Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Letecké záznamy pro iOS pomocí moderních architektur a FRP

Bc. Martin Žid

Vedoucí práce: Ing. Dominik Veselý

3. dubna 2018

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 3. dubna 2018 .....

Odkaz na tuto práci

Žid, Martin. *Letecké záznamy pro iOS pomocí moderních architektur a FRP.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce realizuje iOS aplikaci pro evidenci letů. V první části analyzuji obdobné aplikace a předpisy pro piloty České republiky, podle nichž probíhá návrh funkcionality vytvářené aplikace. Podle návrhu je následně zvolena vhodná architektura a vytvořeno uživatelského rozhraní v podobě wireframů.

Aplikace je implementována s použitím zvolené architektury a pomocí principů FRP. V průběhu implementace aplikace jsou realizovány jednotkové testy a na konci jsou provedeny uživatelské testy. Na základě výsledků testů je aplikace upravena do finální podoby.

V poslední části práce popisuji výhody a nevýhody, které přinesly postupy FRP. Také hodnotím časovou a implementační náročnost oproti standardním postupům a architektuře MVC.

V práci jsem vytvořil funkční iOS aplikaci s využitím moderní architektury a principů FRP. Aplikace bude sloužit pilotům České republiky pro elektronickou evidenci letů a bude jim také ulehčovat administrativu s evidencí spojenou.

V příloze této diplomové práce je možné nalézt všechny zdrojové kódy jak aplikace, tak i testů společně s vytvořenými wireframy.

Klíčová slova mobilní aplikace pro evidenci letů, iOS, Swift, FRP, ReactiveCocoa, MVVM architektura

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords flight records mobile application, iOS, Swift, FRP, ReactiveCocoa, MVVM architecture

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době, kdy existují mobilní aplikace na téměř vše, mě zarazil fakt, že u pilotů tomu tak nemusí být. Aplikace na evidenci letů samozřejmě existují, však je tu hned několik problémů. Tyto aplikace jsou často velice drahé, nemusí odpovídat leteckým přepisům České republiky nebo nemají vyhovující funkcionalitu.

Z tohoto důvodu jsem se rozhodl vytvořit iOS aplikaci na evidenci letů. Tato aplikace bude pomáhat pilotům zaznamenávat elektronicky své lety, bude také kontrolovat předpisy a umožňovat export do formátu pro tisk.

Začínám analýzou podobných aplikací, a to pro zařízení iOS i Android. Poté navrhuji vhodnou funkcionalitu a vytvářím návrh uživatelského rozhraní.

Dalším tématem, které ve své práci řeším, jsou softwarové architektury při vývoji iOS aplikace. Zde analyzuji alternativy k architektuře MVC ve spojení s funkcionálně reaktivním programováním neboli FRP.

Tuto analýzu následně aplikuji v praxi, kdy se zvolenou architekturou a FRP implementuji společně s jednotkovými testy dříve zmíněnou aplikaci. Nakonec aplikaci podrobím uživatelským testům a podle jejich výsledků upravím aplikaci do finální podoby.

V poslední části své práce se snažím zhodnotit postupy FRP společně se mnou zvolenou moderní architekturou a jejich časovou a implementační náročnost oproti klasickému MVC.

Kapitola 1

Cíl práce

Cílem této práce je navrhnout a implementovat aplikaci k evidenci letů pro platformu iOS, a to pomocí postupů FRP (funkcionálně reaktivního programování) a s využitím moderní softwarové architektury jako např. MVVM nebo VIPER. Tato aplikace bude sloužit pilotům České republiky k elektronické evidenci letů. Tento cíl je rozdělen do několika podúkolů.

V první části analyzuji podobné aplikace pro evidenci letů, a to jak pro platformu iOS, tak i pro Android. Na základě této analýzy navrhnu vhodnou funkcionalitu pro vytvářenou aplikaci. Podle navržených funkcionalit si zvolím architekturu a navrhnu uživatelské rozhraní v podobě wireframů.

V dalším kroku aplikaci implementuji pomocí postupů FRP a se zvolenou architekturou. V průběhu realizace aplikace budou vytvářeny také testy a dokumentace aplikace.

Dále bude aplikace podrobena uživatelským testům, podle kterých bude vhodně upravena.

V poslední části budu popisovat výhody a nevýhody, které přinesly postupy FRP. Budu také hodnotit časovou a implementační náročnost oproti standardním postupům a architektuře MVC.

Kapitola 2

Tvorba iOS aplikací

2.1 Možnosti vývoje

Vývoj iOS aplikace je možný hned několika způsoby, každý má své výhody a nevýhody, právě ty bych rád v této kapitole rozebral. Mezi možné způsoby vývoje bych rád zmínil nativní aplikace, hybridní aplikace a mobilní webové aplikace.

2.1.1 Nativní aplikace

Nativní aplikace jsou vyvíjeny specificky pro jednu platformu. Díky tomu mají přístup ke všem funkcím daného zařízení jako např. GPS, kamera nebo kontakty. Mohou fungovat i pouze offline, tedy bez nutnosti internetového připojení. [1]

Však pokud bychom chtěli aplikaci distribuovat na více platforem, tak s tímto přístupem by bylo nutné vytvořit pro každou platformu vlastní aplikaci. To by prodloužilo vývoj a znesnadnilo následnou údržbu aplikací.

Co se týče iOS vývoje, je možné si zvolit z dvou programovacích jazyků – Objective-C nebo Swift. [2] [3]

2.1.2 Hybridní aplikace

Hybridní aplikace jsou aplikace tvořené nejčastěji pomocí HTML5 a JavaScriptu, následně jsou spuštěné v nativním kontejneru. [4] Jako příklad je možné uvést např. Apache Cordova. Tento kontejner umožňuje přístup k funkcím daného přístroje, podporuje použití aplikace offline a dává možnost publikace vytvořené aplikace do obchodu tzv. app store. [5]

Však výhodou nativních aplikacím proti hybridním je to, že jsou vytvářeny přesně pro danou platformu, a tudíž jejich vzhled a výkon bude vždy lepší. [6]

2.1.3 Mobilní webové aplikace

Poslední možností jsou mobilní webové aplikace. Tyto aplikace jsou pouze upravené webové stránky do podoby a chování nativních aplikací. Přestože běží pouze v prohlížeči, mohou mít i tyto aplikace přístup k určitým (ne však ke všem) nativním funkcím. [1]

Tím, že jsou mobilní webové aplikace spouštěny v prohlížeči a nejsou stahovány přes obchody, je ulehčena údržba a vývoj, protože si uživatel nemusí vždy stahovat novou verzi aplikace.

Však tento postup má i své nevýhody. Jak již bylo zmíněno dříve, aplikace nemá přístup ke všem nativním funkcím daného přístroje. S dalším problémem se můžeme setkat u offline ukládání dat, a to se zabezpečením, které nemusí být tak dokonalé nebo uživatelsky přívětivé, jako u nativních aplikací. [4]

2.1.4 Zvolené řešení

Pro svou práci jsem si zvolil možnost nativní mobilní aplikace z důvodu zaměření pouze na platformu iOS. Bude se tedy jednat pouze o jednu aplikaci, která bude moci využít všech nativních funkcionalit, výkonu i vzhledu.

2.2 Architektury při tvorbě iOS aplikací

Při tvorbě iOS aplikace je možné si vybrat z několika architektur. V této kapitole budu rozebírat pouze MVC, MVVM a VIPER.

2.2.1 MVC

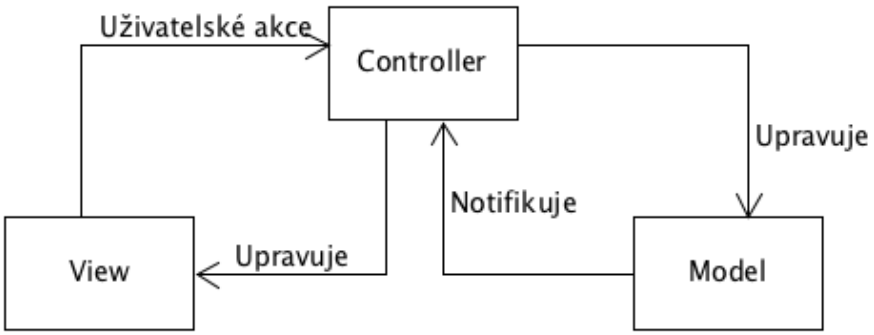
Architektura MVC je zkratka pro „Model-View-Controller“ neboli tři komponenty, ze kterých se architektura skládá. Jedná se o softwarovou architekturu, které se velice často používá při tvorbě aplikací s uživatelským rozhraním. [7]

- *Model* definuje jaká data aplikace obsahuje, a pokud dojde k jakékoliv změně, tak informuje buď *Controller* nebo *View* (tzv. své observery). [8]
- *View* vrstva je prezentována samotnému uživateli. Tedy jsou zde zobrazena aplikační data a je zachycována uživatelova práce s aplikací.[7]
- *Controller* je vrstva mezi *View* a *Model* zajišťující logiku aplikace. Stará se o prezentaci změn do *View* pokud se změní *Model*. Zároveň provádí úpravy v *Modelu* při uživatelově manipulaci s *View*. [8]

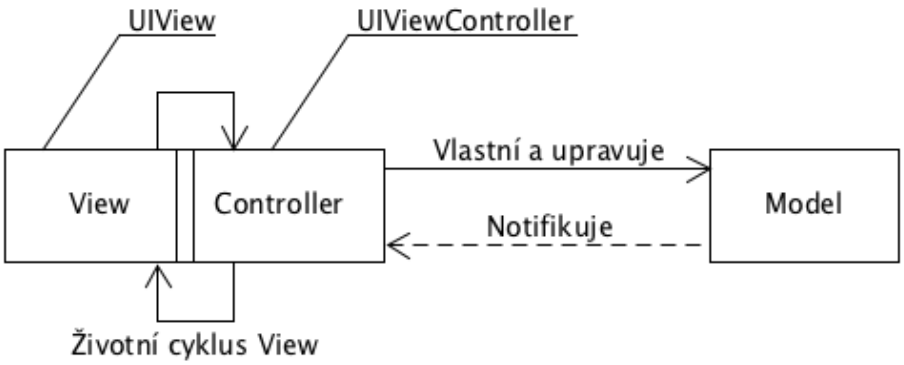
Však co se týče iOS vývoje, vrstvy *View* a *Controller* jsou téměř spojeny, protože *Controller* je příliš úzce zapojený do životního cyklu *View*, což následně způsobuje velký nárůst *Controlleru*.[9]

Základní myšlenku MVC a MVC při vývoji iOS aplikace ukazují obrázky 2.1 (převzato a přeloženo z originálu [10]) a 2.2 (převzato a přeloženo z originálu [11]).

MVC je základní architekturou pro tvorbu iOS aplikací. Není však jedinou možností.



Obrázek 2.1: Model-View-Controller diagram



Obrázek 2.2: Model-View-Controller při vývoji iOS aplikace

2.2.2 MVVM

Architektura MVVM má obdobné koncepce jako MVC. Jedná se také o zkratku, tentokrát „Model-View-ViewModel“. [12]

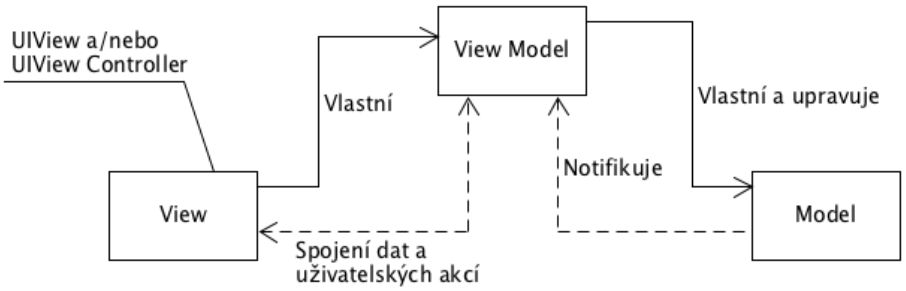
- *Model* je totožný s *Model* vrstvou architektury MVC, jedná se tedy o datovou část aplikace.
- *View* prezentuje aplikační data uživateli a monitoruje jeho akce. Však, jak již bylo zmíněno dříve, u iOS aplikací se jedná spíše o vrstvu *View/Controller*. Tato vrstva obsahuje pouze minimum logiky aplikace a reaguje hlavně na *ViewModel*. [13]
- *ViewModel* spojuje *View* a *Model* a zajišťuje hlavní logiku aplikace. *ViewModel* tedy komunikuje s *Modelem* a jeho metodami a následně připravuje data pro *View*. Obsahuje také implementaci funkcí, které reagují a zpracovávají akce uživatele např. kliknutí na tlačítko. [12]

Tedy pro shrnutí rozdílů MVC a MVVM u iOS bych zmínil to, že iOS MVC má ve výsledku téměř jen dvě vrstvy *View/Controller* a *Model*. Když potom uvažujeme architekturu MVVM, *View/Controller* je opravdu pouze jednou vrstvou a mezi ní a *Modelem* je vložena nová vrstva *ViewModel*, která je spojuje, a do které je přesunuta i většina aplikační logiky.

Mezi výhody architektury MVVM oproti MVC patří např.:

- poskytnutí návrhového principu tzv. separation of concerns, neboli oddělení zájmů;
- zlepšení možnosti testovatelnosti aplikace.

Architekturu MVVM zobrazuje obrázek 2.3 (převzato a přeloženo z originálu [14]).



Obrázek 2.3: Model-View-ViewModel architektura

2.2.3 VIPER

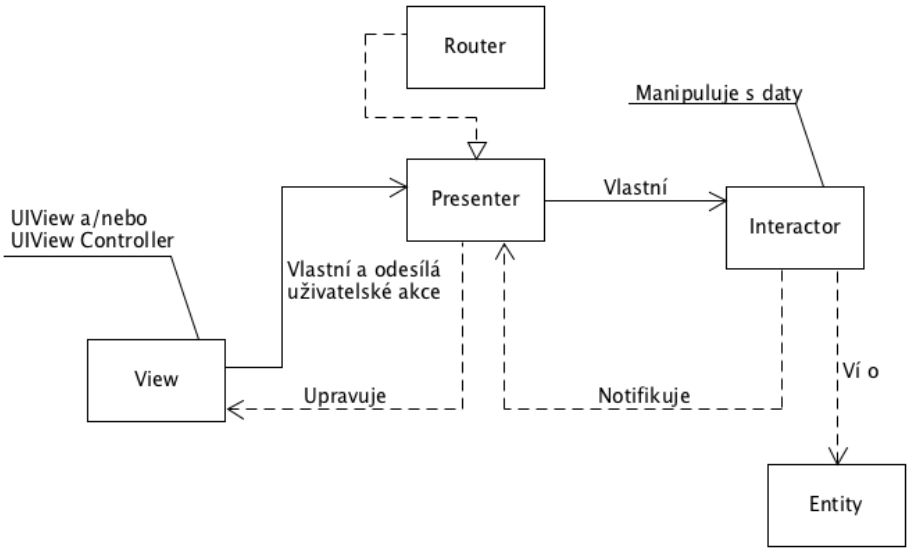
VIPER je poslední analyzovanou možností, co se týče architektur. I zde je název složen z prvních písmen jednotlivých vrstev architektury, tedy „View, Interactor, Presenter, Entity, Router“.

