplodnění matky.

Pokud uživatel zakládá novou změnu narození, potom je zde navíc položka skupina. Po jejím kliknutí je uživatel přesměrován na výběr dostupných skupin. Pokud je vybraná nějaká aktuální stáj, potom jsou zobrazeny pouze skupiny na aktuální stáji.

Pro uložení změny je nutné vyplnit číslo kusu a pohlaví. U čísla kusu je nutné vyplnit zemi a id kusu. Kodex je dobrovolný, jelikož se v jiných státech nemusí používat. Pokud je pořizována

nová změna potom je nutné vybrat i skupinu, kam bude zvíře přidáno.

Po uložení je v systému založena nová změna narození a umístění. Zároveň je vytvořen nový záznam zvířete. Takto nově vniklé zvíře má hodnoty vygenerované na základě dat zadaných do změny narození a zvíře je umístěno do skupiny vybrané při zadávání této změny.

4.4.11.2 Označení

Změna označení má v hlavičce aktuální číslo elektronické identifikace (EID) a číslo vitalimetru daného zvířete. Tělo má tři textová pole. Vlevo je vždy popisek, co dané pole představuje, vpravo potom uživatel píše reálnou hodnotu. Uživatel může vždy zadávat pouze celá čísla. První textové pole je obojek, druhé je EID a poslední vitalimetr. Všechny atributy jsou volitelné.

4.4.11.3 Vážení

Vážení má v hlavičce u zvířete skupinu. U skupiny je vždy zobrazen jak její název, tak číslo. Následuje datum posledního vážení, které má v závorce umístěno počet dní od posledního vážení. Potom je uvedena hmotnost při posledním vážení. Poslední údaj je aktuální hmotnost. Tu může vypočítat server, pokud jsou v podniku definované konstanty nutné pro výpočet. Pokud definované nejsou, potom je hodnota prázdná.

V těle má uživatel možnost zadat pouze samotnou váhu v kilogramech. Lze zadávat pouze celá kladná čísla.

4.4.11.4 Poznámka

Poznámka obsahuje v hlavičce informace o aktuální skupině zvířete. V těle má uživatel možnost zadávat typ poznámky a samotnou poznámku. Pro zadání typu uživatel klikne na řádek s typem. Pro výběr typu je uživateli zobrazen dialog. Má možnost vybrat typ jeden z čísel od 1 do 6, popřípadě možnost ”žádný”, pokud si u poznámky žádný typ nepřejde. Po typu je samotná poznámka. Poznámka se vyplňuje do textového pole. Toto pole je výjimečně přes celou obrazovku a pokud je prázdné, je zde informační text ”Sem pište poznámku”. Typ i samotná poznámka jsou volitelné atributy a změnu lze pořídit i pokud jsou prázdné.

4.4.11.5 Říje

Říje v hlavičce má informace o skupině, název a číslo, datum posledního otelení, datum poslední říje a datum poslední inseminace. Všechny tři data mají v závorce počet dní od daného data. Tělo navíc obsahuje pouze jediný atribut a to je výsledek. Výsledek je výběrový atribut. Pro výběr je uživateli zobrazen dialog. Může vybrat následující hodnoty: normální, tichá, vyvolaná, vitalimetr a pozorovaná.

4.4.11.6 Inseminace

V hlavičce změny inseminace je skupina, datum poslední inseminace, pořadí poslední inseminace, datum posledního otelení a pořadí laktace. U dat je vždy počet uběhnutých dní. V těle jsou následující položky: pořadí, charakter, býk, technik, dávka a sexované sperma.

Všechny položky jsou výběrové. Začneme charakterem. Pro charakter má uživatel možnost vybrat 4 položky a to inseminaci, reinseminaci, embryotransfer a přirozenou plemenitbu. Přirozená plemenitba je historický charakter a nově je nahrazena samostatnou změnou, proto do budoucna bude odstraněna.

Pořadí je výběrová položka. Po kliknutí je uživatel přesměrován na obrazovku, kde může vybrat hodnoty mezi 1 až 20. Zároveň mu však aplikace nabízí předpokládané pořadí inseminace

u daného zvířete. Předpokládané pořadí se vypočítává z aktuálního pořadí + 1. Toto předpokládané pořadí je v listu zvýrazněno šedivou barvou a je vždy na první pozici. Pokud je charakter inseminace reinseminace, potom je položka pořadí skryta a nastavena na nulovou hodnotu.

Další položkou je býk. Pokud si uživatel přeje zvolit býka, je přesměrován na obrazovku popsanou v sekci 4.4.8. Když má zvíře definované býky pomocí připařovacího plánu, potom jsou tito býci zobrazeni na prvních pozicích a jsou zvýrazněny.

Položka technika reprezentuje technika, který provedl inseminaci. Po kliknutí je uživatel přesměrován na list techniků. Kliknutím na technika zvolí příslušného technika. Pokud je uživateli předdefinován technik pomocí systému oprávnění, potom tato položka není zobrazena a pro inseminaci je automaticky nastaven definovaný technik.

Dávka představuje dávku spermatu podanou při inseminaci. Jedná se o desetinné číslo. Uživatel má možnost vybrat hodnoty 0, 0,25, 0,5, 0,75 a hodnoty celých čísel mezi 1 a 10. Výchozí hodnota je 1.

Sexované sperma reprezentuje pouze ano či ne. Hodnota je reprezentovaná pomocí přepínače, který uživatel může nastavit na požadovanou hodnotu.

Pokud uživatel vybere jako charakter hodnotu embryotransfer, potom je zobrazena položka matka. Zde by uživatel měl vyplnit identifikační číslo matky embrya. Je zde tlačítko pro výběr země a dvě textová pole. První představuje id kusu a druhé kodex. Do obou hodnot lze psát pouze čísla.

Pokud uživatel zakládá novou změnu, aplikace se mu sama pokusí nastavit nejlepší výchozí hodnoty. Pokud aplikace najde poslední inseminaci a ta není starší než 15 dní, potom automaticky nastaví charakter na reinseminace. Zároveň nastaví položku býk na býka použitého při této inseminaci. Pokud je poslední inseminace starší než 15 dní, potom jako výchozí charakter nastaví inseminaci. Pořadí se nastaví na předpokládané pořadí.

Pokud uživatel zakládá změnu inseminace a zároveň poslední změna otelení je starší více než 150 dní, pak je uživateli nabídnuta možnost pořídit i změnu otelení. Aplikace předpokládá, že nastala chyba v datech a změnu otelení do systému někdo zapomněl zadat. Pokud jsou podmínky splněny, potom se v těle inseminace objeví nová sekce otelení.

Vedle nadpisu otelení je přepínač, kterým si uživatel určí, zdali si přeje otelení zadávat. Sekce otelení obsahuje stejné položky jako samotná změna otelení, tedy datum otelení a pořadí laktace. Tyto položky však nejsou předvyplněny a uživatel je musí zadat.

Pokud si uživatel přeje změnu uložit, pak musí být vybrán býk inseminace. Pokud charakter není reinseminace, potom musí být i vybráno pořadí inseminace. Pokud je k dispozici sekce otelení a uživatel neodškrtl přepínač pro zadávání otelení, potom musí vyplnit obě položky otelení, jinak nedojde k uložení. Pokud jsou všechny podmínky pro uložení splněny, vznikne změna inseminace. Pokud byla k dispozici sekce otelení a uživatel jí zadal, potom vznikne i nová změna otelení.

4.4.11.7 Březost

Březost ukazuje v hlavičce informace o aktuální skupině, datum poslední inseminace a status březosti zvířete. V těle je výsledek březosti. Výsledek je výběrová hodnota 4 hodnot. Jsou to březí, jalová, otevřená a zmetání.

4.4.11.8 Otelení

Otelení obsahuje v hlavičce údaje o skupině. Tělo obsahuje pořadí laktace. Jedná se o výběrovou položku. Po kliknutí je uživatel přesměrován na obrazovku výběru pořadí. Aplikace nabízí uživateli předpokládané pořadí otelení. Předpokládané je aktuální pořadí otelení + 1. Tato hodnota je zároveň výchozí hodnotou pro nové změny otelení.

4.4.11.9 Zaprahnutí

Zaprahnutí je jednoduchá změna, která v hlavičce obsahuje navíc pouze informace o aktuální skupině zvířete. V těle je jenom datum a nejsou zde žádné speciální atributy.

4.4.11.10 Selekce

Hlavička změny selekce obsahuje navíc pouze informace o skupině. V těle změny selekce jsou následující položky: důvod, skupina, pořadí, jednotka a splněno.

Důvod je výběrová položka. Pro výběr je uživateli zobrazen dialog důvodu selekce. Důvody selekce jsou následující: inseminace, zjišťování březosti, řízená reprodukce, řízené zaprahování, veterinární ošetření, přesuny mezi skupinami, jatka, hromadné akce, úprava obojků/číslování, dojírna a ostatní. Skupina je výběrová položka určující, kam bylo zvíře během selekce přesunuto. Pro výběr je dostupný seznam skupin dostupných na aktuální stáji. Pořadí je výběrová položka s možností výběru hodnot od 0 do 5. Další položkou je jednotka. Jednotka je textové pole, kam uživatel může zadávat pouze kladná celá čísla.

Poslední položkou je splněno. S tou uživatel nemůže interagovat a je pouze pro čtení. Ukazuje, zdali došlo ke splnění selekce, například že zvíře bylo selektováno na selekční brance. Tuto položku nastavuje server.

Pro uložení změny selekce nejsou žádné omezení. Všechny položky jsou volitelné.

4.4.11.11 Umístění

V hlavičce u této změny uživatel nalezne aktuální skupinu zvířete. Tělo má navíc pouze jednu položku a to samotnou skupinu, kam je zvíře umístěno. Jedná se výběrovou položku. Po kliknutí je uživatel přesměrována na obrazovku výběru skupin.

Skupiny jsou zobrazeny v listu. U každá skupiny je na prvním řádku napsán její název. Druhý řádek obsahuje číslo. Pokud skupina nemá název, potom je zobrazeno pouze její číslo. Všechny skupiny jsou seřazeny abecedně a obrazovka podporuje textové vyhledávání. Uživatel aktivuje textové vyhledávání tažením prstem dolů. Vždy jsou zobrazeny pouze skupiny dostupné uživateli, tedy skupiny na aktuální stáji a nebo skupiny pouze na stájích, kam má uživatel přístup, pokud žádná aktivní stáj není vybraná. Stejná obrazovka je použita i při výběru skupiny ve změně narození a selekce.

Pro uložení změn je nutné, aby uživatel vybral skupinu. Pokud uživatel nevybere skupinu změnu nelze uložit a uživatel je na chybu upozorněn chybovou hláškou.

4.4.11.12 Paznehty

Hlavička změny paznehty je jednoduchá. Navíc obsahuje informace pouze o skupině. Oproti tomu tělo této změny je celkem komplexní. První položka je textová poznámka. Jedná se o textové pole. Uživatel není nijak limitován obsahem tohoto pole. Druhou položkou je typ. Typ je výběrová položka. Při kliknutí je uživateli zobrazen dialog s dvěma dostupnýma hodnotami. Jsou to FUP a kontrola. FUP je výchozí hodnota pro tento typ.

