VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MOBILNÍ APLIKACE PRO PODPORU REVIZNÍCH KONTROL NA STAVBÁCH

MOBILE APPLICATION FOR REVISION CONTROLS ON CONSTRUCTION SITES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ŠIMON ŠESTÁK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN HRUBÝ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Šesták Šimon**
Program: Informační technologie
Název: **Mobilní aplikace pro podporu revizních kontrol na stavbách**
Mobile Application for Revision Controls on Construction Sites
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte programování aplikací pro iOS, knihovny pro analýzu obrazu, knihovny pro ukládání dat a knihovny pro správu souborů PDF.
2. Navrhněte aplikaci, která bude podporovat pracovníka při revizním sběru dat na stavbách. Smyslem aplikace má být pořizování dat z měření kvality stavebních ploch (zdí, střech, podlah, apod.). Aplikace musí veškerá data archivovat lokálně i na vzdáleném úložišti. Aplikace o revizní kontrole vygeneruje protokol ve formátu PDF.
3. Aplikaci implementujte pro operační systém iOS.
4. Aplikaci testujte na simulované revizní kontrole, avšak se zapojením reálných měřících přístrojů.

Literatura:

- Keur, Ch., Hillegass, A.: iOS Programming: The Big Nerd Ranch Guide, Big Nerd Ranch Guides; 6 edition (January 6, 2017), ISBN-13: 978-0134682334

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hrubý Martin, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Bakalárska práca sa zaoberá vytvorením návrhu aplikácie pre špecifickú skupinu užívateľov a jej implementácie pre operačný systém iOS.

Vyvinutá aplikácia užívateľom umožní jednoducho vyplniť potrebné dáta, nahrať fotografie pre vytvorenie výstupného protokolu a dáta archivovať lokálne, ale aj na vzdialenom serveri. V jednotlivých kapitolách sú postupne opísané metódy ukladania a získania uložených správ zo vzdialeného servera, metóda detekcie a klasifickácie obrazu, návrh a implementácia výslednej aplikácie. Záver obsahuje možné rozšírenia a celkové zhodnotenie aplikácie.

Aplikácia bola navrhnutá po vlastnej skúsenosti a konzultácii s cieľovým užívateľom.

Abstract

The bachelor's thesis deals with the creation of an application design for a specific group of users and its implementation for the iOS operating system.

The developed application allows the users to easily fill in the necessary data, upload photos to create an output protocol and archive the data locally as well as on a remote server. The individual chapters gradually describe the methods of storing and retrieving stored protocols from a remote server, the method of image detection and recognition, the design and implementation of the resulting application. The conclusion contains possible extensions and an overall evaluation of the application.

The application was designed after personal experience and consultation with the target user.

Kľúčové slová

Ukladanie na vzdialený server, mobilná aplikácia, SwiftUI, iOS, rozpoznanie čísel v obraze.

Keywords

Saving to remote server, mobile application, SwiftUI, iOS, recognition of numbers in an image.

Citácia

ŠESTÁK, Šimon. *Mobilní aplikace pro podporu revizních kontrol na stavbách.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hrubý, Ph.D.

Mobilní aplikace pro podporu revizních kontrol na stavbách

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne a pod vedením Ing. Martina Hrubého, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Šimon Šesták
7. mája 2021

Poďakovanie

Chcel by som poďakovať vedúcemu práce pánovi Ing. Martinovi Hrubému, Ph.D., ktorý mi umožnil realizovať prácu pod jeho vedením. Ďakujem za poskytnuté odborné rady a konštruktívnu kritiku počas návrhu a vývoja aplikácie.

Obsah

Kapitola 1

Úvod

V posledných rokoch sa výrazne a rýchlo mení vzhľad našich miest. Nemenia sa len mestá, ale s výstavbou ich infraštruktúry sa mení vzhľad celej krajiny. Vďaka vplyvu technologickej revolúcie v stavebníctve sa uplatňujú nové spôsoby výstavby, využívajú sa nové stroje, moderné technológie, inovatívne materiály a to najmä v stavebnej chémii. Vďaka tejto revolúcii sa stavia oveľa rýchlejšie a modernejšie.

V tejto hektickej dobe výstavby je mnoho stavenísk, na ktorých je veľmi dôležité vykonávať rôzne kontrolné testy potrebné na schválenie práce stavebným dozorom, či boli dodržané stanovené podmienky. Jednou z dôležitých kontrol na stavbách je napríklad dodržanie použitia správnych materiálov, ale aj dodržanie stanoveného množstva použitého materiálu. Takéto kontroly sa v dnešnej dobe vykonávajú formou testov, z ktorých sú vypracovávané zápisy, respektíve protokoly. Tieto protokoly vypracovávané priamo na stavenisku sa často v pracovnom zhone na stavbe strácajú. Veľakrát sa stáva, že protokoly z tej istej stavby, a z toho istého typu testu majú rôzne písomné formy, sú štrukturálne rozdielne, a teda sú pre stavebný dozor neprehľadné.

Jedným z najčastejšie používaných kontrolných testov na stavbách je odtrhová skúška. Jedná sa o odtrhové skúšky priľnavosti podľa STN EN ISO 4624: 2016. Táto norma je slovenskou verziou európskej normy EN ISO 4624: 2016. Odtrhové skúšky by sa mali vykonávať na všetkých stavbách, keďže sa vykonávajú na kontrolu povrchov (fasády, podlahy,...). Jedným z ďalších používaných testov sú napríklad odtrhové skúšky kotviacich prvkov.

Motiváciou pre vytvorenie tejto práce boli vlastné skúsenosti z pracovnej brigády na stavbe. Pracoval som pre firmu, ktorá mala v jednom čase rozpracovaných niekoľko stavenísk. Firma sa zaoberá hydroizoláciami priemyselných objektov, priemyselnými epoxidovými a polyuretánovými liatymi podlahami, ako aj nátermi. Na všetkých stavbách boli vykonávané odtrhové skúšky. Na konci mesiaca bola sumarizácia všetkých protokolov odtrhových skúšok z jednotlivých stavieb, podľa jednotlivých miest na danej stavbe, s chronologickým zoradením podľa dátumu a času. Pre osobu zodpovednú za stavbu to bola často časovo náročná a zaťažujúca činnosť.

Pre zjednodušenie práce s odtrhmi sa používajú rôzne odtrhové prístroje, ktoré na výstupe nemajú len namerané ťahové napätie odtrhu ale aj grafický priebeh odtrhu, číslo odtrhu a ďalšie parametre. Žiadne z používaných prístrojov neposkytujú možnosť priamo generovať protokoly o skúškach.

V dobe priemyselnej revolúcie 4.0 mi pripadalo zastaralé prepisovanie jednotlivých odtrhových skúšok do jednotného formulára, najmä ak sú zodpovední ľudia na stavbe vybavení modernou komunikačnou technikou. Nakoľko smartphony s iOS systémom sú druhé naj-

používanejšie medzi koncovými odberateľmi, myslím, že vyvíjaná aplikácia by bola ľahko dostupná na každom stavenisku.

Vyvíjaná aplikácia sa preto zameriava na jednoduché a rýchle vytvorenie protokolu o odtrhovej skúške.

Cieľom aplikácie je možnosť načítať výstupné údaje z rôznych prístrojov využívaných pri odtrhových skúškach, vytvárať fotografie, ktoré budú priložené ku konkrétnemu protokolu, rýchly prístup k dátam o počasí na mieste vykonávania skúšky v danom čase, možnosť jednoduchého vyplnenia informácií o meracom prístroji a samotné vyplnenie polí, ktoré sa automaticky načítajú alebo priradia bez úkonu užívateľa (číslo skúšky, dátum,...). Aplikácia výrazne zjednoduší a sprehľadní prácu používateľovi aj pri väčšom počte stavenísk.

Kapitola 2

Zmysel vývoja aplikácie

Táto kapitola sa zaoberá zdôvodnením vytvorenia mobilnej aplikácie pre podporu revíznych kontrol na stavbách. Taktiež obsahuje informácie o najčastejšie používanej revíznej skúške, vďaka ktorej vznikol nápad na vytvorenie aplikácie a spôsob akým sa protokol z tejto skúšky vyhotovuje. Na konci kapitoly sa nachádza prieskum existujúcich riešení.

2.1 Odtrhové skúšky

Pre každú stavbu je vytvorený technologický postup, ktorý obsahuje všetky technologické a technické postupy stavby „ako sa to vykoná a čím sa to bude vykonávať“. Na základe technologického postupu sa vytvára kontrolný a skúšobný plán, ktorý obsahuje akým spôsobom sa majú jednotlivé fázy projektu otestovať. Kontrólny a skúšobný plán na základe noriem stanoví jednotlivé typy skúšok pre jednotlivé fázy stavby.

Odhtrhové skúšky patria pod kontrolný a skúšobný plán. Väčšina stien, podláh a stropov má v dnešnej dobe povrchovú úpravu. Odtrhové skúšky slúžia pre zabezpečenie predčasného porušenia povrchových úprav. Odtrhová skúška testuje pevnosť (dokonalosť spojenia) medzi dvoma a viacerými vrstvami testovaného objektu, zároveň slúži aj na otestovanie kvality podkladu, na ktorý sa bude nanášať povrchová úprava. Zisťuje maximálne ťahové napätie, ktoré je potrebné vynaložiť k rozdeleniu vrstiev testovacieho objektu.

2.1.1 Priebeh

Odtrhovú skúšku priľnavosti je možné prevádzať pomocou normy *EN ISO 4624*. Priebeh odtrhovej skúšky priľnavosti sa dá zhrnúť do štyroch častí:

- Do skúšanej povrchovej úpravy sa zareže až do základného materiálu, buď korunkovým vrtákom, alebo ručnou frézou. Presná veľkosť meranej plochy je definovaná v normách.
- Vhodným lepidlom (epoxidová alebo polyesterová živica) sa na konkrétne miesto testovaného materiálu, upraveného v predchádzajúcom bode, nalepí testovacie teliesko (ďalej ako skúšobný terč).
- Po vytvrdnutí lepidla sa pripojí skúšobný terč k meraciemu prístroju. Otáčaním ramena prístroja sa skúšobný terč zaťažuje.
- Skúšobný terč sa zaťažuje až do odtrhnutia. Po odtrhnutí sa na obrazovke prístroja zobrazí maximálna dosiahnutá sila (ťahové napätie), potrebná na prekonanie sily väzby

medzi vrstvami povlaku (priľnavosť), medzi povlakom a podkladom (adhézia), alebo v povlaku (kohézia).

2.2 Vyhotovenie protokolu

Pre vytvorenie výstupného protokolu o odtrhovej skúške je potrebné získať všetky dáta o danej skúške. V sekcii 2.1.1 je vysvetlené ako sa získavajú jednotlivé dáta pre jednu z častí výsledného protokolu. Ostatné dáta musí užívateľ získať iným spôsobom. Číslo protokolu je podľa vnútorných pravidiel firmy zväčša zložené z čísla posledného protokolu zväčšeného o jedna a roku, v ktorom sa zákazka vykonáva. Ďalej je potrebné zaznamenať deň, v ktorom bola skúška vykonaná a zároveň špecifikovať miesto (sekciu) stavby, na ktorej bola vykonaná. Pred vykonaním konkrétnej odtrhovej skúšky je potrebné, aby vykonávajúci zistil vlhkosť a teplotu ovzdušia ako aj konštrukcie, nakoľko tieto veličiny majú značný vplyv na výsledok odtrhovej skúšky. Musí uviesť podklad testovaného materiálu, informácie o prístroji, ktorý bude využívať na vyhotovenie jednotlivých odtrhových testov, prípadne môže rozšíriť každý odtrhový test o popis miesta odtrhu. Dôležitou súčasťou testu, je vytvorenie fotografie, ktorá by mala v ideálnom prípade obsahovať tri objekty záujmu. Týmito objektami sú *spodná strana odtrhovaného telesa*, *prístroj s nameranou hodnotou* a *miesto, v ktorom bola vykonaná skúška*. Fotografia môže obsahovať iba nameranú hodnotu na prístroji a spodnú stranu odtrhovaného telesa. Pre minimálnu validáciu testu je možné odfotiť iba nameranú hodnotu na prístroji. Z takto daných dát vieme vytvoriť výsledný protokol, ktorý obsahuje štruktúru popísanu v sekcii 2.2.1.

2.2.1 Výsledok odtrhovej skúšky

Výsledkom odtrhovej skúšky je protokol o odtrhovej skúške, ktorý obsahuje:

- číslo protokolu,
- miesto odtrhu,
- firmu ktorá vykonáva skúšku,
- objednávateľa,
- deň výkonu skúšky,
- vlhkosť a teplotu ovzdušia a objektu,
- podklad testovaného objektu,
- typ použitého meracieho prístroja,
- jednotlivé maximálne ťahové napätia odtrhov,
- fotky jednotlivých nameraných hodnôt,
- vyhodnotenie odtrhovej skúšky, alebo popis výsledku v prípade nevyhovujúcich výsledkov.

2.3 Existujúce riešenia

Táto podkapitola sa zaoberá analýzou existujúcich aplikácií, ktoré nachádzajú uplatnenie na stavbách. Prvý impulz na vytvorenie tejto bakalárskej práce bol v roku 2019, a preto prvý prieskum existujúcich riešení pre česko-slovenský trh bol vykonaný v tom istom roku. Nakoľko v tom čase žiadna z existujúcich aplikácií sa nezaoberala priamou tvorbou konkrétnych protokolov o vykonaní revíznych kontrol na stavbách, aplikácia mala vyplniť túto dieru na trhu. Implementácia aplikácie sa oddialila o rok, bol v roku 2021 vykonaný druhý prieskum existujúcich riešení, ktorý už obsahoval konkurenčnú aplikáciu, ktorá je bližšie opísaná v sekcii 2.3.3. Ak existuje iba jedna aplikácia rovnakého charakteru, je možné, že užívatelia stále o nej nevedia, a tak sa ku nim vyvíjaná aplikácia nemusí dostať, alebo užívatelia nebudú chcieť prejsť zo svojich zaužívaných metód na inovatívnejší spôsob tvorby protokolov.

