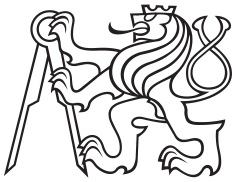
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katerdra počítačů

Mobilní aplikace pro sběr dat v oblasti zemědělství a životního prostředí

Michael Slavev

Vedoucí: Ing. Ivo Malý, Ph.D.
Studijní program: Softwarové inženýrství a technologie
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Slavev** Jméno: **Michael** Osobní číslo: **483819**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mobilní aplikace pro sběr dat v oblasti zemědělství a životního prostředí

Název bakalářské práce anglicky:

Mobile application for data collection in environmental engineering projects

Pokyny pro vypracování:

Analyzujte existující aplikace pro dotazování a sběr dat při experimentech. Zaměřte se na oblast environmentálního inženýrství, např. proces růstu rostlin nebo kontrolu kvality vodních ploch.
Dále navrhněte architekturu aplikace pro generování vhodných typů otázek souvisejících s doménou zemědělství a sběr odpovědí. Zaměřte se na možnost sběru dat automaticky (např. poloha uživatele) nebo manuálně (formulář).
Pro realizaci vyberte vhodné knihovny a datové úložiště splňující požadavky na souběžnou práci více uživatelů. Navržené řešení implementujte pomocí vývojového nástroje Flutter se zaměřením na platformu Android.
Aplikaci otestujte jak softwarovými testy, tak i s alespoň třemi uživateli.

Seznam doporučené literatury:

1. Kraus Jan, Mobilní aplikace pro sběr dat z chodníkové sítě, Bakalářská práce, České vysoké učení technické v Praze, 2020.
2. Flutter, https://flutter.dev/
3. OpenWeatherMap API, https://openweathermap.org/api

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ivo Malý, Ph.D., katedra počítačové grafiky a interakce FEL

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

Datum zadání bakalářské práce: **12.02.2021** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Ivo Malý, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. _____ Datum převzetí zadání

_____ Podpis studenta

Poděkování

Na tomto místě bych rád poděkoval vedoucímu Ing. Ivu Malému, Ph.D. za jeho ochotu a cenné rady při zpracovávání bakalářské práce již od samotných začátků. Dále bych rád poděkoval svým kolegům a všem zúčastněným na testování aplikace a své přítelkyni za korekturu textu. V neposlední řadě patří mé poděkování mé matce za podporu během celého mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 21. května 2021

Abstrakt

Cílem této bakalářské práce je vytvořit mobilní aplikaci pro platformu Android sloužící ke sběru dat z oblasti environmentálního inženýrství. Nejdříve jsou v práci porovnána existující řešení, dále jsou analyzované aplikační požadavky, případy užití a je představen high-fidelity prototyp aplikace. Následně je provedena technická analýza, která má za cíl zvolení vývojových nástrojů. Výsledkem práce je implementovaná mobilní aplikace na základě aplikačních požadavků, případů užití a prototypu. Funkčnost aplikace byla ověřena za pomoci softwarových a uživatelských testů.

Klíčová slova: Vývoj mobilních aplikací, Android, Flutter, Firebase, Google maps, Material Design

Vedoucí: Ing. Ivo Malý, Ph.D.

Abstract

The purpose of this bachelor thesis is to create a mobile application for the Android platform used to collect data in the field of environmental engineering. In the beginning, the existing solutions are compared, followed by application requirements, use cases, and an introduction to the high-fidelity prototype of the application. After that, the technical analysis is performed with the aim of development tools selection. The result of this bachelor thesis is an implemented mobile application based on application requirements, use cases, and prototype. Application functionalities were verified by software unit tests and user tests.

Keywords: Mobile development, Android, Flutter, Firebase, Google maps, Material Design

Title translation: Mobile application for data collection in environmental engineering projects

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Cílem této bakalářské práce je navrhnout a implementovat moderní mobilní aplikaci sloužící ke sběru dat při experimentech z oblasti environmentalního inženýrství související s doménou zemědělství. Navržená aplikace bude implementována pomocí vývojového nástroje Flutter pro platformu Android.

V textu práce se nejprve věnuji mimo jiné analýze existujích řešení a tvorbě aplikačních požadavků, dále se pak přesouvám na návrh aplikace obsahující případy užití a představuji high-fidelity prototyp aplikace. Následuje technická analýza porovnávající různé vývojové nástroje pro následující implementaci. Implementace vychází z definovaných aplikačních požadavků, případů užití a prototypu. Implementace je poté ověřena softwarovými unit testy a uživatelskými testy. Ná závěr vyhodnotím dosažený výsledek.

Kapitola 2

Analýza

2.1 Analýza existujících řešení

V následujících částech této kapitoly se věnuji popisu existujících řešení, které se zabývají stejnou nebo podobnou problematikou.

2.1.1 GroZone Tracker

Webová aplikace GroZone Tracker[1] přináší pěstitelům, potažmo i zaměstnancům, možnost sledování malého množství základních parametrů, pro rostliny (pH, EC) a pro vodní zdroje (pH, EC, zásaditost), dále pak lze vytvořit sledované prostředí a přidat jeho lokaci na mapě. Přednostní funkcionalitou je zakládání podřazených účtů pro zaměstnance, umožňující přidávat nová měření.

Prostředí aplikace není zcela přehledné a často jsem se potýkal se zmizením menu. Výrazné znepokojení ve mě vzbudilo potvrzení registrace, kdy mi bylo zasláno zadané heslo v prostém textu.

2.1.2 Gardenize

Mobilní aplikace Gardenize[2] je řešením v podobě sociální sítě pro domácí pěstitele. I když se jedná spíše o sociální síť než aplikaci pro sběr dat a jejich analýzu, částečně se svojí podobou přibližuje konceptu, který popíši detailně v další kapitole.

[1] `http://grozonetracker.com/`
[2] `https://www.gardenize.com/`

Aplikace má čtyři hlavní funkce:

- Oblasti - Umožňuje vytvořit pěstební oblast jako je např. skleník nebo záhon.
- Květiny - Umožňuje vytvořit květinu a přiřadit ji k oblasti, ke každé květině pak lze přidávat průběžné fotografie a vlastní poznámky.
- Události - Umožňuje vytvoření událostí pro oblasti i květiny s možností nastavení upozornění např. pro zalití nebo zastřihnutí, k proběhnutým událostem lze přidat fotografie a vlastní poznámky.
- Přátelé - část sloužící k propojení s lidmi z celého světa, lze sledovat ostatní uživatele i získat sledující. Možnost podívat se na záhony či květiny ostatních uživatelů a tak získat rady nebo inspiraci.

Prostředí aplikace je moderní a uživatelsky přívětivé.

2.1.3 DeviceMagic

DeviceMagic[3] je formulářové řešení vhodné převážně pro společnosti. Tato aplikace umožňuje široké množství řešení sběru dat pro zemědělství, energetiku, logistiku, telekomunikace, výrobu atd. Umožňuje customizaci všech použitých dotazovacích formulářů dle aktuální potřeby. Jde o velice komplexní řešení nevhodné pro individuální použití na menším měřítku.

2.2 Aplikační požadavky

Na základě analýzy existujících řešení této problematiky jsem se rozhodl pro vytvoření customizovatelných formulářů pro sběr dat z procesu vývoje růstu rostlin nebo z kontroly kvality vodních ploch. Sledované položky tvoří rostliny a vodní plochy s individuálně přiřazeným prostředím reprezentujícím existující lokalitu pomocí GPS pozice. Vytvořené lokality lze zobrazit na mapě nebo v seznamu.

Důležitým prvkem je možnost zobrazení přehledných vývojových grafů a fotografií sledovaných položek z nasbíraných dat. Položky je možné po dokončení sledování archivovat a tím bezpečně uchovat důležitá data.

Získané aplikační požadavky jsem rozdělil do dvou kategorií:

[3] `https://www.devicemagic.com/`

- Funkční požadavky: Z publikace [1] definují systémovou funkci nebo akci. Mohou obsahovat výpočty, manipulaci dat a další specifické funkcionality, které jsou nezbytné pro vykonání. V následujících sekcích jsou značeny Fn, kde n je číslo funkčního požadavku.
- Nefuknční požadavky: Dle článku [3] definují požadavky na aplikační prostředí, rozhraní a architekturu. V následujících sekcích jsou značeny Nn, kde n je číslo nefunkčního požadavku.

2.2.1 Funkční požadavky

Seznam všech funkčních požadavků je vyobrazen dále.

- F1 - Přihlášení pomocí účtu Google

  Aplikace bude umožňovat neautentizovanému uživateli přihlášení do aplikace skze aktivní Google účet.

- F2 - Odhlášení z aplikace

  Aplikace bude umožňovat autentizovanému uživateli odhlásit se z aplikace.

- F3 - Vytvoření a editace prostředí

  Aplikace bude umožňovat autentizovanému uživateli vytvářet prostředí. Každé prostředí bude mít následující atributy: název, popis, typ prostředí (Skleník, Foliovník, Pařeniště, Záhon, Vyvýšený záhon, Dům, Byt, Zahrada, Vodní plocha) a umístění pomocí aktuální polohy zařízení.

  Takto vytvořenému prostředí lze následně v aplikaci upravit jednotlivé atributy (název, popis a typ). V případě potřeby aplikace umožní prostředí smazat (včetně všech obsažených položek).

- F4 - Zobrazení a filtrace prostředí

  Aplikace bude umožňovat autentizovanému uživateli zobrazit vytvořené prostředí v seznamu nebo na mapě. Aplikace umožní uživateli filtrovat jednotlivá prostředí dle typu definovaného v požadavku F3. Ze zobrazení v seznamu nebo na mapě bude možné přejít na konkrétní prostředí.

- F5 - Vytvoření a editace položky

  Aplikace bude umožňovat autentizovanému uživateli vytvářet v prostředí nové položky. Každá položka bude mít následující atributy: název, popis, typ položky (Rostlina, Vodní plocha), seznam sledovaných atributů dle typu položky.

  - Pro rostliny: velikost, okolní teplota, okolní vlhkost, teplota půdy, vlhkost půdy, pH půdy, pH zálivky, množství zálivky, EC zálivky, alkalinita zálivky, tvrdost zálivky, celkové rozpuštěné soli, celkový

obsah nerozpustných látek, dusík, fosfor, draslík, měď, boron, molybden, vápník, nikl, hořčík, foto průběžného stavu, foto defektů, foto škůdců, jiné foto

- Pro vodní plochy: stav hladiny, teplota vody, okolní vlhkost, průtok, pH, EC, alkalinita, tvrdost, celkové rozpuštěné soli, celkový obsah nerozpustných látek, dusík, fosfor, draslík, měď, boron, molybden, vápník, nikl, hořčík, foto průběžného stavu, jiné foto

Posledním atributem při vytváření položky je četnost výzev k sběru dat, kterou má uživatel možnost nastavit dle potřeby v rámci rozmezí dnů, případně tuto funkci nepoužívat.

Takto vytvořené položce lze následně v aplikaci upravit jednotlivé atributy (název, popis a frekvenci notifikací). Dále podle potřeby uživatele aplikace umožní položku archivovat a smazat.

- F6 - Zobrazení položek

  Aplikace bude umožňovat autentizovanému uživateli zobrazit vytvořené položky prostředí. Ze zobrazení bude možné přejít na konkrétní detail položky (požadavek F7) proklikem nebo naskenováním QR kódu položky v příslušném prostředí, tzn. v prostředí nelze načíst položku pomoci QR kódu z jiného prostředí.

- F7 - Zobrazení detailu položky

  Aplikace bude umožňovat autentizovanému uživateli zobrazit detail vytvořené položky, který bude obsahovat název, datum vytvoření, popis a historii měření. Detail položky také umožní zobrazení QR kódu (požadavek F8) a zobrazení vývoje měřených parametrů (požadavek F9).

- F8 - Zobrazení, uložení a tisk QR kódu

  Aplikace bude umožňovat autentizovanému uživateli zobrazit QR kód generovaný ke každé položce s možností ho uložit do zařízení nebo vytisknout.

- F9 - Zobrazení vývoje měřených parametrů

  Aplikace bude umožňovat autentizovanému uživateli zobrazit nasbíraná data položek v podobě grafů pro všechny nefotografované atributy a v podobě datovaných galerií pro fotografované atributy.

- F10 - Vytvoření nového měření

  Aplikace bude umožňovat autentizovanému uživateli přidat nové měření položky ze zvolených parametrů při jejím vytvoření.