Třetí položkou je LCS. LCS může mít 6 stavů. Jsou to nehodnoceno a potom celá čísla od 1 do 5. Hodnota je napsaná ihned po nadpisu položky LCS. Příklad pro nehodnoceno je zde napsáno ”LCS - Nehodnoceno”. Pokud je položka nehodnocená, potom je přepínač v pravé části vypnutí a spodní bar pro nastavení hodnoty je zašedlý. Pokud si uživatel přeje LCS ohodnotit, potom nastaví požadovanou hodnotu tažením prstu po spodním baru. Nastavenou hodnotu opět vidí za pomlčkou. Pokud uživatel jakkoliv interaguje se spodním barem, potom je položka automaticky přepnuta do pozice ohodnoceno.

Tím jsme prošli položky týkajícího se celkového vyšetření. Následují položky se zaměřují na jednotlivé ošetřované končetiny. Uživatel vidí obrys zvířete. Kliknutím na jednu ze čtyř končetin vybere požadovanou končetinu a je přesměrován na její detail.

V horní navigační liště vidí název aktuálně ošetřované končetiny. První položka je přepínač označující, zdali je dané končetina ošetřena či nikoliv. Pokud je ošetřena, pak se zobrazí další položky. Další položky na této obrazovce jsou závažnost, ddscore, poznámka a tlačítko léčit.

Položka závažnost může mít 5 různých hodnot. Jsou to Nehodnoceno a čísla od 1 do 4. Nastavování probíhá stejně jako u LCS. Pouze rozsah hodnot je jiný. Podobně je tomu i u DDScore. To má hodnoty nehodnoceno, čísla od 1 do 4 a speciální hodnotu 4.1. Pro nastavení je opět použit stejný styl jako u LCS. Další položkou je poznámka. Poznámka je textové pole, kam uživatel může psát libovolnou poznámku k ošetření dané končetiny. Poslední položkou je tlačítko léčit. Po kliknutí na něj je uživatel přesměrován na obrazovku léčení dané končetiny.

Zde může vybrat diagnózy a léčiva, která byla podaná pro odstranění problému. Může vybrat libovolný počet diagnóz a léčiv. Vybrané položky mají šedivé pozadí. Po vrácení zpět na předchozí obrazovku je zde zobrazena nová sekce se záznamy léčby. Uživatel těchto záznamů může pořídit libovolné množství tlačítkem léčit.

Každý záznam obsahuje na prvním řádku končetinu, které léčení patří. Druhý řádek obsahuje zvolené diagnózy oddělené čárkou. Třetí řádek obsahuje podaná léčiva oddělená čárkou. Pokud si uživatel přeje záznam upravit, může tak učinit kliknutím na něj v listu. Pro smazání záznamu stačí dlouze podržet prst na záznamu. A poté potvrdit dialogové okno upozorňující na fakt, že se chystá daný záznam odstranit. Stejná varianta záznamů je i na hlavní obrazovce změny paznehtů. Zde jsou však záznamy na všech končetinách. Záznamy léčby v detailu končetiny zobrazují pouze záznamy spojené s danou končetinou.

Pokud uživatel pořizuje novou změnu a nastaví typ FUP, potom jsou automaticky všechny končetiny nastaveny na ošetřeno. Pokud nastaví Kontrola, tak jsou všechny nastaveny na neošetřeno. Ošetřené končetiny pozdá uživatel na hlavní obrazovce následujícím způsobem. V obrysu zvířete jsou ošetřené končetiny vykresleny červeně. Tím uživatel přehledně vidí, kde se provádělo ošetření paznehtů.

Pro uložení změny je potřeba kliknout na tlačítko pro uložení na hlavní obrazovce změny. Všechny položky jsou volitelné a pro uložení změny nejsou žádná omezení. Potvrzení není potřeba ani na detailu končetiny ani na obrazovce léčení končetiny. Zde jsou hodnoty ukládány automaticky, když uživatel odejde z obrazovky.

4.4.11.13 Nemoc dojírna

V hlavičce změny Nemoc dojírna uživatel najde skupinu zvířete. V těle změny jsou 3 textové položky. První je Blokace dojení. Druhá a třetí jsou Nemoci 1 a Nemoc 2, respektive kódy těchto nemocí. Všechny tři položky lze zadávat pouze jako kladná celá čísla.

4.4.11.14 Léčení

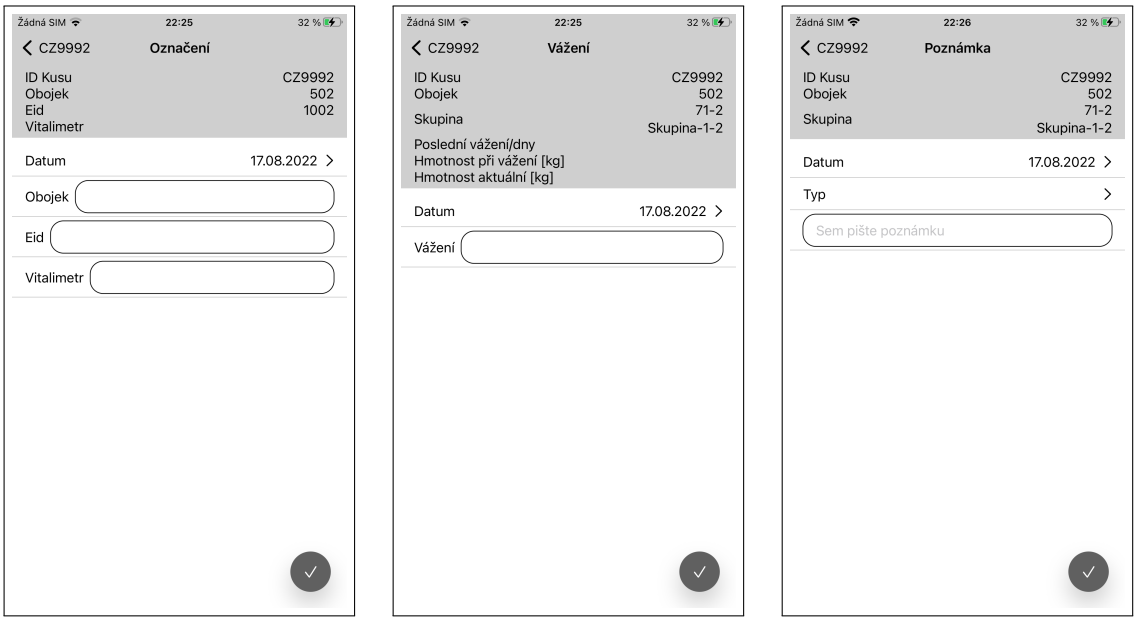
Léčení je jediná změna, kterou v aplikaci nelze zadávat. Je k dispozici tedy pouze pro čtení. Hlavička léčení obsahuje navíc pouze skupinu zvířete. Tělo je rozděleno na dvě sekce diagnozy a léčiva. Nadpisy sekcí mají šedivé pozadí.

V sekci Diagnózy uživatel nalezne diagnózy přiřazené k dané změně. Diagnózy jsou vždy předdefinované, popřípadě si je uživatel může definovat v PC aplikaci. U každé diagnózy vidí uživatel vždy pouze název diagnózy.

Sekce léčení je složitější. Každý záznam léčení obsahuje tři položky. První řádek obsahuje název podaného léku vlevo a vpravo je podané množství včetně jednotek. Pod názvem uživatel nalezne datum podání léku. Tento datum se může lišit od data změny léčení.

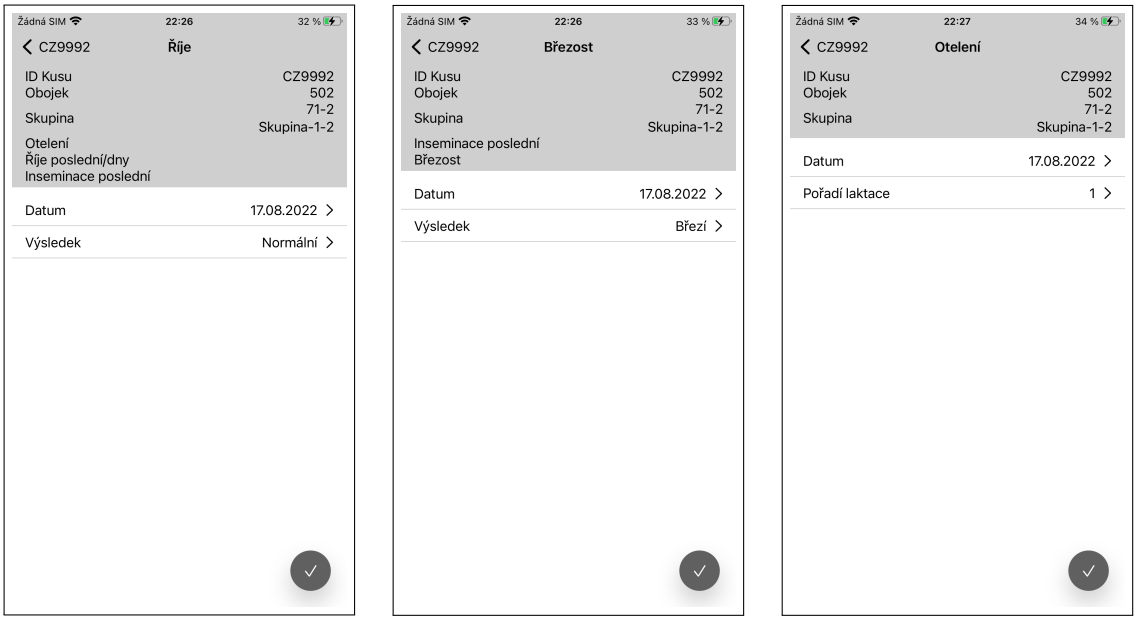
4.4.12 Sestava inseminací

Momentálně jediná dostupná sestava v aplikaci je sestava inseminací. Ta uživateli zobrazuje provedené inseminace za zadané období. Obrazovka nabízí možnosti pokročilého filtrování a



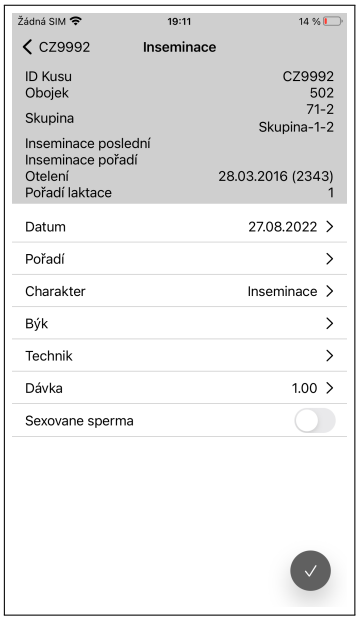
(a) Označení **(b)** Vážení **(c)** Poznámky