Väčšina už existujúcich aplikácií pre operačný systém iOS sa zaoberá tvorbou stavebných denníkov. Stavebný denník obsahuje správu o pracovnom dni na stavbe (teplota, vykonaná práca, úrazy,...).

2.3.1 Construction Daily Log App

Táto aplikácia umožňuje užívateľom vytvárať a simultánne spracovávať viacero projektov. V každom projekte je možnosť tvorby denníka a zobrazenie histórie daného projektu. Vkladanie odpracovaných hodín pre jednotlivých zamestnancov a vytváranie ich mzdy. Vkladanie fotografií ako dokumentovanie postupu prác, prípadne samotných dokumentov. Všetky denníky sú synchronizované so vzdialeným serverom a je ich možné zdieľať aj mimo aplikáciu.

Aplikácia tak spĺňa možnosť zdieľať a synchronizovať protokol o vykonaní revíznej kontroly, no nespĺňa tvorbu protokolu a úpravu dát protokolu.

2.3.2 iNeo Pro Field Daily Reports

Táto aplikácia je podobná ako aplikácia *Construction Daily Log App* 2.3.1 tiež umožňuje vytvárať a simultánne spracovávať viacero projektov. Ale možnosť synchronizácie so vzdialeným serverom, a možnosť tvorby výstupných PDF dokumentov je spoplatnená mesačným predplatným. Táto aplikácia nepodporuje pridávanie a následné zdieľanie externých dokumentov o revíznych kontrolách.

Aplikácia tak nespĺňa ani jednu z výhod vytváranej aplikácie. Nespĺňa tvorbu protokolov o vykonaní revíznej kontroly, synchronizáciu daných protokolov a úpravu dát týchto protokolov.

2.3.3 ElcoMaster

Táto aplikácia prišla na česko-slovenský trh už počas vyhotovovania bakalárskej práce. Jej najväčšou výhodou by mala byť multiplatformovosť, ktorú aplikácia deklaruje. Síce popis aplikácie oznamuje multiplatformovosť, nie je to úplná pravda, nakoľko ju nebolo možné naištalovať na notebook s operačným systémom macOS Big Sur (ver. 11.2.3). Ďalšou z nevýhod aplikácie je možnosť používať ju len s prístrojmi značky Elcometer. Ako poslednú nevýhodu aplikácie vidím v jej užívateľskom rozhraní. Na prvý pohľad síce vyzerá jednoducho a dokonca obsahuje aj jednoduchý návod pri prvom spustení. Ale po skončení návodu a približne dvadsiatich minútach práce s aplikáciou, som nenašiel inú možnosť vytvorenia

protokolu, ako pripojiť určité zariadenie firmy Elcometer a importovať z neho dáta. Taktiež sa užívateľské rozhranie veľakrát vnára do vlastnej štruktúry, a tak nie je jednoduché rozpoznať, kde sa práve užívateľ nachádza.

Aplikácia spĺňa zálohovania a zdieľania vytvorených protokolov, ale nespĺňa jednoduchosť ovládania a možnosť vytvoriť protokol z ľubovoľného testovacieho zariadenia.

Kapitola 3

Vývoj aplikácií pre systémy s operačným systémom iOS

Táto kapitola obsahuje priblíženie vývoja mobilných aplikácií pre systémy s platformou iOS od spoločnosti *Apple*. Je tu opísané vývojové prostredie, v ktorom sa takéto aplikácie vyvíjajú a popis knižníc, ktoré sú v aplikácii využité. Nakoľko ekosystém spoločnosti *Apple* je dosť uzavretý, vývoj aplikácie musí byť uskutočnený na ich produktoch ako sú MacBook, iMac,. . . .

Operačný systém iOS

iOS predtým známy ako iPhone OS, je mobilný operačný systém od spoločnosti *Apple* exkluzívne vyvíjaný pre ich hardware. Je to svetovo druhý najčastejšie sa vyskytujúci mobilný operačný systém. Prvá verzia tohto operačného systému bola dostupná 29.6.2007 a najnovšia verzia *14.4.2* je dostupná od 26.3.2021. Hlavné verzie iOS sú vydávané v ročnom cykle [2].

Apple developer program

Apple developer program je dôležitý pre vývoj aplikácií, ktoré si chcú nájsť uplatnenie medzi užívateľmi a dostať aplikáciu na App Store. Program pre jednotlivcov stojí 99$ ročne. V prípade, že aplikáciu publikuje firma je nutné aby si zakúpila Apple Developer Program Enterprise, ktorý stojí 299$ ročne. Tieto predplatné okrem možnosti zverejňovať vytvorené aplikácie na App Store ponúkajú možnosť rozšíriť vlastnosti aplikácie o využívanie vzdialených serverových úložísk (*CloudKit* bližšie v podkapitole 3.5), Game Centra, ApplePay, možnosť nákupov v aplikácii a mnoho ďalších [12].

3.1 Xcode

Pre vyvíjanie aplikácií bežiacich na platforme iOS sa používa integrované vývojové prostredie *Xcode*. Xcode taktiež podporuje vyvíjanie aplikácií bežiacich na macOS, watchOS, tvOS, iPadOS. Bol vyvinutý firmou *Apple* v roku 2003. Jeho posledná verzia *12.4* vyšla 26.1.2021. A beta verzia *12.5* je dostupná od 2.3.2021. Xcode je možné využívať len na počítačoch s operačným systémom macOS.

Súčasťou *Xcode* je textový editor, debugger program pre ladenie vyvíjanej aplikácie, interface builder program pre vytvorenie grafického užívateľského rozhrania aplikácie, podpora správy zdrojového kódu pomocou riadenia verzií GIT, ktorý povoľuje vytvárať užívateľovi GIT archívy. V neposlednom rade *Xcode* obsahuje väčšinu vývojovej dokumentácie od *Applu* a simulátor zariadení od spoločnosti *Apple* pre testovanie vytváraných aplikácií. *Xcode* implicitne podporuje tieto programovacie jazyky: C, C++, Objective-C, Java, AppleScript, Swift, SwiftUI. [5]

3.2 Swift

Swift je multi-paradigmatický, kompilovaný programovací jazyk vyvíjaný firmou *Apple* od roku 2014. Je to nástupca programovacieho jazyka *Objective-C*, nakoľko Objective-C nebolo veľmi menené od roku 1980, chýbali mu vymoženosti moderných programovacích jazykov. Jednou z najväčších výhod *Swiftu* je možnosť spolupracovať s už existujúcim kódom napísaným v Objective-C.

Swift podporuje koncept rozširovania protokolmi pre triedy, štruktúry a typy. Tento koncept prezentuje spoločnosť *Apple* ako naozajstnú zmenu a takýto jazyk nazývajú „protokolovo orientované programovanie“ [4].

3.2.1 SwiftUI

SwitftUI je framework, ktorý poskytuje jednoduchú tvorbu užívateľského rozhrania deklaratívnou cestou programovania a ľahkú implementáciu reakcií aplikácie na užívateľské vstupy [16]. Nie je potrebné definovať prechody medzi jednotlivými časťami užívateľského rozhrania. Poskytuje možnosť vytvoriť základné elementy užívateľského rozhrania pre multiplatformovú znovapoužiteľnosť v rámci operačných systémov iOS, macOS, tvOS a watchOS.

3.3 Perzistenté úložiská

Väčšina aplikácií potrebuje svoje dáta ukladať lokálne a nestratiť ich pri ukončení behu aplikácie. Pre takéto ukladanie dát sa využívajú perzistenté úložiská, do ktorých sa dáta zapíšu a sú v nich uložené, pokiaľ ich odtiaľ užívateľ nevymaže. V rámci programovania pre operačné systémy iOS sa k tomuto účelu može využiť trieda *UserDefaults* 3.3.1 pre uloženie malých objemov dát alebo využiť framework *Core Data* 3.3.2, ktorý spravuje objekty modelovej vrstvy v aplikácii.

3.3.1 UserDefaults

Je to interface umožňujúci prístup k užívateľovej štandartnej databáze. Funguje ako slovník. To znamená, že sa dáta ukladajú pre určitý kľúč, vďaka ktorému môžeme potom k nim pristupovať. Využíva sa primárne pre ukladanie preferencií užívateľa, pre prispôsobenie behu aplikácie, ako je napríklad určiť rýchlosť prehrávania médií, prípadne aké merné jednotky chce užívateľ používať.

UserDefaults si svoje dáta ukladá do cache pamäte, aby sa pri každom vyžiadaní uloženej hodnoty nemusela otvárať užívateľova štandartná databáza. Pri uložení novej hodnoty sa daná hodnota zapíše synchrónne s procesom, ktorý ju zmenil ale zápis do perzistentného úložiska prebieha asynchrónne.

Takto uložené dáta sa vyskytujú lokálne iba na konkrétnom zariadení. Pre možnosť zdieľať tieto dáta so všetkými užívateľovými zariadeniami je potrebné využiť triedu *NSUbiquitousKeyValueStore* [18].

3.3.2 Core Data

Poskytuje riešenia na bežné úlohy vykonávané nad životným cyklom objektov a ich perzistentným ukladaním. Vďaka tomuto frameworku sa kód zväčša zmenšuje o 50% - 70% vďaka už naprogramovaným vlastnostiam [19].

Core Data vytvorý súbor s príponou .xcdatamodel, v ktorom je možné vytvoriť databázový model, s ktorým bude aplikácia pracovať. Takýto model obsahuje entity a každá jeho entita obsahuje určité atribúty. Entita musí odpovedať určitej triede a tá musí dediť vlastnosti z triedy *NSManagedObject*, vďaka tejto triede nadobúda vlastnosti potrebné pre perzistentné ukladanie dát [19]. Inštanciu takto vytvorenej triedy je možné si predstaviť ako riadok tabuľky, zatiaľ čo atribúty triedy ako jej stĺpce. Ďalej je možné modelovať väzby v rámci modelu medzi jednotlivými entitami. Medzi dôležité vlastnosti vzťahov patrí ich názov, kardinalita a cieľová entita vzťahu.

Core Data Stack

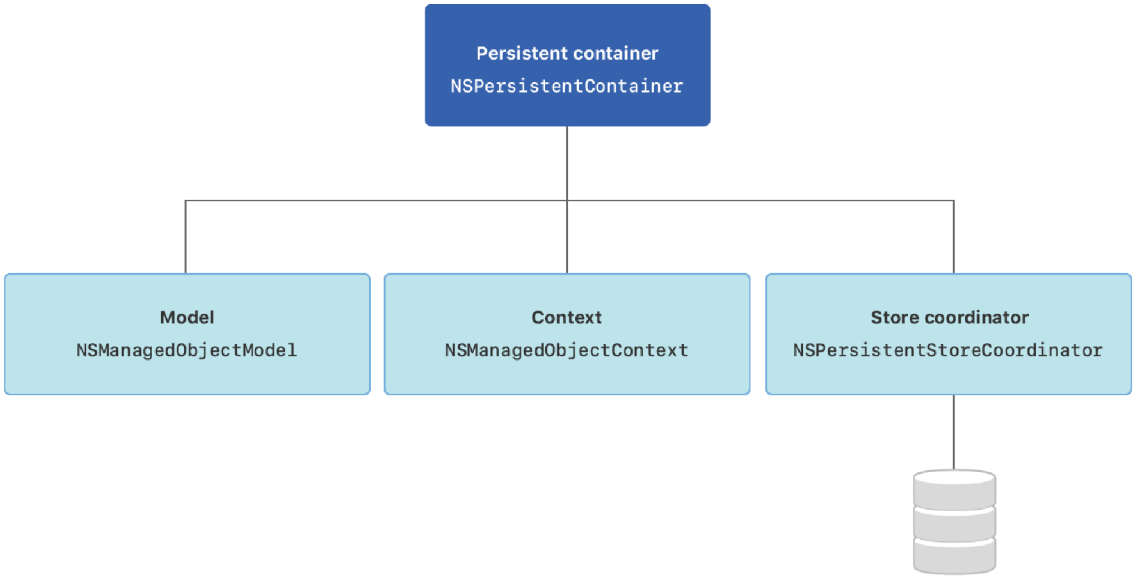
Zaobstaráva všetku interakciu medzi externým úložiskom dát a aplikáciou. *Core Data Stack* je kolekcia objektov architektúry, ku ktorým sa pristupuje pre inicializáciu *Core Data*. Vďaka tomu môže aplikácia komunikovať s externým úložiskom dát [11]. Skladá sa zo štyroch základných objektov:

- NSManagedObjectContext - trieda využívaná pre správu a sledovanie zmien objektov
- NSPersistentStoreCoordinator - koordinátor využívaný triedou *NSManagedObjectContext* pre komunikáciou s perzistentným úložiskom
- NSManagedObjectModel - programová reprezentácia modelu
- NSPersistentContainer - zapúzdrenie samotného *Core Data Stack*

3.4 Kódovanie a dekódovanie

Pre posielanie dát po sieti je nutné dáta zakódovať. Takto zakódované dáta môžu byť odoslané a prijaté. Prijaté dáta je pred ich využitím nutné dekódovať. V rámci jazyka *Swift* podporuje túto činnosť alias **Codable**, ktorý sa skladá z dvoch protokolov: *Encodable* a *Decodable*. Ako je možné rozpoznať z názvov deklarujú, že dátový typ, na ktorý sú použité môže byť zakódovateľný a rozkódovateľný.

V prípade že štruktúra obsahuje prvky, ktoré vyhovujú aliasu *Codable*, je štruktúru možné zakódovať a rozkódovať. V prípade, že štruktúra obsahuje prvky, ktoré nevyhovujú aliasu *Codable*, štruktúru nie je možné zakódovať alebo rozkódovať. V prípade že potrebujeme, aby táto štruktúra vyhovovala aliasu *Codable* je možné určiť pre nevyhovujúce prvky, akým spôsobom sa majú zakódovať a rozkódovať.



Obr. 3.1: Core Data Stack [14]

3.4.1 JSON

JSON je to veľmi rozšírená forma zakódovaných dát. Pre ľudí je táto forma kódovania ľahko čitateľná a zapisovateľná zatiaľ čo pre stroje je ľahko generovateľná a analyzovateľná. Bol odvodený z jazyka JavaScript, ale väčšina moderných jazykov už obsahuje implementáciu pre rozkódovanie alebo zakódovanie dát do tohto formátu. Výnimkou nie je ani swift, ktorý pre tieto akcie obsahuje objekty **JSONEncoder** a **JSONDecoder** [3].