- F11 - Archivovat položku

  Aplikace bude umožňovat autentizovanému uživateli archivovat položky do vytvořených archivačních adresářů s vlastním komentářem a bodovým hodnocením sledování.

  Takto archivovanou položku aplikace umožní uživateli smazat.

- F12 - Vytvoření a editace archivačního adresáře

  Aplikace bude umožňovat autentizovanému uživateli vytvořit nový archivační adresář s vlastním názvem.

  Takto vytvořenému adresáři lze následně v aplikaci upravit atribut názvu. Dále podle potřeby uživatele aplikace umožní smazat adresář (včetně všech obsažených položek).

- F13 - Zobrazení archivací

  Aplikace bude umožňovat autentizovanému uživateli zobrazit archivované položky v archivačních adresářích. Ze zobrazení bude možné přejít na konkrétní detail archivované položky (požadavek F7).

- F14 - Nastavení sítě

  Aplikace bude umožňovat autentizovanému uživateli možnost zvolit využití datové sítě, Wi-Fi nebo jejich kombinaci pro synchronizaci.

2.2.2 Nefuknční požadavky

Seznam všech nefunkčních požadavků je vyobrazen níže.

- N1 - Offline použití

  Aplikace musí poskytovat svoji funkcionalitu i přes ztrátu připojení k internetu.

- N2 - Přizpůsobení velikosti

  Aplikace se musí přizbůsobit rozměrům mobilního zařízení na kterém se má vykreslit.

- N3 - Nezávislost OS

  Aplikace musí fungovat na systému Android (verze 6.0 a vyšší).

Kapitola 3

Návrh

Tato kapitola popisuje návrh aplikace. Sestavil jsem případy užití pokrývající aplikační požadavky (kapitola 2.2) a sestrojil prototyp aplikace.

3.1 Případy užití

Tato kapitola představuje identifikované aktéry interagující s navrhovaným systémem a jednotlivé případy užití. Vztahy jednotlivých případů užití a aktérů jsou zobrazeny na diagramu případů užití na obrázku A.2 v příloze A.

3.1.1 Use cases

Tato podkapitola popisuje případy užití vyobrazené v use case diagramu na obrázku A.2 v příloze A.

- UC1 - Zobrazit přihlášení

  Uživatel je po zapnutí aplikace přesměrován na přihlašovací obrazovku.

- UC2 - Přihlásit

  Uživatel je po zadání správných přihlašovacích údajů třetí strany přihlášen do aplikace. Při úspěšném příhlášení je uživatel přesměrován na úvodní obrazovku (UC4), při zadání neplatných údajů aplikace zobrazí dialogové okno o chybných přihlašovacích údajích.

- UC3 - Nastavit využití sítě

  Přihlášený uživatel vybere typ datového připojení, který je použit pro synchronizaci dat měření.

- UC4 - Zobrazit seznam prostředí

  Přihlášený uživatel klepnutím ve spodním navigačním panelu na "*Prostředí*" zobrazí seznam všech jeho vytvořených prostředí.

- UC5 - Filtrovat prostředí

  Jde o rozšíření případu užití UC4. Přihlášený uživatel klepnutím na ikonku hledání může filtrovat zobrazený seznam prostředí. Aplikovatelný filtr je typ prostředí. Klepnutím na tlačítko "*Filtrovat*" se zobrazí filtrovaný seznam prostředí, klepnutím na tlačítko "*Zrušit*" se žádné filtrování neprovede.

- UC6 - Vytvořit prostředí

  Jde o rozšíření případu užití UC4. Aplikace přihlášeného uživatele po klepnutí na ikonku přidání přesměruje na formulář vyplnění údajů prostředí. Klepnutím na tlačítko "*Další*" aplikace přesměruje na mapu s upřesněním pozice nově přidávaného prostředí. klepnutím na tlačítko "*Vytvořit*" se uživateli zobrazí dialogové okno s potvrzením vytvoření prostředí, klepnutím na "*Vytvořit*" aplikace uživatele přesměruje seznam prostředí (UC4), klepnutím na "*Zrušit*" pak uživatel zůstane na obrazovce s upřesněním pozice.

- UC7 - Editovat prostředí

  Jde o rozšíření případu užití UC4. Aplikace přihlášenému uživateli po klepnutí na ikonku zobrazí dialogové okno možností prostředí, po klepnutí na možnost "*Zrušit*" se zobrazí dialogové okno a jednotlivé atributy prostředí v textových polích. Klepnutím na "*Uložit*" se uloží všechny provedené změny atributů a aplikace přesměruje uživatele na seznam prostředí (UC4), klepnutím na "*Zrušit*" pak uživatel zůstane na obrazovce seznamu prostředí (UC4).

- UC8 - Smazat prostředí

  Jde o rozšíření případu užití UC4. Aplikace přihlášenému uživateli po klepnutí na ikonku zobrazí dialogové okno možností prostředí, po klepnutí na možnost "*Smazat*" se uživateli zobrazí dialogové okno s potvrzením smazání prostředí, klepnutím na "*Smazat*" se prostředí smaže a aplikace uživatele přesměruje na seznam prostředí (UC4), klepnutím na "*Zrušit*" pak uživatel zůstane na obrazovce seznamu prostředí (UC4).

- UC9 - Zobrazit prostředí

  Jde o rozšíření případu užití UC4. Aplikace přihlášeného uživatele po klepnutí na kartu prostředí přesměruje na seznam všech jeho vytvořených položek prostředí.

- UC10 - Zobrazit položky QR kódem

  Jde o rozšíření případu užití UC9. Přihlášený uživatel po klepnutí na ikonku fotoaparátu je přesměrován na fotoaparát pro naskenování QR

kódu. Naskenováním QR kódu položky z příslušného prostředí aplikace přesměruje uživatele na příslušnou položku stejně jako UC15. V případě naskenování QR kódu položky z jiného prostředí se zobrazí chybová hláška.

- UC11- Vytvořit položku

  Jde o rozšíření případu užití UC9. Aplikace přihlášeného uživatele po klepnutí na ikonku přidání přesměruje na formulář vyplnění údajů položky. klepnutím na tlačítko "*Další*" aplikace přesměruje na formulář výběru sledovaných parametrů. klepnutím na tlačítko "*Další*" aplikace přesměruje na formulář výběru četnosti výzev k sběru dat (notifikací). klepnutím na tlačítko "*Vytvořit*" se uživateli zobrazí dialogové okno s potvrzením vytvoření prostředí, klepnutím na "*Vytvořit*" aplikace uživatele přesměruje na zobrazení QR kódu vytvořené položky (UC17), klepnutím na "*Zrušit*" pak uživatel zůstane na obrazovce s výběrem četnosti výzev.

- UC12 - Editovat položku

  Jde o rozšíření případu užití UC9. Aplikace přihlášenému uživateli po klepnutí na ikonku zobrazí dialogové okno možností položky, po klepnutí na možnost "*Upravit*" aplikace přesměruje uživatele na obrazovku s jednotlivými atributy prostředí v textových polích, atributy sledovaných parametrů a frekvence notifikací. Klepnutím na "*Upravit*" se uloží všechny provedené změny atributů a aplikace přesměruje uživatele na prostředí (UC9).

- UC13 - Archivovat položku

  Jde o rozšíření případu užití UC9. Aplikace přihlášenému uživateli po klepnutí na ikonku zobrazí dialogové okno možností položky, po klepnutí na možnost "*Archivovat*" se zobrazí dialogové okno s výběrem vytvořených archivačních adresářů, zvolením umístění archivace uživatel potvrdí archivaci tlačítkem "*Archivovat*", klepnutím na "*Zrušit*" pak uživatel zůstane na obrazovce prostředí (UC9).

- UC14 - Smazat položku

  Jde o rozšíření případu užití UC9. Aplikace přihlášenému uživateli po klepnutí na ikonku zobrazí dialogové okno možností položky, po klepnutí na možnost "*Smazat*" se zobrazí dialogové okno s potvrzením smazání položky, klepnutím na "*Smazat*" se položka smaže a aplikace uživatele přesměruje na zobrazení položek prostředí (UC9), klepnutím na "*Zrušit*" pak uživatel zůstane na obrazovce prostředí (UC9).

- UC15 - Zobrazit položku

  Jde o rozšíření případu užití UC5. Aplikace přihlášeného uživatele po klepnutí na kartu položky přesměruje na detail vytvořené položky.

- UC16 - Přidat měření

Jde o rozšíření případu užití UC15. Aplikace přihlášeného uživatele po klepnutí na ikonku přidání přesměruje na formulář vyplnění sledovaných atributů. Uživatel vyplní naměřená data, případně pořídí fotografie. Klepnutím na tlačítko "*Dokončit*" se zobrazí dialogové okno s potvrzením dokončení měření, klepnutím na "*Ano*" se měření dokončí a zaznamená, aplikace poté uživatele přesměruje na zobrazenou položku (UC15), klepnutím na "*Ne*" pak uživatel zůstane na obrazovce přidat měření (UC16).

- UC17 - Zobrazit QR kód

  Jde o rozšíření případu užití UC15. Aplikace přihlášeného uživatele po klepnutí na ikonku QR kódu přesměruje na obrazovku příslušného QR kódu.

- UC18 - Zobrazit vývoj sledovaných parametrů

  Jde o rozšíření případu užití UC15. Aplikace přihlášeného uživatele po klepnutí na ikonku grafů přesměruje na obrazovku grafů naměřených hodnot. Uživatel může vybrat zobrazení grafu vývoje určitého atributu.

- UC19 - Vytisknout QR kód

  Jde o rozšíření případu užití UC17.

- UC20 - Uložit QR kód

  Jde o rozšíření případu užití UC17.

- UC21 - Zobrazit prostředí na mapě

  Přihlášený uživatel klepnutím ve spodním navigačním panelu na "*Mapy*" zobrazí mapu se značkami lokací již vytvořených prostředí. Uživatel tento bod může otevřít a pomocí klepnutí na tlačítko "*Otevřít prostředí*" přejít na konkrétní prostředí (UC9).

- UC22 - Zobrazit archivační adresáře

  Přihlášený uživatel klepnutím ve spodním navigačním panelu na "*Otevřít prostředí*" je přesměrován na obrazovku s listem vytvořených archivačních adresářů.

- UC23 - Vytvořit archivační adresář

  Jde o rozšíření případu užití UC22. Přihlášený uživatel je po klepnutí na ikonku přidání přesměrován na formulář vytvoření archivačního adresáře. Klepnutím na tlačítko "*Vytvořit*" je uživatel přesměrován na obrazovku archivačních adresářů (UC22).

- UC24 - Editovat archivační adresář

  Jde o rozšíření případu užití UC22. Aplikace přihlášenému uživateli po klepnutí na ikonku zobrazí dialogové okno možností archivačního adresáře, po klepnutí na možnost "*Upravit*" aplikace přesměruje uživatele na obrazovku s jednotlivými atributy archivačního adresáře v textových

polích. Klepnutím na "*Uložit*" se uloží všechny provedené změny atributů a aplikace přesměruje uživatele na obrazovku archivačních adresářů (UC22).

- UC25 - Smazat archivační adresář

  Jde o rozšíření případu užití UC22. Aplikace přihlášenému uživateli po klepnutí na ikonku zobrazí dialogové okno možností archivačního adresáře, po klepnutí na možnost "*Smazat*" se zobrazí dialogové okno s potvrzením smazání adresáře, klepnutím na "*Smazat*" se adresář smaže a aplikace uživatele přesměruje na zobrazení prostředí (UC22), klepnutím na "*Zrušit*" pak uživatel zůstane na obrazovce prostředí (UC22).

- UC26 - Zobrazit archivované položky

  Jde o rozšíření případu užití UC22. Aplikace přihlášeného uživatele po klepnutí na kartu archivačního adresáře přesměruje na list archivovaných položek.

- UC27 - Zobrazit nastavení

  Přihlášený uživatel klepnutím ve spodním navigačním panelu na "*Nastavení*" je přesměrován na obrazovku s nastavením aplikace.

- UC28 - Odhlásit

  Přihlášenému uživateli se v nastavení (UC27) klepnutím na "*Odhlásit se*" zobrazí dialogové okno s potvrzením odhlášení. Potvrzení klepnutím na tlačítko "*Odhlásit*" je uživatel přesměrován na obrazovku přihlášení, při klepnutí na tlačítko "*Zrušit*" dialogové okno zmizí.

3.2 Prototyp

V poslední části této kapitoly prezentuji vytvořený prototyp[1] uživatelského rozhraní, který je díky využití pravidel Material Designu využit pro snadnější implementaci. U každé obrazovky popíšu její vlastnosti a funkce ovládání, které nejsou zřejmé z prototypu.