Obrázek 4.14 Označení/Vážení/Poznámky

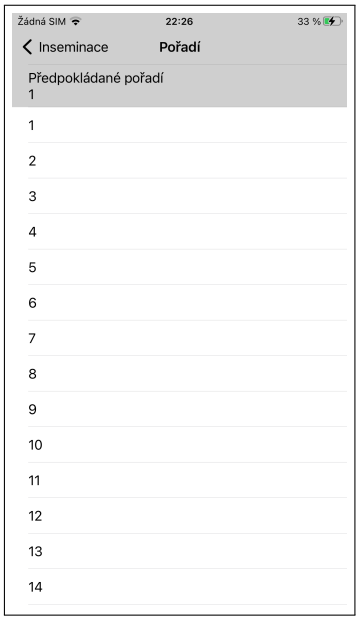


(a) Říje **(b)** Březost **(c)** Otelení

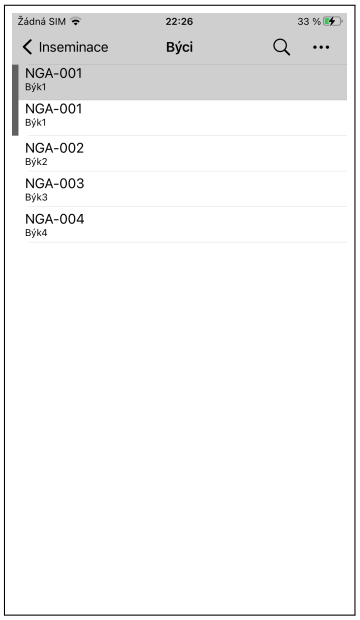
Obrázek 4.15 Říje/Březost/Otelení



(a) Inseminace

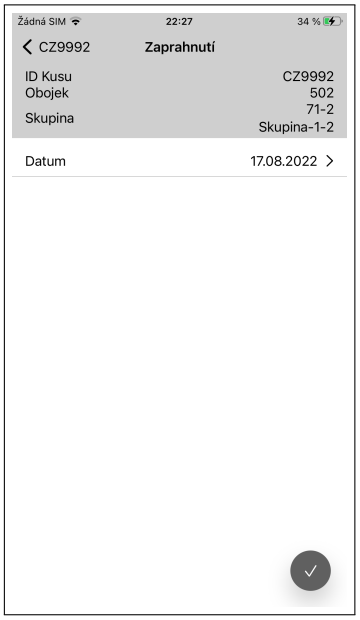


(b) Výběr pořadí inseminace

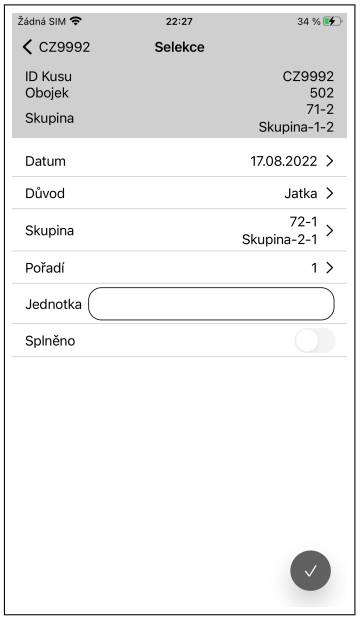


(c) Výběr inseminačního býka

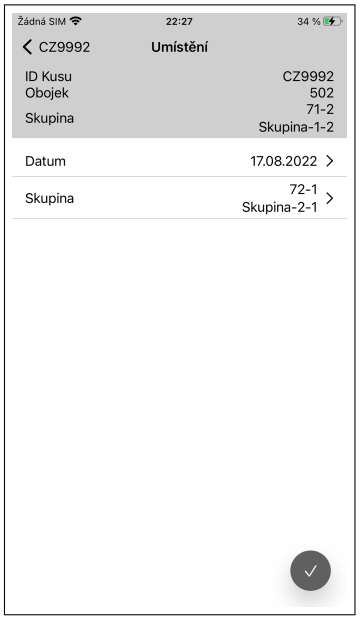
Obrázek 4.16 Inseminace



(a) Zaprahnutí

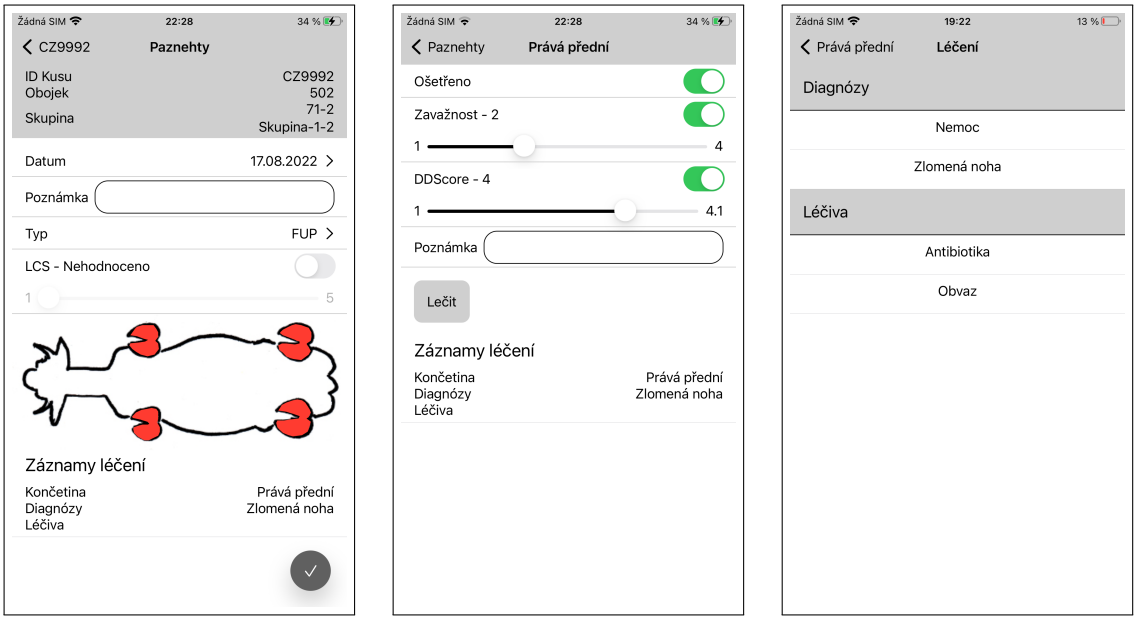


(b) Selekce



(c) Umístění

Obrázek 4.17 Zaprahnutí/Selekce/Umístění



(a) Paznehty **(b)** Obrazovka končetiny **(c)** Léčení končetiny

Obrázek 4.18 Paznehty

zobrazování sestavy. Dále je zde možnost exportu výsledné sestavy do formátu PDF a exportu pro Plemdat.

Obrazovka je list položek. První položka listu je vždy souhrn informací o aktuálně zobrazované sestavě. Tento souhrn má šedivé pozadí. Uživatel zde nalezne počet inseminací v aktuální sestavě a aktuální nastavení filtru sestavy.

Následují samotné položky inseminace. Každá položka má na prvním řádku číslo kusu, na kterém byla inseminace provedena. Následuje datum inseminace a stáj, kde byla provedena. Tato stáj se může lišit od aktuální stáje, ve které se zvíře momentálně nachází. Vpravo uživatel vidí základní informace o inseminaci. Jsou to charakter inseminace, pořadí a býk použitý pro inseminaci. Pří kliknutí na položku je uživatel přesměrován na detail změny inseminace.

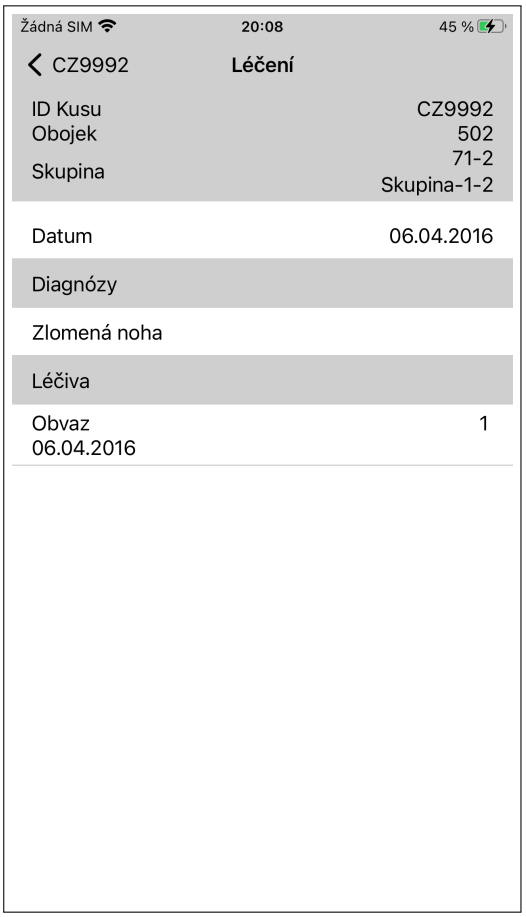
V navigační liště jsou dvě tlačítka. První je pro export aktuální sestavy a druhý je pro zobrazení filtru. Uživatel může filtrovat inseminaci za určité období. Potom mu jsou zobrazeny pouze inseminace vykonané v daném období. Může zvolit technika. V takovém případě mu aplikace ukáže pouze inseminace vykonané daným technikem. Poslední položkou filtrování je stáj. Uživatel si může vybrat N různých stájí. V sestavě se potom zobrazí pouze inseminace provedené na vybraných stájích.

Na obrazovce filtrování se nachází ještě položka ”Seskupit podle”. Ta slouží pro slučování jednotlivých inseminací v sestavě. Slučovat lze podle datu a podle stáje. Pokud uživatel seskupuje podle stáje, potom ve výsledném listu vidí inseminace provedené na jedné stájí u sebe. Pro seskupení podle data je situace obdobná. Pokud uživatel neaplikuje žádné seskupení, potom jsou položky seřazeny podle data inseminace od nejstarších po nejnovější.

Po kliknutí na tlačítko exportovat je uživatel přesměrován na obrazovku exportu. Zde vidí možné varianty exportu. Momentálně jsou zde dvě a to PDF a Plemdat. Při kliknutí na PDF aplikace vytvoří PDF dokument ekvivalentní aktuální sestavě. Dokument můžete vidět na obrázku 4.21. Po úspěšném exportu aplikace uživatele přesměruje zpět na obrazovku sestavy a zobrazí mu systémový dialog pro sdílení souborů. V dialogu se uživatel může rozhodnout o dalších akcích s exportovaným souborem.