- *View* zobrazuje data uživateli a předává uživatelovi vstupy vrstvě *Presenter*.
- *Interactor* obsahuje logiku aplikace spojenou s daty (*Entity*).
- *Presenter* vrstva má na starosti *View* logiku. Reaguje tedy na uživatelovy akce a komunikuje s vrstvou *Interactor*, od ní také přijímá nová data. [9]
- *Entity* jsou datové objekty aplikace přístupné pouze části *Interactor*.
- *Router* obsahuje navigační logiku. [15]

Mezi výhody architektury VIPER znovu patří např.:

- dobré rozdělení odpovědností;
- zlepšení možnosti testovatelnosti aplikace. [9]

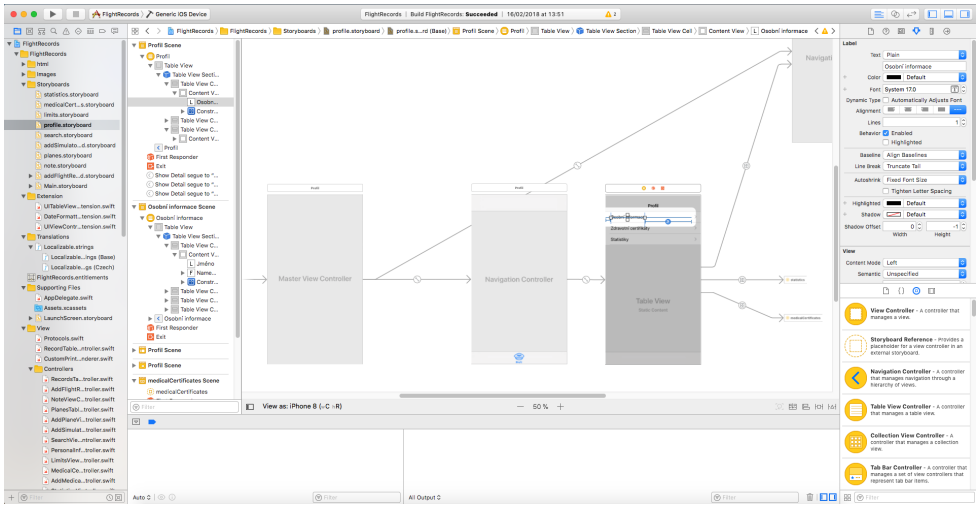
Tato architektura však může být zbytečně složitá pro menší aplikace. [9]
Architekturu VIPER zobrazuje obrázek 2.4 (převzato a přeloženo z originálu [16]).



Obrázek 2.4: VIPER architektura

2.3 Tvorba uživatelského rozhraní

Při vývoji iOS aplikace existují tři možnosti, jak vytvořit uživatelské rozhraní – *storyboard*, *Xib* a kód. Každá z těchto možností má své výhody a nevýhody. [17]



Obrázek 2.5: Storyboard (pouze ilustrační)

2.3.1 Storyboard

Storyboard představuje grafickou reprezentaci uživatelského rozhraní – zobrazuje jednotlivé obrazovky, jejich obsah a propojení těchto obrazovek. Xcode mají zabudovaný editor pro tvorbu a úpravu *storyboardu* (viz. obrázek 2.5). Jednotlivé prvky jsou vkládány jednoduchým přetažením a je u nich možné upravit obsah, velikost, barvu a podobně, ihned v editoru. Tento editor také umožňuje propojení jednotlivých prvků a jejich akcí s kódem aplikace. [18]

Mezi výhody *storyboar*ů patří rychlost tvorby uživatelské rozhraní, což zahrnuje i rychlou tvorbu prototypů aplikace. Dále vizuální zobrazení, nejen prvků, ale také propojení jednotlivých obrazovek, a to bez nutnosti kompilování aplikace.

Nevýhodou *storyboard*ů může být výkon při více nebo složitějších obrazovkách. Tento problém se však dá zredukovat rozdělením obrazovek do více *storyboard* souborů. Další nevýhodou je problematické znovupoužití jednotlivých prvků (obrazovek) nebo komplikovaná práce v týmu způsobená složitým spojením změn při verzování (i tento problém řeší z části více souborů se *storyboardy*). Posledním nedostatkem je to, že ne vše se dá vytvořit ve *storyboardu* (hlavně dynamické prvky) a z toho důvodu dochází k tomu, že část uživatelského rozhraní je v kódu a část ve *storyboardu*. [17] [19]

2.3.2 Xib

Xib je starším přístupem pro tvorbu uživatelského rozhraní iOS aplikací než *storyboardy*, starší však neznamená zastaralý. Vytváří se jednoduché *view* elementy stejným způsobem jako o *storyboard*ů. Právě tato skutečnost přináší stejné výhody i problémy, s tím rozdílem, že se jedná o drobnější prvky, což

ulehčuje např. verzování, znovupoužitelnost nebo nezpůsobuje zpomalovaní vývojového prostředí Xcode. [17] [19]

2.3.3 Uživatelské rozhraní v kódu

Poslední možností jak tvořit uživatelské rozhraní je vlastní kód. Tento způsob přináší mnoho výhod (hlavně tím, že redukuje nedostatky *storyboard*ů a *Xib*), ale má i své nevýhody.

Mezi výhody vlastního kód patří: znovupoužitelnost, bezproblémová práce ve více lidech (verzování) a to, že se vše nachází na jedno místě.

Nevýhodou vytváření uživatelské rozhraní v kódu je rychlost tvorby bez možnosti svižného vytvoření prototypu aplikace. Dále také tento přístup postrádá vizuálnost a vyžaduje kompilaci. [17] [19]

2.3.4 Zvolené řešení

Pro tvorbu aplikace vytvářené v rámci této diplomové práce, jsem si zvolil možnost tvorby uživatelského rozhraní pomocí *storyboard*ů. Tato volba vyplývá z výše zmíněných výhod *storyboard*ů a z toho důvodu, že aplikaci budu vytvářet sám.

2.4 Funkcionálně reaktivní programování

Funkcionálně reaktivní programování je kombinací funkcionálního a reaktivního programování, díky němuž dokáže aplikace dynamicky měnit stav a chování v závislosti na událostech přicházejících za daný čas. [20]

Pro vysvětlení, co je to reaktivní programování, cituji André Staltze: „reaktivní programování je programování s asynchronními datovými toky“. [21]

Na spojení funkcionální a reaktivního programování může dívat i jako na návrhový vzor *observer*. [22] Pozorujeme tedy např. určité vstupní pole, tlačítko nebo i dotaz na server a jsme informování o každé změně v podobě asynchronního datového toku. Na tyto datové toky je možné aplikovat funkcionální programování. Je tedy možné toky:

- spojovat (*merge*),
- filtrovat (*filter*) např. pouze události, které nás zajímají,
- mapovat (*map*) jeden tok na nový, a další. [21]

2.4.1 FRP frameworky pro iOS

V této kapitole jsou pouze rozebrány základy jednotlivých frameworků, podrobnější vysvětlení (zvoleného frameworku) společně s ukázkami je k nalezení v kapitole Realizace.

2.4.1.1 ReactiveSwift

ReactiveSwift je prvním frameworkem pro iOS podporující FRP. Obsahuje řadu základních prvků (*Signal*, *SignalProducer*, *Property*, *Action*...) a operátorů podporujících myšlenku „streams of values over time“. [23]

2.4.1.2 ReactiveCocoa

ReactiveCocoa je další z FRP frameworků pro iOS. ReactiveCocoa rozšiřuje různé aspekty Apple Cocoa frameworku základními prvky frameworku ReactiveSwift. Umožňuje vazbu na prvky uživatelského rozhraní, u interaktivních prvků napojuje *Signal* a *Action* pro kontrolu událostí a změn. Dále také umožňuje vytvářet signály na volání metod (např. i pro UIKit třídy). [24]

2.4.1.3 RxSwift

RxSwift je Swift verzí knihovny Reactive Extensions (Rx). [25] Tato knihovna umožňuje vytvářet aplikace založené na událostech a asynchronních datových tocích pomocí tzv. Observables. [26] I přesto, že RxSwift není striktně FRP frameworkem, [27] je zde uváděn, a to z důvodu velkého využití knihovny Reactive Extensions i na jiných platformách např.: JavaScript, C#, Python. [28]

2.5 Perzistence dat

Perzistence dat, neboli jejich uchování a uložení, je velice důležitou funkcionalitou většiny aplikací. V této kapitole jsou rozebírány tři možnosti zajišťující perzistenci dat iOS aplikací – Core Data, iCloud a Realm.

2.5.1 Core Data

Core Data je Apple framework, který má na starosti model vrstvu aplikace. Stará se o životní cyklus objektů, jejich vztahy (objektový graf) i perzistenci. [29]

Core Data má více možností jakým způsobem data uložit např. SQLite a XML. Jedná se tedy o uložení dat na disku daného zařízení. [30]

Výhodou tohoto frameworku je to, že je zcela zdarma a má podporu přímo v Xcode. [31]

2.5.2 iCloud

iCloud je cloudové úložiště od společnosti Apple. Umožňuje ukládat aplikační data i dokumenty a přistupovat k nim na všech Apple zařízeních a na webu. [32]

Při vývoji iOS aplikací se pro využití iCloudu používá framework CloudKit. CloudKit zajišťuje rozhraní pro komunikaci dané aplikace a iCloudu. [33] Poskytuje ověření uživatele, tři druhy databáze – soukromou, veřejnou a sdílenou. Dále také analytický nástroj CloudKit Dashboard, který umožňuje analýzu dat, měření aktivity uživatelů a další. [34]

CloudKit je dostupný pro členy Apple Developer programu. [35] Tento program stojí ročně v přepočtu 2150 Kč. [36]

2.5.3 Realm

Realm, s oficiální stránkou `https://realm.io`, je multiplatformní mobilní databáze, která je připravená pro jazyky Java (Android), Swift, Objective-C, JavaScript a Xamarin. Hlavní myšlenkou je kontejner objektů tzv. Realm. V těchto kontejnerech jsou uložena data, na které je možné se dotazovat, tyto data filtrovat a podobně. Na rozdíl od klasických např. SQL databází, zde pracujeme přímo s „živými“ objekty, tedy pokud máme instanci Realm objektu a cokoliv v aplikaci tuto instanci změní, tato změna je ihned viditelná.

Realm je rozdělený na dvě části – mobilní databáze Realm (Realm Mobile Database) a objektový server Realm (Realm Object Server). Jak je již z názvů možné usuzovat mobilní databáze je pouze na mobilním zařízení, jedná se tedy o offline uložení dat. Pokud však chceme data např. sdílet na více zařízení, je možné se připojit na objektový server Realm a s tím se synchronizovat. Realm se řídí strategii „nejprve offine“ – čtení a zápis probíhá nejprve lokálně a až poté probíhá synchronizace se serverem.

Jedna aplikace může využívat hned několik Realm kontejnerů, a to jak lokální, tak i vzdálené, kde každý z nich může mít různá oprávnění pro různé uživatele.

Realm Mobile Database je open source, tedy zdarma. [37]

Kapitola 3

Analýza evidence letů

3.1 EASA

EASA, neboli European Aviation Safety Agency, je agentura spadající pod Evropskou Unii, která má na starosti technické přepisy, bezpečnost, regulace a certifikace v oboru letectví. [38]

Pro tuto diplomovou práci je EASA důležitá, protože vydává i pokyny např. pro evidenci letů nebo limity odlétaných hodin. [39]

3.1.1 Pokyny pro evidenci letů

Pokyny pro evidenci letů udává předpis FCL.050. Tento předpis specifikuje povinné položky každého leteckého záznamu. [40]

„ Každý záznam letů by měl obsahovat minimálně tyto informace:

1. osobní informace: jméno a adresu pilota;
2. každý záznam letu by měl obsahovat:
   - jméno velícího pilota (PIC – Pilot-in-command),
   - datum letu,
   - čas a místo odletu a příletu,
   - typ, značku, model, variantu a registraci letadla,
   - označení zda je letadlo jednomotorové (SE – single engine) nebo vícemotorové (ME – multi engine),
   - čas letu,
   - celkový čas letu;
3. každý záznam z výcvikového zařízení pro simulaci letu (FSTD – flight simulation training devices) by měl obsahovat:

- typ a kvalifikační číslo výcvikového zařízení,
- instrukce výcvikového zařízení pro simulaci letu,
- datum,
- čas,
- celkový čas;

4. funkce pilota:
   - velící pilot (včetně sólového, studenta (Student PIC) nebo velícího pilota pod dohledem (PICUS – pilot-in-command under supervision)),
   - druhý pilot,
   - dvojí pilot (dual),
   - instruktor (FI – Flight Instructor) nebo zkoušející (FE – Flight Examiner);
5. provozní podmínky – pokud se let uskutečnil v noci nebo pokud byl prováděn podle pravidel pro let podle přístrojů.

“ [40] (překlad vlastní)

3.1.2 Limity

Limity letového času a času ve službě obsahuje předpis ORO.FTL.210.

„Celková doba služby, na kterou může být člen posádky přidělen, nesmí překročit:

1. 60 hodin služby za 7 po sobě jdoucích dnů;
2. 110 hodin služby za 14 po sobě jdoucích dnů; a
3. 190 hodin služby za 28 po sobě jdoucích dnů, rozdělených co nejrovnoměrněji během tohoto období.

Celkový čas, na který je jedinec přidělen jako člen provozní posádky, nesmí překročit:

1. 100 hodin letu za 28 po sobě jdoucích dnů;
2. 900 hodin letu v kalendářním roce; a
3. 1000 hodin letu během 12 po sobě jdoucích kalendářních měsících.