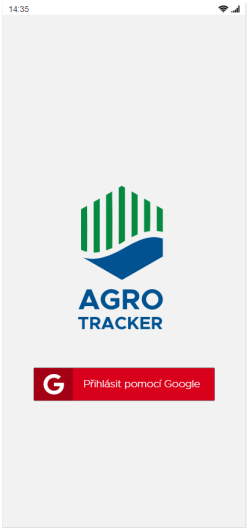
Material Design[2], vytvořený společností Google, je soubor komponentů řídící se stanovenými zásadami. Inspirace Material Designu přicházející z reálného světa mají reprezentovat textury povrchů včetně odrazů světla a vrhání stínů. Slouží k vytvoření komponentů, které berou ohled na uživatele. Prvky Material Designu musí být konzistentní, předvídatelné a responzivní [6].

V prostředí celé aplikace bude možné se pohybovat pomocí systémových gest nebo tlačítek.

[1] `https://65s46p.axshare.com/`
[2] `https://material.io/design`

3.2.1 Přihlašovací obrazovka

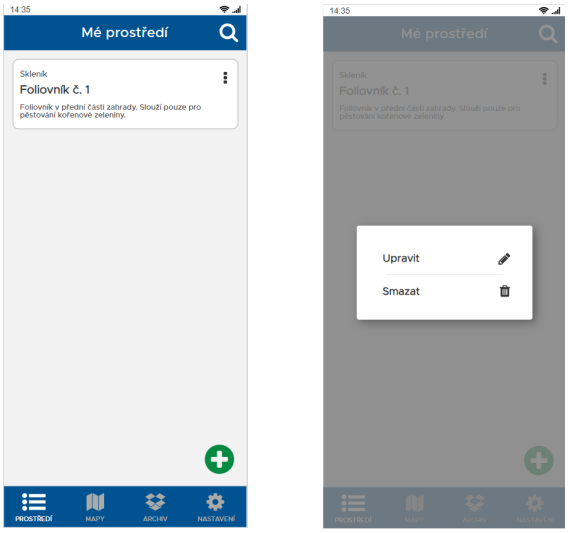
Přihlašovací obrazovka (obrázek 3.1) je vstupní branou do aplikace, umožňuje přihlášení pomocí účtu Google.



Obrázek 3.1: Přihlašovací obrazovka

3.2.2 Seznam prostředí

Tato obrazovka je obrazovkou úvodní a jde o hlavní navigační místo aplikace (obrázek 3.2a), uživatel zde vidí přehled všech svých prostředí. Umožňuje filtrovat zobrazená prostředí pomocí kategorí, vytvořit/editovat (obrázek 3.2b) prostředí a přejít na prostředí.

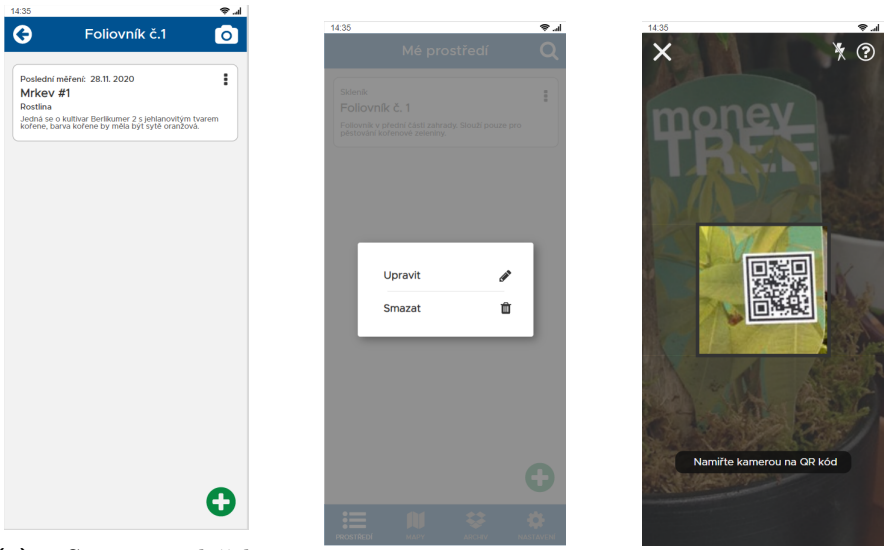


(a) : Seznam prostředí **(b) :** Nastavení

Obrázek 3.2: Mé prostředí

3.2.3 Seznam položek prostředí

Obrazovka seznamu položek (obrázek 3.3a) je přehledem jedotlivých prostředí, uživatel zde vidí přehled všechny své příslušné položky. Umožňuje uživateli načíst/zobrazit detail položky pomocí QR kódu, vytvořit/editovat/archivovat (obrázek 3.3b) položky a přejít na detail položky.

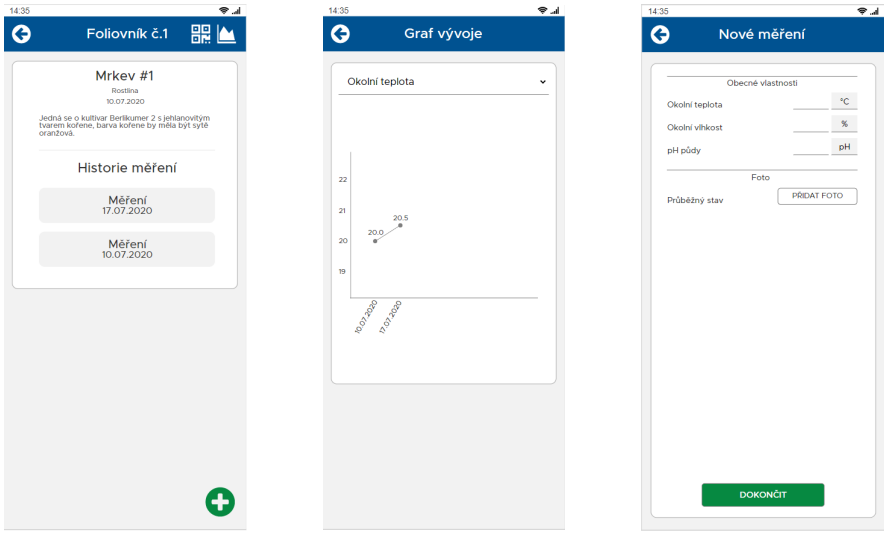


(a) : Seznam položek prostředí **(b) :** Nastavení **(c) :** QR Sken

Obrázek 3.3: Položky prostředí

3.2.4 Detail položky

Na této obrazovce (obrázek 3.4a) uživatel najde veškeré informace o položce - data proběhlých měření, graf vývoje sbíraných parametrů (obrázek 3.4b), případně galerii fotografií z měření. Zde uživatel přidává nové měření (obrázek 3.4c).

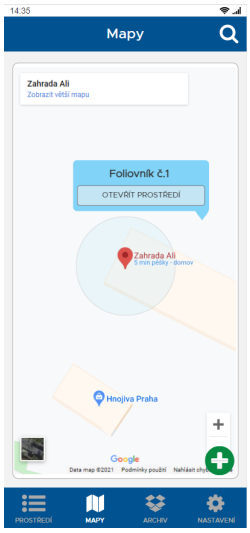


(a) : Detail položky **(b) :** Graf vývoje **(c) :** Nové měření

Obrázek 3.4: Detail položky

3.2.5 Mapy

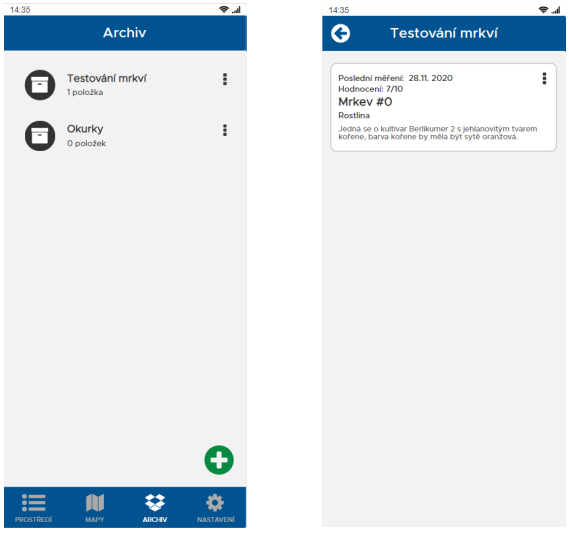
Tato obrazovka (obrázek 3.5) umožňuje uživateli zobrazit mapy, na kterých uvidí body reprezentující prostředí. Umožňuje uživateli přejít na prostředí přímo z map nebo vytvořit prostředí.



Obrázek 3.5: Mapy

3.2.6 Archiv

Obrazovka sloužící k zobrazení archivačních adresářů (obrázek 3.6a). Umožňuje uživateli přejít do archivačního adresáře (obrázek 3.6b) a vytvořit/editovat adresář. V archivačním adresáři uživatel vidí přehled archivovaných položek stejně jako na obrazovce přehledu položek, umožňuje uživateli položky editovat a přejít na jejich detail.

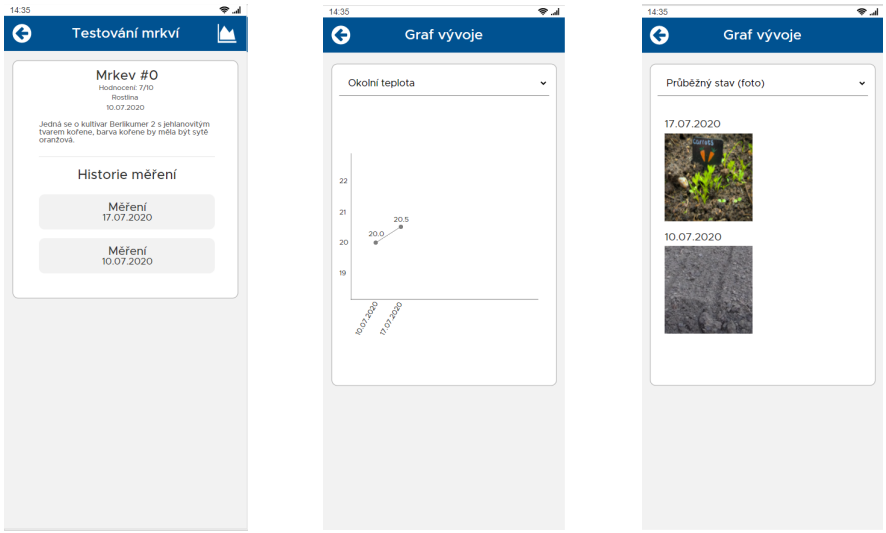


(a) : Archiv **(b) :** Archivační adresář

Obrázek 3.6: Archiv a archivační adresář

3.2.7 Archivovaná položka

Na této obrazovce, podobně jako na obrazovce detailu položky (obrázek 3.7a), uživatel vidí veškeré informace o archivované položce - data proběhlých měření, graf vývoje sbíraných parametrů (obrázek 3.7b), případně galerii fotografií z měření (obrázek 3.7c).



(a) : Detail archivované položky

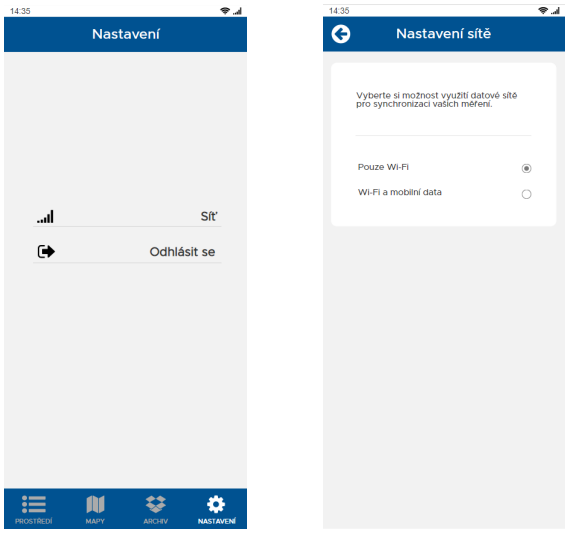
(b) : Graf vývoje archivované položky

(c) : Fotografie vývoje archivované položky

Obrázek 3.7: Archiv a archivační adresář

3.2.8 Nastavení

Na této obrazovce (obrázek 3.8a) uživatel vidí možnosti odhlášení z aplikace a možnost nastavení sítě. Obrazovka nastavení sítě (obrázek 3.8b) umožňuje uživateli nastavit využití sítě pro synchronizaci měření.