3.5 CloudKit

Pre synchronizáciu aplikácie bežiacej na viacerých zariadeniach je potrebné implementovať komunikáciu aplikácie so vzdialeným serverom. Pre takéto uľahčenie danej implementácie bol vyvinutý framework *CloudKit*, ktorý poskytuje rozhranie pre prenos dát medzi zariadeniami a vzdialeným serverom. Tento vzdialený server spoločnosť *Apple* nazvala iCloud. iCloud obsahuje kontajnery 3.5.1 a v každom z nich sa vyskytujú tri rôzne databázy, do ktorých môže užívateľ vkladať svoje dáta, bližšie o týchto databázach sa dočítate v sekcií 3.5.2.

Na rozdiel od iných implementácií vzdialených serverov CloudKit nevyžaduje veľa práce s nastavením. Nemusí sa vyberať, nastavovať, prípadne inštalovať vzdialený server. Stačí sa zaregistrovať do programu pre vývojárov iOS (*iOS Developer Program*) a pridať schopnosť aplikácie komunikovať s iCloudom [23].

Ďalšia z výhod CloudKitu je možnosť využívať webové rozhranie *CloudKit Dashboard*. V ňom je možnosť zobraziť podrobnosti o komunikácií aplikácie so serverom, zobraziť využívanú šírku pásma. V neposlednom rade zobraziť uložené dáta, prípadne ich upravovať alebo upraviť uloženú databázovú schému. Možnosť upravenia schémy je dostupná iba v prípade že aplikácia ešte nebola zavedená do produkcie.

Priebeh komunikácie

Na priebeh komunikácie sa používajú serverové API, ktoré sú implementované vo frameworku *CloudKit*. Ten nenahrádza už vytvorené dátové objekty, ale poskytuje ich synchronizáciu so vzdialeným serverom. V prípade, že aplikácia chce ukladať svoje dáta do privátnej databázy užívateľa, je nutné, aby bol užívateľ prihlásený svojim platným Apple ID na zariadení, ináč komunikácia s privátnou databázou nie je možná.

Pre komunikáciu sú využívané *CKRecord* takzvané záznamy. Záznam je slovník obsahujúci kľúč a hodnotu uloženú pre daný kľúč. Hodnota daná týmto kľúčom sú dáta, ktoré chceme uložiť na server, alebo sme ich zo servera obdržali. Záznam je potrebné najskôr vytvoriť v pamäti a až potom je možné ho využívať pre synchronizáciou so vzdialeným serverom. Pre ukladanie, mazanie alebo obdržanie takto vytvorených záznamov na server alebo zo servera sa využívajú asynchrónne operácie.

„*Ale ako vie server, ktorý záznam chcem získať prípadne vymazať ?*“ Každý záznam obsahuje svoje *CKRecord.ID* takzvané ID. Vďaka tomuto ID vie aplikácia alebo server rozoznať, o ktorý záznam sa jedná. Nakoľko ale aplikácia implicitne tieto ID neukladá, odporúča sa ich ukladať v perzistentých úložiskách napríklad v Core Datach 3.3.2. Nakoľko táto komunikácia prebieha asynchrónne, každá funkcia po dokončení vyvolá *completion handler*, ktorý obsahuje výsledok operácie alebo chybu.

3.5.1 Kontajnery

CloudKit kontajnery predstavujú iCloud úložiská. Každá aplikácia má svoj vlastný kontajner, ktorý spravuje jej obsah. *CKContainer* spravuje všetky pokusy o prístup k jeho dátam [9]. Každý kontajner rozlišuje medzi súkromnými a verejnými dátami, ktoré ukladá do samostatných databáz, ktoré vlastní 3.5.2.

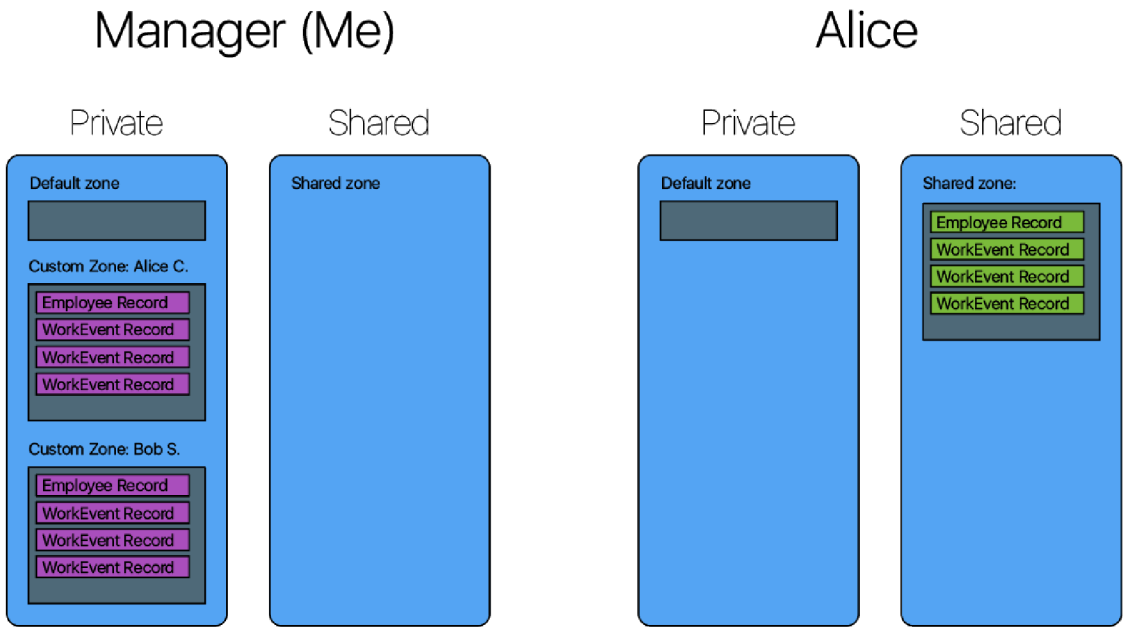
3.5.2 Databázy

Ako bolo zmienené v sekcii CloudKit 3.5 tak iCloud obsahuje tri rôzne databázy, s ktorými môže aplikácia komunikovať. Každá z nich obsahuje minimálne jednu zónu. Čo znamená zóna je vysvetlené neskôr v sekcii 3.5.3.

Privátna databáza v nej môže existovať niekoľko zón, a ako naznačuje názov k dátam vyskytujúcim sa v privátnej databáze môže pristupovať iba užívateľ, ktorý ich tam nahral. Pre prístup do privátnej databázy je potrebné, aby bol užívateľ prihlásený svojim Apple ID na zariadení. V privátnej databáze je možnosť využívať operáciu *CKFetchRecordZoneChangesOperation*, ktorá sa rozoberá v sekcii 3.5.4.

Verejná databáza k tejto databáze má prístup každý užívateľ, ktorý si nainštaloval aplikáciu ktorá prístup k tejto databáze podporuje. Avšak užívateľ, ktorý nie je prihlásený na zariadení svojim Apple ID, môže z tejto databázy dáta iba čítať. Sú tu uložené dáta všetkých užívateľov využívaných danú aplikáciu. Jedna z nevýhod tejto databázy je nemožnosť využívať operáciu *CKFetchRecordZoneChangesOperation*.

Zdieľaná databáza do tejto databázy sa zdieľajú záznamy, ktoré užívateľ A označí a tým sa odošle link pre zdieľanie určitému užívateľovi B. Užívateľovi B príde notifikácia o zdieľaní a možnom prijatí záznamu. Užívateľ A môže zrušiť možnosť zdieľania záznamov ale aj užívateľ B má možnosť zrušiť svoju väzbu na daný záznam. Záznamy v tejto databáze užívateľ B nevlastní, má ich iba možnosť zobraziť, prípadne upraviť podľa udelených práv od užívateľa A [15].



Obr. 3.2: Ukážka zdieľanej databázy [22]

3.5.3 Zóny

Zóny sú dôležitou súčasťou organizácie dát. Každá zóna zaobaľuje uložené záznamy v nej. Vďaka tomu sa môžu vytvárať nové zóny a tak zaobaľovať záznamy, ktoré spolu súvisia. Každá vytvorená zóna môže využívať aj výhody ako je napríklad atomická operácia zápisu viacerých záznamov. V rámci zóny sa môžu vytvárať väzby medzi záznamami [10].

Privátna a verejná databáza obsahuje svoju predvolenú zónu. Názov tejto zóny v obidvoch databázach je *Default zone*.

Typy záznamov

CKRecordType je hodnota ktorú definuje aplikácia. Je využívaná len na rozlíšenie druhov záznamov s ktorými aplikácia operuje.

3.5.4 Predplatné

„*Ako zistí aplikácia, že nastala zmena na vzdialenom úložisku ?*“ Pre tento účel je využívaná trieda *CKNotification*, ktorá dokáže zachytávať upozornenia vyvolané zmenami na vzdialenom úložisku a vďaka tomu vyvolať reakciu aplikácie na danú zmenu. Pre možnosť takto odchytávať upozornenia musí mať aplikácia povolenú schopnosť vzdialených notifikácií.

Vďaka možnosti odchytávať vykonané zmeny na vzdialenom úložisku je ľahké implementovať vyvolanú reakciu. Jednou z najčastejších môže byť kontrola zmien na vzdialenom úložisku od určitej časovej značky. Pre túto reakciu slúži operácia *CKFetchRecordZoneChangesOperation*. Dvoma argumentami tejto operácie sú: pole zón a ich možnosti, ktoré obsahujú poslednú časovú značku pre konkrétnu zónu. Táto operácia asynchrónne vráti nové alebo zmenené záznamy a pre záznamy, ktoré boli zo servera vymazané vráti ich ID. V prípade, že počas operácie nenastala žiadna chyba, navracia novú časovú značku pre každú zónu, ktorú je treba uložiť pre ďalšie volanie.

3.6 UIDocument

Trieda využívaná na správu dát aplikácie. Hlavné výhody dokumentov tohto typu sú:

- Asynchrónne čítanie a zápis na pozadí aplikácie.
- Automatické ukladanie dokumentu pri zmene jeho obsahu.
- Bezpečné ukladanie dokumentu, najskôr sa vytvorí dočasný súbor, ktorý potom nahradí už existujúci dokument.
- Podpora zistenia konfliktov verzií dokumentu.
- Možnosť využívania integrovaných cloudových služieb.

Hlavný atribút tohto typu dokumentu je URL adresa k dokumentu v rámci SandBoxu 3.7 aplikácie. Pri vytváraní dokumentu je potrebné zadať adresu kde bude dokument vytvorený. Z tejto adresy sa automaticky vyčíta názov a typ ukladaného dokumentu.

Nakoľko operácie s dokumentom sú asynchrónne, každá obsahuje *completion handler* ktorý, sa vyvolá po dokončení danej operácie. *Completion handler* obsahuje hodnotu typu *Bool*, ktorá oznamuje či operácia prebehla bez problému alebo sa operácia nepodarila [17].

3.7 SandBox

SandBoxing je ukladanie a spracovanie dát aplikácie v adresári, ku ktorému má prístup iba daná aplikácia. Výrazne zvyšuje bezpečnosť operačného systému limitovaním prístupu aplikácie ku dátam. Ak aplikácia potrebuje prístup ku dátam mimo svojho SandBoxu, môže k nim pristúpiť iba skrz systémový interface [20].

SandBox aplikácie štandartne obsahuje:

- *property list* - je to list vlastností aplikácie, ktorý využíva operačný systém.
- *Documents* - priečinok do ktorého by sa mali ukladať dáta priamo súvisiace s užívateľom.
- *Library* - obsahuje dva priečinky:
  - Cache v ktorom ako názov naznačuje je ideálne ukladať dáta, ktoré boli načítane do cache pamäte.
  - Preferences, ktorý obsahuje dáta uložené do štandartnej databázy aplikácie.
- *tmp* - tento priečinok by mal obsahovať dočasné dáta, nakoľko obsah tohto priečinku operačný systém raz začas vymaže.

3.8 Model - View - ViewModel

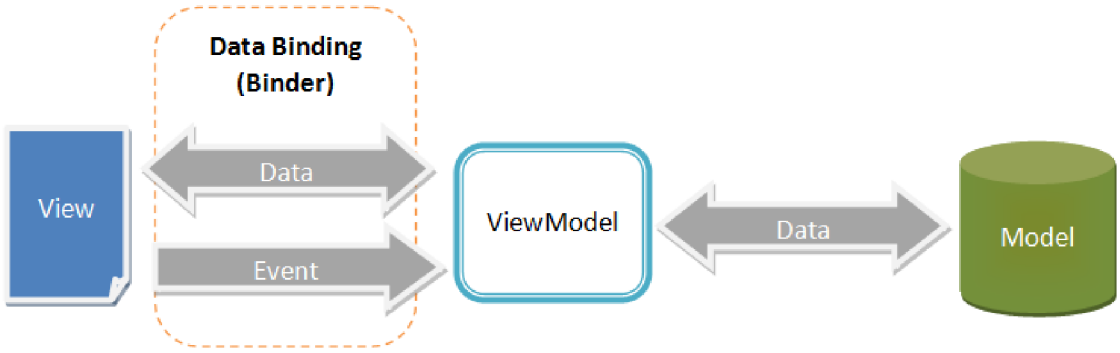
Model - View - ViewModel (ďalej len MVVM) je druhý najpopulárnejší architektonický vzor používaný pri vytváraní aplikácií na operačný systém iOS. Bol vynájdený v jazyku *Smalltalk* v roku 1988. Základná myšlienka tohto vzoru je oddeliť prezentačnú a obchodnú logiku od užívateľského rozhrania. Toto oddelenie zaručí lepšiu udržateľnosť aplikácie.