Poletová služba se počítá do doby služby. “ [41] (překlad vlastní)

3.1.3 Zdravotní certifikáty

Informace o zdravotních certifikátech obsahuje předpis Part-MED. Certifikáty jsou tří druhů – zdravotní certifikát třídy 1 (Class 1 medical certificate), zdravotní certifikát třídy 2 (Class 2 medical certificate) a zdravotní certifikát pro licence na lehká letadla (LAPL – Light Aircraft Pilot Licence). Každý z těchto certifikátů má jinak nastavenou dobu platnosti a je pro jiné typy pilotních licencí.

LAPL certifikát je pouze pro pilotní licence na lehká letadla. Platnost je 60 měsíců u pilotů do věku 40 let, poté je platnost pouze 24 měsíců.

Zdravotní certifikát třídy 2 je pro pilotní licence PPL (Private Pilot Licence), SPL (Sailplane Pilot Licence) a BPL (Balloon Pilot Licence), tedy pro piloty soukromých letadel, kluzáků a balónů. Platnost licence se znovu odvíjí od věku pilota – 60 měsíců u pilotů do věku 40 let, následně 24 měsíců do věku 50 let a nakonec platnost licence klesá na 12 měsíců.

Zdravotní certifikát třídy 1 je certifikát nejvyšší úrovně. Je pro pilotní licence CPL (Commercial Pilot Licence), MPL (Multi-crew Pilot Licence) a ATPL (Airline Transport Pilot Licence), tedy pro piloty komerčních, vícečlenných a dopravních letadel. Platnost licence je 12 měsíců. To neplatí u pilotů starších 40 let, létajících jednopilotní komerční lety s cestujícími nebo u pilotů starších 60 let, zde se platnost licence snižuje na 6 měsíců. [42]

3.2 Analýza existujících aplikací pro evidenci letů

Tato kapitola se zabývá analýzou již existujících aplikací pro evidenci letů. Pro analýzu bylo vybráno pět aplikací – tři pro iOS a dvě pro platformu Android.

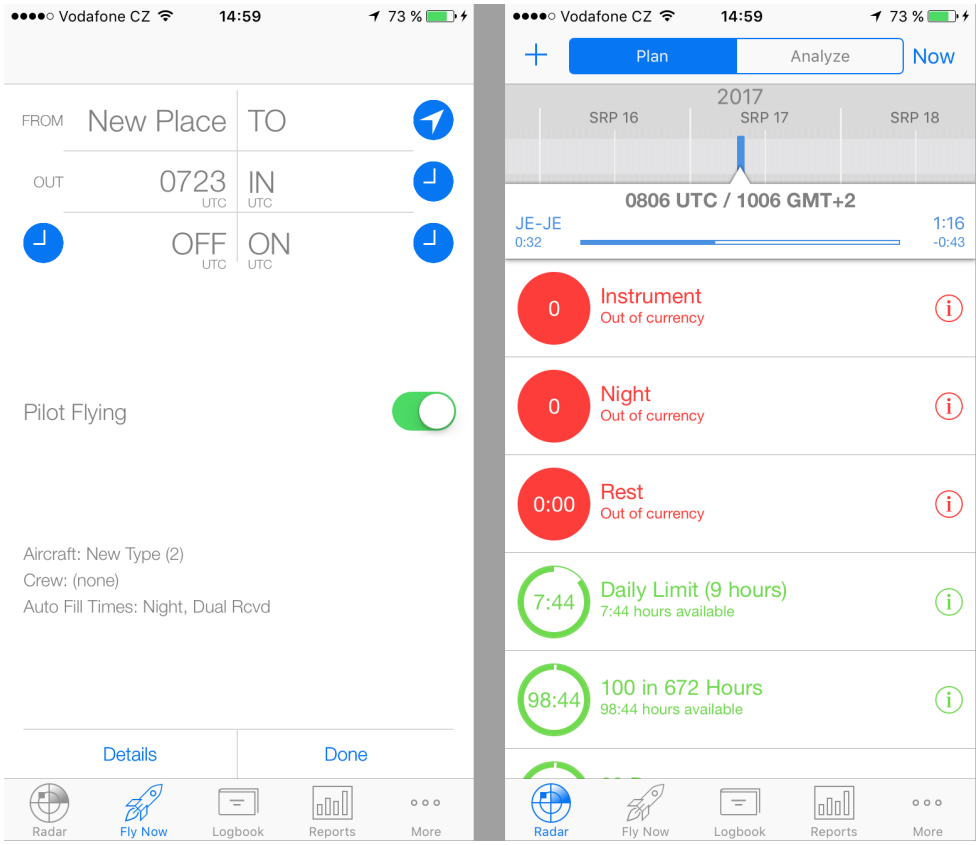
1. LogTen Pro X – iOS aplikace v angličtině vyvíjená společností Coradine Aviation. [43]

2. Logbook Pro Aviation Flight Log for Pilots – druhá iOS aplikace, také v angličtině, vytvořená NC Software, Inc. [44]

3. Safelog Pilot Logbook – poslední z analyzovaných iOS aplikací. I tato aplikace je v anglickém jazyce. Publikována Dauntless Software. [45]

4. FlyLogio - Pilot Logbook – česká aplikace vyvinutá pro platformu Android společností FlyLogio.com. [46]

5. Smart Logbook – anglická Andriod aplikace vydána firmou Kviation, Inc. [47]



Obrázek 3.1: LogTen Pro X

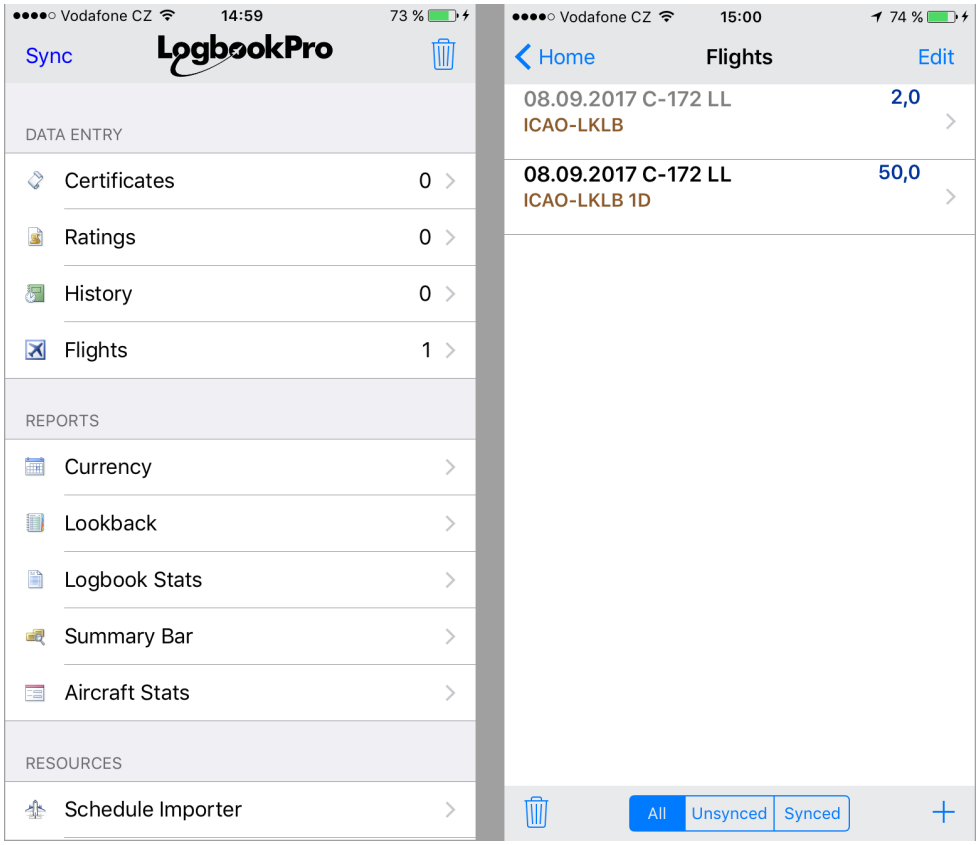
3.2.1 Kritéria hodnocení

Kritéria hodnocení byla rozdělena do několika kategorií – přihlášení, platby, napojení externích databází, doplňkové položky při vkládání záznamu, limity a certifikáty, reporty a zálohování.

Při vkládání záznamu jsou brány v potaz pouze doplňkové položky, protože aplikace bude tvořena podle předpisu EASA FCL.050, který udává povinné údaje při evidování letu. Také položka v hodnocení – reporty podle EASA, je analyzována z pohledu předpisu FCL.050.

Všechny položky hodnocení jsou uváděny z pohledu mobilní/tablet aplikace.

3.2.2 Srovnávací tabulka



Obrázek 3.2: Logbook Pro Aviation Flight Log for Pilots

Tabulka 3.1: Srovnávací tabulka

Kritéria	Log Ten Pro X	Logbook Pro	Safelog	FlyLogio	Smart Logbook	Výsledek
Přihlášení						
Aplikace funkční bez přihlášení	✓	✗	✗	✗	✗	1/5
Možnost přihlášení	✓	✓	✓	✓	✓	5/5
Platby						
Platby jednorázové	✗	✗	✗	✗	✓	1/5
Opakované platby	✓	✓	✓	✓	✓	5/5
Napojení externích databází						

Tabulka 3.1: Srovnávací tabulka

Kritéria	Log Ten Pro X	Logbook Pro	Safelog	FlyLogio	Smart Logbook	Výsledek
Napojení na databázi letišť	✗	✓[1]	✓	✓	✓	4/5
Napojení na databázi letadel	✗	✗	✓	✓	✗	2/5
Doplňkové položky při vkládání záznamu						
Možnost přidání fotky	✓	✗	✓	✗	✗	2/5
Možnost přidání dokumentu	✓	✗	✓	✗	✗	2/5
Limity a certifikáty						
Kontrola limitů	✓	✓	✓[2]	✗	✓	4/5
Certifikáty	✓	✓	✓[3]	✗	✓	4/5
Reporty						
Generování reportů	✓	✗	✓[4]	✗	✓	3/5
Reporty podle EASA	✗	✗	✓	✗	✓	2/5
Jiné reporty	✓	✓	✓	✗	✓	2/5
Perzistence dat						
iCloud/Google	✓	✗	✗	✓	✓	3/5
Vlastní řešení	✗	✓	✓	✗	✗	2/5
Synchronizace více zařízení	✓	✓	✓	✓	✓	5/5

3.2.3 Výsledky a vlastní zhodnocení

3.2.3.1 LogTen Pro X

LogTen Pro X je z analyzovaných iOS aplikací nejvíce uživatelsky přívětivá. Zobrazuje přehledně limity a certifikáty, i vkládání je intuitivní. Má však i několik nedostatků:

- není napojená na databázi letišť, tudíž uživatel musí vyplnit všechny informace o daném letišti sám, bez automatického doplnění nebo našeptávání;
- neumožňuje generování reportů podle předpisu EASA FCL.050;
- aplikace je placená ročně –

[1] Logbook Pro umí nalézt pouze nejbližší letiště.
[2] Safelog zobrazuje limity ve webové verzi.
[3] Safelog zobrazuje certifikáty ve webové verzi.
[4] Safelog zobrazuje reporty ve webové verzi.

Obrázek 3.3: Safelog Pilot Logbook

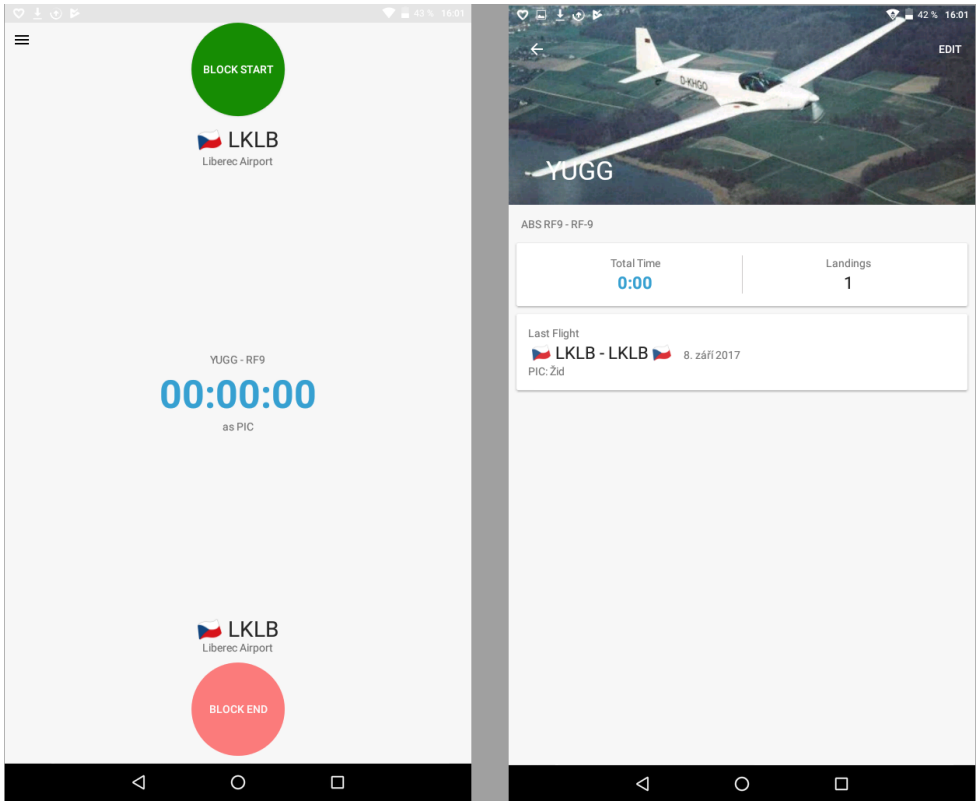
- iPhone + iPad + Mac – 3550 Kč,
- Mac – 3550 Kč,
- iPhone + iPad – 2150 Kč.

Data o aplikaci a cenách jsou získány přímo z aplikace LogTen Pro X.

3.2.3.2 Logbook Pro Aviation Flight Log for Pilots

Aplikace Logbook Pro vyžaduje pro přihlášení stažení PC aplikace (pouze pro Windows). S touto aplikací je následně synchronizován. Některá funkcionalita, např. generování reportů, je dostupná pouze v PC verzi.