(a) : Nastavení **(b) :** Nastavení připojení

Obrázek 3.8: Nastavení

Kapitola 4

Technická analýza

V následujících sekcích popisuji implementační přístupy. Tyto přístupy popíši, porovnám a ve shrnutí technické analýzy vyhodnotím vhodné řešení.

4.1 Vývojové nástroje - SDK

Pro vývoj mobilních aplikací lze zvolit jeden ze dvou přístupů - nativní a multiplatformní.

Dle zadání práce je již stanoven cíl a tím je implementace pomocí multiplatformního frameworku Flutter[1], vyvíjeném společností Google. Výběr multiplatformních řešení je dnes již dost rozsáhlý a každý má své různé výhody a úskalí. V další části dokumentu jsem porovnal nativní přístup vývoje proti frameworku Flutter. Existuje mnoho dalších frameworků jako např. Xamarin, React Native, Ionic a další.

4.1.1 Nativní aplikace

Nativní aplikace jsou dnes vyvíjeny převážně pro Android a iOS, jelikož Windows Mobile už není relevantní vzhledem k ukončení jeho vývoje v roce 2017. Pro systém iOS je specifický vývoj v jazyce Swift a Objective C, pro Android poté Java a Googlem preferovaný Kotlin.

- Výhody
  1. Přímý přístup k funkcím operačního systému zaručující vysokou stabilitu

[1] `https://flutter.dev/`

2. Konzistence UI komponent
3. Optimalizace pro operační systém

- Nevýhody

1. Použitelný pouze pro vyvíjený operační systém
2. Větší náklady na vývoj

4.1.2 Flutter

Definice Flutteru, dle oficiální dokumentace [5], je open-source sada vývojových nástrojů sloužící pro multiplatformní vývoj. Hlavním programovacím jazykem je Dart, který je kompilován do nativního jazyku zvlášť pro Android (pomocí Android NDK) a iOS (pomocí LLVM).

- Výhody

1. Stejné UI na různých verzí OS
2. Hot reload
3. Jednotná codebase
4. Využití GPU

- Nevýhody

1. Menší komunita vzhledem k vytvoření v roce 2018
2. Možnost neexistujích balíčků - nezaručení všech funkcionalit
3. Velikost aplikace

4.2 State management

Každá aplikace potřebuje jistou správu kontroly svých vnitřních stavů, tyto vnitřní stavy reprezentují rozhraní se kterými uživatel interaguje, např. textové pole nebo tlačítka, ale také jejich hodnoty jako je např. barva nebo hodnota. Vzhledem k jisté obsáhlosti aplikace lze předpokládat, že se s touto komplexní kontrolou budu potýkat již při prvotním vývoji a je tedy nutné zvolit vhodný způsob správy těchto stavů. Existují různá řešení tohoto problému, která popíši v následujích podkapitolách.

4.2.1 Flutter - InheritedWidget and InheritedModel

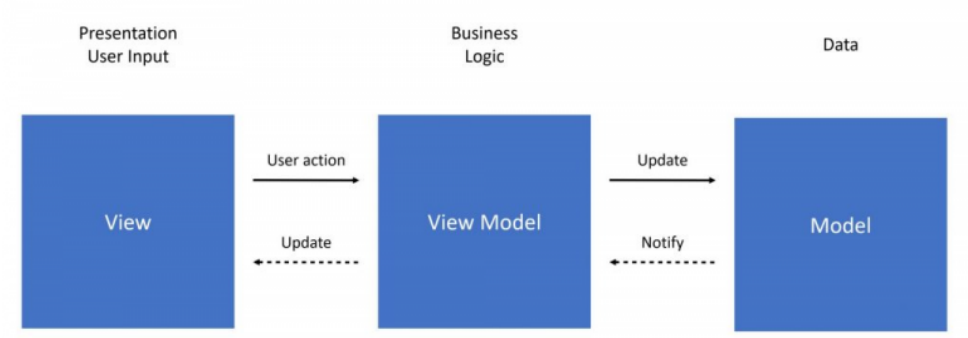
Základním řešením, které je součástí Flutter frameworku, je InheritedWidget a InheritedModel. Tento typ řešení funguje na principu rodičů a potomků v podobě jednotlivých widgetů. S rostoucí náročností aplikace vzniká velký kořenový widget s mnoha vrstvami, a s narůstajícím počtem vrstev narůstá náročnost správy toku dat přes jednotlivé widgety, je tedy vhodné spíše pro malé aplikace.

4.2.2 Provider

Provider[2] je další jednoduché řešení, které je popisováno [7] jako wrapper základního InheritedWidgetu. Právě díky přidaným vlastnostem vytvoření, vystavení, naslouchání a změně naslouchání objektů a prostředků, a jejich snadnější správě, je tento způsob vhodným kandidátem na implementaci MVVM vzoru.

MVVM vzor

V článku Juliana Bissekkou [2] o Model-View-ViewModel vzoru, jde o návrhový vzor oddělující vykreslovací logiku od business logiky aplikace. Hlavním cílem MVVM je přesun stavů a logiky z View do oddělené entity zvané ViewModel, která obsahuje také business logiku a slouží jako prostředník mezi View a Modelem. Narozdíl od architektury MVC je View a Model úplně oddělené.



Obrázek 4.1: MVVM vzor [2]

- View reprezentuje zobrazovací rozhraní pro uživatele, zaručující interakci s ViewModelem a vykreslení aktuálních dat.

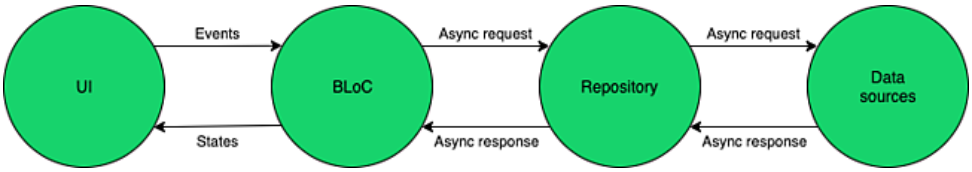
[2] `https://pub.dev/packages/provider`

- ViewModel je esenciálním prvkem MVVM architektury, je to část separující View a Model. Obstarává ukládání, manipulaci a přístup k Modelu.
- Model je Poslední částí MVVM, reprezentuje čistá data.

4.2.3 BLoC

V článku Kacper Kaguta [8], je Business Logic Components (BLoC) podobný návrhovému vzoru MVVM, od kterého se ale liší kompletním oddělením logiky od vykreslování taktéž za pomocí event streamů. Skládá se ze čtyř základních vrstev:

- UI - Vykreslovací vrstva, viditelná uživatelem. Proběhlými eventy informuje nižší vrstvu.
- BLoC - Vrstva business logiky reagující na vstupy UI vrstvy. Je prostředníkem mezi vrstvou repozitáře a UI, pomocí získaných dat nastavuje stav aplikace.
- Repozitář - Vrstva sloužící k získaní a manipulaci dat z jednoho nebo více zdrojů.
- Data - Poskytuje aplikaci nutná data, podobně jako databáze.



Obrázek 4.2: BLoC [8]

4.2.4 GetIt

Balíček GetIt[3] slouží ke správě instancovaných objektů v celé aplikaci i správě singleton instancí a ulevuje tím od zbytečné práce garbage collectoru. Je vhodným doplňkem při použití kteréhokoliv state managementu.

[3] `https://pub.dev/packages/get_it`

4.3 Databáze

Většina aplikací vyžaduje určitou formu ukládání dat mimo zařízení. Tradiční přístup pro řešení tohoto problému je využití databáze, pro SQL databáze např. PostgreSQL[4] a pro NoSQL databáze např. Redis[5]. Bohužel žádné z těchto řešení přímo neposkytuje funkcionalitu autentizace třetích stran a tento tradiční přístup by přinesl do implementace nutnost vývoje vlastního backendu.

Naštěstí je na trhu více řešení této problematiky a jedna z nich je Backend-as-a-Services (BaaS) [9]. Společnost Google v této oblasti poskytuje cloudové řešení Google Firebase [4], u kterého lze využít různé typy autentizace např. email a heslo nebo služby třetích stran, dále pak umožňuje aktualizace v reálném čase, offlinové použití díky možnosti uložení akcí provedených bez připojení, serverové push notifikace atd. Podobných cloudových řešení je ovšem více, např. Amazon Web Services, Microsoft Azure, Cloudflare apod.

4.4 Mapy

V neposlední řadě je třeba zvolení frameworku pro mapy. Hlavní řešení, které se hned nabízí je balíček Google Maps[6], dalé pak Leaflet[7] či Mapbox[8]. Všechny tři řešení poskytují potřebné funkcionality jako je sledování polohy, customizovatelné značky a vyskakovací informační okna.

Výhodou použití Mapbox map je rychlejší načítání díky použití vektorového vykreslení oproti rastrovému.

4.5 Shrnutí

V provedené technické analýze v předchozích sekcích jsem mluvil o technologických řešeních vhodných pro vývoj aplikace. Nyní bych chtěl odůvodnit svoje volby.

První volbu, vývojový nástroj, jsem již učinil při výběru samotné práce. Pro state management jsem se rozhodl pro použití balíčků Provider a GetIt, díky snadnému instancování objektů a přístupu k nim. Pro oddělení business

[4] `https://www.postgresql.org/`
[5] `https://redis.io/`
[6] `https://pub.dev/packages/google_maps_flutter`
[7] `https://leafletjs.com/`
[8] `https://pub.dev/packages/mapbox_gl`

logiky pak MVVM vzor, který je narozdíl od BLoC vzoru výrazně lehčí naimplementovat bez předchozích zkušeností. Při volbě ukládání dat, autentizace a map byl výběr srovnatelný a rozhodl jsem se pro zachování Google platformy a zvolil tedy Google Firebase a Google Maps.

Kapitola 5

Implementace

V této kapitole přiblížím aplikaci samotnou, proces vývoje, použité balíčky a detailněji představím její zajímavé části. Aplikace byla vyvíjena pro platformu Android. Všechny odkazované ukázky kódu nalezneme v příloze B.

5.1 Projekt

Základní strukturu projektu jsem přizpůsobil vybranému návrhovému vzoru MVVM, o kterém jsem se zmiňoval v předešlé kapitole. Strukturu jsem tedy rozdělil na:

- models

  Adresář models ukrývá veškeré modely, které reprezentují data (dokumenty) uložené v databázi.

- providers

  V tomto adresáři najdeme třídy typu ViewModel poskytující data jednotlivým View.

- views

  Zde nalezneme všechny View třídy a widgety, zajišťující interakci s uživatelem.

- services

  Tento adresář obsahuje služby volané ViewModel třídami.

5.2 Služby

Vytvořené služby bylo třeba zpřístupnit v celém rozsahu aplikace pro snadné použití. Tento problém jsem řešil již zmíněným balíčkem GetIt, kde po zaregistrování jednotlivých objektů služeb coby singleton instancí, jsem k těmto instancím schopen přistoupit z kteréhokoliv místa v aplikaci. Způsob registrace objektu je vidět na ukázce 1 a přístup k registrované instanci na ukázce z třídy EnvironmentsProvider 2.

5.2.1 Autentizace

Služba pro autentizaci společně s přihlašovací obrazovkou pro mě byly základním kamenem aplikace, právě kvůli nemožnosti komunikace s databází jako neautentizovaný uživatel. Díky balíčkům firebase_auth[1] a google_sign_in[2] jsem nemusel implementovat logiku pro předávání autentizačních tokenů a další logiku spojenou přímo s ověřením. Přes širokou škálu nabízených možností pro autentizaci (např. email a heslo, Facebook účet, Microsoft účet atd.) jsem vzhledem k platformě Android zvolil přihlášení pomocí účtu třetí strany, a to skrze Google Sign In.

Ukázka implementované metody pro přihlášení do aplikace je vidět na ukázce 6.

5.2.2 Komunikace s databází a serializace

Komunikace s Firebase databází probíhá za pomocí tzv. dokumentů (JSON-like objektů), které jsou na zařízení ukládány jako "DocumentSnapshot" objekty obsahující kromě samotných dat i metadata zajišťující offline fukcionalitu. Standardní reprezentace JSON dat v jazyce Dart je za pomocí map typu klíč-hodnota.