Aby sa v rámci View[1] minimalizovala rozhodovacia logika a zároveň bola presunutá do ViewModelu[2] musí View spĺňať tieto pravidlá:

- View nesťahuje údaje z ViewModelu.
- View nezodpovedá za svoje obnovenie pri zmene vo ViewModele.
- View má svoj stav, ktorý spravuje ViewModel.

Dátové toky idú v oboch smeroch ako zobrazuje obrázok 3.3 Užívateľova interakcia, ktorá je obslúžená za pomoci View, vyvolá reakciu ViewModelu. Následne ViewModel ďalej deleguje reakciu do Modelu. Takto delegovaná reakcia patrí medzi CRUD[3] operácie nad modelom a dátami.

V opačnom smere to vyzerá nasledovne. Model načíta dáta z databázy alebo iného dátového úložiska. Načítané dáta sa odovzdajú do ViewModelu. ViewModel upraví dáta do podoby, v akej ich je možné zobraziť v rámci View. View vykreslí obdržané dáta [6].



Obr. 3.3: Ukážka komunikácie v rámci MVVM [8]

3.9 Swift Package Manager

Swift Package Manager je nástroj pre jednoduchú distribúciu a znovu použiteľnosť zdrojových kódov. Je integrovaný so systémom vytvárajúcim výstupné súbory, tak aby zautomatizoval proces sťahovania, kompilovania a vytvárania závislostí.

Module, Swift organizuje svoj kód do modulov. Zdrojový kód môže obsahovať všetko v jednom module, alebo obsahovať viacero modulov. Väčšina závislostí vyžaduje stiahnutie a vytvorenie daného kódu na jeho využívanie. Pri využívaní modulov pre určité oblasti problematiky vzniká možnosť takto vytvorený kód znova použiť.

Package, obsahuje súbory so zdrojovým kódom a manifest súbor obsahujúci metadáta pre daný balíček. Manifestový súbor pomenovaný „Package.swift“ definuje meno balíčka a jeho obsah pomocou modulu „PackageDescription“. Balíček obsahuje jeden, alebo viacero cieľov. Každý cieľ špecifikuje produkt, ktorý môže deklarovať jednu, alebo viacero závislostí.

Target, môže ako svoj produkt vytvoriť knižnicu, alebo spustiteľný súbor. Knižnica obsahuje modul, ktorý môže byť importovaný do zdrojového kódu.

[1] *View* - časť programu, ktorá sa zobrazuje užívateľovi a zachytáva užívateľovu interakciu
[2] *ViewModel* - model obsahujúci dáta potrebné pre zobrazenie konkrétneho View
[3] *CRUD* - operácie vytvor, načítaj, uprav a vymaž

Dependecies, závislosti cieľa sú moduly, ktoré vyžadujú kód v balíčku. Závislosť pozostáva z relatívnej alebo absolútnej cesty ku balíčku a požadovanej verzie daného balíčka.

Hlavná úloha *Swift Package Manager*-a pozostáva zo zníženia náročnosti koordinácie procesu sťahovania a vytvárania všetkých závislostí, nakoľko je tento proces rekurzívny (závislosť môže obsahovať vlastné závislosti) vytvára sa graf závislostí [13].

3.10 Externý framework ZIP

Je framework jazyka Swift pre vytváranie súborov typu ZIP a zároveň aj extrahovanie ich dát. Je to externý framework s MIT licenciou, ktorá povoľuje voľné používanie frameworku. Pôvodný tvorca tohto frameworku je Roy Marmelstein, nakoľko je framework open source verziou obsahuje veľa úprav od komunity [21].

Preferované zavedenie do aplikácie je uvedené prostredníctvom Swift Package Manager 3.9.

3.11 Spracovanie obrazu

Spracovanie obrazu je súčasťou počítačového videnia, ktoré je v súčasnosti na vzostupe, najmä kvôli možnosti využívať strojové učenie. Priebeh spracovania obrazu sa môže jednoducho vysvetliť tak, že na vstupe je obrázok na ktorý použijeme filter. Filter je program, ktorý algoritmicky skúma pixely a aplikuje na nich určitý efekt vďaka ktorému je možné vygenerovať nový výstupný obrázok. Skratka obrazového prvku, pixel je hodnota, ktorá predstavuje intenzitu farieb, ktoré tvoria obraz.

Počítačové videnie je oblasť, kde programujeme tak, aby počítač videl prostredie okolo nás, vďaka čomu dokáže rozpoznávať a extrahovať informácie z objektov v reálnom čase. Počítačové videnie je priamo súvisiace so strojovým učením [7].

3.11.1 Vison

Je framework vytvorený spoločnosťou *Apple*. Vykonáva vysoko výkonnú analýzu obrazu pre identifikovanie čiarových kódov, detekovanie textu, tvárí, ale za pomoci integrácie *CoreML* 3.11.2 môže detekovať aj gestá človeka, mimiku tváre,... Analýza môže prebiehať v rámci práve natáčaného videa, uloženého videa, alebo fotografie.

3.11.2 Core ML

Je framework zabezpečujúci integráciu modelu strojového učenia do aplikácie. Poskytuje unifikovanú reprezentáciu pre všetky modely.

Model je výsledkom aplikovania algoritmu strojového učenia nad zvolenými trénovacími dátami. Využíva sa získanie predpovede z nových vstupných dát zadaných užívateľom aplikácie. Je možné naučiť model napríklad na kategorizáciu fotiek, detekovanie objektov v obraze,...

Modely formátu *CoreML* sa vytvárajú za pomoci aplikácie *Create ML*, ktorá je súčasťou *Xcode*. Alebo vďaka pomoci nástroja *Core ML Tool* je možné konvertovať iné modely do tohto formátu.

Podporuje frameworky *Vison* pre analyzovanie obrázkov, *Natural Language* pre spracovanie textu, *Speech* pre konvertovanie audia do textu a *Sound Analysis* pre identifikovanie zvukov vo zvuku [1].

3.12 PDFKit

Dokument typu PDF bol vytvorený firmou *Adobe*, stal sa z neho otvorený štandart, ktorý obsahuje text, obrázky a interaktívne prvky. PDF je elektronický dokument ktorý, má podobu tlačeného dokumentu.

Framework PDFKit prišiel s verziou *iOS 11.0*. Umožňuje tvorbu, manipuláciu a zobrazenie dokumentov typu PDF v rámci aplikácie.

Tvorba PDF dokumentu

Najdôležitejšou triedou je *UIGraphicsPDFRenderer*. V nej je obsiahnutá funkcia *pdfData(actions:)*, ktorá umožnuje tvorbu PDF dokumentu a navracia vyprodukované dáta. Vytvorená inštancia triedy *UIGraphicsPDFRenderer* umožňuje zadanie veľkosti strany, naplnenie metadát o dokumente a vytvorenie štruktúry dokumentu.

Zobrazenie PDF dokumentu

Pre zobrazenie zvoleného PDF dokumetu sa využíva trieda *PDFView*, okrem zobrazenie poskytuje možnosť priblížiť dokument, prechádzať odkazmi vloženými do dokumentu a kopírovať určité časti textu. Vytvorená inštancia tejto triedy očakáva naplnenie svojej premennej *document*, PDF súborom ktorý sa má zobraziť.

Kapitola 4

Návrh aplikácie

V tejto kapitole je popísaný celý návrh aplikácie. Výsledky komunikácie s potencionálnym koncovým užívateľom. Návrh užívateľského rozhrania, návrhy databázových schém (lokálnej a na vzdialenom úložisku). Odôvodnenie výberu použitých technológií.

4.1 Cieľová skupina

Pre návrh cieľovej aplikácie je dôležité zistiť cieľovú skupinu užívateľov. Nakoľko vyvíjaná aplikácia má zrýchliť a zjednodušiť vytváranie výstupných protokolov z revíznych kontrol vykonaných na stavbách, cieľová skupina je jasne stanovená, revízny kontrolóri ktorí budú primárne pracovať s aplikáciou, prípadne ich zamestnávatelia a stavebný dozor, ktorý bude interagovať iba s výstupnými dátami aplikácie.

Návrh aplikácie na základe tejto cieľovej skupiny prebiehal komunikáciou s technikom, ktorý určil požadovanú funkčnosť aplikácie, na základe ktorej mu bol predvedený návrh užívateľského rozhrania. Po drobných úpravách rozhrania a následnom odsúhlasení technika bol návrh predstavený jeho zamestnávateľovi, s ktorým bola rozobratá štruktúra výstupných dát.

4.2 Životný cyklus protokolu revíznej kontroly

Na zabezpečenie správneho návrhu aplikácie je potrebné vysvetliť životný cyklus protokolu revíznej kontroly. To zaručí jednoduchšie vytvorenie životného cyklu aplikácie.

Vytvorenie protokolu o revíznej kontrole môže vznikať v krátkom časovom úseku po vykonaní revíznej kontroly alebo retrospektívne na základe uložených dát o revízii. Na vytvorenie protokolu je potrebné získať dáta. Zber týchto dát je opísaný v sekcii 2.2. Po vytvorení životný cyklus protokolu nekončí. Jeho vygenerovaný výstup je odoslaný firme, ktorá si revíznu kontrolu objednala. V prípade, že časti revíznej kontroly nesplnili požiadavky pre danú revíziu je potrebné kontrolu opakovať. Pri opakovaní kontroly sa zbierajú nové dáta iba z časti, ktorá nesplňovala požiadavky. Ostatné dáta sú znovapoužiteľné. Po vykonaní opakovanej revíznej kontroly je vygenerovaný nový výstup s upravenými dátami a zvýšeným interným počítadlom. Všetky výstupy sú spolu s vyhotovenými fotografiami uschované, pokiaľ opakovaná kontrola nesplní všetky požiadavky revízie.

V prípade, že revízna kontrola spĺňa všetky požiadavky, môže byť daný protokol označený ako *uzatvorený*. Uzatvorenie protokolu znamená nemožnosť vytvárať opakované revízne kontroly. Sú odstránené všetky fotografie okrem tých, ktoré sa vyskytujú v posled-

nom výstupnom protokole. Takže uzavretý protokol obsahuje históriu všetkých výstupných protokolov vo forme PDF a súbor typu ZIP obsahujúci fotografie, na ktoré sa odkazuje v poslednom vytvorenom výstupnom protokole.

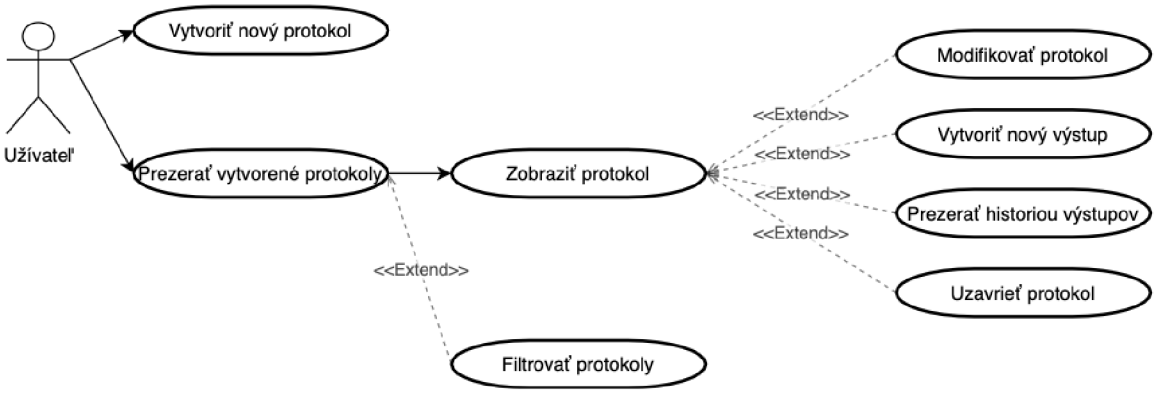
4.3 Užívateľské rozhranie

Užívateľské rozhranie je jedným z najdôležitejších faktorov pri hodnotení aplikácie užívateľom. Malo by byť ľahko pochopiteľné a užívateľ by sa mal v ňom jednoducho orientovať. Samostatné časti aplikácie by sa od seba nemali veľmi odlišovať dizajnom. Každá užívateľova interakcia by mala vyvolať čo najrýchlejšiu reakciu rozhrania a tak oznamovať užívateľovi že aplikácia daný dotaz vykonala alebo je dotaz spracovávaný.

Na základe dát získaných od budúceho užívateľa bol vytvorený „use case“ diagram, ktorý znázorňuje obrázok 4.1. Ako je možné vyčítať z diagramu, za užívateľa je braný iba technik, nakoľko jediná požiadavka zamestnávateľa bola možnosť vytvorený protokol jednoducho zdieľať, ako ľahko čitateľný súbor PDF spolu so súborom ZIP obsahujúcim fotografie, na základe ktorých bol vytvorený PDF dokument.

Pre jednoduché ovládanie aplikácie bol zvolený prístup zobrazenia skrz „tab view“. Aplikácia bude obsahovať, dve karty, kde na jednej môže užívateľ vytvárať čisto nový protokol a na druhej sa mu zobrazí zoznam už vytvorených protokolov. Zobrazený zoznam bude mať možnosť filtrovať na základe klienta pre ktorého sa daný protokol vytvára, dátumu, staveniska, prípadne čísla protokolu. Aplikácia by mala obsahovať znova použiteľné View pre zobrazenie detailu protokolu, alebo vytvorenie nového protokolu, nakoľko tieto dve zobrazenia sú skoro totožné. Ďalej by mala obsahovať jednoduchý list s fotografiami a ich metadátami obsiahnutými v protokole, možnosť vybrať z kalendára dátum vytvorenia protokolu v prípade, že sa protokol nevytváral priamo na stavenisku.

Rozhranie umožňuje užívateľovi sledovať postup práce vďaka rýchlej spätnej reakcie rozhrania. Užívateľ pri tvorbe protokolu o revíznej kontrole môže postupovať po riadkoch formulára alebo v ľubovoľnom poradí. Riadky formulára obsahujú názvy jednotlivých častí výstupného dokumentu a ich farebné vyhotovenie oznamuje, či boli do nich vložené dáta a zároveň sa kontroluje validita týchto dát. Vďaka farebnému rozlíšeniu užívateľ ľahko spozoruje prípadnú nevyplnenú časť potrebnú pre tvorbu výstupného dokumentu. Užívateľ môže rozpracovaný dokument uložiť aj bez vloženia všetkých dát a tak neprísť o už naplnené informácie, ale generovanie výstupných súborov je užívateľovi sprístupnené až po zadaní všetkých potrebných dát.