PC verze aplikace je zadarmo pouze ve zkušební verzi, poté základní verze stojí v přepočtu 1800 Kč. iOS verze aplikace se platí ročně v přepočtu za 1045 Kč, je nutné si zaplatit i zálohování a další funkcionality. [48]



Obrázek 3.4: FlyLogio

3.2.3.3 Safelog Pilot Logbook

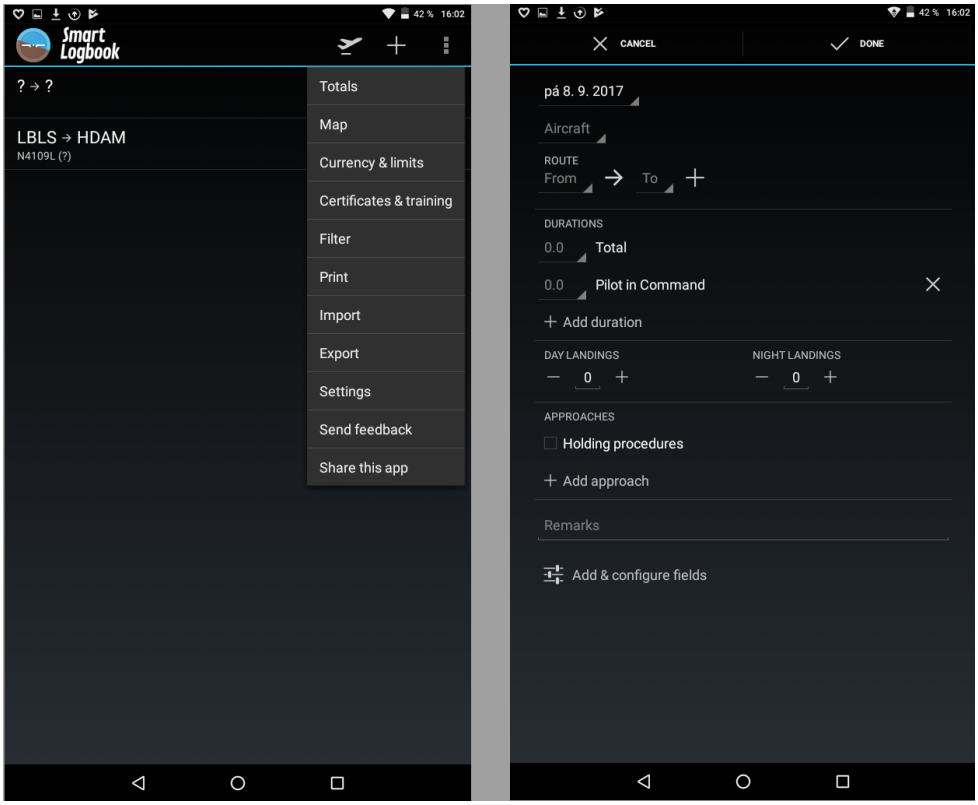
Safelog Pilot Logbook obsahuje pouze některé funkce přímo v aplikaci, u ostatních je uživatel odkázán do webového rozhraní (SafelogWeb Cloud) viz. 3.3. Toto webové rozhraní zobrazené v aplikaci však není přizpůsobené pro mobilní zařízení, často je zobrazena pouze část stránky a není možné např. vyplnit všechna pole formuláře.

Tato aplikace však, pokud budeme brát v potaz i funkce ve webovém rozhraní, obsahuje nejširší spektrum funkcionalit.

Samotná aplikace je zadarmo, ale pro plnou verzi aplikace je nutné předplatné. To se pohybuje od 1320 Kč za jeden rok až po 8990 Kč za deset let. Data o aplikaci a cenách jsou získány z aplikace Safelog Pilot Logbook.

3.2.3.4 FlyLogio - Pilot Logbook

Android aplikace FlyLogio - Pilot Logbook je jedinou aplikací kompletně zadarmo. Z uživatelského pohledu se jedná o přehlednou a jednoduchou aplikaci. Však neobsahuje takovou funkcionalitu jako ostatní placené aplikace, např. chybí generování jakýchkoliv reportů nebo kontrolování limitů.



Obrázek 3.5: Smart Logbook

3.2.3.5 Smart Logbook

Smart Logbook je poslední analyzovanou aplikací, jedná se o Android aplikaci. Obsahuje mnoho funkcionalit – generování reportů podle mnoha norem, zobrazení mapy se zaznamenáním jednotlivých letů, hlídání limitů i expirace certifikátů.

Tato aplikace je zadarmo pouze ve zkušební verzi, následně je nutné aplikaci zakoupit za 300 Kč. Je nutné také platit za zálohování a synchronizaci dat, a to buď 20 Kč za měsíc, nebo 120 Kč za rok. Informace o cenách jsou získány z aplikace Smart Logbook.

3.2.3.6 Závěr analýzy

Tato analýza posloužila při návrhu funkcionalit iOS aplikace ve formě případů užití, při návrhu uživatelského rozhraní a také při výběru vhodného nástroje na perzistenci dat.

Kapitola 4

Návrh

4.1 Funkční a nefunkční požadavky

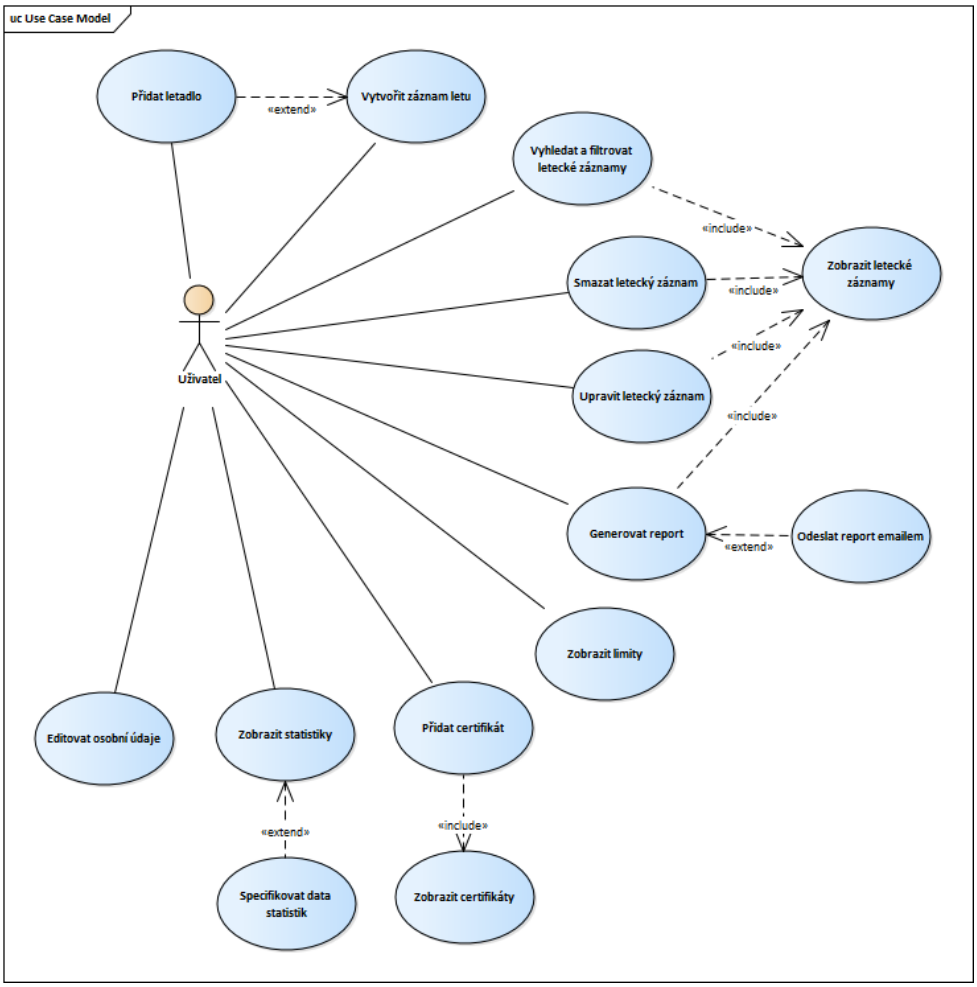
4.1.1 Funkční požadavky

Funkční požadavky jsou uvedeny pouze jmenovitě. Podrobnější popis je uveden v podobě případů užití.

1. Evidování leteckých záznamů.
2. Vyhledávání v leteckých záznamech.
3. Kontrola limitů.
4. Evidence zdravotních certifikátů.
5. Zobrazení statistik.
6. Generování reportů do formátu PDF.

4.1.2 Nefunkční požadavky

1. Funkční pro iOS 11 – aplikace bude dostupná pro verzi operačního systému iOS 11.
2. Aplikace pro iPhone a iPad – uživatelské rozhraní bude přizpůsobeno jak telefonům, tak tabletům.
3. Stejná data na více uživatelových zařízeních – všechna data budou zálohována online a uživatel k nim bude mít přístup na všech zařízeních s iOS 11 pod svým účtem.
4. Dostupnost dat offline – uživatel bude mít přistup ke všem dříve staženým/vytvořeným záznamům. Aplikaci bude možné používat i bez internetového připojení, k synchronizaci dojde až ve chvíli, kdy bude internetové připojení k dispozici.



Obrázek 4.1: Případy užití

5. Aplikace podle EASA – aplikace se bude řídit předpisy EASA a to přesně: FCL.050 u evidence letů, ORO.FTL.210 v případě limitů a Part-MED při evidenci zdravotních certifikátů.

4.2 Navržená funkcionalita v podobě případů užití

Případy užití (use cases) byly využity při návrhu funkcionalit iOS aplikace. Jsou obsaženy přímo v práci z důvodu přehledného zobrazení navržených funkcionalit a možnosti dovysvětlení u některých z nich.

Diagram případů užití zobrazuje obrázek 4.1. Tento diagram společně se projektem Enterprise Architect je k nalezení v příloze této práce.

4.2.1 Vytvořit záznam letu

Vytvoření leteckého záznam umožňuje uživateli vložit nový záznam letu do aplikace.

Hlavní scénář –

- Případ užití začíná, když chce uživatel evidovat svůj let.
- Systém zobrazí formulář umožňující zadat: jméno velícího pilota, datum letu, čas a místo odletu a příletu, letadlo, časy letu, celkový čas letu, počet vzletů a přistání, pilotovu funkci při letu a provozní podmínky.
- Uživatel vyplní formulář.
- Aplikace uloží informace o letu.

Alternativní scénář –

- Případ užití začíná, když chce uživatel evidovat záznam z výcvikového zařízení pro simulaci letu.
- Systém zobrazí formulář umožňující zadat: typ a kvalifikační číslo výcvikového zařízení, datum a čas.
- Případ užití pokračuje 3. krokem hlavního scénáře.

Aplikace nebude napojena na databázi letadel, protože uživatel u každého letadla musí vyplnit minimálně registrační číslo. Pokud by uživatel musel letadlo najít a zeditovat, je výhodnější pokud si sám letadlo přidá. Dalším důvodem k tomu rozhodnutí je, že pilot létá velmi často pouze s jedním letadlem a proto funkcionalitu přidání letadla nebude používat příliš často.

4.2.2 Přidat letadlo

Přidání letadla dává uživateli možnost přidat letadlo, které pak může vkládat do záznamů o letu.

- Případ užití začíná, když chce uživatel přidat nové letadlo.
- Systém zobrazí formulář umožňující zadat: typ, značku model, variantu, registrační číslo letadla a zda je letadlo jednomotorové nebo vícemotorové.
- Uživatel vyplní formulář.
- Aplikace uloží letadlo.

4.2.3 Zobrazit letecké záznamy

Zobrazení leteckých záznamů zobrazuje jednotlivé záznamy v podobě tabulky, kde u každého záznamu jsou vidět základní informace. Mezi tyto informace patří: místo odletu a přílet, datum, čas letu a letadlo.

4.2.4 Vyhledat a filtrovat letecké záznamy

Tato funkcionalita umožňuje uživateli vyhledávání a filtrování leteckých záznamů.

- Případ užití začíná, pokud chce uživatel vyhledat nebo vyfiltrovat letecké záznamy.
- Include(Zobrazit letecké záznamy).
- Aplikace zobrazí formulář, který umožňuje: zadat hledaný text, nastavit zda se jedná o záznam letu nebo o záznam z výcvikového zařízení, zvolit typ letadla nebo přesné letadlo a nastavit období.
- Uživatel vyplní pole, podle kterých chce vyhledávat/filtrovat.
- Systém zobrazí pouze záznamy odpovídající zvoleným parametrům.

4.2.5 Smazat letecký záznam

Smazání leteckého záznamu umožňuje uživateli smazat letecký záznam, který předtím sám vytvořil.

- Případ užití začíná, když chce uživatel smazat jeden ze svých leteckých záznamů.
- Include(Zobrazit letecké záznamy).
- Uživatel si zvolí záznam, který chce smazat.
- Aplikace zobrazí potvrzovací dialog.
- Uživatel potvrdí smazání.
- Aplikace odstraní položku ze seznamu.

4.2.6 Upravit letecký záznam

Upravení leteckého záznamu umožňuje uživateli upravit všechny položky zvoleného letecké záznamu.

- Případ užití začíná, když chce uživatel upravit letecký záznam.

- Include(Zobrazit letecké záznamy).
- Uživatel si zvolí záznam, který chce upravit.
- Scénář pokračuje krokem 2 Vytvořit záznam letu.

4.2.7 Zobrazit limity

Tato funkcionalita slouží ke zobrazení limitů a kontrole zda jsou všechny limity v normě.

4.2.8 Zobrazit certifikáty

Tato funkcionalita umožňuje uživateli zobrazit všechny jeho certifikáty, společně s kontrolou platnosti a počtem dní do jejich expirace.

4.2.9 Přidat certifikát

Tato funkce umožňuje uživateli přidat certifikát a to buď dle šablony pro zdravotní certifikáty (LALP, třídy 1 a třídy 2), nebo vlastní.

- Případ užití začíná, když chce uživatel vytvořit nový certifikát.
- Include(Zobrazit certifikáty).
- Aplikace zobrazí formulář s možností vytvoření vlastního certifikátu nebo dle šablony. Ve formuláři je následně možné zadat: název certifikátu, datum vydání, datum expirace a popis.
- Uživatel vyplní formulář.
- Aplikace uloží certifikát.

4.2.10 Zobrazit statistiky

Zobrazení statistik zobrazuje nalétané hodiny (celkově, v noci, podle přístrojů a v různých pilotových funkcích).