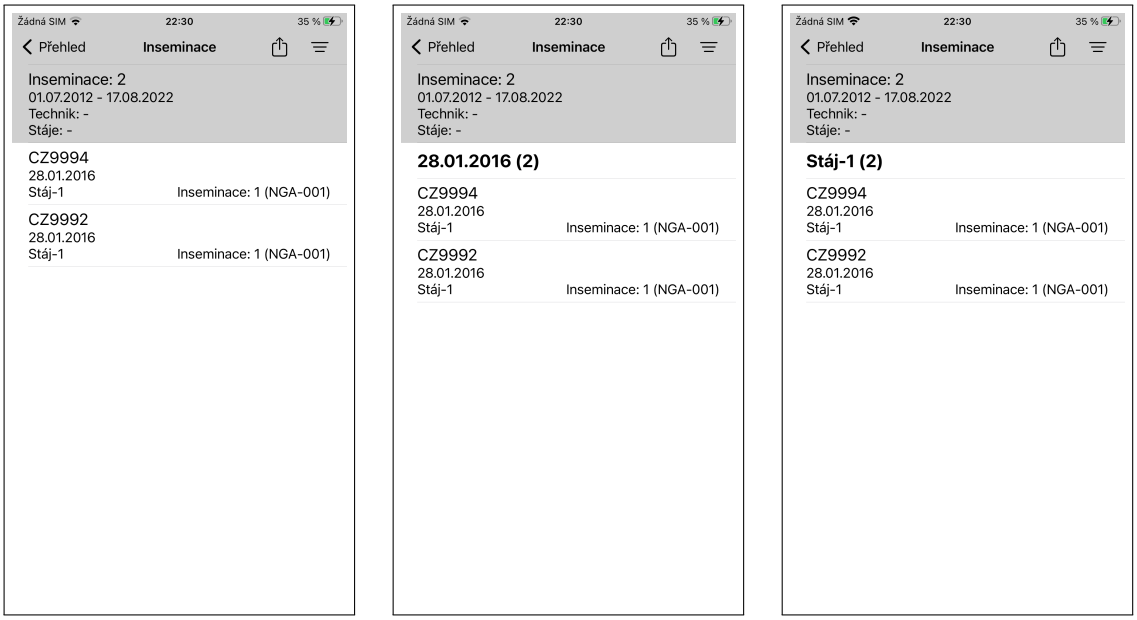
Při exportu pro plemdat je situace složitější. Export probíhá podle specifikace plemdatu. Pro

(a) Nemoc dojírna



(b) Léčení

Obrázek 4.19 Nemoc dojírna/Léčení



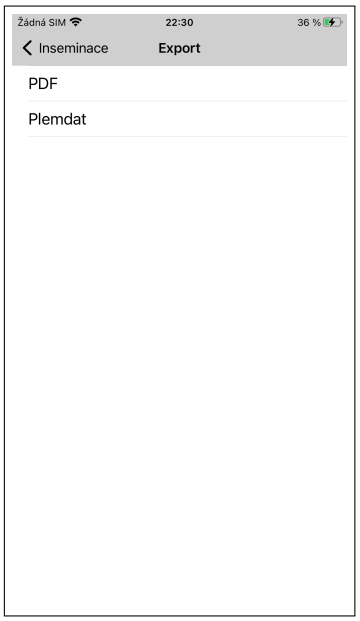
(a) Bez seskupení **(b)** Seskupení podle data **(c)** Seskupení podle stáje

Obrázek 4.20 Sestava inseminace

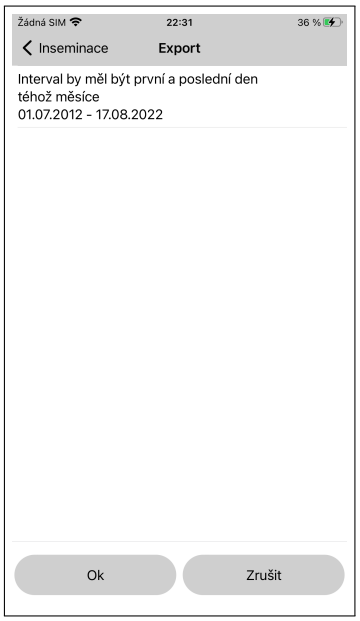
export potřebuje aplikace znát identifikátor organizace. Ten se nastavuje v nastavení aplikace popisované v částí 4.4.4. Pokud uživatel identifikátor nevyplnil, je přesměrován na obrazovku nastavení identifikátoru a je mu zobrazen dialog s chybovou hláškou. Po vyplnění identifikátoru je identifikátor vidět na obrazovce možných exportů u položky plemdatu.

Pokud uživatel vybere export pro Plemdat a má vyplněný identifikátor, potom aplikace může zobrazit obrazovku s varování. Obrazovka se zobrazí pouze v případě, pokud si aplikace myslí, že export pro Plemdat je špatně nastaven. Aplikace může vygenerovat dva typy varování. První je, pokud interval sestavy není nastaven od prvního do posledního dne téhož měsíce. Druhý typ je, pokud aplikace nedovede převést registrační číslo býka na formát požadovaný systémem Plemdat. Pokud je varování zobrazeno, uživatel má možnost kliknout na Ok a tím je mu soubor vygenerován bez ohledu na varování. Druhou možností je tlačítko zrušit, kterým se vrátí zpět na obrazovku sestavy inseminace.

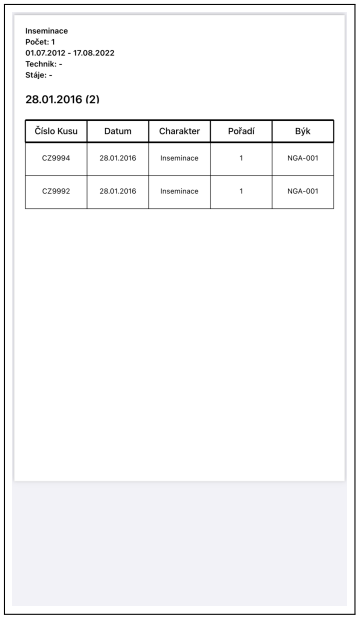
Při exportu pro Plemdat aplikace vygeneruje textový soubor ve formátu .ins. Název soubrou je SHIXX_YY_mes.ins. XX je datum uzavření exportu. Datum uzavření je datum nastavení v položce ”do”při filtrování sestavy. YY je identifikační číslo organizace. V samotném souboru je na prvním řádku uložena hlavička. Hlavička obsahuje identifikační číslo organizace, datum uzavření exportu, počet inseminací a čas vygenerování. Po hlavičce následují samotné inseminační záznamy. Pokud aplikace nedokáže záznam převést na požadovaný formát, potom se tato položka do výsledného vygenerovaného exportu nedostane. Celý postup generování odpovídá specifikaci. Po vygenerování souboru je uživatel přesměrován na obrazovku sestavy a je mu zobrazen systémový dialog pro sdílení souboru.



(a) Možnosti exportu



(b) Varování při exportu



(c) Výsledný export do formátu PDF

Obrázek 4.21 Export sestavy inseminace

Kapitola 5
Testování

Testování softwarových aplikací se typicky rozděluje na několik částí. Můžeme použít unit testy, které vytváří programátor a tím testuje kód na úrovni funkcí. Dále můžeme využít funkční testy, kterými tester ověří funkční aplikace jako celku. A v neposlední řadě uživatelské nebo akceptační testy, kterými ověříme, že výsledná aplikace splňuje požadavky uživatelů, kteří jí budou používat.

Software Farmsoft je software dělaný na míru pro použití v zemědělství. Nemá se jedna o masově používaný software, který budou používat miliony uživatelů. Uživatelů budou desítky, nebo spíše nižší stovky. Uživatelé jsou typicky starší lidé, kteří umí svoji práci vykonávat i bez moderních vymožeností. Farmsoft jim jejich práci však velice usnadňuje a za roky jeho používání si na něj velice zvykli. Někteří si již bez něj nedovedou svoji práci představit a používají ho téměř každý den.

Nyní bych rád popsal, jak jsem postupoval při testování iOS aplikace pro Farmsoft, tak aby se uživatelé na ni stále mohli spolehnout a rádi jí používali.

5.1 Testování návrhu

I když se může zdát, že návrh a analýza iOS aplikace vznikl přes noc, není tomu tak. Při návrhu bylo využito téměř 5 let vývoje Android aplikace. U Androidu vývoj probíhá iterativně. Vzhledem k tomu, že Android verzi vyvíjím pouze sám, můžu si určovat velikost jednotlivých cyklů mezi vydáními nové verze podle potřeby. Typicky jsem při přidání nové funkcionality vydal novou verzi.

Přidání funkce vždy inicializuje pan Smolík. Panu Smolíkovi typicky zavolá uživatel s požadavkem na přidání funkce. Ten následně rozhodne, zdali se bude funkce implementovat a zdali je nutné jí přidat i do mobilní aplikace. Následuje konzultace se mnou, kde rozhodneme, jak by daná funkcionalita měla v mobilu vypadat. Po její implementaci je verze poskytnuta panu Smolíkovi. Pokud se mu provedení líbí, verze jde mezi uživatele. V případě problému se verze vrátí k přepracování.

Když se nová verze dostane mezi lidi, proces nekončí. Typicky se kontaktuje uživatel, který požadoval funkci. Ten dostane pár dní na použití v ostrém provozu a pokud má výhrady, je návrh upraven.

Návrh Android verze aplikace vznikl z velké části právě podle potřeb uživatelů, kteří aplikaci používají. A díky tomu jsem měl téměř jistotu, že pokud dodržím stejné principy i na iOS a návrh iOS aplikace bude vycházet z Android verze, tak uživatelé budou mít to, co potřebují.

Rád bych zde zmínil člověka, který nám pří implementaci nových funkcí byl asi nejvíce nápomocen. Jedná se o pan Stanislava Skalického. Pan Skalický je bývalý inseminátor pracující pro Reprogen a.s. Reprogen a.s je jeden z hlavních zákazníků využívající systém Rskot. Pan Skalický

se nyní stal hlavním administrátorem aplikace Rskot a téměř veškeré nové funkcionality jsou na jeho vyžádání.

Chtěl bych zde vypíchnout několik funkcí, které byly implementovány přímo na požadavek pana Skalického a postupným iterativním vývojem se dostaly do fáze, kdy mu naprosto vyhovovaly.

První funkce byla automatické přednastavování charakteru inseminace na reinseminaci, pokud poslední inseminace byla provedena před méně než 15 dni. Dále pak funkce pro predikování pořadí inseminace, jenž se později použila i při pořizování otelení. Přibila možnost zadávat otelení při zadávání nové inseminace. Prakticky celý návrh systému oprávnění jak v Rskotu tak v Farmsoftu vychází z jeho potřeb jakožto uživatele aplikace. Nápad na čtečku QR kódu z karet zvířat je taky nápad od pana Skalického.

Poslední položkou, kterou bych rád zmínil, je sestava inseminací. Její aktuální podoba vychází téměř celá z potřeb uživatelů Rskotu v čele s panem Skalickým. Původní návrh sestavy obsahoval pouze filtrování podle data. Ostatní položky filtrování a exportování byly dodefinovány časem zákazníkem.

Samozřejmě pan Skalický není jediný z uživatelů, kdo svými poznámkami přispěl k vývoji funkcí ve Farmsotu. Například funkce propojení mobilní verze s bluetooth zařízeními popisované v sekci 2.3.9 vznikla z potřeb jednoho z podniků Farmsoftu. Při návrhu této funkce jsme s panem Smolíkem jeli přímo do tohoto podniku na prezentaci mojí implementace v Android. Podnik tuto funkci s bluetooth využívá a do budoucna se plánuje její implementace i pro iOS aplikaci.