Obr. 4.1: Akcie vykonateľné užívateľom

4.4 Databáza

Databáza sa delí na dve verzie. Týmito verziami je lokálna databáza a vzdialená databáza. V lokálnej databáze sú úložené prevažne metádáta k jednotlivým prvkom protokolu (napríklad cesta k lokálnej verzii fotografie). Na rozdiel od lokálnej, vzdialená databáza obsahuje aj samotné súbory potrebné pre vytvorenie protokolu ako sú fotky, vytvorené výstupné súbory a zašifrovaný protokol za pomoci *JSONEncoder*-u 3.4.1.

4.4.1 Lokálna

Pre lokálne ukladanie informácií o vytvorených protokoloch, ich fotkách a výstupoch bolo vybrané perzistentné úložisko CoreData 3.3.2. Vďaka voľbe tohto úložiska je možné jednoducho vytvárať nové objekty a následne ich ľahko spravovať. Databázový model je teda zložený z troch entít **DatabaseArchive**, **OutputArchive** a **MyPhoto**. Entity neobsahujú žiadne väzby, no každý má atribút *ProtoID*, vďaka ktorému je možné priraďovať jednotlivé záznamy k sebe. Každá z entít obsahuje *CKRecord.ID*, v prípade vymazania alebo modifikovania jej inštancie je vďaka tomuto atribútu možnosť upraviť aj jej vzdialenú verziu.

- **DatabaseArchive** obsahuje informácie o vytvorených protokoloch pre ich možné filtrovanie.
  - *client* [String] - názov klientovej firmy
  - *date* [Date?] - dátum vykonávania revíznej skúšky
  - *local* [Bool] - flag[1] signalizujúci existenciu lokálnej kópie protokolu
  - *protoID* [Int16] - identifikačné číslo konkrétneho protokolu
  - *recordID* [CKRecord.ID?] - identifikačné číslo vzdialeného záznamu odpovedajúcemu konkrétnemu lokálnemu záznamu
  - *construction* [String] - názov stavby, na ktorej bola vykonávaná revízna skúška
- **OutputArchive** obsahuje informácie o vytvorenom výstupe určitého protokolu. Či výstup obsahuje oba súbory (pdf, zip) a koľký v poradí je to výstup daného protokolu.
  - *recordID* [CKRecord.ID?] - identifikačné číslo vzdialeného záznamu odpovedajúcemu konkrétnemu lokálnemu záznamu
  - *zip* [Bool] - flag signalizujúci existenciu lokálnej kópie súboru ZIP
  - *pdf* [Bool] - flag signalizujúci existenciu lokálnej kópie súboru PDF
  - *protoID* [Int16] - identifikačné číslo konkrétneho protokolu
  - *internalID* [Int16] - identifikačné číslo verzie konkrétneho protokolu
- **MyPhoto** obsahuje všetky potrebné informácie o konkrétnej časti revíznej kontroly pre pridanie vytvorenej fotografie do výstupného PDF súboru.
  - *descriptionOfPlace* [String] - rozširujúci popis konkrétneho miesta odtrhu
  - *local* [Bool] - flag signalizujúci existenciu lokálnej kópie odpovedajúcej fotografie
  - *name* [Int] - názov fotografie v rámci protokolu (poradové číslo fotografie)
  - *protoID* [Int16] - identifikačné číslo konkrétneho protokolu

[1] *flag* - hodnota signalizujúca určitý stav objektu, zväčša nadobúdajúca maximálne dvoch hodnôt

  - *recordID* [CKRecord.ID?] - identifikačné číslo vzdialeného záznamu odpovedajúcemu konkrétnemu lokálnemu záznamu
  - *targetDiameter* [Double] - veľkosť použitého terča
  - *value* [Double] - nameraná hodnota

4.4.2 Vzdialená

Nakoľko aplikácia využíva vzdialené úložisko pre synchronizáciu dát medzi viacerými zariadeniami, je nutné vytvoriť vzdialenú databázu, ktorá bude ukladať nielen metadáta o konkrétnych častiach protokolu ale aj samotné súbory. Pre každú entitu vyskytujúcu sa v lokálnej databáze existuje príslušný *record type* 3.5.3.

Záznam vzdialenej databázy sa vytvára alebo modifikuje po úspešnom vyhotovení alebo modifikácií lokálnej entity. Takto vytvorený záznam je naplnený dátami danej entity a asynchrónne odoslaný na uloženie do vzdialenej databázy. Vďaka tomu ostáva vzdialená databáza aktualizovaná.

Protokol môže byť modifikovaný a aby nedochádzalo k odosielaniu dát po každej modifikácii, uloženie zmien a ich asynchrónne odoslanie do vzdialenej databázy sa vykoná v momente keď užívateľ zatvorí zobrazenie daného protokolu.

- Entite *OutputsArchive* odpovedá record type **Outputs** obsahujúci:
  - *protoID* [Int] - identifikačné číslo konkrétneho protokolu
  - *internalID* [Int] - identifikačné číslo verzie konkrétneho protokolu
  - *pdf* [Asset] - zakódovaný súbor PDF v podobe prílohy
  - *pdfLocal* [Int] - flag určujúci či má byť PDF súbor uložený aj lokálne
  - *zip* [Asset] - zakódovaný súbor ZIP v podobe prílohy
  - *zipLocal* [Int] - flag určujúci či má byť ZIP súbor uložený aj lokálne
- Entite *DatabaseArchive* odpovedá record type **Protocols** obsahujúci:
  - *encodedProto* [String] - obsahuje vytvorený zakódovaný protokol (bez príslušných súborov fotiek a verzií)
  - *protoID* [Int] - identifikačné číslo konkrétneho protokolu
- Entite *MyPhoto* odpovedá record type **Photos** obsahujúci:
  - *diameter* [Int] - veľkosť použitého terča
  - *desc* [String] - rozširujúci popis konkrétneho miesta
  - *local* [Int] - flag určujúci či má byť fotografia uložená aj lokálne
  - *name* [Int] - názov fotografie v rámci protokolu (poradové číslo fotografie)
  - *photo* [Asset] - zakódovaná fotografia v podobe prílohy
  - *protoID* [Int] - identifikačné číslo konkrétneho protokolu
  - *value* [Double] - nameraná hodnota

4.5 Generovanie PDF dokumentu

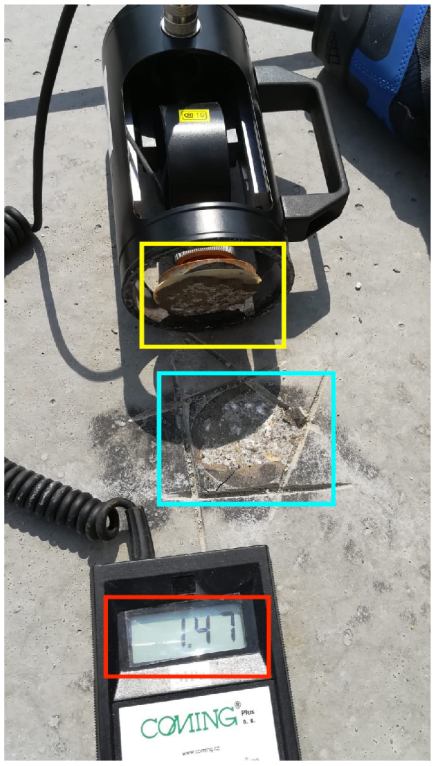
Pre generovanie a následné zobrazenie PDF dokumentu bol vybraný framework *PDFKit* 3.12. Vygenerovaný dokument by mal obsahovať jednoduchú a ľahko čitateľnú štruktúru. Návrh tejto štruktúry sa nachádza v prílohe A.

Dokument obsahuje v hlavičke logo, údaje o firme, ktorá daný dokument vytvorila a číslo protokolu. Ďalej dokument obsahuje údaje o stavenisku, na ktorom sa revízna kontrola vykonala, údaje o počasí automaticky získané aplikáciou. Údaje o firme, ktorá si objednala revíznu kontrolu. Výrobcu, názov, merné jednotky a sériové číslo zariadenia, ktoré bolo využité pri meraní. Opis metódy skúšky a pracovného postupu, na základe ktorých bola vykonaná revízia. V neposlednom rade dáta o jednotlivých častiach merania, dátum výkonu skúšky a materiál, prípadne podklad pre ktorý bola skúška realizovaná.

4.6 Rozpoznanie hodnoty vo fotografii

Ako je opísané v sekcii 2.2, je potrebné ukladať fotografie, ktoré minimálne zobrazujú prístroj s nameranou hodnotou. Nakoľko je takáto fotografia potrebná pre každý vykonaný odtrh (časť skúšky), poskytuje to príležitosť veľkého zrýchlenia procesu vytvárania výstupného protokolu. Užívateľ zadá fotografiu s nameranou hodnotou a vďaka frameworku *Vision* 3.11.1 sa z fotografie získa nameraná hodnota.

Na obrázku 4.2 sa nachádza ukážka fotografie z revíznej odtrhovej kontroly. V časti fotografie, ktorá je označená červenou farbou sa nachádza displej prístroja obsahujúci výslednú hodnotu odtrhu. Žltý štvorec označuje spodnú stranu odtrhového telesa a tyrkysový štvorec označuje miesto testovanej plochy.



Obr. 4.2: Fotografia časti revíznej odtrhovej kontroly

Kapitola 5

Implementácia

Pre implementáciu aplikácie bolo využité vývojové prostredie XCode verzie 12.4. Aplikácia je implementovaná pomocou jazyka Swift. Súčasťou implementácie je framework *UIKit* pre možnosť tvorby UIDocumentov, framework *CoreData* pre prácu s perzistentnými dátami, framework *CloudKit* pre umožnenie komunikácie so vzdialeným úložiskom. Mimo frameworku od spoločnosti *Apple* bol využitý aj externý framework *ZIP* pre vytváranie súborov typu ZIP a extrahovanie dát z týchto súborov.

V tejto kapitole je opísaná implementácia štartu aplikácie, akým spôsobom bolo vytvorené užívateľské rozhranie. Ďalej obsahuje opísanie získavania hodnoty z vytvorených fotografií. Ukážku ukladania a získavania dát z perzistentných úložísk. Na konci kapitoly je priblížená komunikácia aplikácie so vzdialeným úložiskom.

5.1 Štart aplikácie

V tejto sekcii je opísaná implementácia štartu aplikácie. Po defaultnom a bezproblémovom spustení aplikácie je vyvolaná funkcia zobrazená na obrázku 5.1 obsiahnutá v triede *AppDelegate*. Funkcia najskôr získa cesty, na ktorých by sa mali nachádzať priečinky, ktoré budú potrebné pre ukladanie vytvorených fotiek, protokolov a výstupných súborov aplikácie. V prípade, že tieto priečinky ešte neexistujú, tým pádom sa jedná o štart aplikácie na novom zariadení, kde ich vytvorí.

Následne funkcia pristupuje k objektu obsluhujúcemu komunikáciu so vzdialeným úložiskom. Skontroluje existenciu zóny, v rámci ktorej bude aplikácia ukladať všetky svoje záznamy. Ďalej zaregistruje aplikáciu pre možnosť odchytávania vzdialených notifikácií a skontroluje existenciu predplatného na vzdialenom úložisku. Úspešne zaregistrované predplatné bude vytvárať tiché notifikácie, ktoré aplikácia odchytáva a reaguje na nich rozdielovým fetchom. Ako poslednú akciu, zavolá funkciu *doDiffFetch()*, ktorá vykoná rozdielový fetch pre získanie najnovších zmien zo vzdialeného úložiska, toto zaručí najnovšie dáta, v prípade že aplikácia nebola na zariadení nainštalovaná, alebo počas neaktivity neobdržala notifikácie o zmene na vzdialenom úložisku.

5.2 Užívateľské rozhranie

Táto sekcia obsahuje implementáciu navrhnutého užívateľského rozhrania zo sekcie 4.3. Implementácia užívateľského rozhrania prebiehala pomocou frameworku *SwiftUI*. Tento fra-

```swift
func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions:
    ↪ [UIApplication.LaunchOptionsKey: Any]?
) -> Bool {
    guard let imagesPath = Dirs.shared.getImagesDir() else
    { return false }
    guard let documentsPath = Dirs.shared.getProtosDir() else
    { return false }
    guard let outputPath = Dirs.shared.getOutputDir() else
    { return false }
    guard Dirs.shared.createDir(at: imagesPath.path) == true else
    { return false }
    guard Dirs.shared.createDir(at: documentsPath.path) == true else
    { return false }
    guard Dirs.shared.createDir(at: outputPath.path) == true else
    { return false }

    Cloud.shared.startZone()
    UIApplication.shared.registerForRemoteNotifications()
    Cloud.shared.startSubscript()
    Cloud.shared.doDiffFetch()
}
```

Obr. 5.1: Funkcia volaná po úspešnom štarte aplikácie

mework dovoľuje vytvárať časti užívateľského rozhrania ako štruktúry, ktoré spĺňajú *View* protokol. Časť užívateľského rozhrania je ďalej v texte zastúpená slovom zobrazenie.

Pre jednoduché orientovanie v rámci hlavných zobrazení rozhrania bola zvolená štruktúra *TabView*. Každý element TabView (ďalej karta) je zaobalený v štruktúre *NavigationView*. Vďaka zvolenému prístupu sa jednotlivé vnárania v rámci kariet navzájom neprekrývajú, a tak nedochádza k neželanému správaniu užívateľského rozhrania. Túto implementáciu je možné nájsť v súbore *ContentView.swift*

Vysúvacie modálne okná

Ďalšou súčasťou užívateľského rozhrania sú vysúvacie modálne okná. Je to okno, ktoré sa zobrazí nad celou, práve používanou plochou aplikácie. Ich hlavnou výhodou je zobrazenie nového obsahu bez zbytočného zanárania sa hlbšie do užívateľského rozhrania. Za pomoci týchto okien sú implementované zobrazenia:

- **CreatorView** - zobrazí sa pri prvom spustení aplikácie na zariadení ako jednoduchý formulár, ktorý žiada užívateľa o vyplnenie údajov o jeho firme. V prípade nevyplnenia týchto údajov je možné pokračovať vo vytváraní protokolu, no nie je možné vytvoriť výstupné súbory, nakoľko súčasťou protokolu je aj firma, ktorá vytvára protokol.
- **CalendarView** - zobrazí sa pri zadávaní dátumu do protokolu. Obsahuje kalendár a tlačidlo na potvrdenie vloženého dátumu.