4.2.11 Specifikovat data statistik

Tato funkcionalita umožňuje specifikovat data, ze kterých jsou zobrazeny statistiky. Je možné specifikovat: text, zda jsou ve statistikách zobrazeny záznamy letů a/nebo záznamy ze simulátoru, typ letadla nebo přesné letadlo a období.

4.2.12 Editovat osobní údaje

Editace osobních údajů umožňuje uživateli upravit své osobní informace. Mezi tyto informace patří: jméno a příjmení, adresa a věk.

U osobních údajů je důležitý hlavně věk, který hraje roli u zdravotních certifikátů.

4.2.13 Generovat report

Generování reportu umožňuje uživateli vygenerovat report ve formátu PDF z nímž zvolených záznamů.

- Případ užití začíná, jestliže se uživatel rozhodne vygenerovat report.
- Include(Zobrazit letecké záznamy).
- Aplikace zobrazí náhled generovaného reportu.

4.2.14 Odeslat report emailem

Funkcionalita odeslání reportu emailem umožňuje uživateli, poté co vygeneroval report, odeslat tento report přes email.

4.3 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní přímo navazuje na návrh funkcionalit. Pro tento návrh byl použit wireframe.

4.3.1 Wireframe

Wireframe je technika, která se používá v brzké fázi vývoje softwaru [49] pro rozvržení základních prvků obrazovek a navržení hlavních průchodů aplikací (navigace). Protože se jedná pouze o návrh, nemusí wireframe podporovat dynamicky měnící se obsah nebo např. nemusí zobrazovat chybové hlášky.

Wireframe může posloužit k první heuristické analýze a k uživatelským testům ještě před vývojem dané aplikace. [50]

Wireframe byl pro tuto práci vytvořen ve webovém nástroji NinjaMock (`https://ninjamock.com`). Z tohoto nástroje byl wireframe následně exportován do formátu pdf a ninjamock package. Tyto exporty jsou k nalezení v příloze této diplomové práce.

Exportovaný wireframe ve formátu pdf je uložený vždy ve dvou provedeních: se zobrazeným mobilním zařízením a bez něj, je tomu tak, protože ve verzi se zobrazeným mobilním zařízením u obrazovek s více položkami (např. přidání záznamu letu) nejsou všechny položky vidět. U provedení bez zobrazeného mobilního zařízení jsou vždy všechny položky vidět, ale wireframe působí neúplným dojmem. Pro úplnost je přidána verze ninjamock package,

kterou je možné importovat do webového nástroje NinjaMock. V této verzi jsou funkční i odkazy mezi jednotlivými obrazovkami.

4.3.2 Heuristická analýza

Heuristická analýze je metoda, která se používá pro hledání problémů použitelnosti (usability problems) v uživatelském rozhraní. [51] Jakob Nielsen vytvořil deset základních principů pro uživatelské rozhraní –

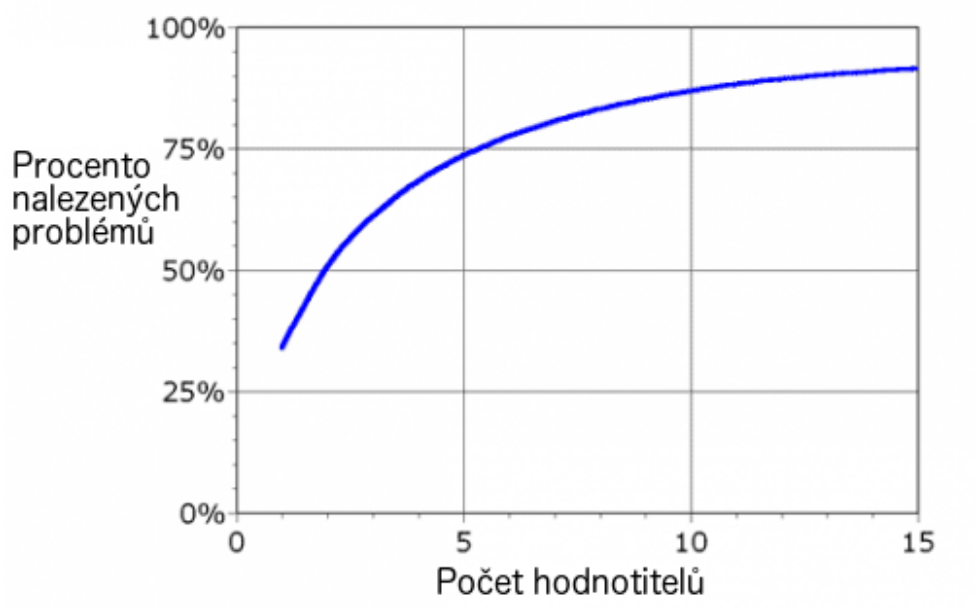
1. „viditelnost stavu systému,
2. spojení systému a reálného světa,
3. uživatelská kontrola a volnost,
4. konzistence a standardizace,
5. předcházení chyb,
6. rozpoznání místo vzpomínání,
7. flexibilita a efektivita použití,
8. estetika a minimalismus,
9. pomoci uživatelům rozpoznat, pochopit a vzpamatovat se z chyb,
10. nápověda a dokumentace“ [52] (překlad vlastní).

Tato pravidla byla použita při heuristické analýze vytvořeného wireframu. Však nebyla kontrolována pravidla:

- „předcházení chyb“ – z důvodu, že ve fázi návrhu není možné např. vyplnit formulář a nebo smazat záznam letu;
- „pomoci uživatelům rozpoznat, pochopit a vzpamatovat se z chyb“ – také není možné vyvolat chybu;
- „nápověda a dokumentace“.

Článek Jakoba Nielsena [51] doporučuje více hodnotitelů, kteří provedou nezávisle heuristickou analýzu nad daným návrhem uživatelského rozhraní. Počet nalezených problémů v závislosti na počtu hodnotitelů zobrazuje obrázek 4.2. Z grafu je možné usuzovat vhodný počet hodnotitelů pět až šest. Protože se však nyní jedná pouze o návrh, kde není možné testovat všechna pravidla, byli použiti pouze dva hodnotitelé.

Po provedení heuristické analýzy byla vytvořena tabulka 4.1 zobrazující nalezené problémy společně s jejich řešením a ohodnocením 1–5, kde číslo 1 identifikuje pouze drobný problém a číslo 5 velice závažný problém. Byla také vytvořena nová verze wireframe, která je také k nalezení v příloze této práce.



Obrázek 4.2: Počet nalezených problémů v závislosti na počtu hodnotitelů. [53] (překlad vlastní)

4.3.3 Uživatelské testy

Uživatelské testy byly provedeny po heuristické analýze (na upraveném wireframu) ještě před samotnou implementací. Testy také nebylo možné provést v plné míře a sloužili převážné k odhalení nejvíce problematických částí návrhu uživatelského rozhraní.

Jakob Nielsen [54] doporučuje pět uživatelů na uživatelské testování. Však ze stejných důvodů jako u heuristické analýzy, byly zvoleni nyní pouze dva uživatelé.

Rozsáhlejší testování a heuristická analýza bude provedeno až po dokončení prototypu aplikace.

Testovaní uživatelé nebyly děleni v této fázi do skupin z důvodu jejich malého počtu. Podmínky na uživatele však kladeny byly – oba uživatelé museli používat operační systém iOS minimálně po dobu jednoho rok a alespoň jeden z nich se musel pohybovat v leteckém průmyslu.

Pro oba uživatele byly vytvořeny stejné testovací scénáře. Uživateli, který neměl znalosti z letectví, byly nejprve vysvětleny základní pojmy nutné k absolvování scénářů a až poté bylo provedeno testování.

4.3.3.1 Scénáře

1. Právě jste dokončil(a) svůj první let z Brna do Prahy. Z Brna jste odlétal(a) v 7:00 a do Prahy jste přiletěl(a) v 8:45, neměl jste žádné me-

Pravidlo	Závažnost	Provedená úprava
1	3	Při přidávání nového záznamu se bude měnit nadpis (title) podle toho zda je přidáván záznam o letu nebo ze simulátoru.
2	2	Tlačítko „Zavřít“ bude přejmenováno na „Zrušit“. Tato změna se týká obrazovek – přidat záznam, přidat letadlo a přidat certifikát.
2	3	U nastavení hledání bude tlačítko „Zpět“ přejmenováno na „Zrušit“.
2	4	Přejmenování záložky „Nastavení“ na „Profil“ a změnění ikonky.
2	4	Přejmenování „Můj profil“ (v bývalé záložce „Nastavení“) na „Osobní informace“.
2	2	Doplnění „Certifikáty“ na „Zdravotní certifikáty“.
8	2	U zobrazení všech záznamů jsou zredukovány zobrazené informace na datum, čas, místo a letadlo.

Tabulka 4.1: Nalezené problémy použitelnosti

zipřistání. Celou doby jste zastával(a) funkci vedoucího pilota a letěl(a) jste letadlem Boeing. Zaznamenejte svůj let.

2. Zjistil(a) jste, že jste u minulého záznamu chybně zaznamenal(a) čas. Upravte tedy čas příletu do Prahy na 7:45.

3. Chcete si přidat další záznam, tentokrát z výcvikového zařízení. Zadejte dnešní datum, čas na zařízení dvě hodiny a znovu jste zastával(a) funkci vedoucího pilota.

4. Aplikaci používáte již delší dobu a máte tedy i více záznamu. Zobrazte si pouze lety, které byly provedeny za poslední týden.

5. Jeden ze záznamů z minulého kroku smažte.

6. Zobrazte si svůj profil a zkontrolujte, zda jsou všechny položky vyplněny správně. Pokud ne, opravte chybně vyplněné nebo chybějící pole.

7. U letadla Boeing z prvního kroku chcete změnit model. Proveďte tuto úpravu.

8. Za poslední tři týdny jste absolvoval(a) mnoho letů, zkontrolujte, zda jste nepřekročil jeden z limitů.

9. Dnes Vám vydali zdravotní certifikát třídy jedna (Class 1). Vložte ho do aplikace.

10. Vytvořte report ze všech záznamů letů i záznamů z výcvikového zařízení za posledních čtrnáct dní.

4.3.3.2 Výsledky testů a provedené úpravy

Testovaní uživatelé byli při jednotlivých testech pozorováni, aby bylo zjištěno co a proč dělají špatně. Z čehož byly následně vyvozeny nutné úpravy v uživatelském rozhraní.

1. Smazání záznamu budu možné nyní ještě u detailu tohoto záznamu. Tato funkcionalita bude ještě znovu podrobena testu u prototypu aplikace. Gesto *swipe* se u wireframu špatně testuje.

2. „Má letadla“ již nebudou pod záložkou „Profil“, ale budou představovat samostatný *tab*.

3. Vytvoření reportu bude nyní označeno ještě navíc textem.

4. Při přidávání nového záznamu si uživatel nejprve vybere zda se jedná o záznam letu nebo o záznam ze simulátoru.

5. U zobrazení seznamu záznamů budou informace o jednotlivých záznamech v jiném pořadí – datum, čas, místa vzletu a příletu a letadlo.

Po provedení testování s pilotem byla diskutována navržená funkcionalita. Výsledkem této diskuze bylo:

- přidání celkové statistiky nalétaných hodin,
- filtrování bude nyní možné i podle typu letadla,
- záznamy letů budou zobrazovat nyní registrační číslo letadla místo jeho modelu, typu a varianty.

Případy užití zobrazené v této práci již zahrnují tuto přidanou a upravenou funkcionalitu. Po uživatelských testech a diskuzi byla vytvořena i poslední verze wireframe, také k nalezení v příloze této diplomové práce.

4.4 Navržená řešení pro tvorbu aplikace

Tato kapitola popisuje a zdůvodňuje zvolená řešení architektury, perzistence dat a FRP framework.

4.4.1 Architektura

Pro svou práci jsem si na základě navržených funkcionalit zvolil architekturu MVVM. Architektura MVVM eliminuje nevýhodu MVC – příliš mnoho logiky a kódu ve vrstvě *Controller*, a na druhou stranu není zbytečně složitá (pro navrženou funkcionalitu) jako VIPER.

4.4.2 Perzistence dat

Pro tvorbu aplikace na evidenci letů jsem si zvolil možnost perzistence dat pomocí Realmu. Prvním z důvodů této volby je podpora online i offline uložení dat. To umožňuje uživateli mít stejná data na více zařízeních, však i společně s možností používat aplikaci offline. Dalším důvodem bylo to, že je Realm zcela zdarma, což mu dává výhodu oproti iCloud – Apple řešení.

4.4.3 FRP framework

Pro realizaci aplikace byla zvolena ReactiveCocoa jako FRP framework. ReactiveCocoa byla vybrána, protože příhodně rozšiřuje framework ReactiveSwift, má také přehlednou dokumentaci a existují kvalitní příklady demonstrující použití tohoto frameworku.

Kapitola 5

Realizace

5.1 Programovací jazyk Swift

Swift je programovací jazyk vytvořený společností Apple. Jedná se o nástupce jazyku C a Objective-C a používá se pro tvorbu iOS, macOS, watchOS a tvOS aplikací.

Swift je objektově orientovaný open source jazyk. Mezi jeho hlavní přednosti patří:

- automaticky spravovaná paměť,
- podpora funkcionálního programování (filter, map, reduce),
- možnost vracení více hodnot z funkcí,
- podpora tzv. generics,
- nativní zpracovávání chyb pomocí try, cache, trow,
- podpora protokolů a tzv. extensions (rozšíření již existujících tříd). [3]

Swift je relativně mladý programovací jazyk, první oznámení proběhlo roku 2014 a nyní je nejnovější verzí Swift 4. Právě v této verzi byla implementována aplikace v rámci této diplomové práce. [55]

V této kapitole jsou dále rozebírány některé funkcionality a vlastnosti jazyka Swift společně s ukázkami kódu z vytvářené aplikace.

5.1.1 Optionals

Swift je tzv. typově bezpečný jazyk, což zaručuje to, že daná proměnná bude mít vždy stejný typ. Tedy např. pokud náš kód vyžaduje *String* není možné zadat *Int*. Proměnné se definují klíčovým slovem *var* a konstanty *let*. Proměnné musí být vždy inicializované před použitím a objekty nikdy nemohou být *nil*. Pro práci s hodnotami *nil* jsou ve Swiftu zavedeny tzv. *Optinals*.