Android verze je prověřená přímo z praxe uživateli a její návrh odpovídá očekávání systému Farmsoft. Uživatelé Android verzi používají téměř každý den. Díky všem těmto faktů jsem věděl, že pokud iOS verze bude funkcemi odpovídat Android verzi, potom bude stejně dobrá jako verze pro Android. Při návrhu iOS aplikace jsem tedy využil znalostí získaných z Android verze. Díky tomu jsem byl schopný vytvořit návrh iOS aplikace podobné již ověřené verzi pro Android.

5.2 Testování aplikace

Samozřejmě dobrý návrh pro kvalitní aplikaci nestačí. Je nutné tento návrh správně realizovat. Testování probíhalo v několika vlnách. Při první prezentaci panu Smolíkovi byl použit prototyp aplikace, který nebyl ještě napojený na server. Testování probíhalo spíše na uživatelské úrovni, kdy jsme hodnotili, zdali iOS verze odpovídá Android verzi.

Po prvním kole testování došlo k napojení aplikace na server. Pro napojení jsem použil aplikaci prezentovanou v této práci, kterou jsem následně rozšířil o napojení na API Farmsoft.

Při napojení na server jsem však při lokálním testování narazil na zajímavý problém, který bych rád zmínil. Jelikož aplikace při synchronizaci pracuje s velkým množstvím dat, zvláště u Rskotu, kde mluvíme o stovkách MB, bylo potřeba při synchronizaci otestovat i práci aplikace s pamětí.

Při tomto testování jsem zjistil, že aplikaci občas při synchronizaci dochází paměť. Toto chování bylo opravdu zvláštní, protože veškerá implementace byla správná. Soubor byl načítán postupně. Při postupném načítání ze souboru byl vytvořen list položek o maximální velikosti 1024 položek. A tyto položky byly následně v transakci ukládány do lokální databáze zařízení pomocí CoreData. Po úspěšném zpracování transakce byl list vymazán a proces se opakoval. Implementace se zdála správná, ale paměť docházela. Po několika dnech hledání chyby a ujištění se, že k memory leakům nedochází v mé části kódu, jsem konečně našel řešení. Problém vyřešil `autoreleasepool`.

Swift sice implementuje vlastní správu paměti a tato implementace byla schopná alokovanou paměť po synchronizaci uvolnit, ale během synchronizace jí bez použití `autoreleasepool` uvolnit nedokázal. To se může stát, pokud ve Swiftu použijeme staré Objective-C. Já jsem sice Objective-C nikde v mém kódu nepoužil, ale CoreData jsou tímto jazykem napsaná. Po vyřešení tohoto problému jsme mohli přejít k dalšímu testování aplikace.

V druhém kole byla aplikace již napojena na server. V této verzi už jsme ladili pouze finální úpravy. Především jsme zde sjednocovali texty iOS a Android verze. Po validaci druhé verze jsme se rozhodli aplikaci nahrát na AppStore a dát jí k otestování zákazníkům. Testování probíhalo pomocí TestFlightu dostupného v App Store Connect.

Aplikace byla poskytnuta panu Skalickému. Pan Skalický prošel základní funkcionalitu. V té době však již aktivně neinseminoval, a proto aplikaci předal jednomu uživateli na otestování přímo v terénu. Tento uživatel aktivně využíval Android verzi Rskotu a přechod na iOS verzi proběhl bez problému. Aplikaci používal v reálném provozu tak, jak byl zvyklý používat verzi s Androidem. Uživatel mi po 14 dnech poslal přes pana Skalického výsledek testování.

Testovací uživatel byl z aplikace nadšený a konstatoval, že základní funkce fungují výborně. Líbila se mu rychlost a plynulost aplikace. Z testování vyplynulo, že základní funkce aplikace, tedy přihlášení, pořizování změn a synchronizace se serverem, fungují. Uživatel v aplikaci nalezl pouze drobné chyby, typicky špatné textace. Tyto chyby byly následně opraveny a aplikace mohla přejít do finální produkce.

5.2.1 Aplikace v produkci

Jakmile aplikace byla vydaná mezi uživatele, bylo nutné kontrolovat stabilitu a kvalitu aplikace i pro další vydání. Obvyklý postup je, že se aplikace před vydáním nové verze celá znovu otestuje testerem. Bohužel z kapacitních důvodu v týmu neexistuje žádný člověk zaměřený pouze na testování. Toto testování bych tedy musel dělat já osobně a to z časových důvodů není možné.

Proto jsem se rozhodl při každém vydání nové verze testovat pouze nejdůležitější funkce aplikace. Za nejdůležitější funkce aplikace považuji přihlášení, pořízení změny a synchronizace pořízených změn se serverem. Pro toto testování jsem sestavil následující testovací scénář.

1. Spustit aplikaci a přihlásit se pod testovacím účtem.
2. Aktivovat účet a spustit synchronizaci.
3. Po synchronizaci v nastavení povolit odesílání změn.
4. Vrátit se zpět na přehled a kliknout na stádo.
5. Ve stádu vyhledat libovolnou samici.
6. U vybraného zvířete pořídit všechny dostupné změny.
7. Po zadání všech změn se vrátit na úvodní obrazovku a synchronizovat aplikaci
8. Pokud synchronizace projde, potom je test dokončen.

Tento scénář není dokonalý a nemá šanci najít všechny chyby. Jeho vyplnění však zabere maximálně 10 minut. Díky němu zajistím základní funkcionalitu aplikace a jeho vyplnění mě nestojí příliš mnoho času, který potom můžu investovat do dalšího vývoje.

Samozřejmě občas nějaká chyba pronikne až do produkčního prostředí. V takovém případě jsou dva způsoby odhalení. Pokud aplikace padá, potom tyto pády vidím v Firebase consoli v sekci Crashlytics. Díky tomu jsem schopný odhalit pády aplikace a z logů následně vyčíst jejich příčinu a tu opravit.

Druhou možnou chybou je chování aplikace, které nezpůsobí pád. Takové chyby, pokud nejsou odhaleny během vývoje, musí nahlásit až uživatel. Uživatelů je poměrně málo a pan Smolík zná velkou část z nich osobně. Právě díky této osobní známosti nám uživatelé chybu typicky sami nahlásí. Pokud se chyba týká mobilní aplikace, potom mi jí pan Smolík popíše a chyba je opravena. Pokud toto není možné, pak dostanu přímo kontakt na uživatele a s jeho popisem je chyba následně opravena. Veškerá komunikace s uživateli probíhá pomocí telefonních hovorů.

Momentálně se tedy uživatelé používají jako testeři, kteří odhalí skutečně všechny chyba a ty nám nahlašují. I když to určitě není správný způsob, jak vytvářet aplikace, tak nám se osvědčil. Díky malému počtu uživatelů se chyby v aplikaci projeví jenom jedincům. Samozřejmě pokud chyba znemožní používání aplikace, potom musí být opravena a to se typicky děje do dalšího pracovního dne.

Tímto postupem se nám podařilo snížit náklady na vývoj, kdy není potřeba platit dalšího člověka. Toto snížení je sice za cenu zhoršení kvality aplikace, čehož jsme si vědomi, ale tento postup se nám vyplácí.

Do budoucna je však tento postup neudržitelný a bude potřeba najít člověka, který bude aplikace kontrolovat. Taky bude potřeba v aplikaci doplnit automatické testy. iOS aplikace je pro automatické testování připravena a od začátku byla designována, aby šla automaticky testovat pomocí Unit testů.

Všechny třídy zásadně komunikují pomocí rozhraní, což je za mě základní podmínka pro možné unit testování. Potom jsou všechny závislosti vždy vkládány pomocí konstruktoru třídy. Díky tomu lze třídě podstrčit mockovanou závislost. Tyto dvě nejdůležitější podmínky pro testování jsou tedy splněny a tvorbě automatických Unit testů by nic bránit nemělo. Momentálně však unit testy nejsou implementované. Do budoucna však jejich implementaci určitě plánuji.

Kapitola 6

Automatické nasazení na AppStore

Nyní bych rád představil možnosti automatického nasazení aplikace do AppStore. Momentálně, pokud jsem chtěl udělat deploy aplikace a nahrát její novou verzi na AppStore, musel jsem provést následující kroky, zvednou číslo verze aplikace, v rozhraní Xcode kliknout na "archivovat", čím se vytvořil archiv připravený pro nahrání do AppStore. Po doběhnutí buildu se v Xcode zobrazí tabulka s možností distribuce do AppStore. Po kliknutí na tu možnost je spuštěn průvodce, který programátora provede nahrávacím procesem. Po dokončení je archiv nahrán do AppStore. Tam jí zkontroluje automatická kontrola a potom je aplikace připravena k distribuci. Pro distribuci musí v AppStoreConnect programátor kliknout na tlačítko plusu pro přidání nové verze, popsat změny v této verzi a přiřadit jí nahraný balíček s aplikací.

Jak je vidět, proces je celkem náročný a musí se opakovat dvakrát, jednou pro Farmsoft a podruhé pro Rskot. Proto přišla myšlenka tento proces alespoň částečně automatizovat. Automatizaci jsem rozdělil na dvě části, automatizace samotného sestavení a nahrání aplikace do AppStore a automatické spuštění toho procesu při nahrání aplikaci do gitu.

6.1 Automatizace sestavení a nahrání aplikace

Pokud chceme sestavit aplikaci pomocí příkazů příkazové řádky, můžeme k tomu použít nativní metodu. Pro sestavení iOS aplikace nativní metodou můžeme využít program xcodebuild. Tento program využívají na pozadí i samotné Xcode pro sestavování aplikace. [33]

Xcodebuild nám umožní pouze sestavení aplikace, pro nahrávání aplikace do AppStore potřebujeme další programy jako jsou altool, agvtool, codesign a security. Jak je vidět, programů je hodně a jejich naučení by zabralo velké množství času. [33]

Místo nativního řešení jsem se tedy rozhodl použit Fastlane. Fastlane je nástroj pro automatický deploy aplikací. Podporuje iOS i Android. Pro podporované platformy umí aplikaci sestavit a nahrát jí do požadovaného obchodu. Dokonce umí i vygenerovat screenshoty obrazovek aplikace. Tuto možnost jsem však nezkoušel. [34]

Sestavení pro Android zde probírat nebudu a zaměřím se pouze na sestavení pro iOS. Nejdříve je nutné program nainstalovt. Po instalaci je nutné v kořenovém adresáři projektu spustit příkaz `fastlane init`. Tím se vytvoří všechny nutné soubory pro fungování fastlane v projektu.

Příkaz nám v projektu vytvoří složku fastlane. Ta obsahuje několik souborů. Mezi ty důležité patří Appfile a Fastfile. Jsou to textové konfigurační soubory. V souboru Appfile jsou konfigurovány proměnné nutné pro komunikaci s Apple. Jsou to email AppleId účtu, který má přístup k vývojářskému účtu a dále id týmu, pod kterým bude aplikace nahrána.