- **FilterView** - je vyvolané pomocou tlačidla pre filtrovanie protokolov v zobrazení pre vytvorené protokoly. Obsahuje jednoduchý vyberač, v ktorom si užívateľ vyberie na základe čoho chce vytvorené protokoly filtrovať a textové pole, do ktorého zadá hľadanú hodnotu.

Protocol View

Je zobrazenie, ktoré je používané pri vytváraní nového protokolu, ale aj pri editácii už vytvoreného. Nakoľko tieto dve zobrazenia (nový protokol, vytvorený protokol) sa od seba líšia iba tlačidlami na spodnej časti zobrazenia, bolo jednoduchšie využiť iba jednu štruktúru, ktorá implementuje obe zobrazenia. Rozlíšenie týchto zobrazení je už pri samotnej inicializácií štruktúry a to vďaka voliteľnému argumentu *protoID*. V prípade, že argument je prázdny jedná sa o nový protokol, ináč sa jedná už o vytvorený protokol.

V prípade inicializácie vytvoreného protokolu sa zobrazenie inicializuje s nenaplnenými dátami. Keď je zobrazenie zvolené a má sa zobraziť na zariadení, vyvolá to funkciu *openDocument()*, ktorá na základe hodnoty uloženej v premennej *protoID* otvorí príslušný *UIDocument* s obsahom príslušného protokolu. Nakoľko UIDocument obsahuje zakódovanú podobu protokolu bez fotiek, vyplnenie zobrazenia dátami je skoro okamžité a tak má užívateľ malú pravdepodobnosť vidieť zobrazenie pred vyplnením príslušnými dátami. Po dokončení úprav protokolu a následnej zmene zobrazenia sa *UIDocument* zatvára. V prípade modifikácie sa ukladá jeho nová verzia lokálne a modifikuje aj záloha na vzdialenom úložisku.

Takýto prístup zaručuje otvorenie práve jedného *UIDocumentu* a vykonávanie akcií nad ním.

Photo View

Jedným z elementov zobrazenia protokolu je aj možnosť vnorenia užívateľského rozhrania do zobrazenia fotiek príslušných danému protokolu. Toto vnorenie je implementované pomocou štruktúry *NavigationLink*, tá berie ako parameter štruktúru, ktorá sa má užívateľovi zobraziť v tomto prípade to je štruktrúra *PhotosView*.

PhotosView je formulár, ktorý obsahuje riadky s fotografiami a ich dátami a tlačidlo pre vyvolanie listu akcií, na ktorom si užívateľ môže zvoliť či chce fotku importovať z knižnice fotografií alebo vytvoriť novú. Každá fotografia sa načítava asynchrónne. Vďaka tomu je užívateľské rozhranie viac responzívne, no užívateľovi sa môže stať, že kým sa príslušné fotky nenačítajú bude vidieť zástupnú fotografiu takzvaný placeholder.

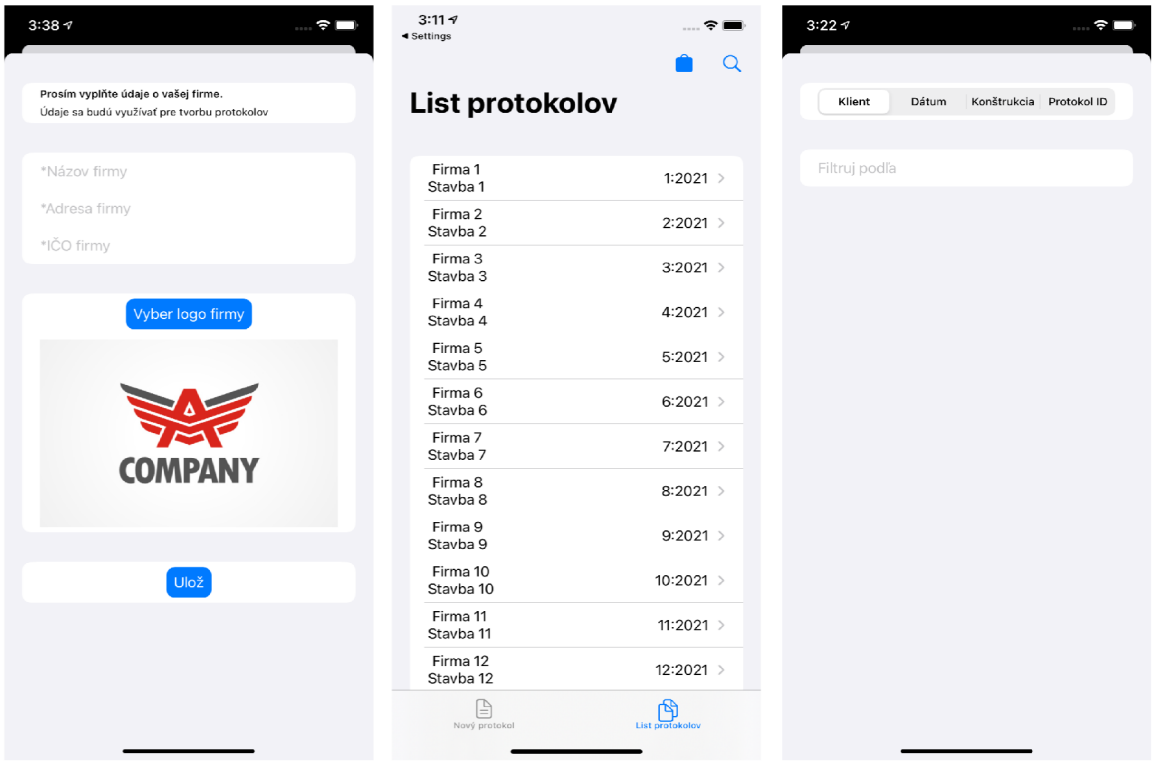
Každý riadok obsahuje fotografiu a dáta, ktoré sú zobrazené aj v prípade, že fotografia ešte nebola načítaná. Po dvojitom kliknutí na dáta sa zobrazí užívateľovi výsuvné modálne okno s možnosťou úpravy dát danej fotografie.

ProtocolList View

ProtocolListView je formulár obsahujúci všetky protokoly uložené v lokálnom perzistentnom úložisku. Každý riadok predstavuje jeden protokol, na riadku je vyobrazený názov spoločnosti, pre ktorú bol protokol vytváraný, názov stavby, na ktorej bol protokol vytváraný a číslo protokolu.

Ďalej zobrazenie obsahuje v hornom pravom rohu dve ikony. Ikona lupy zobrazí vysúvacie modálne okno, v ktorom môže užívateľ nastaviť podľa čoho chce filtrovať zobrazené protokoly. Implementácia filtrovania je vysvetlená v sekcii 5.4.1. Druhá ikona zobrazujúca prázdny, alebo plný kufrík predstavuje, či užívateľ vyplnil informácie o firme. V prípade

kliknutia na ikonu sa užívateľovi zobrazí vysúvacie modálne okno. V ňom má možnosť vyplniť, prípadne upraviť informácie o firme. Ukážka tohto zobrazenia sa nachádza na obrázku 5.2.



Obr. 5.2: Zobrazenie protokolov

5.3 Získanie dát z fotografie

Nakoľko fotografie využívané užívateľom v aplikácii budú špecifické, bol zvolený prístup vyhodnocovania hodnoty získanej z fotografií dvomi od seba nezávislými metódami. Obidve metódy využívajú framework *Vision* 3.11.1 od spoločnosti *Apple*. A pre získanie správnejšej hodnoty sa porovnávajú dôveryhodnosti (*confidence*) rozpoznaných hodnôt. Dôveryhodnosť je získaná pri rozpoznaní hodnoty.

Fotografia sa získa v podobe *UIImage*. V prípade, že bola fotografia práve vyhotovená, môže sa stať, že orientácia fotografie je zapísaná ako *.right* aj keď sa fotografia vykreslí normálne, kvôli presnejšiemu výsledku je fotografia otočená tak, aby jej orientácia bola označená ako *.up*. Po korekcii fotografie je jej reprezentácia v podobe *CGImage* odoslaná do funkcie *recognize*, ktorá patrí objektu *TextRecognizer*. Ten rozpoznáva hodnotu vo fotografii.

Získanie validnejšej hodnoty

Funkcia *recognize* beží vo vedľajšom vlákne. Vďaka tomu môže užívateľ ďalej pracovať s aplikáciou počas rozpoznávania hodnoty.

Najskôr sa hodnota získava za pomoci modelu strojového učenia s názvom *MNISTClassifier*, ktorý bol stiahnutý z vývojárskych stránok *Applu* a jeho licencia umožňuje jeho voľné využívanie. Model bol trénovaný na obrázkoch, ktoré obsahovali čísla písané rukou. Nakoľko

model dokáže rozpoznať len jednotlivé číslice a nie desatinné čísla, bol problém so zistením desatinnej čiarky. Tento problém je asi najväčší s používaním takéhoto rozpoznávania. Na základe existencie tohto problému prebehla komunikácia s potencionálnym užívateľom, kde sa zisťoval približný rozsah reálnych hodnôt. Užívateľ oznámil, že hodnota viac ako 90% vykonaných testov je menšia ako 10. Na základe toho bola upravená implementácia, tak aby po rozpoznaní prvej hodnoty bola vložená desatinná čiarka a pokračovalo sa ďalej v rozpoznávaní. Tento postup je postačujúci, spolu s použitím porovnania takto získanej hodnoty, s hodnotou získanou z druhej metódy. Výpočet validity tejto hodnoty je súčet dôveryhodnosti každého použitého čísla, vydelený počtom čísel.

Druhá metóda získava celú hodnotu za pomoci triedy *VNRecognizeTextRequest*. Táto trieda vytvára žiadosť na rozpoznanie textu z obrázka. Takéto rozpoznávanie navracia rozpoznaný text z celého obrázku v poli s hodnotami typu *Any*. Hodnoty v poli sa pretypujú na *VNRecognizedTextObservation*, vďaka tomu sa môže pristúpiť na hodnoty a ich dôveryhodnosti. Každá hodnota poľa prechádza porovnaním minimálnej dôveryhodnosti, v prípade že jej hodnota presahuje dôveryhodnosť 50% a zároveň sa rozpoznaná hodnota skladá iba z čísel. Uloží sa ako potenciálna rozpoznaná hodnota a dôveryhodnosť tejto hodnoty ďalej slúži ako minimálna dôveryhodnosť, ktorá musí byť prekonaná pre zapísanie novej potenciálnej hodnoty.

Pre získanie výslednej rozpoznanej hodnoty sa porovnajú dôveryhodnosti rozpoznaných hodnôt a uloží sa tá s vyššou dôveryhodnosťou. V prípade, že ani jedna z metód nebola úspešná, navráti sa hodnota -1,0.

5.4 Využitie CoreData

Pre ukladanie informácií o vytvorených protokoloch, fotkách a verziách protokolov bol využitý framework *CoreData*. Hlavným dôvodom jeho využitia bolo ukladanie na lokálnom zariadení bez využitia internetového pripojenia a jeho možnosť jednoduchého ukladania, filtrovania a načítavania uložených dát. Entity obsiahnuté vo vytvorenom dátovom modeli sú opísané v sekcii 4.4.1.

Pomocné funkcie Každá entita obsahuje pomocné funkcie pre prácu s jej príslušnými súbormi, uloženými v sandboxe 3.7 aplikácie.

- *MyPhoto* obsahuje funkcie pre ukladanie a vymazávanie fotografie zo sandboxu aplikácie.
- *OutputArchive* obsahuje funkcie pre vytváranie a vymazávanie výstupných súborov typu PDF a ZIP.
- *DatabaseArchive* obsahuje funkcie pre ukladanie, modifikovanie a vymazanie UIDokumentov obsahujúcich všetky informácie o protokole.

5.4.1 Filtrovanie protokolov

Na vytváranie filtrov nad dátovým modelom vytvoreným pomocou frameworku CoreData sa využíva trieda *NSPredicate*, reprezentuje logické podmienky, vďaka ktorým je možné vyhľadať objekty obsiahnuté v modeli.

Na zobrazenie filtrovaných protokolov bola vytvorená štruktúra *Filtred*, ktorá zobrazí všetky protokoly pre vytvorený predikát. Na inicializáciu danej štruktúry je potrebné jej predať kľúč a hľadaný výraz. Kľúč predstavuje premennú obsiahnutú v entite dátového

modelu, nad ktorou sa vytvára filter. Hľadaný výraz je obsah danej premennej. Nie je potrebné zadať celý obsah premennej pre vyhľadávanie objektu. V prípade, že obsah výrazu je prázdny, predikát sa nevytvára a štruktúra zobrazí všetky protokoly zoradené podľa ich čísla vzostupne. Kód s inicializáciou štruktúry sa nachádza na obrázku 5.3

```
1    init(filterKey: String, filter: String, content: @escaping (T) ->
     ↪ Content){
2        let sortDesc: [NSSortDescriptor] = [NSSortDescriptor(key:
         ↪ "protoID", ascending: true)]
3        if filter != "" {
4            fetchedRequest = FetchRequest<T>(entity: T.entity(),
             ↪ sortDescriptors: sortDesc, predicate: NSPredicate(format:
             ↪ "%K BEGINSWITH %@", filterKey, filter))
5        }
6        else{
7            fetchedRequest = FetchRequest<T>(entity: T.entity(),
             ↪ sortDescriptors: sortDesc)
8        }
9
10       self.content = content
11   }
```

Obr. 5.3: Inicializácia štruktúry zobrazujúcej vyfiltrované objekty

5.5 Využitie UserDefaults

Úložisko *UserDefaults* bolo využité pre uloženie informácií o firme, ktorá vytvára protokol a jej logo. Vďaka tomu môže užívateľ iba raz vyplniť informácie o firme, pre ktorú pracuje a ďalej vytvárať výstupné protokoly, ktoré ako hlavičku obsahujú danú firmu s jej logom.