Možností jak se vyhnout manuální deserializaci a serializaci existuje několik a to opět s pomocí různých balíčků, jeden z balíčků na které jsem narazil je např. json_serializable[3], který dokáže generovat nové třídy a metody pro serializaci a deserializaci pomocí přídání několika anotací k datovým objketům, toto řešení mi však přišlo vzhledem k obsáhlosti mých datových objektů zbytečné implementovat. Zvolil jsem tedy v tu chvíli pro mě snadnější cestu s manuální deserializací pro interpretaci do vlastních objektů (ukázka dotazu 3 a deserializace 5) a zpětnou serializací nutnou pro zápis (ukázka serializace v dotazu 4).

[1] `https://pub.dev/packages/firebase_auth`
[2] `https://pub.dev/packages/google_sign_in`
[3] `https://pub.dev/packages/json_serializable`

5.2.3 Notifikace

Poslední významnou službou je služba zprostředkovávající notifikace. Zde jsem se rozhodoval mezi variantou serverových push notifikací a lokálními notifikacemi.

- Server push notifikace
  - + nezávislost na operačním systému a zařízení
  - - je třeba připojení k internetu
  - - placená funkce Google Firebase
- Lokální notifikace
  - + offline funkcionalita
  - + zdarma
  - - závislost na operačním systému a zařízení
  - - složitá implementace přenositelnosti na jiné zařízení

S předpokladem, že se uživatel bude přesouvat na jiné zařízení maximálně jednou za 1-3 roky v rámci výměny zařízení, jsem přenos notifikací mezi zařízeními nepovažoval za kritickou funkcionalitu. Notifikace jsem implementoval za pomocí balíčku flutter_local_notifications[4]. Metodu scheduledNotification registrující plánované notifikace nalezneme v ukázce 7.

5.3 Datový model a optimalizace

Díky předešle vytvořenému prototypu v kapitole 3.2 jsem měl představu, s jakými daty se budu potýkat, a bylo potřeba vymodelovat jednoduché třídy reprezentující dokumenty. Jak jsem již zmiňoval, komunikace s Firebase databází probíhá za pomocí tzv. "JSON-like dokumentů", které podléhají určitým pravidlům. Nejzásadnějším omezením je velikost dokumentu v databázi, který nemůže přesáhnout 1MiB. Další rozhodující vlastnost je cena za každé provedené čtení, zápis a mazání dokumentu. Pro psaní dotazů a tím i modelování samotné struktury databáze tyto vlastnosti hrají klíčovou roli.

Pro optimalizaci počtu čtení, jsem na všech obrazovkách zobrazující vícero dokumentů implementoval nekonečné scrollování s výchozím limitem počtu dokumentů na 15. Kontroleru příslušného ListView widgetů jsem vytvořil jednoduchý scrollListener (ukázka 8), který si doptáváním a naslouchání na přidružený ViewModel (ukázka 9) obstará nová data.

[4] `https://pub.dev/packages/flutter_local_notifications`

V návaznosti na předchozí optimalizaci počtu čtení, další optimalizací pak byla nutnost duplicity určitých parametrů dokumentů z podružených kolekcí. Pro nastínění této problematiky uvažujme následovně: Máme vytvořených např. 40 různých prostředí s unikátní lokací a chceme na mapě zobrazit všechny lokace v podobě bodů. S předešlou optimalizací je načtených pouze prvních 15 prostředí (a stejný počet bodů na mapě), v případě dotázání bez optimalizace platíme 40 čtení z databáze + přihlášení. Elegantním řešením je duplikace parametrů polohy a názvu prostředí přímo do dokumentu uživatele. Přihlášením (tj. 1 přečtení dokumentu uživatele a zobrazení prvních 15 prostředí v ListView - dohromady 16 čtení) získám všechny polohy a názvy prostředí uživatele. Toto řešení však nese i negativa, nutnost rozšíření logiky dotazů, zvýšený počet zápisů při úpravě nebo vytvoření. Příklad takového volání můžete vidět v ukázce 5. Výsledné schéma databáze lze vidět v příloze A na obrázku A.1.

5.4 Mapy a GPS lokalizace

Součástí aplikace je i využití map pro zobrazení uživatelských prostředí. Balíček map google_maps_flutter, společně s balíčkem location[5] pro získání polohy uživatele, tvoří základ poskytované funkcionality. S využitím lokalizačního balíčku jsem byl schopen spravovat i přidělení oprávnění uživatelem (ukázka 10). Získání aktuální polohy je řešeno naslouchání na stream změny polohy (ukázka 11), která je předávána zpět do View.

Po zjištění, že nelze upravit základní info boxy v balíčku google_maps_flutter[6], jsem pro zachování již navrhnuté funkcionality (obrázek 3.5) zvolil balíček custom_info_window[7], který obaluje kontroler Google map a umožňuje zaměnit základní info box za mnou vytvořený widget se zachováním stejného ovládání. Ukázka kódu pro přidání bodů dle aktuálně zvolených typů filtrovaných prostředí tvoří ukázku 12.

5.5 QR čtečka a uložení

QR čtečka je funkcionalita implementovaná nad rámec zadání, která mě napadla již při samotném prototypování aplikace. Podobně jako body jednotlivých prostředí na mapě, QR čtečka slouží k rychlému přechodu na detail položky v případě většího množství položek.

Pro implementaci čtečky jsem využil balíčku qr_code_scanner[8], který mi

[5] `https://pub.dev/packages/location`
[6] `https://pub.dev/packages/google_maps_flutter`
[7] `https://pub.dev/packages/custom_info_window`
[8] `https://pub.dev/packages/qr_code_scanner`

poskytl metodu pro naslouchání datového streamu fotoaparátu a v případě detekce QR kódu následné uložení výsledku jako objekt Barcode nesoucího informace naskenovaného QR kódu (ukázka 13). Data naskenovaného objektu Barcode poté porovnám s kolekcí všech QR kódů daného prostředí a vyhodnotím zda uživatel naskenoval validní QR kód (ukázka 14).

Pro uložení QR kódu do zařízení včetně názvu a popisu položky jsem použil balíčků hned několik:

- qr_flutter[9]

  Balíček sloužící k vykreslení QR kódu z dat. V tomto případě data tvoří unikátní id položky, přidělené vytvořením záznamu v databázi (ukázka 15).

- screenshot[10]

  Jednoduchý balíček pro vytvoření jpg obrázku z vykreslených widgetů (ukázka 16).

- image_gallery_saver[11]

  Balíček pro uložení vytvořeného obrázku do zařízení (ukázka 17).

- permission_handler[12]

  Balíček pro správu oprávnění (ukázka 18).

5.6 Zobrazení naměřených dat

Nedílnou součástí kolekce dat je i jejich zobrazení. Implementace View pro jejich zobrazení byla jednou ze složitejších částí aplikace především kvůli zmatené dokumentaci použitého balíčku syncfusion_flutter_charts[13], který i přes svoje mírné nedostatky převyšoval funkčností všechna ostatní řešení.

Oprostíme-li se od logiky pro dynamické vytváření widgetů z mapy naměřených hodnot, pak samotná implementace widgetu grafu není tak rozsáhlá (ukázka 19). Jelikož mezi sbíraná data patří i fotografie, implementoval jsem jednoduchou galerii (obrázek 3.7c) s pomocí balíčku photo_view[14], který obaluje klasický Hero widget a přidává možnost přiblížení a rotace za pomocí klasických gest. Třídu obalující Hero widget použitou pro galerii nalezneme jako ukázku 21.

[9] `https://pub.dev/packages/qr_flutter`
[10] `https://pub.dev/packages/screenshot`
[11] `https://pub.dev/packages/image_gallery_saver`
[12] `https://pub.dev/packages/permission_handler`
[13] `https://pub.dev/packages/syncfusion_flutter_charts`
[14] `https://pub.dev/packages/photo_view`

Kapitola 6

Testování

V této kapitole se zabývám testováním aplikace za pomocí softwarových unit testů a čtyř dobrovolníků pro uživatelské testování. V závěru kapitoly pak vyhodnotím proběhlé testování.

6.1 Unit testy

Unit testy jsou vhodné pro odhalení chyb na úrovni kódu s atomizací přímo na jednotlivé metody či procedury. Unit testy lze spouštět při každé změně a tak zachovat stálou funkcionalitu.

Flutter poskytuje dva balíčky zajišťující testovatelnost aplikace, balíček flutter_test[1] pro samotné psaní testů a balíček test[2] poskytující přidané funkcionality pro vyhodnocování a testování widgetů. Flutter má pro testy účelový adresář test přímo v projektu, z kterého testy provádí. Pro zaregistrování testu je třeba každý název testovacího souboru ukončit _test.

V následujících podkapitolách popisuji unit testy, které pokrývají autentizační a databázovou službu.

6.1.1 Test autentizace

Pro otestování autentizace jsem vzhledem ke zvolenému přihlašování za pomocí Google účtu použil balíček google_sign_in_mocks[3] a firebase_auth_mocks[4],

[1] `https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html`
[2] `https://pub.dev/packages/test`
[3] `https://pub.dev/packages/google_sign_in_mocks`
[4] `https://pub.dev/packages/firebase_auth_mocks`

které zastřešují balíčky mockito a použité balíčky firebase_auth a google_sign_in. Díky těmto balíčkům jsem byl schopný namockovat testovacího uživatele 22 a za pomocí mockované autentizační služby 23 otestovat identické metody jako při standardním použití.

Test přihlášení a odhlášení pomocí mockovaného uživatele lze vidět v přílohách 24 a 25

6.1.2 Test třídy database_service

Další testy proběhly na testování služby pro komunikaci s databází. Pro tyto testy jsem zvolil balíček cloud_firestore_mocks[5], který umožňuje na mockované Firestore databázi provádět omezené množství úkonů oproti klasické službě. Využití tohoto balíčku pro mě bylo především o vyzkoušení struktury dotazů před zahájením samotného vývoje.

Zde najdeme porovnání mockovaného testu aktualizace uživatele (ukázka 26) a výsledné metody (ukázka 27). Dalším testem je vytvoření dokumentu v uživatelské kolekci (ukázka 28).

6.2 Testování během vývoje

Během vývoje probíhaly průběžné smoke testy nově implementovaných funkcionalit a obrazovek prováděné kolegy z ročníku. Tyto smoke testy odhalily drobné chyby s textací nebo špatné vykreslení widgetů při jiných velikostech zařízení, v pokročilé fázi vývoje pak odhalily zásadní chybu při otevření různých obrazovek, které se kvůli buildu aplikace v release módu načítaly rychleji než data samotná a takto načtená obrazovka způsobila pád aplikace. Všechny tyto chyby byly opraveny.

6.3 Uživatelské testy

Uživatelské testování je vhodná metoda pro odhalení chyb, které se při samotném vývoji dají snadno přehlédnout kvůli vlastní zaujatosti a znalosti problematiky. Pro získání kvalitní odezvy je vhodné zvolit testery, které zapadají do cílové skupiny uživatelů. Díky jinému úhlu pohledu a neúplné znalosti samotné aplikace pak právě uživatelské testování ušetří budoucí práci.

V následujících podkapitolách Vám představím testery, testovací scénaře, doplňující otázky a tabulku jejich vyhodnocení.

[5] `https://pub.dev/packages/cloud_firestore_mocks`

6.3.1 Testeři

- Tester #1

  Tento tester se podílel na testování prototypu aplikace.

  - Zařízení: Samsung Galaxy S10+
  - Verze Android: 11

- Tester #2

  - Zařízení: OnePlus 5T
  - Verze Android: 10

- Tester #3

  Tento tester trpí poruchou barvocitu.

  - Zařízení: Xiaomi Mi A1
  - Verze Android: 9

- Tester #4

  Tento tester je absolventem bakalářského programu "Rostlinná produkce" České zemedělské univerzity v Praze, a v současnosti studentem magisterského programu "Rostlinolékařství".

  - Zařízení: OnePlus 6T
  - Verze Android: 10

6.3.2 Testovací scénáře

Scénáře jsem vytvořil tak, aby se uživatelé při průchodu přiblížili reálnému cyklu použití a zároveň byly pokryty případy užití (obrázek A.2). Detaily následujících scénářů nalezneme v příloze C.

- Spustit aplikaci a přihlásit se
- Vytvořit nové prostředí a zobrazení v seznamu i na mapě
- Filtrovat prostředí
- Vytvořit novou položku a zobrazit její detail
- Uložit QR kód položky
- Zobrazit detail položky pomocí QR kódu
- Editace položky

- Přidat měření a zobrazit vývoj sledovaných parametrů
- Vytvořit archivační adresář
- Archivovat položku
- Zobrazit vývoj sledovaných parametrů archivované položky
- Editace prostředí
- Editace archivu
- Odhlásit se

6.3.3 Doplňující otázky

Každému testerovi jsem mimo jiné položil i následující otázky pro získání obsáhlejší zpětné vazby z průchodů testů:

1. Co byste v aplikaci změnili nebo zlepšili?
2. Co Vám v aplikaci chybí za funkcionality?
3. Na jaký problém/problémy jste během testování narazili?
4. Co se Vám v aplikaci líbilo?