Výpis kódu 6.1 Část souboru Fastfile pro sestavení aplikace

```
platform :ios do
  desc "Push a new beta build to TestFlight"
  lane :upload_to_tf do
    get_certificates           # invokes cert
    get_provisioning_profile   # invokes sigh
    build_app(workspace: "Farmsoft.xcworkspace", scheme: ENV['SCHEME'])
    upload_to_testflight
  end
end
```

Fastfile je soubor pro konfiguraci samotného procesu. V tomto souboru si můžeme definovat libovolný fastlane script pomocí fastlane příkazů. Mezi příkazy patří například ”build_app”pro sestavení aplikace, či ”upload_to_testflight”pro nahrání aplikace do Testflight.

Fastfile, které byly použity pro sestavení tohoto projektu, vidíte zde: 6.1. První řádek je pouze určení platformy. Druhý řádek je popis skriptu. Zajímavý je třetí řádek, kde začíná definice našeho skriptu. Obsahuje název skriptu, který je potom použit pro spuštění v programu fastlane. V tomto případě je název skriptu ”upload_to_tf”.

Samotný skript je tvořen 4 fastline příkazy. První dva jsou pouze pro získání certifikátů z provázaného účtu definovaného v Appfile. Třetí příkaz slouží k sestavení samotné aplikace. Příkaz má několik argumentů. Workspace je projekt, který si přejeme sestavit. Schema je target tohoto projektu. V mém případě je zde ”ENV['SCHEME']”. Tím fastlane dáme najevo, že si přejeme tuto hodnotu získat z proměnné prostředí. Více si o tom povíme později. Posledním příkazem v skriptu je ”upload_to_testflight”. Jak název napovídá, tento příkaz nahraje sestavenou aplikaci do testflightu.

Ještě bych se rád věnoval proměnným v fastlane. Fastlane nám nabízí možnost spustit jeden příkaz s různými proměnnými. Proměnné se definují v textových souborech s názvem ”.env.PROSTREDI”. Tyto souboru musí být ve složce fastlane. Proměnné v těchto souborech se definují ve formátu NÁZEV=HODNOTA, podobně jako tomu je například v prostředí shell. Když si přejeme použít takto definované proměnné, přidáme k fastlane příkazu `--env PROSTREDI`. Tím fastlane řekneme, kterém proměnné si přejeme využít.

Pro sestavení a nahrání aplikace na testflight nám tedy stačí následující příkazy. Pro Farmsoft je to `fastlane upload_to_tf --env Farmsoft` a pro Rskot `fastlane upload_to_tf --env Rskot`. Po provedení příkazu je daná verze sestavena a nahrána na testflight. Zde je nutné jí manuálně přidat k požadovanému vydání, doplnit changelog a aplikace je připravena k produkčnímu vydání.

6.2 Automatizace spuštění procesu po nahrání na git

Mít jeden příkaz pro sestavení aplikace se může zdát jako dostačující. Pro plné pohodlí během vývoje jsem však chtěl mít možnost automatického spuštění tohoto procesu při nahrání verze do gitu.

Samozřejmě můžeme použít prehooks, které jsou k dispozici přímo gitu. Ty se však spouští pouze lokálně a výstup těchto operací se nikam automaticky nezaznamenává. Já jsem se rozhodl použít Gitlab CI/CD.

Toto není přímo funkce gitu, ale aplikace postavené nad gitem. V našem případě se jedná o Gitlab. Gitlab umožňuje spouštění různých skriptů na základě různých událostí jako je například nahrání nové verze, vytvoření tagu atd. [35]

Gitlab CI/CD se skládá z několika základních komponent. Nejednoduší je Job. Job si lze představit jako nejmenší jednotku. Pro každý Job můžeme definovat skript, který si chceme spustit a za jakých okolností se má Job zapnout. [35]

Další je Stage. Stage může seskupovat libovolný počet Jobů. Poslední je Piplene. Pipeline je několik Stagí obsahující různé Joby. Zároveň Pipline definuje, v jakém pořadí se mají jednotlivé Stage spouštět. Normálně všechny Joby v dané Stagi čekají, než se dokončí předchozí Stage, respektive všechny Joby v této Stagi. Toto chování však lze upravit. [35]

Všechny spuštěné Pipliny jsou vidět přímo ve webovém prohlížeči v rozhraní Gitlabu v záložce CI/CD. Ukázku vidíme na obrázku 6.1. Je zde vidět stav jednotlivých Pipeline a jestli byly jednotlivé Stage úspěšně dokončeny. Kliknutím na status zobrazíme detail Pipeliny.

Detail vidíme na obrázku 6.2. Momentálně moje Pipline obsahuje dva Joby, jeden pro nahrání Farmsoftu do TestFligthu a druhý pro nahrání Rskotu.

Kliknutím na jednotlivé Joby zobrazíme log. To vidíme na obrázku 6.3. Logy jsou barevně rozlišeny. Vidíme zde runnera, který script provedl a další důležité informace pro ladění.

Již jsme si popsali jednotlivé části, které nám Gitlab CI/CD poskytuje a nyní si popíšeme konfiguraci samotné Pipliny. Ke konfiguraci se používá soubor s názvem ".gitlab-ci.yml". Pokud Gitlab najde tento soubor, nakonfiguruje nám Pipliny podle definic v tomto souboru. [35]

Konfigurace je opravdu rozsáhlá. Definujeme, co vše naše Pipline potřebuje pro běh, prostředí pro běh Pipliny, proměnné, Stage, Joby atd... Mnou vytvořenou konfiguraci vidíme ve výpisu kódů 6.2.

Moje konfigurace definuje jednu Stage "deploy". Tato Stage obsahuje dva Joby, "farmsoft_TF_upload_job"a "rskot_TF_upload_job". Oba se spouští podle pravidla definovaného v ".run_when_tag_is_created". Toto pravidlo říká, že Job je spuštěn při tvorbě nového tagu. A konečně u každého Jobu je příkaz, který se má provést. V obou případech je to nám známý příkaz pro sestavení a nahrání aplikace popsaný v sekci 6.1.

Tím bych uzavřel popis konfigurace Pipliny. Nyní tedy víme, co Pipline dělá, ještě je třeba říct, kde se provádí. Gitlab k běhu Pipliny využívá takzvaného Gitlab runnera. Jedná se o program, který managuje běh Pipline. [36]

Program nám umožňuje instalovat "runnera". Runner je prostředí, kde běží skript jednotlivých Jobů. Může to být terminál počítače, na kterým je Gitlab runner nainstalován, popřípadě to může být úplně jiný počítač připojený pomocí ssh, anebo to může být docker image. Možností je hodně. Gitlab runner může mít v jednu chvíli nainstalovaných několik runnerů různého typu.

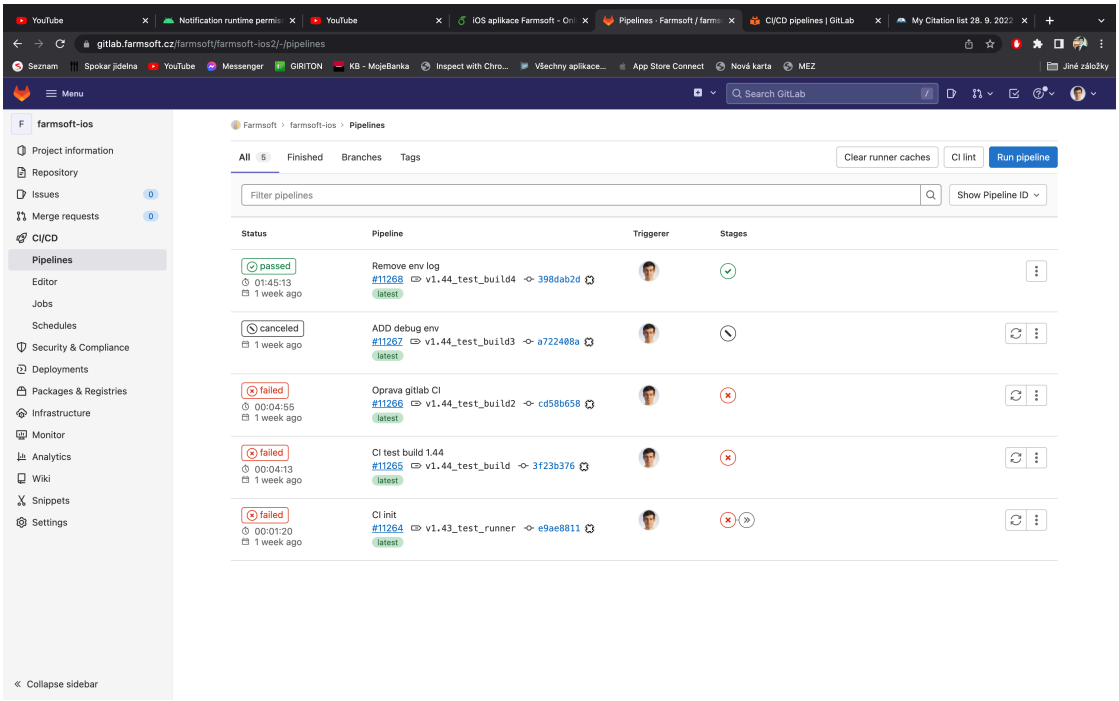
Při instalaci je nutné runnera registrovat do Gitlabu. Registraci lze provést globálně. Potom bude runner dostupný pro všechny projekty. Pro toto nastavení jsem však neměl oprávnění. Druhá možnost je registrace runnera pouze pro jeden projekt. Já si zvolil druhou možnost.

Registrace probíhá zadáním url adresy Gitlabu a tokenu. Oba tyto údaje jsou zadávány při instalaci runnera do programu Gitlab runner. Token je vždy unikátní pro každý projekt a lze jej najít v nastavení projektu ve webovém rozhraní Gitlabu.

Runner pro kompilaci mého skriptu potřebuje přístup k programu fastlane a Xcode. Proto jsem se rozhodl jej spouštět přímo v terminálu mého MacBooku, kde mám všechny tyto věci nainstalované. Má to však nevýhodu a to, že pokud běží nějaká Pipline, tak můj MacBook musí být zapnutý, dokud Pipline nedoběhne a zároveň během běhu runneru je značně zatížen. Lepší řešení by bylo instalace runneru na stále dostupný server. Bohužel momentálně nemám přístup k žádnému takovému stroji. Výhodou však je, že runner je spuštěn automaticky při startu systému.

Ve finále práce s CI vypadá následovně. S každou verzí tagu odeslaného do gitu se spustí CI na runneru v mém počítači, který zkompiluje aplikaci a výsledek pošle do TestFlightu. Došlo k drobnému zrychlení nahrávání, jelikož se kompiluje a nahrává Farmsoft a Rskot současně. Kromě toho již nemusím manuálně proklikávat průvodce nahrávání v Xcode a ani nemusím manuálně spouštět skript pro nahrání. Stačí pouze vytvořit tag v gitu a vše je provedeno automaticky, zatímco já se můžu věnovat jiné činnosti.