Druhé využitie *UserDefaults* bolo pre ukladanie časových značiek, využívaných operáciou *CKFetchRecordZoneChangesOperation*.

5.6 Získanie informácií o počasí

Pre získanie informácií o teplote a vlhkosti ovzdušia, na aktuálnej lokácii užívateľa, bol využitý objekt *CLLocationManager*. Vďaka nemu má aplikácia možnosť pristúpiť k aktuálnej polohe zariadenia. Implementácia získavania polohy sa nachádza v triede *WeatherService*. Pri inicializácii triedy aplikácia žiada povolenie o zdieľaní polohy počas využívania aplikácie. Pokiaľ užívateľ povolí zdieľanie polohy, aplikácia pri zobrazení nového protokolu volá funkciu *loadWeatherData*, ktorá asynchrónne navracia získané dáta o počasí.

Dáta sa získavajú po načítaní aktuálnej polohy skrz API od spoločnosti *OpenWeather*. Poskytujú možnosť vytvoriť 60 volaní API za minútu bez platenia, alebo poskytujú aj možnosť využívať ich API ako študent zadarmo. Po obdržaní dát vo formáte JSON sa dáta dekódujú do štruktúry *APIResponse*. Ukážka volania API sa nachádza na obrázku 5.4.

```swift
func makeDataRequest(for coordinates: CLLocationCoordinate2D) {
        guard let urlString =
        "https://api.openweathermap.org/data/2.5/weather?
        lat=\(coordinates.latitude)&lon=\(coordinates.longitude)&
        appid=\(API_KEY)&units=metric"
        .addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)
        ↪ else { return }
        guard let url = URL(string: urlString) else { return }
        URLSession.shared.dataTask(with: url) { data, response, error in
            if let err = error {
                printError(from: "makeDataRequest", message:
                ↪ err.localizedDescription)
                return
            }
            guard let data = data else { return }
            if let response = try? JSONDecoder().decode(APIResponse.self,
            ↪ from: data) {
                self.completitionHandler?(Weather(response: response))
            }
        }.resume()
    }
```

Obr. 5.4: Funkcia pre získanie dát o aktuálnom počasí na lokácii zariadenia

5.7 Práca s CloudKitom

V tejto sekcií je opísaná implementácia komunikácie so vzdialenou databázou navrhnutou v sekcii 4.4.2 za pomoci využitia informácií získaných zo sekcie 3.5.

Pre všetku komunikáciu bol vytvorený jedináčik (singleton), ktorý dedí z triedy *Cloud*. Pristupuje sa k nemu skrz triednu premennú *shared*. Vďaka tomuto prístupu je možné sa vyvarovať duplicite objektov pri ich sťahovaní zo vzdialeného úložiska.

Priebeh sťahovania zmien

Po obdržaní a zaregistrovaní upozornenia od vzdialeného úložiska sa vykoná funkcia *doDiffFetch()*. Tá zabezpečí uloženie všetkých zmien vykonaných na vzdialenom úložisku od poslednej kontroly označenej časovou značkou uloženou v *UserDefaults*. Tieto zmeny sú záznamy, ktoré sa ukladajú do privátnych triednych premenných. Záznamy sa na základe svojich typov (*CKRecordType*) uložia do premenných, ako objekty príslušných entít, ktoré ale ešte nie sú vložené do perzistentného úložiska.

Po dokončení ukladania zmenených, nových alebo zmazaných záznamov sa vykoná funkcia *insertFetchChangeIntoCoreData()*. Táto funkcia prechádza do hlavného vlákna nakoľko ide pristupovať k lokálnej databáze, vďaka ktorej sú záznamy zobrazené užívateľovi. Ako prvé funkcia načíta všetky už lokálne uložené záznamy (protokoly, fotografie, výstupy) pre porovnávanie novo obdržaných záznamov. V prípade, že bol záznam zo vzdialeného úložiska vymazaný, nájde sa jeho lokálna kópia a taktiež sa vymaže. Porovnávanie zaručí modifikáciu lokálneho záznamu namiesto vloženia nového duplicitného záznamu.

Ukladanie na vzdialenom úložisku

Každý typ (*CKRecordType*) záznamu má svoju vlastnú funkciu, ktorá vykonáva ukladanie do vzdialenej databázy. Pre jednoduché používanie týchto funkcií sú metódy preťažované, ich prvým argumentom je typ záznamu kvôli kontrole, aby funkcia nepristúpila k dátam, ktoré jej neboli predané, prípadne neuložila dáta do nesprávnej štruktúry záznamu.

Po vytvorení záznamu sa zavolá operácia nad vzdialenou privátnou databázou pre asynchrónne uloženie záznamu. V prípade neúspešného uloženia záznamu vracia chybovú hlášku. Ak ukladanie na vzdialené úložisko prebehlo úspešne, vracia jeho štruktúru, ktorá obsahuje aj identifikátor daného záznamu (*CKRecord.ID*) na vzdialenom úložisku. Navrátený identifikátor sa uloží do príslušného objektu v CoreData. Ukážka funkcie ukladania protokolu sa nachádza na obrázku 5.5.

```
1  func saveToCloud(recordType: CKRecord.RecordType, protoID: Int,
   ↪  encodedProto: String, completition: @escaping (CKRecord.ID?) -> ()) {
2          guard recordType == RecordType.protocols else {
3              printError(from: "save to cloud [protocol]", message: "Record
                ↪  type is not correct")
4              return
5          }
6          let recordID = CKRecord.ID(zoneID: zoneID)
7          let record = CKRecord(recordType: recordType, recordID: recordID)
8          record["protoID"] = protoID as CKRecordValue
9          record["encodedProto"] = encodedProto as CKRecordValue
10
11         db.save(record) { record, err in
12             DispatchQueue.main.async {
13                 if let err = err {
14                     printError(from: "cloud save [protocol]", message:
                        ↪  err.localizedDescription)
15                     completition(nil)
16                 } else {
17                     guard let record = record else {
18                         printError(from: "cloud save [protocol]", message:
                            ↪  "Returned record is nil")
19                         completition(nil)
20                         return
21                     }
22                     print("Protocol saved on cloud")
23                     completition(record.recordID)
24                 }
25             }
26         }
27     }
```

Obr. 5.5: Funkcia pre ukladanie vytvoreného protokolu na vzdialené úložisko

Modifikovanie na vzdialenom úložisku

Tak ako každý typ záznamu má svoju vlastnú funkciu pre uloženie na vzdialené úložisko má aj vlastnú funkciu pre modifikovanie uloženého záznamu. Funkcia pozostáva z dvoch volaní asynchrónnych operácií *fetch* a *save*.

Pre modifikovanie záznamu je potrebné sa uistiť o existencii záznamu na vzdialenom úložisku. K tomu slúži operácia *fetch*, ktorá vyhľadá záznam na základe jeho identifikátora, ktorý je uložený v perzistentnom úložisku *CoreData* a je predaný funkcii skrz argument. Operácia *fetch* asynchrónne navráti záznam zo vzdialenej privátnej databázy. V prípade existencie záznamu, navrátený záznam je nenulový, sa dáta predané funkcií skrz argument v prípade potreby zakódujú pomocou *JSONEnkodéru* a je nimi naplnená navrátená štruktúra záznamu. Takto modifikovaný záznam je potrebné uložiť na vzdialené úložisko. Uloženie prebieha pomocou operácie *save*, ktorá je opísaná v sekcii 5.7. Ukážka funkcie využívaná pre modifikovanie záznamu obsahujúci protokol sa nachádza na obrázku 5.6.

```
1  func modifyOnCloud(recordID: CKRecord.ID, proto: Proto) {
2          db.fetch(withRecordID: recordID) { record, err in
3              if let err = err {
4                  printError(from: "modify protocol on cloud", message:
                   ↪  err.localizedDescription)
5                  return
6              }
7              guard let record = record else { return }
8              let encoder = JSONEncoder()
9              encoder.outputFormatting = .prettyPrinted
10             guard let encodedProto = try?
               ↪  encoder.encode(proto).base64EncodedString() else {
11                 printError(from: "modify protocol on cloud", message:
                   ↪  "Cannot encoded proto \(proto.id)")
12                 return
13             }
14             record["encodedProto"] = encodedProto as CKRecordValue
15             self.db.save(record) { record, err in
16                 if let err = err {
17                     printError(from: "modify protocol on cloud", message:
                       ↪  err.localizedDescription)
18                     return
19                 }
20                 guard let _ = record else { return }
21                 print("Protocol modified on cloud")
22                 return
23             }
24
25         }
26     }
```

Obr. 5.6: Funkcia pre modifikovanie záznamu obsahujúceho protokol

Vymazávanie zo vzdialeného úložiska

Nakoľko pre vymazanie záznamu zo vzdialeného úložiska je potrebné vedieť iba identifikátor záznamu, postačuje implementácia jednej funkcie, ktorá zabezpečuje vymazanie všetkých typov záznamov. Funckcii je odovzdaný identifikátor záznamu pre vymazanie. V rámci funkcie je volaná asynchrónna operácia *delete*, ktorá obsluhuje vymazanie záznamu zo vzdialenej privátnej databázy. Pri úspešnom vymazaní je navrátený identifikátor vymazaného záznamu.

Kapitola 6

Testovanie

V kapitole je rozobratý prístup testovania aplikácie. Fyzické zariadenia, na ktorých bola aplikácia testovaná sú iPhone X a iPhone SE. Ďalej bola aplikácia testovaná aj na zariadeniach vytvorených pomocou simulácie, tieto zariadenia boli iPhone 11, iPhone 11 Pro, iPhone 12. Prvý bude v kapitole uvedený rozbor testovania funkčnosti aplikácie, následne rozbor testovania užívateľského rozhrania. Na konci kapitoly sa nachádza sekcia s vykonaním simulovanej revíznej kontroly.

6.1 Priebežné testovanie

Počas vývoja bola aplikácia neustále testovaná. Každej novej funkcii bol venovaný čas pre zaistenie správneho fungovania. V tomto štádiu bola aplikácia testovaná prevažne na simulátoroch poskytovaných vývojovým prostredím *Xcode*. Tento druh testov sa vykonával primárne na simulovanom zariadení iPhone 11 ale testy sa vykonávali aj na iPhone 11 Pro a iPhone 12. Keď aplikácia mala už implementovanú väčšiu časť svojej očakávanej funkcionality, testovanie sa prenieslo aj na reálne zariadenie iPhone X. Počas priebežného testovania bola upretá pozornosť najmä na správne vymazávanie už nepotrebných dát z lokálneho zariadenia ale aj zo vzdialeného úložiska a na lokálne ukladanie zmien vytvorených na vzdialenom úložisku bez vytvárania duplicitných záznamov.

6.2 Užívateľské rozhranie

Užívateľskému rozhraniu je potrebné venovať veľkú časť vývoja. Aby zabezpečovalo intuitívne a prehľadné zobrazenia. Testovanie užívateľského rozhrania prebiehalo primárne tak, aby jednotlivé prvky boli rozmiestnené logicky a ich pozícia dávala význam užívateľovi. Ďalej bola testovaná funkčnosť daných prvkov. Počas tohto testovania bola napríklad pridaná deaktivácia tlačidla pre vytváranie výstupných súborov počas zapisovania nových fotografií do SandBoxu aplikácie. Tento prístup užívateľovi nedovolí vytvoriť výstupné súbory bez fotografií, ktoré síce zadal, ale ešte sa nestihli asynchrónne zapísať.

6.3 Simulovaná revízna kontrola

Na vykonanie simulovanej revíznej kontroly na stavenisku v Bratislave, mi bolo udelené povolenie od firmy *Syncol s.r.o.*. Zároveň mi poskytli informácie o stavenisku a prístroj na vykonanie revízie od spoločnosti *Coming Plus a.s.* s výrobným číslom 424. Pred príchodom

na stavbu som si do aplikácie zadal logo firmy *Syncol s.r.o.* spolu s jej identifikačnými údajmi, nakoľko protokol musí obsahovať v hlavičke údaje tejto firmy.

Po príchode na sekciu stavby, kde bolo potrebné vykonať revíznu odtrhovú kontrolu, som pripravil a nalepil desať odtrhových teliesok. Počas tvrdnutia lepidla som mal možnosť v aplikácii vyplniť informácie potrebné pre tvorbu protokolu, ktoré mi boli poskytnuté.

Po vytvrdnutí lepidla som urobil desať odtrhových meraní. Výsledky týchto meraní sa nachádzajú v tabuľke 6.1. Z výsledkov vykonanej kontroly vyplýva, že revízna kontrola nevyhovuje, preto musí byť znovu vykonaná. Z desiatich odtrhov iba šesť splnilo požadovanú hodnotu, tri merania hodnotu nesplnili a jedno meracie teliesko sa odlepilo pri nasádzaní prístroja. V prípade, že by ostatných deväť meraní splnilo požadovanú minimálnu hodnotu, pokazené meranie by som opakoval. Nakoľko ale niektoré merania nesplnili minimálnu hodnotu, bude sa skúšaný materiál technologicky upravovať a následne sa vykoná opravné meranie. Preto nebol dôvod pokazené merania v danej chvíli opakovať.