Listing 5.1: Příklad práce s Optinal

```
// optinal variable initialization
var delegate: NoteViewControllerDelegate? = nil
...
private func save(note: String) {
    // optional variable unwarpping using optional binding
    if let delegate = delegate {
        delegate.save(note: note)
    }
    ...
}
```

Optinal je jakýsi obal (wrapper) klasických typů (*String*, *Int*, *Double* apod.), který reprezentuje dvě možnosti: hodnota je nastavena a je tedy možné tzv. *unwrap optinal* a tím získat danou hodnotu, nebo hodnota není nastavena (jedná se o *nil*). [56]

Příklad 5.1 zobrazuje nejprve inicializaci *optional* proměnné, kde otazník indikuje *optinal* hodnotu uvnitř proměnné. Následně je ve funkci *save* použito pro *unwrap* proměnné tzv. *optional binding*. To zpřístupní hodnotu *optional* proměnné (pokud byla hodnota nastavena) v podobě dočasné konstanty. V případě, že je hodnota předána dočasné konstantě, můžeme k ní přistupovat klasicky a v našem případě delegovat funkci *save*.

5.1.2 Extensions

Extension, neboli rozšíření, je klíčové slovo jazyka Swift, které umožňuje přidat novou funkcionalitu nebo atributy k již existující třídě, struktuře, protokolu apod. Toto zahrnuje i třídy, u kterých není možné upravit jejich zdrojový kód, tedy např. třídy UIKit.

Extensions umožňují:

- definovat nové metody,
- přidat konstruktory,
- upravit typ tak, aby splňoval daný protokol a další. [57]

Ukázka 5.2 předvádí rozšíření třídy $UITableViewController$ o novou generickou metodu. Tato metoda je použita u dynamických tabulek zobrazujících data z Realmu. Díky principů FRP může tabulka jednoduše podporovat např. smazání záznamu, protože je neustále pozorován *Signal* se změnami a podle změn jsou následně provedeny vhodné animace na straně $UITableView$.

Dalšího klíčového slova, kterého je možné si v ukázce všimnout je *guard*. *Guard* je možné vnímat podobně jako tzv. assert. Tedy zadáváme podmínku,

Listing 5.2: Ukázka Extension

```
extension UITableViewController {
    func observeSignalForTableDataChanges<T>(
      with signal: Signal<RealmCollectionChange<Results<T>>,
      NoError>) {
      // observing input signal
      signal.observeValues{ [weak self] changes in
         guard let tableView = self?.tableView
            else { return }
         // switch on changes type
         switch changes {
         case .initial:
            tableView.reloadData()
         case .update(_, let deletions, let insertions,
                      let modifications):
            tableView.beginUpdates()
            tableView.insertRows(at: insertions.map
               ({ IndexPath(row: $0, section: 0) }),
               with: .automatic)
            tableView.deleteRows(at: deletions.map
               ({ IndexPath(row: $0, section: 0)}),
               with: .automatic)
            tableView.reloadRows(at: modifications.map
               ({ IndexPath(row: $0, section: 0) }),
               with: .automatic)
            tableView.endUpdates()
         case .error(let error):
            fatalError("\(error)")
         }
      }
   }
}
```

u které chceme aby byla splněna, pokud splněna není dostáváme se do povinné *else* větve. Klíčové slovo *guard* je jednou z možností pro *unwrap optional* proměnné a také zamezuje několikanásobné podmínce $if-else$ (ve složitějších případech), čímž dochází k zpřehlednění kódu. [58]

5.1.3 Protokoly

Protokoly představují ve Swiftu předpis metod, vlastností a dalších náležitostí, které jsou nutné pro splnění daných úloh. Podmínky protokolu mohou splňovat

Listing 5.3: Definice protokolu

```
protocol NoteViewControllerDelegate {
    func save(note: String)
}
```

třídy, struktury a další tím, že poskytnou implementaci všech požadavků. [59]

V rámci implementace aplikace bylo definováno několik protokolů. Například v ukázce 5.1, musel *delegate* odpovídat protokolu $NoteViewControllerDelegate$ a tedy mít implementovanou metodu *save* (ukázka definování protokolu viz. 5.3). Jak je možné vidět, protokol neurčuje jak má být metoda implementována, pouze určuje její definici.

Princip delegování je v jazyce Swift velice častý. I mnoho tříd např. z UIKit vyžaduje pro podporu určitých funkcionalit splnění daného protokolu. Ukázka 5.4 ukazuje implementaci protokolu $UIPickerViewDelegate$ a $UIPickerViewDataSource$. Kdy po implementaci příslušných metod a nastavení $UITableViewController$ jako delegáta, je možné ve $View$ pracovat s komponentou $UIPickerView$.

5.1.4 Automatické počítání referencí

Automatická správa paměti, jak je již řečeno dříve, je jednou z předností jazyka Swift. Tato správa je zajištěna metodou automatického počítání referencí. Toto počítání zajišťuje udržení reference dané instance pouze po dobu, co je opravdu potřebná. V době, kdy instanci není nutné udržovat (již neexistují žádné reference), dojde k uvolnění předem alokované paměti.

Swift definuje tři druhy referencí – *strong*, *weak* a *unowned*. Nejčastějším a základním případem je *strong* reference, které zaručuje, že instance, na kterou reference ukazuje, nebude dealokována dokud reference existuje. Může však nastat případ nazývaný cyklus *strong* referencí, který způsobí tzv. paměťový *leak* (nedojde k uvolnění veškeré alokované paměti). Tento cyklus může být způsoben vzájemnými referencemi dvou instancí nebo např. pokud třídní *closure* zachytí $self$. Z tohoto důvodu jsou vytvořeny další dva typy referencí. $Weak$ reference neudržuje *strong* referenci na instanci a není tedy bráněno dealokaci dané instance. Toto chování umožňuje i to, že *weak* reference může mít hodnotu *nil*. $Weak$ reference se používá pokud má instance v referenci kratší životnost než samotná instance vlastnící referenci. $Unowned$ reference se používá v opačném případě a nemůže mít nikdy hodnotu *nil*. [60]

5.1.5 Použité Apple frameworky

Aplikace využívá třech Apple frameworků – UIKit, Foundation a CloudKit.

UIKit je základním frameworkem pro $View$ část aplikace. Obsahuje části pro:

Listing 5.4: Implementace protokolu

```
class AddMedicalCertificateViewController:
RecordTableViewController,
UIPickerViewDelegate,
UIPickerViewDataSource {

   private let picker = UIPickerView()

   override func viewDidLoad() {
      // setting self as delegate
      picker.delegate = self
   }

   // MARK: - UIPickerViewDataSource

   func numberOfComponents(in pickerView: UIPickerView)
       -> Int {
      return 1
   }

   func pickerView(_ pickerView: UIPickerView,
         numberOfRowsInComponent component: Int) -> Int {
      return viewModel.getTypesCount()
   }

   func pickerView(_ pickerView: UIPickerView,
         titleForRow row: Int, forComponent component: Int)
       -> String? {
      return viewModel.getType(for: row)
   }

   func pickerView(_ pickerView: UIPickerView,
         didSelectRow row: Int,
         inComponent component: Int) {
      typeTextField.text = viewModel.getType(for: row)
      viewModel.typeString.value =
         viewModel.getType(for: row)
   }
}
```

- vytvoření uživatelského rozhraní;
- zajištění komunikaci mezi uživatelem, systémem a aplikací;
- zpracování uživatelských interakcí;
- kreslení a tisknutí;
- animace a mnohé další. [61]

Foundation obsahuje základní datové typy, kolekce, třídy atd. Umožňuje i perzistenci dat, práci s datem a časem nebo poskytuje funkce pro práci s textem. [62]

CloudKit framework umožňuje práci s iCloudem, tedy ukládání dat a přístup k k uloženým datům z více zařízení. V této aplikaci byl tento framework použit pouze pro autentizaci uživatele (dále popsáno v kapitole: Přihlašování uživatele).

5.2 Reactive Cocoa

Tato kapitola se zabývá jednotlivými prvky frameworků Reactive Cocoa a Reactive Swift, společně s ukázkami a vysvětlením jak byly v aplikaci použity. Na ukázkách je také znázorněno, jak principy FRP fungovaly s MVVM architekturou.

5.2.1 Signal

Prvním základním prvkem frameworku Reactive Swift je *Signal*. Jedná se datový tok, který posílá v průběhu svého života události (Events) – hodnoty (Value) nebo chyby (Error). Všichni pozorovatelé daného *Signalu* dostávají stejné události ve stejný čas a nejsou schopni *Signal* svou činností ovlivnit. [63]

Příklad 5.5 zobrazuje použití prvku *Signal*. Společně se *Signalem* je vytvořen ještě *Observer*. Tato dvojice je vytvořena pomocí funkce *pipe*, která vrací společně navázanou dvojici *Signal* a *Observer*. Tato dvojice následně pracuje spolu – všechny události odeslané *Observeru* jsou možné pozorovat na *Signalu*. *Signal* zobrazený v ukázce se stará o informování o změnách v kolekci Realm objektů. Tento druh *Signalu* je ten, který následně zpracovává příslušný $UITableViewController$ pomocí rozšíření (extension) zobrazené na příkladu 5.2.

5.2.2 SignalProducer

Dalším prvkem je *SignalProducer*. *SignalProducer* vytváří *Signal* a je využíván společně s *Action*. *Signal* je vytvářen pomocí funkce *start*(). Tato

Listing 5.5: Ukázka použití prvku Signal

```
class RealmTableViewModel<T: Object>: RealmViewModel {
    ...
    // observed collection (using Realm notifications)
    internal var collection: Results<T>?
    let collectionChangedSignal:
        Signal<RealmCollectionChange<Results<T>>, NoError>
    private let collectionChangedObserver: Signal
        <RealmCollectionChange<Results<T>>, NoError>.Observer

    override init() {
        // signal and observer initialization
        let (arrayChangedSignal, arrayChangedObserver) =
            Signal<RealmCollectionChange<Results<T>>, NoError>
            .pipe()
        self.collectionChangedSignal = arrayChangedSignal
        self.collectionChangedObserver = arrayChangedObserver
        super.init()
    }
     ...
    internal func collectionChangedNotificationBlock(
        changes: RealmCollectionChange<Results<T>>) {
        collectionChangedObserver.send(value: changes)
    }
}
```

funkce vytvoří vždy nový *Signal* a z toho důvodu různé *Signaly* mohou mít různé verze událostí nebo se mohou úplně lišit. [64]

SignalProducer je zobrazen v ukázce 5.7 společně s prvkem *Action*.

5.2.3 MutableProperty

MutableProperty je dalším prvkem frameworku Reactive Swift. *MutableProperty* obsahuje hodnotu, u které je možné pozorovat její změny. Tento prvek také obsahuje vlastní prvky typu *Signal* a *SignalProducer*. [65]

MutableProperty v aplikaci představuje nejčastější způsob namapování *View* na *ViewModel*. Ukázka 5.6 zobrazuje hned několik propojení jednotlivých vrstev architektury. První propojení přestavuje proměnná *fromTextField*, která je označená jako *@IBOutlet*. Právě toto označení se používá při spojení prvků *storyboardu* s kódem. Další propojení, tentokrát již vrstev *View* a *ViewModel*, je zobrazeno ve funkci *bindViewModel*, kde dojde nejprve k nastavení výchozí hodnoty pole *UITextField* a následně jsou pozorovány

Listing 5.6: Ukázka propojení *View* a *ViewModel*

```
// View
class AddFlightRecordTableViewController:
RecordTableViewController {
    // storyboard reference
    @IBOutlet weak var fromTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        ...
        bindViewModel()
    }

    private func bindViewModel() {
        // setting default value
        fromTextField.text = viewModel.from.value
        // bind UITextField to MutableProperty
        viewModel.from <~ toTextField.reactive
            .continuousTextValues.filterMap{ $0 }
        ...
    }
}

// ViewModel
class AddFlightRecordViewModel: RealmViewModel {
    let from: MutableProperty<String?>
    ...
    init(with record: Record?) {
        from = MutableProperty(record?.from ?? nil)
        ...
    }
}
```

všechny změny. Toto pozorování umožňuje framework Reactive Cocoa, který přidává UI prvkům vlastnost *reactive*.

Při nastavování výchozí hodnoty *UITextFieldu* závisí na tom zda se jedná o vytváření nového záznamu letu nebo o úpravu již existujícího. Podle toho je vytvořen *ViewModel* (se záznamem letu nebo s hodnotou *nil*) i *MutableProperty*.

5.2.4 Action

Action je posledním použitým prvkem frameworku Reactive Swift. Hlavní částí prvku *Action* je tzv. *execute* reprezentovaný $SignalProducerem$. Tento *execute* je zadán ve formě closure. *Action* může obsahovat podmínku $enabledIf$, která určuje zda je umožněno akci provést.

Použití prvku *Action* je zobrazeno v ukázce 5.7. Akce je znovu napojena pomocí vlastnosti *reactive* frameworku Reactive Cocoa. Samotná akce pouze odesílá na stisknutí tlačítka proměnou *note*, která je následně delegována do jiného $Controlleru$. Tato akce je však proveditelná pouze pokud je splněna podmínka v části $enabledIf$, tedy *note* nemá nulovou délku. Díky propojení s tlačítkem ve $View$ se proveditelnost akce promítá i do vlastností tlačítka $UIBarButtonItem$. Po provedení akce je informován *observer* ve $View$, který pozoruje *Signal* vytvářený $SignalProducerem$ akce.

5.2.5 Použité základní operátory

Reactive Swift obsahuje mnoho operátorů, které umí nějakým způsobem upravovat pozorované toky dat. Tato kapitola se zabývá jednotlivými použitými operátory společně s ukázkami jejich použití v aplikaci.

5.2.5.1 Map

Operátor *map* je jeden z nejzákladnějších operátorů funkcionálního programování. V případě Reactive Swift přetváří hodnoty jednoho datového toku na jiný datový tok. [66]

V aplikaci byl operátor *map* velice užitečný. Byl použit hlavně pro spojení dvou $MutableProperty$ s odlišnými typy. Toto spojení probíhalo např. při práci s objekty typu $UIDatePicker$ (výběr datu), kdy jedna $MutableProperty$ získává uživatelovi vstupy ve formátu *Date* a tento tok hodnot je průběžné transformován jako *String* do druhé $MutableProperty$. Ukázka je zobrazena v kódu 5.8. Na ukázce je možné vidět, že přepis transformace je zadán jako již existující funkce, stejného výsledku je však možné dosáhnout i při zadání přepisu ve formě closure.