6.3.4 Vyhodnocení scénářů a otázek testery

Testeři během testování nenarazili na problémy vyšší závažnosti, které by měly vliv na schopnost dokončit průchod scénáři.

Zpětná vazba z položených otázek:

- Tester #1
  1. “Tlačítko odhlášení je zbytečné na obrazovce nastavení, kde není co nastavit.”
  2. Bez odpovědi.
  3. “Lehce mě zmátl průchod některými scénáři.”
  4. “Jednoduchost použití aplikace.”
- Tester #2
  1. “Tlačítko pro zvolení vypnutí a zapnutí notifikací nevypadá přidruženě k tomu co ovládá.”

2. “Chybí možnost tmavého motivu.”
3. “Je třeba zavírat klávesnici pro potvrzení formulářů při vyplňování parametrů.”
4. “Vzhled aplikace.”

- Tester #3

1. “Při prvním použití aplikace nejasnost v rozlišení, zda se nacházím v prostředí nebo už na detailu položky. Po uložení QR kódu do zařízení bych očekával přesměrování zpět do prostředí. Nevýrazný přechod na detail položky při použití QR čtečky.”
2. Bez odpovědi.
3. “Nejasnost některých kroků v testovacích scénářích.”
4. “Vzhled a rychlost odezvy. Myšlenka - pohodlné použití QR kódu pro správu rostlin.”

- Tester #4

1. “Odebrat povinnost vyplnit všechny sledované parametry při ukládání měření.”
2. “Chybí možnost úpravy/přidání vlastních parametrů pro sledování. Očekávala bych možnost hromadného uložení QR kódů pro celé prostředí. Přidání nápovědy při vytváření prostředí a položky.”
3. “Nejasné kroky scénářů při referování na prostředí. Neustálé otevírání klávesnice při snaze dokončit měření.”
4. “Prostup na jednotlivé položky pomocí QR kódu. Velký výběr z sledovaných parametrů.”

6.4 Vyhodnocení testů

Z výše zmíněných výsledků uživatelských testů se ukázalo, že největším problémem byly samotné scénáře, přesto uživatelé byli s aplikací spokojení. Ze zbylých relevantních problémů a návrhů jsem vytvořil tabulku požadavků společně s možným řešením vhodným pro budoucí zapracování (tabulka 6.1).

Požadavek	Navrhnuté řešení
Skrýt klávesnici	Přidat focus pro všechny existující formuláře
Odebrat povinnost vyplnit všechny sledované parametry	Odstranit validaci existence hodnoty pro formulář uložení měření
Odhlášení v kartě nastavení	Zapracovat funkční požadavek F14
Oddělenost tlačítka pro zapnutí a vypnutí notifikací	Přidat ohraničení nastavení notifikací ovládané tlačítkem

Požadavek	Navrhnuté řešení
Přesměrování po uložení	Po úspěšném uložení obrázku do zařízení se vrátit na původní kontext
Vyjasnění funkčností	Přidat informační poučky k obrazovkám zahrnující případy užití UC6, UC10, UC11
Hromadné uložení QR kódů	Přidat k výběru úpravy a odstranění možnost uložení všech QR kódů položek
Úprava parametrů pro sledování	Na kartě nastavení přidat možnost úpravy předdefinovaných parametrů pro sledování

Tabulka 6.1: Vzniklé požadavky a navrhnuté řešení

Kapitola 7

Závěr

Podařilo se mi zapracovat téměř všechny aplikační požadavky. V aktuální verzi aplikace je uživateli umožněno založit různé typy prostředí (požadavek F3 kapitola 2.2) společně s jejich umístěním na mapě. V prostředí lze dále vytvářet položky (rostlina, vodní plocha) a individuálně volit sledované parametry (požadavek F5 kapitola 2.2) s možností nastavení notifikací ke kontrole. U takto vytvořených položek může uživatel zaznamenávat jednotlivá měření dle zvolených parametrů a výsledky měření zobrazovat v podobě grafů či pořízených fotografií. Pro rychlou orientaci v prostředí lze položky načítat pomocí QR čtečky. Po ukončení sledování položky má uživatel na výběr mezi odstraněním položky a její archivací v archivačním adresáři.

Rozhodnutí neimplementovat případ užití UC3, tedy možnost výběru typu datového připojení použitého pro synchronizaci dat, vzešlo z použití větší komprese pro nahrávané fotografie, což má za důsledek nižší datovou zátěž. Druhý a zároveň poslední neimplementovaný případ užití je UC19 - možnost tisku QR kódu - jelikož se během testování prototypu ukázalo, že se jedná o zřídka používanou volbu.

Výsledný vzhled aplikace se od prototypu definovaného v sekci 2.3 výrazně neliší, kromě absence neimplementovaných případů užití, díky dodržování zásad Material Designu již při vytváření prototypu.

7.1 Budoucí práce

Pro další rozvoj této aplikace se primárně nabízí zapracování chyb a navrhnutých vylepšení z proběhlých uživatelských testů viz tabulka 6.1. Dále pak po přechodu všech použitých balíčku na Flutter verzi 2 může dojít k migraci aplikace do tzv. “null safety”, která s sebou přinese kontrolu null a tím větší stabilitu aplikace.

Veliký krok pro další posunutí aplikace by pak byla integrace iOS platformy, vytvoření sdílených prostředí mezi uživateli a výsledná publikace na App Store a Google Play.

Literatura

[1] U. S. G. U. Army. Defense acquisition university press - system engineering fundamentals. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf, 2001. [Navštíveno: 10.12.2020].

[2] J. Bissekkou. App architecture: Mvvm in flutter using dart streams. https://quickbirdstudios.com/blog/mvvm-in-flutter/, 2018. [Navštíveno: 19.12.2020].

[3] U. Eriksson. Why is the difference between functional and non-functional requirements important? https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/, 2012. [Navštíveno: 10.12.2020].

[4] Google. Choose a database: Cloud firestore or realtime database. https://firebase.google.com/docs/database/rtdb-vs-firestore. [Navštíveno: 25.12.2020].

[5] Google. Flutter documentation. https://flutter.dev/docs. [Navštíveno: 19.12.2020].

[6] Google. Material system - introduction. https://material.io/design/introduction. [Navštíveno: 16.10.2020].

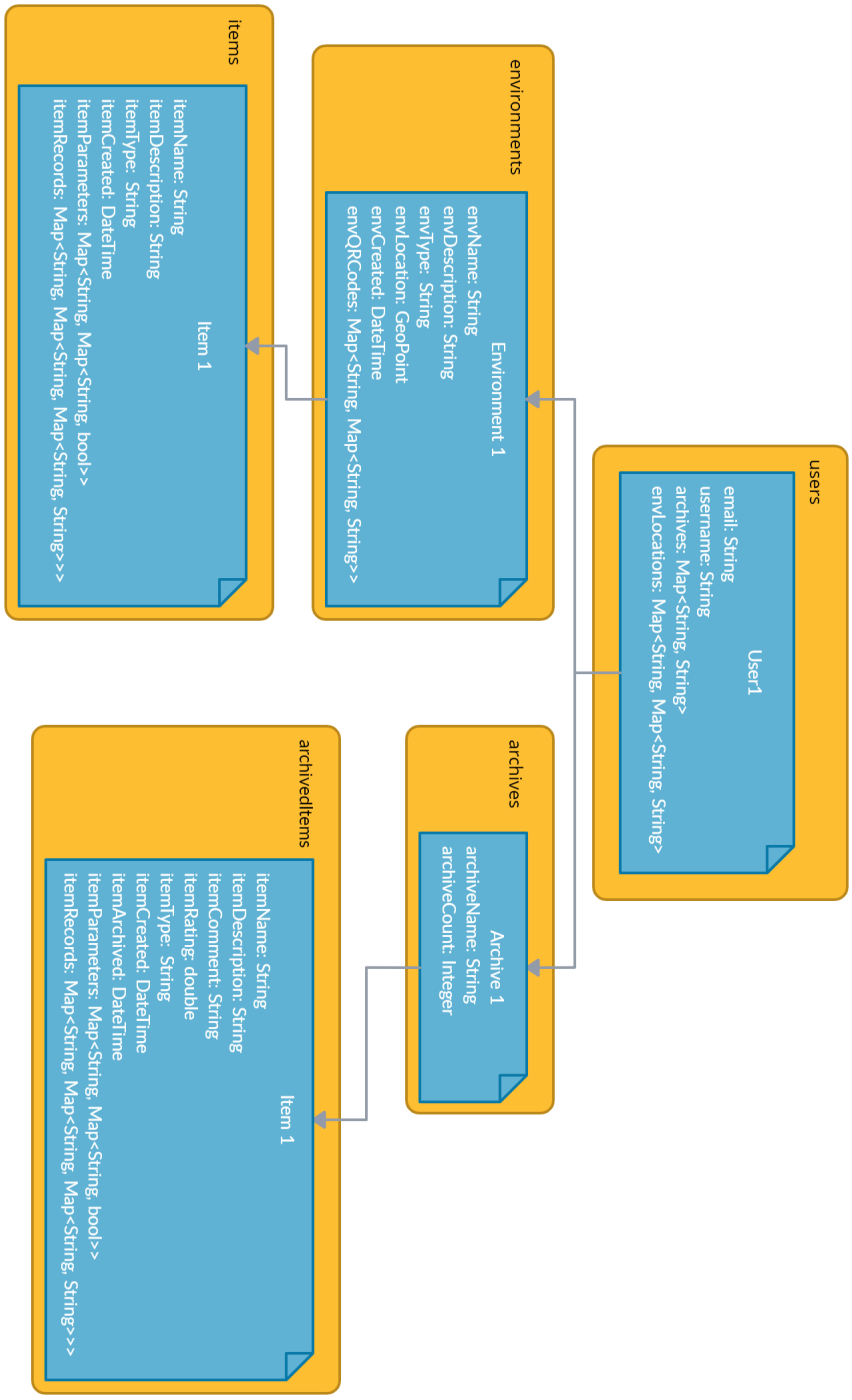
[7] Google. Simple app state management. https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple. [Navštíveno: 19.12.2020].

[8] K. Kogut. Getting started with flutter bloc. https://www.netguru.com/codestories/flutter-bloc/, 2019. [Navštíveno: 19.12.2020].