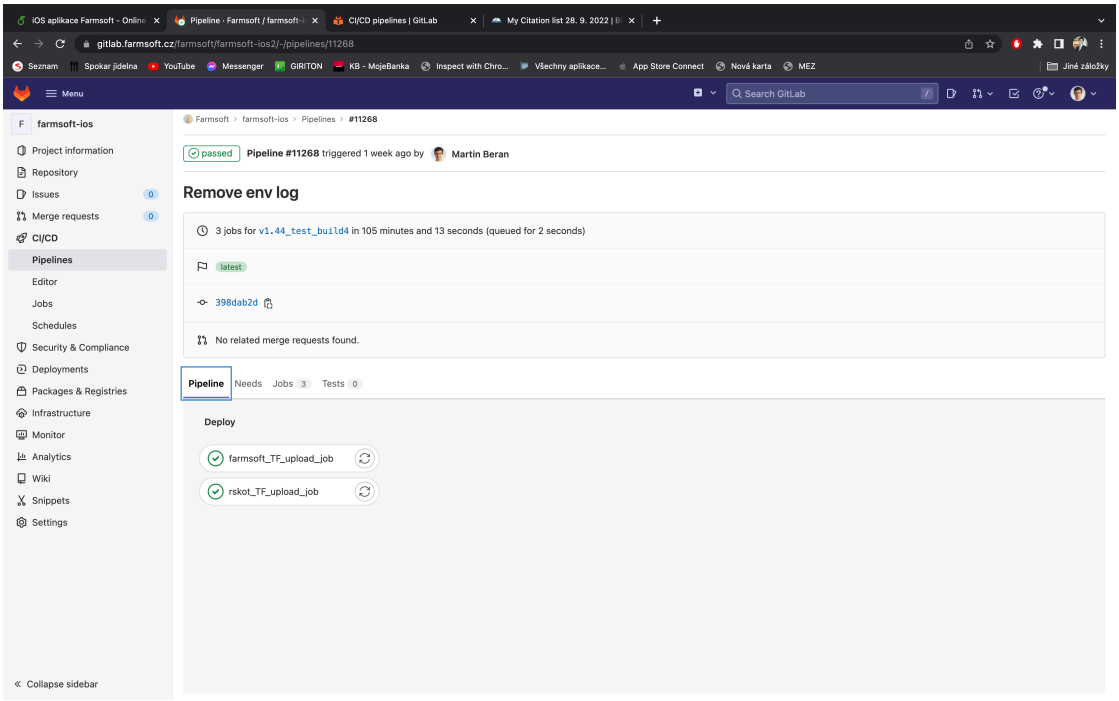
Na závěr bych se ještě rád věnoval alternativám pro Gitlab CI/CD. Samozřejmě existuje velké množství alternativ, které lze využít. Všechny nabízí ve směs to samé a to spuštění nějakého skriptu na nějakém zařízení. Výhoda Gitlab CI/CD je, že využíváme Gitlab pro ukládání kódů a Gitlab CI/CD je v něm přímo integrovaný. Navíc v provedení, které jsem uskutečnil já, je úplně zdarma.



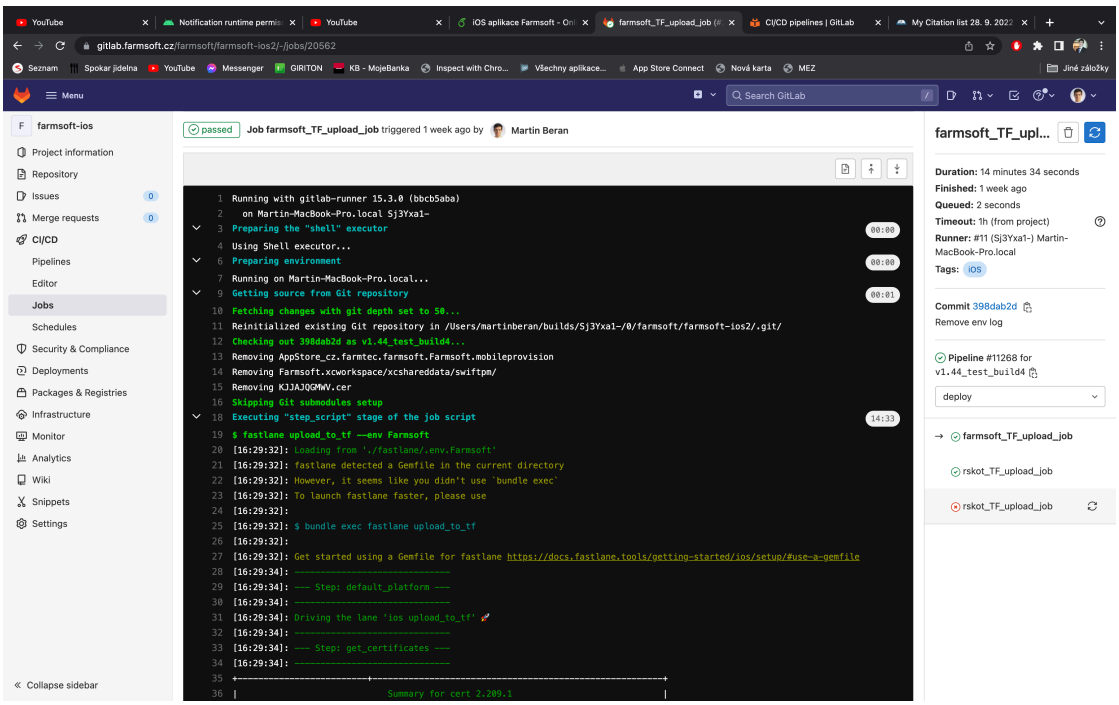
Obrázek 6.1 Přehled všech spuštěných Pipline

Alternativou k tomuto řešení je Azure DevOps od Microsoftu. V základu nabízí dost podobnou funkcionalitu jako Gitlab CI/CD. Na rozdíl od Gitlab CI/CD konfigurace neprobíhá pomocí konfiguračního souboru, ale pomocí uživatelského rozhraní, kde si uživatel definuje své Pipliny přímo pomocí webového editoru. To je výhodné, pokud se člověk nechce učit jak konfigurovat Gitlab CI/CD. Taktéž nabízí velice dobré napojení na ostatní služby od Microsoftu jako například Azure Portal.

Momentálně však pro nás žádné jeho výhody nepřináší dostatečný výnos a osobně budu preferovat využití Gitlab CI/CD hlavně díky jeho integraci do Gitlabu, a tudíž možnosti mít všechny potřebné věci na jednom místě.



Obrázek 6.2 Ukázka detailu pipeliny



Obrázek 6.3 Ukázka logů konkretního Jobu

Výpis kódu 6.2 Soubor .gitlab-ci.yml pro konfiguraci Gitlab CI/CD

```
default:
  tags:
    - iOS

variables:
  LC_ALL: "en_US.UTF-8"
  LANG: "en_US.UTF-8"

stages:
  - deploy

.run_when_tag_is_created:
  rules:
    - if: $CI_COMMIT_TAG != null
      when: on_success

######################## Deploy ########################
farmsoft_TF_upload_job:
  stage: deploy
  extends:
    - .run_when_tag_is_created
  script:
    - fastlane upload_to_tf --env Farmsoft


rskot_TF_upload_job:
  stage: deploy
  extends:
    - .run_when_tag_is_created
  script:
    - fastlane upload_to_tf --env Rskot
```

Kapitola 7

Zhodnocení a budoucí vývoj aplikace

Práce úspěšně implementovala prototyp aplikace Farmsoft pro mobilní telefony se systémem iOS, na jejímž základě vznikla aplikace, které je momentálně dostupná pro uživatele používající systém Farmsoft. Uživatelé aplikaci používají na denní bázi. Aplikace splňuje základní požadavky uživatelů pro práci, ale vždycky je prostor pro vylepšení.

Mezi prvky, na které bych se v budoucnu chtěl zaměřit, je rychlost samotné synchronizace. Při synchronizaci musí být aplikace zapnutá a mobil se pro uživatele stává nepoužitelným. Navíc pro velké podniky může první synchronizace trvat i desítky minut. Proto bych se rád zaměřil na její zrychlení. Pro zrychlení by se nechala použit funkce "NSBatchInsertRequest". Jedná se o funkci, kterou nabízí CoreData pro vložení velkého množství záznamů přímo do perzistentního úložiště. S funkcí zatím nemám žádnou zkušenost a nejsem schopný odhadnout, o jak velké zrychlení by se jednalo.

Další možnost zrychlení synchronizace by byla možná jejím paralelním zpracováním podobně jako tomu je nyní v aplikaci pro systém Android. Díky implementaci ze systému Android vím, že tato varianta je technicky proveditelná a synchronizaci opravdu zrychlí. Jediný důvod, proč jsem jí zatím neimplementoval, je její poměrně velká implementační náročnost.

Po vyřešení problému se synchronizací bych se rád zaměřil na srovnání verzi iOS a Android aplikace. Momentálně je Android verze napřed oproti iOS. První funkce, kterou bych chtěl do iOS dodělat, je kompletní zadávání paznehtů, jako je tomu v Android verzi. V iOS verzi chybí možnost vybrat léčenou oblast a vyfotit léčenou končetinu. Dále v iOS verzi by mělo jít pořizovat obrázky zvířat. iOS verze aktuálně umí je pouze zobrazovat, zatímco Android verze umí fotky i pořizovat v terénu. Důležitou věcí, která bude do iOS verze přidaná, bude komunikace s Bluetooth zařízeními podobně jako je to nyní možné v Android verzi. Aplikace by měla komunikovat se čtečkami a váhami od firmy Tru-Test. Podobně jako Android verze by měla umět vyhledávat zvířata a pořizovat změnu vážení při zvážení zvířete na váze.

V neposlední řade bych rád implementoval všechny sestavy dostupné v Android verzi, pořizování hromadných změn a průvodce aplikací, který v krátkosti uživateli představí aplikaci. Průvodce je v omezené míře pro Android verzi již implementovaný včetně video návodů ve formě gifů. Tyto materiály by měly jít použít i pro iOS verzi aplikace.

Aplikace momentálně podporuje angličtinu a češtinu. Do budoucna by se určitě počet jazyků měl rozšířit minimálně o polštinu a ruštinu. Právě tyto dva jazyky podporuje android verze a díky tomu již máme překlady téměř hotové. Stačí pouze vytvořit slovník a ten dostat do iOS aplikace. To by však momentálně neměl být problém.

Další věcí, kterou bych do budoucna rád udělal je sjednocení textů napříč aplikacemi. Mobilní aplikace mají texty celkem sjednocené. Ovšem některé části mobilních aplikací a PC verze

programu se liší a bylo by dobré je sjednotit. Pro příklad uvádím pojem laktace. V mobilní aplikaci se jedná o laktaci, v PC verzi je to sice laktace, zároveň je to však i kontrola užitkovosti a rozbory. Proto bych se do budoucna rád zamyslel, zdali je potřeba jednu věc pojmenovávat třemi různými názvy.