5.2.5.2 Reduce

Operátor *reduce* je další ze základních operátorů funkcionálního programování. Používá se pro spojení hodnot datového toku (popř. pole hodnot) do jedné. [66]

Operátor *reduce* byl použit např. při generování PDF souboru se záznamy letů pro výpočet jednotlivých součtů odletů, příletů apod. V ukázce 5.9 je vidět použití closure (pro součet je zbytečné vytvářet speciální funkci), který používá zkrácený zápis argumentů – $1 a $2. Tento zápis velice zpřehledňuje a zkracuje kód jednoduchých closurů.

Listing 5.7: Ukázka použití Action

```
// View
class NoteViewController: UIViewController {
    @IBOutlet weak var saveBtn: UIBarButtonItem!
    ...
    override func viewDidLoad() {
        super.viewDidLoad()
        ...
        bindViewModel()
    }

    private func bindViewModel() {
        // binding action to button
        saveBtn.reactive.pressed =
            CocoaAction(viewModel.saveAction)
        // observing action values
        viewModel.saveAction.values.observeValues(save)
        ...
    }
}

// ViewModel
class NoteViewModel {
    var note: MutableProperty<String?>
    let saveAction: Action<(), String, NoError>

    init(note: String?) {
        self.note = MutableProperty(note)
        // action definition
        saveAction = Action<(), String, NoError>(
            state: self.note, enabledIf: {
            if let note = $0 {
                return note.count > 0
            }
            return false
        }) { note, _ in
            return SignalProducer<String, NoError> {
                observer, _ in
                observer.send(value: note!)
            }
        }
    }
}
```

Listing 5.8: Ukázka použití Map

```
extension DateFormatter {
    struct Format {
        static let date = "dd.MM.yyyy"
    }
    func dateToString(from date: Date) -> String {
        dateFormat = Format.date
        return string(from: date)
    }
}
class PersonalInformationsViewModel: RealmViewModel {
    private let dateFormatter = DateFormatter()
    let birthDay = MutableProperty<Date?>
    let birthDayString = MutableProperty<String?>
    ...
    override init() {
        birthDayString <~ birthDay.producer
            .map(dateFormatter.optinalDateToString)
    }
}
```

Listing 5.9: Ukázka použití Reduce

```
private func reduceSumToString(from values: [Int])
 -> String {
        return String(values.reduce(0) { $0 + $1 })
}
let sumString = reduceSumToString(from: records!
    .map{ Int($0.tkoNight) })
```

5.2.5.3 Filter

Posledním ze základních operátorů je *filter*. Tento operátor zajišťuje ponechání hodnot v datovém toku, ale pouze těch které splňují zadanou podmínku.[66]

5.2.5.4 Spojování datových toků

Z operátorů pro spojování datových toků byl použit operátor *combineLatest*. Tento operátor spojuje dva a více datových toků a odesílá hodnotu až poté, co všechny datové toky odeslaly alespoň jednu hodnotu. Následně je odeslána hodnota při jakékoliv změně kteréhokoliv ze spojovaných datových toků. [66]

Operátor *combineLatest* byl využit pro výpočet celkového času letu, který je vypočten ze dvou vstupních hodnot – z času odletu a času příletu. Vypoč-

Listing 5.10: Ukázka použití spojení dvou datových toků

```
let timeTKO: MutableProperty<Date>
let timeLDG: MutableProperty<Date>
let totalTime = MutableProperty<String>("")

totalTime <~ SignalProducer
    .combineLatest(timeTKO.signal, timeLDG.signal)
    .map(countTotalTime)
    .map(dateFormatter.timeToString)
```

tená hodnota je přepočítána vždy při úpravě jakékoliv z těchto vstupních hodnot. Použití operátoru *combineLatest* je zobrazeno na ukázce 5.10.

5.2.5.5 Binding operátor

Binding operátor umožňuje spojení jakýchkoliv datových toků. Je zapisován symboly <~ a Reactive Cocoa přidává vlastnost i pro prvky UI, která umožňuje použití tohoto operátoru. [24]

Použití tohoto operátoru je zobrazeno např. v ukázce 5.6. Kdy dochází k napojení vstupního textového pole s *MutableProperty*. *Binding* operátor je jedna z nejčastějších možností propojení *View* a *ViewModelu*.

5.3 Realm

Realm byl ve vytvářené aplikaci použit pro ukládání dat a to jak pro lokální uložení, tak i pro synchronizaci s Realm Object Serverem.

Uživatel je do Realmu přihlašován pomocí svého iCloud účtu (popsáno dále) a inicializaci Realmu zpracovává v aplikaci třída *RealmViewModel*.

5.3.1 Model

Objekty uložené v Realmu jsou definované jako klasické třídy ve Swiftu s nutnou dědičnosti od třídy *Object*. Tyto objekty se chovají převážně jako klasické Swift objekty, je tedy možné aby splňovaly protokoly nebo měly vlastní metody. Tyto objekty mohou mít také definovány primární klíče nebo indexy.

Ukázka třídy *Plane* definující Realm objekt je zobrazena v kódu 5.11. V aplikaci byly takto definovány třídy pro záznam letu nebo ze simulátoru, letadlo, osobní informace a zdravotní certifikáty.

Nevýhodou objektů v Realmu je nedokonalá dědičnost. Dědičnost Realm objektů nedovoluje: konverzi typu polymorfních tříd, dotazování se na více tříd najednou a více-třídní kontejnery. [67] Z důvodů těchto nedostatků nemohla být třída definující záznam letu a záznam simulátoru rozdělena do dvou

Listing 5.11: Ukázka modelu Plane

```
final class Plane: Object {
    @objc dynamic var type: String?
    @objc dynamic var model: String?
    @objc dynamic var variant: String?
    @objc dynamic var registrationNumber: String?
    @objc dynamic var engine: Engine = .single

    @objc enum Engine: Int {
        case single = 0
        case multi = 1
    }
}
```

(nebylo by možné nad záznamy vytvářet společný dotaz nebo je zobrazovat v jedné tabulce). Bylo tedy nutné mít jednu třídu pro oba typy záznamů a mezi nimi rozlišovat pomocí příznaku *type*.

5.3.2 Dotazy

Dotaz v Realmu vrací kolekci *Results*, která obsahuje jednotlivé objekty. Všechny dotazy jsou tzv. lazy, dochází ke čtení až v okamžiku, kdy jsou data vyžádána (jsou např. čtena). [67] Základní dotaz je zobrazen na příkladu 5.12.

Poté, co je získána kolekce *Results*, je možné výsledky filtrovat a řadit. Filtrování je možné pomocí *NSPredicate*. [67] *NSPredicate* je třída obsažená v knihovně *Foundation*, tato třída představuje podmínku, jenž může být použita pro filtrování nad kolekcemi objektů. *NSPredicate* může obsahovat klíčová slova – like, between, contains, and, or a mnohé další. [68]

U kolekce *Results* je také možné se registrovat a tím získávat oznámení (notifications) o všech změnách. Tato registrace je možná jak pro kolekce, tak i pro celý Realm. [67] V aplikaci byly notifikace předávány pomocí *Signalu* do *View*, kde byly následně zobrazeny vhodné animace (zobrazeno na příkladech 5.5 a 5.2).

5.3.3 Ukládání a upravování objektů

Ukládání a upravování Realm objektů musí být prováděno v transakci *write*. Pro uložení objektu musí být objekt nejprve vytvořen (konstruktorem) a poté vložen do Realm v transakci *write* pomocí funkce *add*. Po uložení je možné objekt nadále používat i upravovat (znovu v transakci *write*). [67]

Ukázka 5.13 zobrazuje uložení (upravení) objektu *Plane* do Realmu. Pokud již proměnná *plane* obsahovala objekt *Plane*, dojde k upravení (update)

Listing 5.12: Ukázka dotazu v Realmu

```
// Realm query
collection = realm.objects(Record.self)
let type = "Boeing"
// filter Results using NSPredicate and sort
collection = collection?
    .filter("plane.type contains[c] %@", type)
    .sorted(byKeyPath: "date", ascending: false)
// collection notifications initialization
notificationsToken = collection?.observe {
    // closure handling notifications
}
```

Listing 5.13: Ukázka zápisu/úpravy Realm objektu

```
// plane is set or create new one
let plane = self.plane ?? Plane()
try! realm.write {
    plane.type = type.value
    plane.model = model.value
    plane.variant = variant.value
    plane.registrationNumber = registrationNumber.value
    plane.engine = engine.value
    // insert/update plane
    realm.add(plane)
}
```

v Realmu. Pokud byla proměnná *plane nil*, je vytvořen nový objekt *Plane*, který je následně uložen do Realmu.

5.3.4 Přihlašování uživatele

Realm umožňuje několik možností přihlášení (autentizace) uživatele – pomocí přihlašovacích údajů (jméno, heslo), pomocí tokenu získaného od podporované autentizační služby třetí strany (iCloud, Facebook, Google) nebo pomocí tokenu od vlastní autentizační služby. [67]

Pro přihlašování uživatele bylo vybráno řešení pomocí iCloud tokenu. Toto řešení navrhuje i článek Sebastiana Kreutzbergera [69]. Tento článek rozebírá nespokojenost uživatelů, kteří jsou nuceni vyplňovat přihlašovací údaje ihned při prvním spuštění aplikace. Navrhovaným řešením je právě iCloud a to z toho důvodu, že každý uživatel má iCloud účet. Kdy tento účet je nutný minimálně ke stahovaní aplikací z App Storu.

Pro přihlášení uživatele pomocí iCloudu je nutné využít framework CloudKit. CloudKit obsahuje třídu $CKContainer$, která umožňuje získat uživatelův iCloud token pomocí funkce $fetchUserRecordID$. [70] Pomocí tohoto tokenu je uživatel následně přihlášen do Realmu. Získání tohoto tokenu má určitou časovou režii [70], proto je token získáván pouze při prvním spuštění aplikace. Následně je již uživatel uložen v Realm objektu $SyncUser.current$ (dokud nedojde k expiraci přihlášení). O přihlášení uživatele se stará v aplikaci třída $RealmViewModel$.

Aby bylo možné přihlášení uživatele do Realmu pomocí iCloud tokenu, je nutné provést několik kroků. Nejprve je nutné vygenerovat dvoji soukromého a veřejného klíče pro objektový server Realm. Soukromý klíč je následně nutné vložit do CloudKit Dashboardu. CloudKit Dashboard dále vygeneruje $KeyID$, které je nakonec nutné vložit do konfiguračního souboru objektového serveru Realm společně s cestou k dříve vytvořenému soukromému klíči. [71]

5.4 Tvorba uživatelského rozhraní

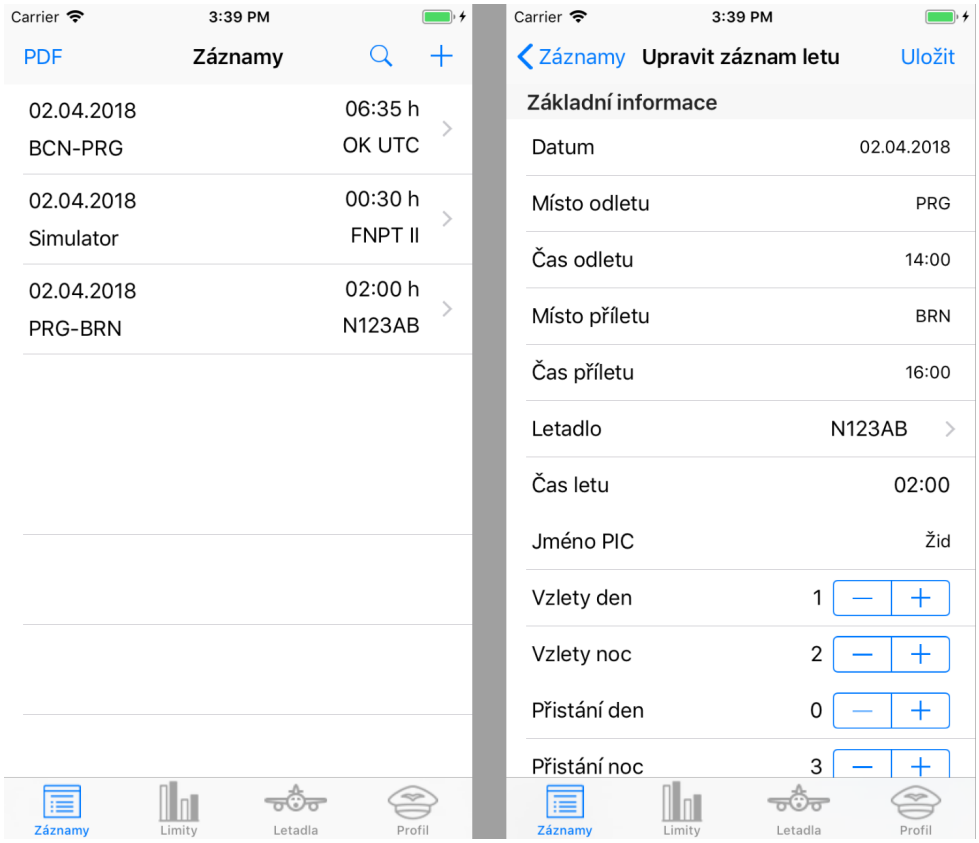
Tvorba uživatelské rozhraní ve formě storyboardů je možná přímo v Xcode pomocí zabudovaného editoru tzv. Interface Builder.

Pro pozicování jednotlivých prvků je vytvořený speciální systém rozložení tzv. Auto Layout. Auto Layout je postavený na myšlence vazeb (constraints). Každý prvek má definované tyto vazby k ostatním prvkům a díky tomu je přesně určeno jak se bude chovat na různých velikostech nebo orientacích displeje. [72]

Pro různé velikosti a orientace displejů zavádí Apple ještě třídy velikostí (size classes). Tyto třídy jsou dvě – *Regular* a *Compact*, jsou definované pro výšku i šířku daného zařízení a mohou se lišit při orientaci na výšku a na šířku. Např. iPhone 6 má při orientaci na výšku definované třídy: pro výšku *Regular* a pro šířku *Compact*, zatímco při orientaci na šířku je to: pro výšku i šířku *Compact*. Tyto třídy umožňují přizpůsobit jednotlivé obrazovky, tak aby byly co nejideálněji zobrazeny na daném zařízení např. tím, že jsou změněny vazby nebo nedojde k zobrazení některých prvků. [73]

Dalším prvkem, který je možné použít pro rozdílná zobrazení na různých velikostech displejů, je tzv. $UISplitViewController$. Tento $Controller$ obsahuje dva $ViewControllery$ – *master* a *detail*. Ty jsou zobrazeny v závislosti na třídách velikosti – u menších displejů se $UISplitViewController$ chová spíše jako klasický $NavigationController$, zatímco u větších displejů (např. tabletů) jsou *master* a *detail* zobrazeny vedle sebe. [74] Pro ukázku byly zvoleny zařízení iPhone 6 (obrázek 5.1 zobrazující $UISplitViewController$ jako $NavigationController$) a iPad Air 2 (obrázek 5.2 zobrazující *master* a *detail* vedle sebe).