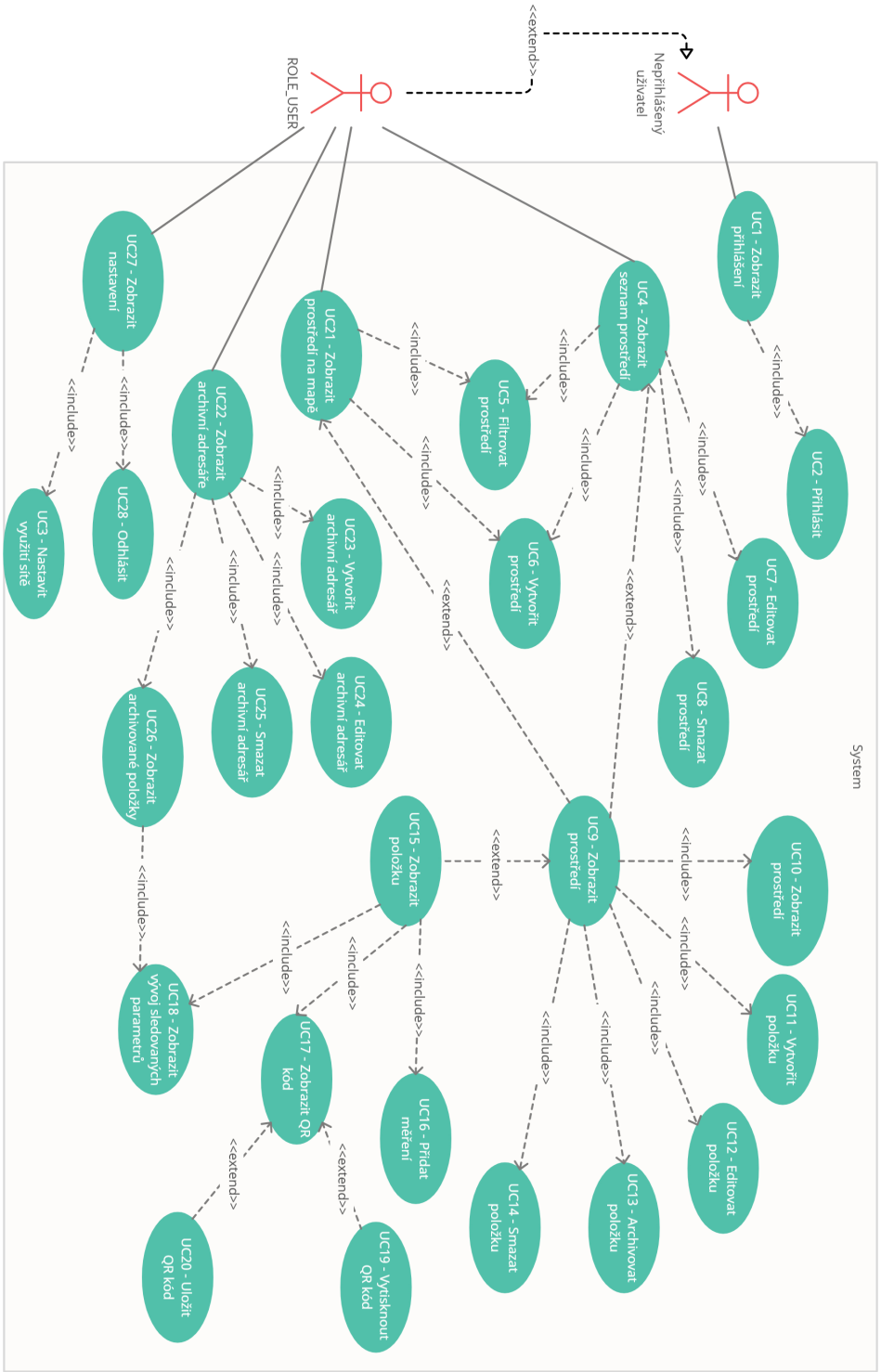
[9] G. VARMA. Firebase vs aws. https://embersoftware.com.au/firebase-vs-aws/, 2020. [Navštíveno: 25.12.2020].

Příloha A

Obrázky



Obrázek A.1: Schéma NoSQL databáze



Obrázek A.2: Diagram případů užití

Příloha B

Ukázky kódu

B.1 Implementace

B.1.1 Služby

```
Future setupLocator() async {
    locator.registerLazySingleton(() => DatabaseService());
    locator.registerLazySingleton(() => EnumConvertorService());
    locator.registerLazySingleton(() => Location());
    locator.registerLazySingleton(() => NotificationsService());
}
```

Kód 1: GetIt - Registrace objektů

```
class EnvironmentsProvider with ChangeNotifier {
    final _dbService = locator<DatabaseService>();
    final _enumConvertorService = locator<EnumConvertorService>();
    .
    .
}
```

Kód 2: GetIt - Použití

```
Future<DocumentSnapshot> getUserEnviroment(String envId) async {
    final enviromentRef = firestore
        .collection("users")
        .doc(FirebaseAuth.instance.currentUser.uid)
        .collection("enviroments")
        .doc(envId);

    return enviromentRef.get();
}
```

Kód 3: Dotaz getUserEnviroment

```
Future<Enviroment> getEnviromentModel(String envId) async {
    var enviroment = await _dbService.getUserEnviroment(envId);

    return Enviroment(
        envId: enviroment.id.toString(),
        envName: enviroment['envName'],
        envDescription: enviroment['envDescription'],
        envType: locator<EnumConvertor>()
            .getEnumFromStringEnviroment(enviroment['envType']),
        envLocation: enviroment['envLocation'],
        envQRCodes: enviroment['envQRCodes']
            ?.cast<String, Map<String, String>>()
    );
}
```

Kód 4: Serializace getUserEnviroment

```
Future createUserEnviroment(
    String envName,
    String envDescription,
    String envType,
    DateTime envCreated,
    GeoPoint envLocation,
    ) async {
    final CollectionReference enviromentCollectionRef = firestore
        .collection("users")
        .doc(FirebaseAuth.instance.currentUser.uid)
        .collection("enviroments");

    return await enviromentCollectionRef.add({
        "envName": envName,
        "envDescription": envDescription,
        "envType": envType,
        "envCreated": envCreated,
        "envLocation": envLocation,
        "envQRCodes": {},
    }).then((value) {
        // add enviroment location to user's envLocations
        firestore
            .collection("users")
            .doc(FirebaseAuth.instance.currentUser.uid)
            .set(
        {
            "envLocations": {
            "${value.id}": {
                "envName": envName,
                "envDescription": envDescription,
                "envType": envType,
                "envLocation": envLocation,
            },
            },
        },
        SetOptions(merge: true),
        );
    });
}
```

Kód 5: Deserializace a dotaz

```
Future login(BuildContext context) async {
    try {
        isSigningIn = true;

        final user = await googleSignIn.signIn();
        if (user == null) {
        isSigningIn = false;
        return;
        } else {
        final googleAuth = await user.authentication;
        final credential = GoogleAuthProvider.credential(
            accessToken: googleAuth.accessToken,
            idToken: googleAuth.idToken,
        );
        await FirebaseAuth.instance.signInWithCredential(credential);
        isSigningIn = false;
        await updateUser();
        }
    } on PlatformException catch (e) {
        isSigningIn = false;
        switch (e.code) {
        case "network_error":
            errorDialog(context, "Chyba internetoveho pripojeni.");
            break;
        default:
            errorDialog(context, e.code.toString());
        }
    }
}
```

Kód 6: Autentizace

```
await _flutterLocalNotificationsPlugin.zonedSchedule(
    itemHash,
    "Plánované měření",
    "Nastal čas sběru dat pro: $itemName",
    dayFrequency > 1
        ? _nextDateTimeCalc(hours, minutes, dayFrequency)
        : _nextDateTime(hours, minutes),
    notifDetails,
    uiLocalNotificationDateInterpretation:
        UILocalNotificationDateInterpretation.wallClockTime,
    androidAllowWhileIdle: true,
    matchDateTimeComponents: dayFrequency > 1
        ? DateTimeComponents.dayOfWeekAndTime
        : DateTimeComponents.time,
);
```

Kód 7: Plánování notifikace

B.1.2 Datový model a optimalizace

```
void scrollListener() {
    if (scrollController.offset >=
        scrollController.position.maxScrollExtent &&
        !scrollController.position.outOfRange) {
        if (widget.enviromentsProvider.hasNextEnviroment) {
        widget.enviromentsProvider.getNextEnviroments();
        }
    }
}
```

Kód 8: Scroll listener

```
Future getNextEnviroments() async {
    if (_isFetchingEnviroments) return;

    try {
        _isFetchingEnviroments = true;
        final snap = await _dbService.getUserEnviroments(
            documentLimit,
            _enviromentsSnapshot.isNotEmpty
                ? _enviromentsSnapshot.last : null,
            getKeysFromTypes());
        _enviromentsSnapshot.addAll(snap.docs);

        if (snap.docs.length < documentLimit) _hasNextEnviroment = false;
        notifyListeners();
    } catch (error) {
        debugPrint("---|| Failed to fetch enviroments: $error");
    }
    _isFetchingEnviroments = false;
}
```

Kód 9: EnvironmentsProvider - getNextEnviroments

B.1.3 Mapy a GPS lokalizace

```
Future getPermissions() async {
    _serviceEnabled = await _location.serviceEnabled();
    if (!_serviceEnabled) {
        _serviceEnabled = await _location.requestService();
        if (!_serviceEnabled) {
        return;
        }
    }

    _permissionGranted = await _location.hasPermission();
    if (_permissionGranted == PermissionStatus.denied) {
        _permissionGranted = await _location.requestPermission();
        if (_permissionGranted != PermissionStatus.granted) {
        return;
        }
    }
}
```

Kód 10: Dotázání na oprávnění

```
Future updateLocation() async {
    _location.onLocationChanged.listen((LocationData currentLocation) {
        latitudeCurrent = currentLocation.latitude;
        longitudeCurrent = currentLocation.longitude;
    });
}
```

Kód 11: Naslouchání na změnu polohy

```
mapsProvider.envLocations?.forEach((key, value) {
    if (mapsProvider.displayTypes[value["envType"]]) {
    GeoPoint location = value["envLocation"];
    _markers.add(Marker(
        markerId: MarkerId(key),
        position: LatLng(location.latitude, location.longitude),
        onTap: () {
        _mapController.addInfoWindow(
            Container(
            child: MyInfoWindowWidget(
                name: value["envName"],
                description: value["envDescription"],
                type: value["envType"],
                id: key),
            ),
            LatLng(location.latitude, location.longitude),
        );
        }));
    }
});
```

Kód 12: Přidání bodů na mapu

B.1.4 QR čtečka a uložení

```
controller.scannedDataStream.listen((scanData) {
    setState(() {
        result = scanData;
    });
});
```

Kód 13: QR scanner

```
child: widget.enviroment.envQRCodes
        .containsKey(result.code)
    ? Column(
        children: [
        Text(
            'Kliknutím přejdete na položku',
            textAlign: TextAlign.center,
            style: TextStyle(
                color: Colors.green),
        ),
        ],
    )
    : Column(
        children: [
        Text(
            'Položka nenalezena',
            textAlign: TextAlign.center,
            style:
                TextStyle(color: Colors.red),
        ),
        ],
    ),
    ),
    onPressed: () {
    if (widget.enviroment.envQRCodes
    .containsKey(result.code)) {
    Navigator.pop(context);
    Navigator.push(
    context,
    MaterialPageRoute(
    builder: (context) => ItemWidget(
        envName: widget.enviroment.envName,
        envId: widget.enviroment.envId,
        itemId: result.code,
    )),
    );
    }
},
```

Kód 14: Kontrola validity QR kódu

```
QrImage(
    data: widget.item.itemId,
    version: QrVersions.auto,
    semanticsLabel: widget.item.itemName,
    padding: const EdgeInsets.all(20),
),
```

Kód 15: Widget pro zobrazení QR

```
screenshotController.capture(
  pixelRatio: 3,
  delay: Duration(milliseconds: 10),
).then((Uint8List image) async {
  _imageFile = image;
  _saveImage(image);
})
```

Kód 16: Logika screenshotu widgetů

```
_saveImage(Uint8List image) async {
    await ImageGallerySaver.saveImage(image).then((value) {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
        behavior: SnackBarBehavior.floating,
        content: Text('QR kód uložen', textAlign: TextAlign.center),
        ));
    });
}
```

Kód 17: Funkce pro uložení QR kódu

```
@override
void initState() {
    super.initState();
    _requestPermission();
}

_requestPermission() async {
    await [Permission.storage].request();
}
```

Kód 18: Dotázání práv správy uložiště

B.1.5 Zobrazení naměřených dat

```
Container(
    height: MediaQuery.of(context).size.height * 0.65,
    child: SfCartesianChart(
        primaryXAxis: CategoryAxis(),
        primaryYAxis: NumericAxis(),
        tooltipBehavior: TooltipBehavior(enable: true),
        zoomPanBehavior: ZoomPanBehavior(
        enablePinching: true,
        enablePanning: true,
        enableDoubleTapZooming: true,
        zoomMode: ZoomMode.xy,
        ),
        series: <LineSeries<_CollectedData, String>>[
        LineSeries<_CollectedData, String>(
            dataSource: _data,
            sortingOrder: SortingOrder.ascending,
            sortFieldValueMapper: (_CollectedData data, _) =>
                DateFormat('dd.MM.yyyy \n HH:mm:ss', 'cs')
                    .format(data.date),
            xValueMapper: (_CollectedData data, _) =>
                DateFormat('dd.MM.yyyy \n HH:mm:ss', 'cs')
                    .format(data.date),
            yValueMapper: (_CollectedData data, _) => data.value,
            dataLabelSettings: DataLabelSettings(isVisible: true),
            markerSettings: MarkerSettings(isVisible: true)),
        ],
    ),
),
```

Kód 19: Widget grafu

```
Hero(
    tag: DateTime.fromMillisecondsSinceEpoch(
        int.parse(key2) + value2["value"].hashCode,
    ),
    child: Container(
        padding: const EdgeInsets.only(bottom: 15),
        width: MediaQuery.of(context).size.width * 0.7,
        child: AspectRatio(
            aspectRatio: 16 / 10,
            child: Image.network(
            value2["value"],
            fit: BoxFit.fitWidth,
            loadingBuilder: (_, child, chunk) =>
                chunk != null ? const Text("loading") : child,
            ),
        )),
),
```