Poslední věc, kterou bych do obou mobilních aplikací rád implementoval, jsou analytické nástroje. Momentálně nemáme vůbec tušení, jak obrazovky uživatelé používají. Pro budoucí vývoj aplikací by určitě chtělo zvážit monitorování uživatelů v aplikaci. Z tohoto monitoringu by následně mohly vznikat další analýzy, kterým směrem bychom chtěli vývoj mobilních aplikací pro Farmsoft udávat.

Momentálně jsem však s aktuální verzí aplikace spokojen. Aplikace plní, co od ní uživatelé očekávání a uživatelé jí používají. Pokud bych mohl něco změnit, tak by to bylo využití jiné architektury pro implementaci. Konkrétně architektury TCA, kterou jsem již v práci popisoval.

Tvorbu aplikace v rámci psaní této práce hodnotím pozitivně. Bylo dobré vědět, že mnou vytvořená aplikace bude mít reálně využití a hodně lidem usnadní jejich práci. Navíc díky psaní práce jsem byl nucen pochopit systém Farmsoft do větší hloubky než by bylo potřeba, pokud bych pouze tvořil již zmíněné mobilní aplikace. To do budoucna určitě pomůže při návrhu nových funkcí či odhalování chyb.

Závěr

Cílem mojí práce bylo implementovat prototypovou aplikaci Farmsoft pro systém iOS. Aplikace sice nemusela být napojena na API Farmsoft, avšak toto napojení mělo být možné doimplementovat tak, aby aplikace mohla být nahrána na AppStore k dispozici zákazníkům.

Aplikace byla implementovaná v jazyku Swift od společnosti Apple. Vychází z designu aplikace pro Android, avšak obsahuje komponenty UI typické pro iOS. Z prototypové aplikace byla následně vytvořena ostrá verze, která byla dána uživatelům k testování.

Prvotní testování probíhalo interně jako konzultace s panem Smolíkem. Po vyladění chyb jsme přešli k ostrému testování aplikace přímo v provozu. Testování probíhalo přímo v ostrém provozu uživatelem. Uživatel měl zkušenosti s verzí pro Android a cílem bylo zjistit, jestli iOS verze nabízí potřebnou funkcionalitu pro práci uživatele. Z testování vyplynulo, že aplikace splňuje všechny požadované funkce. Po úspěšném testování byla aplikace dána k dispozici zákazníkům, kteří si jí nyní mohou stáhnout v aplikaci AppStore.

Dalším úkolem bylo zjistit možnosti automatického nasazení aplikace do AppStoreConnect. K tomu účelu jsem využil kombinaci Fastlane a Gitlab CI/CD. Nyní je aplikace automaticky nahrána na AppStoreConnect při vytvoření nového tagu v prostředí firemního Gilabu.

Nakonec jsem shrnul plánovaný budoucí vývoj aplikace. Vývoj aplikace touto prací nekončí a do budoucna plánuji se na jejím vývoji stále podílet. Do budoucna by aplikace měla být přeložena do dalších cizích jazyků jako je polština a ruština. Do aplikace by měly přibýt ostatní sestavy dostupné v Android verzi a aplikace by měla být schopna komunikovat s Bluetooth zařízeními.

Bibliografie

1. FARMSOFT, e-mail: info@farmtec.cz. *Farmsoft*. [B.r.]. Dostupné také z: https://www.farmsoft.cz/.
2. FARMSOFT, e-mail: info@farmtec.cz. *Podpůrné Technologie*. [B.r.]. Dostupné také z: https://www.farmsoft.cz/podpurne-technologie.html.
3. SMOLÍK Petr, Ing. *Konzultace o projektu Farmsoft*. [B.r.].
4. *Firebird: The true open source database for windows, linux, mac os x and more*. [B.r.]. Dostupné také z: https://firebirdsql.org/en/historical-reference/.
5. ATCHISON, Lee. *Put business logic in the application, not the database*. InfoWorld, 2021. Dostupné také z: https://www.infoworld.com/article/3633005/put-business-logic-in-the-application-not-the-database.html.
6. FRELICH, Jan. *Chov skotu*. Jihočeská univerzita, 2001.
7. FARMTEC. *Vitalimetry Fa 22*. [B.r.]. Dostupné také z: https://www.farmtec.cz/vitalimetry-fa-22.html.
8. NEKRASOV, Alex. *How to write IOS apps without Xcode*. Better Programming, 2022. Dostupné také z: https://betterprogramming.pub/writing-ios-apps-without-xcode-89450d0de21a.
9. INC., Apple. *Xcode*. [B.r.]. Dostupné také z: https://developer.apple.com/support/xcode/.
10. *Programming with objective*. 2014. Dostupné také z: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html.
11. *Top programming languages for IOS app development*. 2022. Dostupné také z: https://www.geeksforgeeks.org/top-programming-languages-for-ios-app-development/.
12. *Flutter tutorial*. 2021. Dostupné také z: https://www.geeksforgeeks.org/flutter-tutorial/.
13. *Flutter: An introduction to the open source SDK by Google*. 2022. Dostupné také z: https://www.geeksforgeeks.org/flutter-an-introduction-to-the-open-source-sdk-by-google/.
14. *Flutter vs native vs react-native: Examining performance*. [B.r.]. Dostupné také z: https://inveritasoft.com/blog/flutter-vs-native-vs-react-native-examining-performance.

15. 6, RyanJMRyanJM; 162K2323 GOLD BADGES270270 SILVER BADGES358358 BRONZE BADGES, bbumbbum; BADGES, XiaoXiao 11.5k22 gold badges2525 silver badges3636 bronze. *IPhone Dev - how important is project.pbxproj?* 1958. Dostupné také z: https://stackoverflow.com/questions/5931788/iphone-dev-how-important-is-project-pbxproj.

16. 4, thndrkissthndrkiss; 17K66 GOLD BADGES5757 SILVER BADGES6767 BRONZE BADGES, gaigegaige; 4, DawnSongDawnSong; 9, user14129006user14129006. *Is the project.xcworkspace file important?* 1959. Dostupné také z: https://stackoverflow.com/questions/10956312/is-the-project-xcworkspace-file-important.

17. *Information property list key reference*. 2018. Dostupné také z: https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Introduction/Introduction.html.

18. ZOZULIA, Kseniia. *Libraries, frameworks, swift packages... what's the difference?* Medium, 2020. Dostupné také z: https://medium.com/@zippicoder/libraries-frameworks-swift-packages-whats-the-difference-764f371444cd.

19. TEAM, CocoaPods Dev. *Cocoapods.org*. [B.r.]. Dostupné také z: https://cocoapods.org/.

20. · BEGINNER, Felipe Laso-Marsetti Mar 18 2020 · Article (25 mins); LASO-MARSETTI, Felipe. *Carthage Tutorial: Getting started*. [B.r.]. Dostupné také z: https://www.raywenderlich.com/7649117-carthage-tutorial-getting-started.

21. TRINH, Jessica. *How to use swift package manager with xcode: An introduction and tutorial*. The Startup, 2020. Dostupné také z: https://medium.com/swlh/how-to-use-swift-package-manager-with-xcode-an-introduction-and-tutorial-6ffb5d3dc44d.

22. *Appcode: Smart swift/objective-C ide for IOS*. [B.r.]. Dostupné také z: https://www.jetbrains.com/objc/.

23. L., Jeroen. *Uikit vs. SWIFTUI - choosing the right framework!* Stream, 2022. Dostupné také z: https://getstream.io/blog/uikit-vs-swiftui/.

24. *The holy grail of uikit: Delegate pattern*. 2021. Dostupné také z: https://tarikdahic.com/posts/the-holy-grail-of-uikit-delegate-pattern/.

25. PANDEY, Vikas. *Uiviewcontroller lifecycle*. Medium, 2019. Dostupné také z: https://medium.com/@vipandey54/uiviewcontroller-lifecycle-7ca2d36f4f07.

26. *Previews in Xcode*. [B.r.]. Dostupné také z: https://developer.apple.com/documentation/swiftui/previews-in-xcode.

27. TOF; (NIKITA BELOV), wtedst; (EIMANTAS), eimantas. *What is the architecture officially recommended by Apple for swiftui applications?* 2021. Dostupné také z: https://forums.swift.org/t/what-is-the-architecture-officially-recommended-by-apple-for-swiftui-applications/44930.

28. *Introduction to model view view model (MVVM)*. 2022. Dostupné také z: https://www.geeksforgeeks.org/introduction-to-model-view-view-model-mvvm/.

29. PIPER, David. *Getting started with the composable architecture*. [B.r.]. Dostupné také z: https://www.raywenderlich.com/24550178-getting-started-with-the-composable-architecture.

30. *App Store*. [B.r.]. Dostupné také z: https://www.apple.com/cz/app-store/.

31. INC., Apple. *Choosing a membership*. [B.r.]. Dostupné také z: https://developer.apple.com/support/compare-memberships/.

32. BADGES, Renan Veloso SilvaRenan Veloso Silva 10344 bronze; 2, Abcd EfgAbcd Efg. *CoreData in main thread is common?* 1963. Dostupné také z: https://stackoverflow.com/questions/31999198/coredata-in-main-thread-is-common.

33. JAGTAP, Shashikant. *Five options for IOS continuous delivery without Fastlane*. XCBlog, 2018. Dostupné také z: `https://medium.com/xcblog/five-options-for-ios-continuous-delivery-without-fastlane-2a32e05ddf3d`.
34. *App Automation done right*. [B.r.]. Dostupné také z: `https://fastlane.tools/`.
35. *CI/CD pipelines*. [B.r.]. Dostupné také z: `https://docs.gitlab.com/ee/ci/pipelines/`.
36. *Gitlab Runner*. [B.r.]. Dostupné také z: `https://docs.gitlab.com/runner/`.

Obsah přiloženého média

- `Farmsoft-ios` ..... Projekt aplikace
  - `Farmsoft` ..... Zdrojové kódy aplikace
    - …
  - `Farmsoft.xcodeproj` ..... Soubor projektu
    - …
  - `Farmsoft.xcworkspace` ..... Soubor workspace
    - …
  - `FarmsoftTests` ..... Zdrojové testů aplikace
    - …
  - `Pods` ..... Zdrojové kódy knihoven vložených pomocí CocoaPods
    - …
  - `fastlane` ..... Konfigurační soubory pro fastlane
    - …
  - `.gitlab-ci.yml` ..... Konfigurační soubory pro Gitlab CI/CD
  - `Farmsoft copy-Info.plist`
  - `Gemfile`
  - `Gemfile.lock`
  - `Podfile` ..... Konfigurační soubory pro CocoaPods
  - `Podfile.lock`
  - `Rskot.entitlements`
  - `info_preprocess_haeder.h`
  - `swiftgen.yml` ..... Konfigurační soubor pro knihovnu SwiftGen
- `thesis` ..... Projekt práce v LaTeX
  - `pictures` ..... Obrázky práce
    - …
  - `templates` ..... Původní šablona
    - …
  - `text` ..... Zdrojové kódy práce
    - …
  - `beranm30-assignment.pdf` ..... Zadání práce
  - `ctufit-thesis.cls`
  - `ctufit-thesis.tex`