Číslo odtrhu	Nameraná hodnota [MPa]	Požadovaná hodnota [MPa]	Popis	Výsledok
1	2.74	1.6	-	Vyhovuje
2	2.22	1.6	-	Vyhovuje
3	1.47	1.6	-	Nevyhovuje
4	2.36	1.6	-	Vyhovuje
5	1.52	1.6	-	Nevyhovuje
6	1.50	1.6	-	Nevyhovuje
7	2.3	1.6	-	Vyhovuje
8	1.98	1.6	-	Vyhovuje
9	1.7	1.6	-	Vyhovuje
10	0.2	1.6	Odlepený terč	Nevyhovuje

Tabuľka 6.1: Tabuľka meraní

Po ukončení revíznej kontroly som mal vyplnenú štruktúru protokolu v aplikácii, k danému protokolu som priložil fotografie a počkal, pokiaľ sa rozpozná ich hodnota. Po rozpoznaní hodnôt z fotografií som prišiel na chybu pri rozpoznávaní. Hodnoty rozpoznané z fotografií, ktoré obsahovali merací prístroj odfotený z väčšej diaľky, neboli rozpoznané. Táto chyba bola riešená pridaním *ML Modelu*, ktorý síce zvýšil pravdepodobnosť získania hodnoty z fotografie, no často tieto hodnoty iba aproximovali k reálnym hodnotám nameraných na fotografiách, čo vyvolalo otázku tvorby vlastného *ML Modelu* v prípade vytvorenia rozšírenia aplikácie.

Po opravení niektorých rozpoznaných hodnôt som vygeneroval výstupný protokol (pozri príloha B). Takto vygenerovaný protokol spolu s fotografiami nachádzajúcimi sa v súbore formátu ZIP, som jednoducho zazdieľal pracovníkovi, ktorý dozeral na môj postup pri revíznej kontrole.

6.4 Výsledky

Vďaka neustálemu priebežnému testovaniu boli odstránené všetky chyby, ktoré sa počas testovania vývojárovi prejavili. Jediná chyba, ktorá je spomenutá v sekcii 6.3, je získavanie presnej hodnoty z fotografie, ktorá ale nebráni hladkému chodu aplikácie. Počas testovania boli pridané funkcie ako dočasné zamedzenie vytvárania výstupných súborov pre zaistenie

správneho obsahu dát spomenuté v sekcii 6.2, vytvorenie možnosti zapísať údaje o užívateľovej firme pri prvom spustení aplikácie, ale aj vytvorenie samostatného tlačidla pre úpravu údajov o danej firme. Možnosť upraviť nielen hodnotu prislúchajúcu fotografii, ale pridať aj jej popis, prípadne možnosť zapísať aké veľké teleso bolo využité pre vykonanie skúšky.

Kapitola 7

Záver

Cieľom bakalárskej práce bolo vytvorenie aplikácie pre jednoduchšie a rýchlejšie vykonávanie revíznych kontrol na stavbách. Aplikácia je kompatibilná so zariadeniami od spoločnosti Apple s operačným systémom iOS 14.4. Hlavnou úlohou aplikácie je lokálne ukladanie a spravovanie dát zadaných užívateľom a zálohovanie ich na vzdialenom serveri. Aplikácia tiež obsahuje prvky, vďaka ktorým má užívateľ možnosť urýchliť proces vytvárania výstupného protokolu o revíznej skúške. Týmito prvkami sú: získanie údajov o počasí na aktuálnej lokácii užívateľa, automatické vyhodnotenie jednotlivých častí protokolu, získanie hodnoty z fotografie zadanej užívateľom, jednoduché zdieľanie vytvorených výstupných súborov a prehľad histórie určitého protokolu.

Užívateľom vytvorené dáta sú uložené a môže k nim pristupovať v ľubovolnom čase. Nakoľko sú dáta zálohované aj na vzdialenom úložisku, užívateľ o svoje dáta neprichádza pri strate zariadenia alebo kúpe nového. Po prihlásení sa svojim Apple ID na zariadení a nainštalovaní aplikácie, sa jeho dáta automaticky stiahnu zo vzdialeného úložiska.

Pre lokálne ukladanie dát bolo využité CoreData a pre menší objem dát úložisko UserDefaults. Úložisko CoreData umožnilo vytvorenie filtrovania pre určité klúče zadané užívateľom. Počas vývoja aplikácie bolo pridané rozšírenie pre automatické načítanie dát o počasí.

7.1 Možné rozšírenia

Jedným z veľmi populárnych rozšírení by sa mohlo stať rozšírenie umožnujúce zdieľanie vytvorených protokolov. Toto rozšírenie by mohlo zaručiť, že v prípade nemožnosti účasti užívateľa na revíznej kontrole, by svoj už vytvorený protokol zazdieľal kolegovi, ktorý tým nadobudne možnosť práce s týmto protokolom a vytvorenie nových verzií.

Druhé rozšírenie by sa malo zaoberať lepším rozpoznávaním hodnoty z fotografie vyhotovenej pri revíznej kontorole. Je možné toto implementovať za pomoci tvorby vlastného modelu strojového učenia alebo dvojstupňového rozpoznávania fotografie. V prvom stupni sa vyznačí časť fotografie, v ktorej sa nachádza prístroj obsahujúci hodnotu vyobrazenú na displeji. V druhom stupni sa vykoná rozpoznávanie hodnoty iba v časti fotografie vymedzenej v prvom stupni.

Dané rozšírenia neobsahuje finálna verzia bakalárskej práce, ale je v pláne ich implementovať.

7.2 Vývoj aplikácie v budúcnosti

V budúcnosti je v pláne pridať rozšírenia zdieľania rozpracovaných protokolov a lepšie rozpoznávanie hodnoty vo fotografii. Taktiež transformovať aplikáciu pre možnosť jej využitia na väčšine zariadení od spoločnosti Apple, ale aj na zariadeniach s operačným systémom Android.

Literatúra

[1] *Core ML* [online]. [cit. 2020-04-17]. Dostupné z: https://developer.apple.com/documentation/coreml.

[2] *IOS* [online]. [cit. 2020-04-08]. Dostupné z: https://en.wikipedia.org/wiki/IOS.

[3] *JSON* [online]. [cit. 2020-04-15]. Dostupné z: https://en.wikipedia.org/wiki/JSON.

[4] *Swift (programming language)* [online]. [cit. 2020-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Swift_(programming_language).

[5] *Xcode* [online]. [cit. 2020-04-08]. Dostupné z: https://en.wikipedia.org/wiki/Xcode.

[6] Bulavin, V. *Modern MVVM iOS App Architecture with Combine and SwiftUI* [online]. [cit. 2020-04-10]. Dostupné z: https://www.vadimbulavin.com/modern-mvvm-ios-app-architecture-with-combine-and-swiftui/.

[7] Concilio, I. *Image Processing and ComputerVision for iOS Applications: An Introduction* [online]. [cit. 2020-04-17]. Dostupné z: https://medium.com/academy-eldoradocps/image-processing-and-computervision-for-ios-applications-an-introduction-fe3171e10be7.

[8] Corporation, P. *MVVM* [online]. [cit. 2020-04-10]. Dostupné z: https://www.zkoss.org/wiki/ZK_Developer%27s_Reference/MVVM.

[9] Inc., A. *CKContainer* [online]. [cit. 2020-04-10]. Dostupné z: https://developer.apple.com/documentation/cloudkit/ckcontainer.

[10] Inc., A. *CKRecordZone* [online]. [cit. 2020-04-10]. Dostupné z: https://developer.apple.com/documentation/cloudkit/ckrecordzone.

[11] Inc., A. *Initializing the Core Data Stack* [online]. [cit. 2020-04-09]. Dostupné z: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/InitializingtheCoreDataStack.html#//apple_ref/doc/uid/TP40001075-CH4-SW1.

[12] Inc., A. *Membership Details* [online]. [cit. 2020-04-08]. Dostupné z: https://developer.apple.com/programs/whats-included/.

[13] Inc, A. *Package Manager* [online]. [cit. 2020-04-11]. Dostupné z: https://swift.org/package-manager/.

[14] Inc., A. *Setting Up a Core Data Stack* [online]. [cit. 2020-04-10]. Dostupné z: https://developer.apple.com/documentation/coredata/setting_up_a_core_data_stack.

[15] Inc., A. *Sharing CloudKit Data with Other iCloud Users* [online]. [cit. 2020-04-10]. Dostupné z: https://developer.apple.com/documentation/cloudkit/shared_records/sharing_cloudkit_data_with_other_icloud_users.

[16] Inc., A. *Swift* [online]. [cit. 2020-04-09]. Dostupné z: https://developer.apple.com/documentation/SwiftUI.

[17] Inc., A. *UIDocument* [online]. [cit. 2020-04-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uidocument.

[18] Inc., A. *UserDefaults* [online]. [cit. 2020-04-10]. Dostupné z: https://developer.apple.com/documentation/foundation/userdefaults.

[19] Inc., A. *What Is Core Data?* [online]. [cit. 2020-04-08]. Dostupné z: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html#//apple_ref/doc/uid/TP40001075-CH2-SW1.

[20] Jacobs, B. *What Is Application Sandboxing* [online]. [cit. 2020-04-10]. Dostupné z: https://cocoacasts.com/what-is-application-sandboxing/.

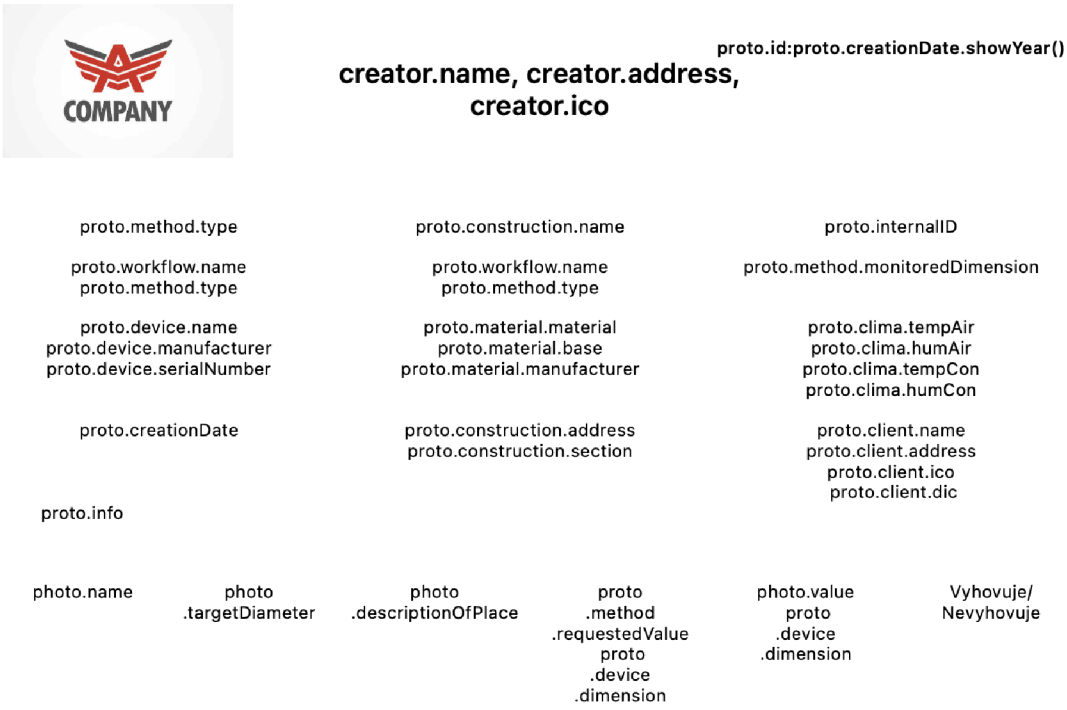
[21] Marmelstein, R. *Zip* [online]. [cit. 2020-04-11]. Dostupné z: https://github.com/marmelroy/Zip.

[22] Miller, A. *CKSharing step by step* [online]. [cit. 2020-04-10]. Dostupné z: https://medium.com/@adammillers/cksharing-step-by-step-33800c8950d2.

[23] Pereira, A. *CloudKit Tutorial: Getting Started* [online]. [cit. 2020-04-09]. Dostupné z: https://www.raywenderlich.com/4878052-cloudkit-tutorial-getting-started.

Príloha A

Príloha



Obr. A.1: Návrh generovaného protokolu

Príloha B

Príloha

Syncol s.r.o., Karlová 53, 038 15 Karlová
IČO: 45338795

Protokol o skúške číslo:

18:2021

Druh skúšky:
Odtrhová skúška

Stavba:
D4 Bratislava

Číslo dokumentu:
1

Použitá metóda skúšky:
Skúška bola zrealizovaná v súlade s pracovným postupom PP-67 "Odtrhová skúška"

Inštalácia meriaceho zariadenia:
Meracie zariadenie bolo nainštalováné v súlade s pracovným postupom PP-67 "Odtrhová skúška"

Sledované veličiny:
MPa

Použitý merací prístroj:

Názov: COMING
Výrobca: COMING Praha a.s.
Výrobné číslo: 424

Podklady pre vypracovanie protokolu a údaje o skúšanom objekte:
Skúšaný materiál: Materiál po tryskaní vodným lúčom
Zhotoviteľ: Syncol s.r.o.

Klimatické podmienky:

Teplota ovzdušia: 19.0 °C
Vlhkosť ovzdušia: 56.0 %
Teplota konštrukcie: 8.0 °C
Vlhkosť konštrukcie: 30.0 %

Dátum realizácie skúšky:
April 30, 2021

Miesto a predmet skúšky:
Jarovce 851 810
Most

Objednávateľ:
Syncol s.r.o.
Karlová 53, 038 15 Karlová
IČO: 45338795
DIČ: 2022946255

Označenie meraného miesta	Priemer terča	Popis meraného miesta	Požadovaná hodnota	Nameraná hodnota	Celkové hodnotenie
1	50.0 mm	-	1.6 MPa	2.74 MPa	Vyhovuje
2	50.0 mm	-	1.6 MPa	2.22 MPa	Vyhovuje
3	50.0 mm	-	1.6 MPa	1.47 MPa	Nevyhovuje
4	50.0 mm	-	1.6 MPa	2.36 MPa	Vyhovuje
5	50.0 mm	-	1.6 MPa	1.52 MPa	Nevyhovuje
6	50.0 mm	-	1.6 MPa	1.5 MPa	Nevyhovuje
7	50.0 mm	-	1.6 MPa	2.3 MPa	Vyhovuje
8	50.0 mm	-	1.6 MPa	1.98 MPa	Vyhovuje
9	50.0 mm	-	1.6 MPa	1.7 MPa	Vyhovuje
10	50.0 mm	Odlepený terč	1.6 MPa	0.2 MPa	Nevyhovuje

Obr. B.1: Vygenerovaný protokol z testovacej revíznej skúšky