Kód 20: Hero widget galerie

```
class HeroPhotoViewRouteWrapper extends StatelessWidget {
    const HeroPhotoViewRouteWrapper({
        @required this.imageProvider,
        this.backgroundDecoration,
        this.minScale,
        this.maxScale,
    });

    final ImageProvider imageProvider;
    final BoxDecoration backgroundDecoration;
    final dynamic minScale;
    final dynamic maxScale;

    @override
    Widget build(BuildContext context) {
        return Container(
        constraints: BoxConstraints.expand(
            height: MediaQuery.of(context).size.height,
        ),
        child: PhotoView(
            imageProvider: imageProvider,
            backgroundDecoration: backgroundDecoration,
            minScale: minScale,
            maxScale: maxScale,
            heroAttributes: const PhotoViewHeroAttributes(),
        ),
        );
    }
}
```

Kód 21: Hero widget v galerii

B.2 Unit testy

```
final testUser = MockUser(
    isAnonymous: false,
    uid: '12345TESTUID',
    email: 'test@testmail.com',
    displayName: 'Test Tester',
    phoneNumber: '123456789',
);
```

Kód 22: User mock

```
MockGoogleSignIn googleSignIn;

setUp(() {
    googleSignIn = MockGoogleSignIn();
});
```

Kód 23: Auth mock

```
test('Login with google', () async {
    final signInAccount = await googleSignIn.signIn();
    final signInAuthentication = await signInAccount.authentication;
    final AuthCredential credential = GoogleAuthProvider.credential(
        accessToken: signInAuthentication.accessToken,
        idToken: signInAuthentication.idToken,
    );

    final auth = MockFirebaseAuth(mockUser: testUser);
    final result = await auth.signInWithCredential(credential);
    final user = result.user;

    expect(signInAuthentication, isNotNull);
    expect(googleSignIn.currentUser, isNotNull);
    expect(user.displayName, equals('Test Tester'));
    expect(signInAuthentication.accessToken, isNotNull);
    expect(signInAuthentication.idToken, isNotNull);
});
```

Kód 24: Login with Google account test

```
test('Null after logout', () async {
    final auth = MockFirebaseAuth(signedIn: true, mockUser: testUser);
    final user = auth.currentUser;

    await auth.signOut();

    expect(auth.currentUser, isNull);
    expect(auth.authStateChanges(), emitsInOrder([user, null]));
});
```

Kód 25: Logout test

```
test('DB test - update user data', () async {
    final instance = MockFirestoreInstance();
    final DocumentReference userRef = instance
            .collection("users")
            .doc(userUid);

    await userRef.set(
        {
        "username": "testName",
        "email": "email@mail.com",
        },
        SetOptions(merge: true),
    );

    final docSnap = await instance
            .collection('users')
            .doc(userUid)
            .get();

    expect(docSnap.get("username"), equals("testName"));
    expect(docSnap.get("email"), equals("email@mail.com"));
});
```

Kód 26: Update user test

```
Future updateUserData(
    String username,
    String email,
    ) async {
    final CollectionReference userCollection = firestore
        .collection("users");

    return await userCollection
        .doc(FirebaseAuth.instance.currentUser.uid)
        .set({
            "username": username,
            "email": email,
        }, SetOptions(merge: true))
    .then((value) => print("---|| User: $email updated"))
    .catchError((error) => print("---|| Failed to add user: $error"));
}
```

Kód 27: Update user final

```
test('DB test - create enviroment', () async {
    final instance = MockFirestoreInstance();
    await instance
        .collection("users").doc(userUid)
        .collection("enviroments")
        .doc(envUid).set({
        "envName": "prostredi",
        "envDescription": "popis",
        "envType": "Sklenik",
        "envCreated": "29 April 2021 at 11:41:32 UTC+2",
        "envLocation": "50.04277545210549, 14.5124351978302",
    });
    await instance.collection("users").doc(userUid).set(
        {
        "envLocations": {
            "$envUid": {
            "envName": "prostredi",
            "envDescription": "popis",
            "envType": "Sklenik",
            "envLocation": "50.04277545210549, 14.5124351978302",
            },
        },
        },
        SetOptions(merge: true),
    );

    final docSnap = await instance
        .collection('users').doc(userUid).get();
    final docSnap2 = await instance
        .collection('users').doc(userUid)
        .collection("enviroments").doc(envUid).get();

    expect(docSnap.get("username"), equals("testName"));
    expect(docSnap.get("email"), equals("email@mail.com"));
    expect(
        docSnap.get("envLocations"),
        equals({
            "testenvUid": {
            "envName": "prostredi",
            "envDescription": "popis",
            "envType": "Sklenik",
            "envLocation": "50.04277545210549, 14.5124351978302",
            },
        }));
    expect(docSnap2.get("envName"), equals("prostredi"));
    expect(docSnap2.get("envLocation"),
    equals("50.04277545210549, 14.5124351978302"));
});
```

Kód 28: Create environment test

Příloha C

Testovací scénáře

C.1 Spustit aplikaci a přihlásit se

1. Zapnout aplikaci AgroTracker
2. Klepnout na tlačítko “Přihlásit pomocí Google”
3. Zobrazí se dialogové okno s vybráním Google účtu pro přihlášení
4. Vybrat účet
5. Uživatel se nachází na kartě “Prostředí”

C.2 Vytvořit nové prostředí a zobrazení v seznamu i na mapě

1. Přejít na kartu “Prostředí”
2. Klepnout na tlačítko “+”
3. Zadat parametry
4. Klepnout na tlačítko “Další”
5. Pomocí oritentace na mapě vybrat umístění prostředí
6. Klepnout na tlačítko “Vytvořit”
7. Zobrazí se dialogové okno s potvrzením
8. Klepnout na “Potvrdit”

9. Vytvořené prostředí je zobrazeno v seznamu
10. Přejít na kartu “Mapy”
11. Vytvořené prostředí je zobrazeno na mapě jako bod

C.3 Filtrovat prostředí

1. Přejít na kartu “Prostředí” nebo “Mapy”
2. Klepnout na ikonu filtrování
3. Zobrazí se dialogové okno s výběrem typů prostředí
4. Vybrat typy
5. Klepnout na “Filtrovat”
6. Na kartě “Prostředí” a “Mapy” se zobrazí pouze vybrané typy prostředí
7. Podle počtu zvolených typů se změní ikona filtrování

C.4 Vytvořit novou položku a zobrazit její detail

1. Přejít do předešle vytvořeného prostředí vybráním v seznamu, případně vybrat bod vyzobrazený na mapě a klepnout na “Přejít na prostředí”
2. Zobrazí se obrazovka prostředí
3. Klepnout na tlačítko “+”
4. Zadat parametry
5. Klepnout na tlačítko “Další”
6. Vybrat parametry k sledování
7. Klepnout na tlačítko “Další”
8. Vybrat si četnost výzev
9. Klepnout na tlačítko “Vytvořit”
10. Zobrazí se dialogové okno s potvrzením
11. Klepnout na “Potvrdit”
12. Vytvořená položka je zobrazena v seznamu prostředí
13. Klepnout na kartu nově vytvořené položky
14. Zobrazí se obrazovka detailu položky

C.5 Uložit QR kód položky

1. Přejít do předešle vytvořeného prostředí
2. Klepnout na ikonu “Více”
3. Zobrazí se dialogové okno s výběrem
4. Klepnout na tlačítko “QR kód”
5. Zobrazí se obrazovka s QR kódem
6. Klepnout na tlačítko “Uložit do zařízení”

C.6 Zobrazit detail položky pomocí QR kódu

1. Přejít do předešle vytvořeného prostředí
2. Klepnout na ikonu “QR kód”
3. Zobrazí se obrazovka čtečky QR kódů
4. Namířit kamerou na QR kód
5. Text “Namiřte kamerou na QR kód” se změní na “Kliknutím přejdete na položku”
6. Klikněte na text
7. Zobrazí se obrazovka detailu položky

C.7 Editace položky

1. Přejít do předešle vytvořeného prostředí
2. Klepnout na ikonu “Více”
3. Zobrazí se dialogové okno s výběrem
4. Klepnout na tlačítko “Upravit”
5. Upravit parametry
6. Klepnout na tlačítko “Uložit”
7. Zobrazí se dialogové okno s potvrzením
8. Klepnout na tlačítko “Potvrdit”
9. Položka je upravena

C.8 Přidat měření a zobrazit vývoj sledovaných parametrů

1. Přejít na detail položky
2. Klepnout na tlačítko “+”
3. Zadat naměřené hodnoty, případně připojit fotografie
4. Klepnout na tlačítko “Dokončit měření”
5. Zobrazí se dialogové okno s potvrzením
6. Klepnout na “Potvrdit”
7. Zobrazí se detail položky
8. Klepnout na ikonu “Graf”
9. Zobrazí se obrazovka s grafem naměřených hodnot / galerie fotografií

C.9 Vytvořit archivační adresář

1. Přejít na kartu “Archiv”
2. Klepnout na tlačítko “+”
3. Zadat parametr
4. Klepnout na tlačítko “Vytvořit”
5. Zobrazí se dialogové okno s potvrzením
6. Klepnout na tlačítko “Potvrdit”
7. Vytvořený archiv je zobrazen v seznamu

C.10 Archivovat položku

1. Přejít do předešle vytvořeného prostředí
2. Klepnout na ikonu “Více”
3. Zobrazí se dialogové okno s výběrem
4. Klepnout na tlačítko “Archivovat”

5. Zadat parametry
6. Klepnout na tlačítko “Archivovat”
7. Zobrazí se dialogové okno s potvrzením
8. Klepnout na tlačítko “Potvrdit”
9. Položka se archivovala
10. Přejít na kartu “Archiv”
11. Přejít do příslušného archivačního adresáře
12. Archivovaná položka se nachází v seznamu

C.11 Zobrazit vývoj sledovaných parametrů archivované položky

1. Přejít do vytvořeného archivačního adresáře
2. Klepnout na kartu archivované položky
3. Klepnout na ikonu “Graf”
4. Zobrazí se obrazovka s grafem naměřených hodnot / galerie fotografií

C.12 Editace prostředí

1. Přejít na kartu “Prostředí”
2. Klepnout na ikonu “Více”
3. Zobrazí se dialogové okno s výběrem
4. Klepnout na tlačítko “Upravit”
5. Upravit parametry
6. Klepnout na tlačítko “Uložit”
7. Zobrazí se dialogové okno s potvrzením
8. Klepnout na tlačítko “Potvrdit”
9. Prostředí je upraveno

C.13 Editace archivu

1. Přejít na kartu “Archiv”
2. Klepnout na ikonu “Více”
3. Zobrazí se dialogové okno s výběrem
4. Klepnout na tlačítko “Upravit”
5. Upravit parametr
6. Klepnout na tlačítko “Uložit”
7. Zobrazí se dialogové okno s potvrzením
8. Klepnout na tlačítko “Potvrdit”
9. Archiv je upraveno

C.14 Odhlásit se

1. Přejít na kartu “Nastavení”
2. Klepnout na tlačítko “Odhlásit se”
3. Zobrazí se dialog s potvrzením
4. Klepnout na “Odhlásit”

Příloha D

Kompilace Android aplikace

D.1 Prerekvizity

- Flutter SDK[1]
- Android SDK - tato instalace je součástí instalace Flutter SDK

D.2 Postup pro kompilaci

Pro kompilaci projektu pro Android je nutné mít nainstalované Flutter SDK a Android SDK viz podkapitola D.1. Do příkazového řádku, v umístění adresáře projektu je poté nutné zadat následující příkazy ve zobrazeném pořadí:

```
1 ...\Zdrojove kody aplikace\agroTracker> flutter pub get
2 ...\Zdrojove kody aplikace\agroTracker> flutter build apk
```

Kód 29: Kompilace Android aplikace

Release build aplikace se po kompilaci nachází v \...\agroTracker\build\app\outputs \flutter-apk\app-release.apk.

[1] https://flutter.dev/docs/get-started/install

Příloha E

Obsah elektronické přílohy

- `Zdrojove kody aplikace.zip`
  - `agroTracker` ..... Praktická část
    - `android`
    - `assets` ..... Doplňový obsah aplikace
    - `ios`
    - `lib` ..... Zdrojový kód aplikace
    - `tests` ..... Testy aplikace
    - `pubspec.lock`
    - `pubspec.yaml` ..... Soubor dependencí
    - `README.md` ..... Návod pro kompilaci práce
  - `bp_latex` ..... LaTeX práce včetně šablony
  - `Prototyp.rp` ..... Prototyp aplikace vytvořený v Axure RP
  - `README.md`