V příloze této práce je k nalezení nafocená celá aplikace na zařízení iPhone 6 společně s obrazovkami (pouze těmi, co se na liší na tabletu) ze zařízení iPad



Obrázek 5.1: UISplitViewController na zařízení iPhone 6

Air 2.

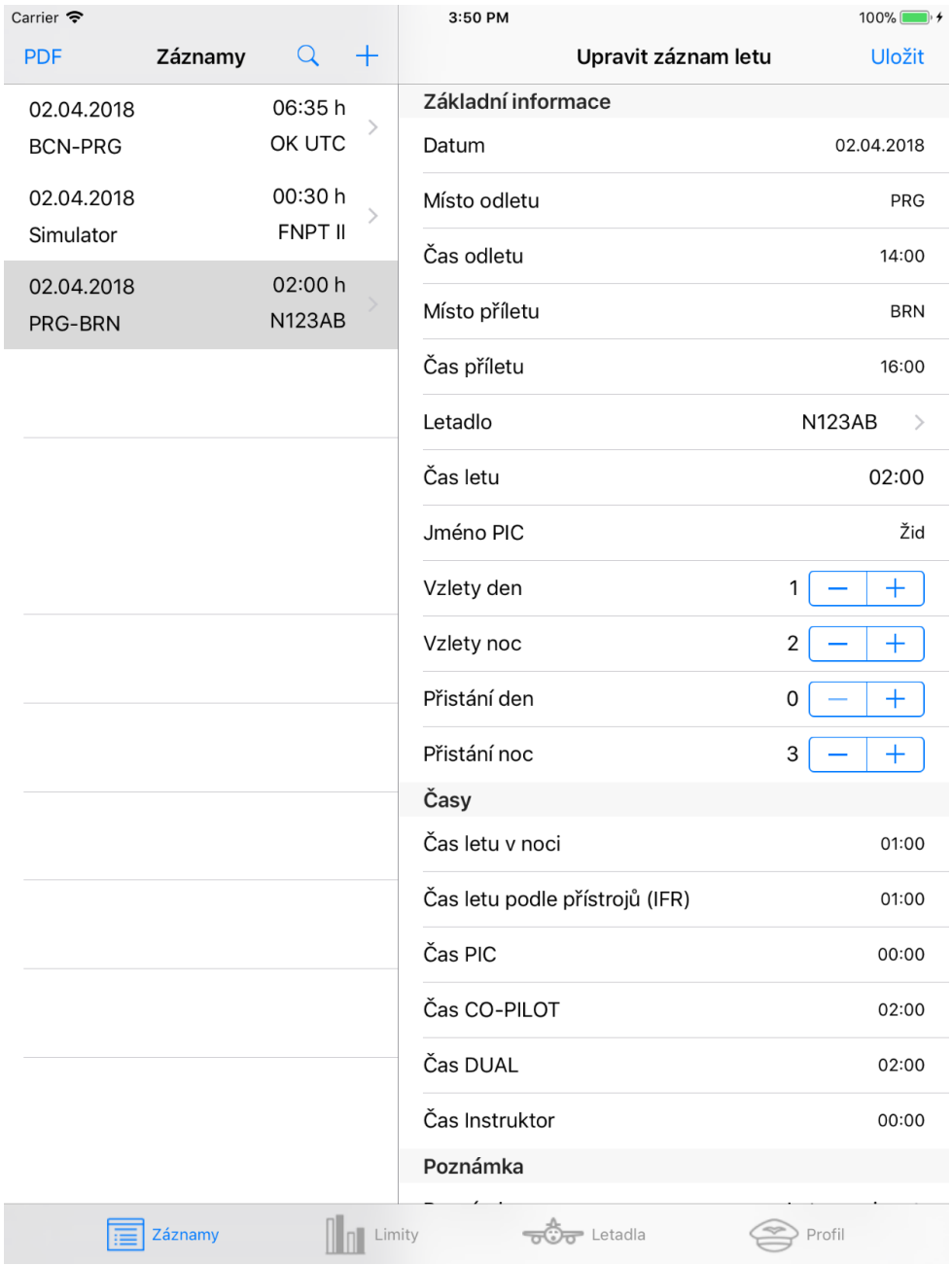
5.5 Překlady

U překladů iOS aplikací je nutné uvést dva pojmy – internacionalizaci a lokalizaci. Internacionalizace je proces zmezinárodnění aplikace, tedy upravení metadat na App Storu, zacílení marketingu, přizpůsobení kulturám daných zemí a lokalizace aplikace. Lokalizace je tedy součást internacionalizace, obsahuje části jako překlady, upravení formátu dat nebo změnění jednotek cen, váhy apod. [75]

Celá aplikace je nyní vytvořena v češtině, ale byla vyvíjená tak, aby ji bylo možné přeložit (lokalizovat) do libovolného jazyka.

Soubory připravené pro překlad aplikace jsou dvojího typy – překlady storyboardů a tzv. *Localizable.string*. Oba typy jsou závislé na tom, jaké jazyky jsou nastaveny v Xcode projektu.

Soubory pro překlad storyboardů vytváří samotné Xcode. Každý UI prvek je identifikován vygenerovaným ID, které je spojeno s textem, který daný



Obrázek 5.2: UISplitViewController na zařízení iPad Air 2

Listing 5.14: Ukázka přípravy storyboardu pro překlad

```
/* Class = "UILabel"; text = "Datum";
 ObjectID = "bPZ-q2-xLN";
 */
"bPZ-q2-xLN.text" = "Datum";
...
```

Listing 5.15: Ukázka použití *Localizable.string*

```
// Localizable.string
"Plane" = "Letadlo";

// get localized Plane
let label = NSLocalizedString("Plane", comment: "")
```

prvek obsahuje. Ukázka storyboardu připraveného pro překlad je zobrazena na příkladu 5.14. Tento příklad zobrazuje českou verzi textů storyboardu. Při překladech by došlo k vygenerování např. anglické verze, kde by všechny prvky měli stejné ID jako u české verze a podle nich by došlo k přeložení příslušných textů. Tento způsob překladů byl použit u neměnících se textů prvků ze storyboardů.

Druhý způsob byl použit u textů, které se mohou měnit po spuštění aplikace nebo nejsou definované ve storyboardech. Tento způsob používá soubory *Localizable.string*, kde každý překládaný jazyk má svůj vlastní soubor. Každý z těchto souborů obsahuje dvojice klíč-hodnota, kde klíč představuje ID a hodnota text. Získání hodnoty je možné pomocí *NSLocalizedString*, který vrací přiřazený lokalizovaný text podle zadaného klíče. Ukázka použití *Localizable.string* je zobrazena na příkladu 5.15.

5.6 Generování PDF

Tato sekce se zabývá způsobem generování reportu do formátu PDF. Jedná se o funkcionalitu, kde je možné z vybraných záznamů letů a záznamů ze simulátoru vygenerovat report do formátu PDF ve stylu zápisníku letů Jeppesen (`http://www.transair.co.uk/sp+Jeppesen-European-Pilot-Logbook-EU-FCL-050+5946`).

Při generování PDF reportu je vytvořena nejprve tabulka ve formátu HTML, ta je vyplněna daty a až z tohoto HTML souboru je generován PDF report. HTML tabulka je v kódu postupně skládána z několika HTML souborů:

- layout.html – obsahuje základní definici HTML stránky společně s CSS

stylováním,

- table.html – obsahuje samotnou definici tabulky tedy rozložení sloupců a hlavičkové buňky,

- všechny ostatní soubory s přívlastkem *row* – jsou řádky tabulky, které jsou v postupně generovány, vyplňovány a nakonec vloženy do tabulky.

Postupné skládání tabulky je zajištěno pomocí značek (začínající znakem #) v HTML souborech. Tyto soubory jsou načteny ve formě *Stringu* a následně, pomocí funkce $replacingOccurrences(of : with :)$, jsou jednotlivé značky nahrazovány reálným obsahem. Ukázku tohoto postupného skládání HTML souboru zobrazuje příklad 5.16. Pomocí tohoto postupu jsou tedy vyplňována uživatelova data, počítány statistiky a připraveny překlady.

Pro následné vytvoření PDF reportu byly použity dvě třídy z UIKit – $UIMarkupTextPrintFormatter$ a $UIPrintPageRenderer$. První z nich se starala o připravení tisknutelného obsahu z HTML souboru. A od druhé dědila vlastní třída $CustomPrintPageRenderer$, která zajišťovala vykreslení (i několika stránek) PDF souboru.

5.7 Dokumentace

Společně s implementací aplikace byla vytvořena i její dokumentace. Při psaní kódu byly dodržovány Apple „Design Guidelines“ a byla tedy snaha o psaní tzv. self explanatory kódu. Kód byl doplňován komentáři – proč daná část kódu dělá to, co dělá.

Samotná dokumentace byla poté zaměřena hlavně na veřejnou část tříd a na složitější metody. Pro tvorbu dokumentace bylo použito oficiální
Apple značení (markup). A po dokončení aplikace byla dokumentace vygenerována pomocí programu „Jazzy“ (`https://github.com/realm/jazzy`) do formátu HTML.

Vygenerovaná dokumentace k je nalezení v příloze této diplomové práce.

5.8 Použité nástroje při vývoji

5.8.1 Xcode

Xcode 9 byly použity pro vývoj, testování a simulování vytvářené aplikace.

Xcode je software od Applu pro operační systém macOS a používá se pro vývoj aplikací na Mac, iPhone, iPad, Apple Watch a Apple TV. Xcode obsahují zabudované nástroje pro psání kódu, kompilaci, testování, vytváření uživatelského rozhraní, spouštění vytvářené aplikace na simulátorech a další. [76]

Listing 5.16: Ukázka postupného skládání HTML souboru

```
// layout.html
<!doctype html>
<html>
...
   <body>
      #TABLE
   </body>
</html>

// ViewModel
private struct Marks {
   static let table = "#TABLE"
   ...
}

func generateHTMLString() -> String? {
   ...
   // get path to layout.html
   let pathToLayout = Bundle.main
      .path(forResource: HTMLFiles.layout, ofType: html)
   // convert HTML file to String
   var layout = try String(contentsOfFile: pathToLayout!)
   // replace #TABLE with actual tables
   layout = layout.replacingOccurrences(of: Marks.table,
               with: try generateTables())
   ...
}

private func generateTables() throws -> String {
   ...
}
```

5.8.2 Git

Software Git byl používán pro zálohování, verzování a sdílení zdrojových kódů aplikace.

Git je open source software, který je zdarma a používá se pro verzování projektů. [77]

Repositář aplikace vyvíjené v rámci této diplomové práce byl uložen na GitHub (`https://github.com/MartinZid/FlightRecords`). Do tohoto repositáře byl přidán vedoucí této práce jako tzv. collaborator. Díky tomu byl stále informován o změnách a mohlo docházet k revizi zdrojových kódů.

5.8.3 Carthage

Carthage je software, který se používá pro správu frameworků. Carthage tyto frameworky stáhne a připraví pro vložení do projektu, dále je také možné pomocí Carthage frameworky aktualizovat na novější verze. [78]

Seznam použitých frameworků je uložen v souboru Cartfile, kde je možné i uvést požadovanou verzi frameworku. Pro stažení a připravení frameworků se používá příkaz *carthage update*. Poté, co jsou frameworky připraveny, je nutné je vložit do Xcode projektu. [78]

Pomocí Carthage byly spravovány frameworky:

- ReactiveCocoa, který obsahuje i ReactiveSwift a Result. Tento framework je licencován pod licencí MIT a je k nalezení na adrese: `https://github.com/ReactiveCocoa/ReactiveCocoa/`.
- Realm k nalezení na adrese: `https://realm.io/docs/swift/latest/`.
- Toast-Swift, který je také licencován pod MIT a je k nalezení na adrese: `https://github.com/scalessec/Toast-Swift`.

5.8.4 Realm Studio

Realm Studio je vývojářský nástroj vytvořený pro správu Realm databází. V Realm Studiu je možné se připojit na lokální Realm, spravovat jakoukoliv instanci objektového serveru Realm nebo se připojit na Realm Cloud. Realm Studio je dostupné pro Mac, Linux i Windows. [79]

Realm Studio umožňuje upravovat záznamy v databázi, vytvářet a spravovat uživatele, dotazovat se nad záznamy, zobrazovat logy a další. [79]

Při vývoje aplikace bylo Realm Studio použito pro testování, ladění a správu databáze.

Kapitola 6

Testování

Tato kapitola se zabývá jednotkovými (unit) testy, které byly tvořeny v průběhu tvorby aplikace, dále také heuristickou analýzou, uživatelskými testy a nakonec také úpravami, které byly po dokončení testování provedeny.

6.1 Unit testy

Unit testy, neboli jednotkové testy, jsou zaměřeny na části (jednotky) kódu. Těmito jednotkami jsou v objektově orientovaném programování např. funkce nebo třídy. [80]

Pro tvorbu unit testů ve vytvářené aplikaci byl použit framework XCTest, který je vytvořen společností Apple a má předpřipravené rozhraní v Xcode. [81]

Všechny testující třídy jsou podtřídou $XCTestCase$. $XCTestCase$ je základní třídou všech testovacích případů. Tato třída obsahuje metody $setUp$ – umožňující nastavit výchozí stav před testováním, $tearDown$ – umožňující „uklizení“ po skončení testů. Dále tato třída také obsahuje funkce pro vyhodnocení jednotlivých testů. Mezi tyto funkce patří: $XCTAssertEqual$ potvrzující, že dvě vstupní hodnoty jsou stejné; $expectation$ jenž je spojováno s $waitForExpectations$, které čeká danou dobu na to, zda je $expectation$ vyplněno ($fulfill$); a další. [82]

Funkce definující testy jsou v testovací třídě označeny klíčovým slovem $test$, které je na začatku jejich názvu. Takto definované funkce jsou automaticky spouštěny a vyhodnocovány. Díky klíčovému slovu $test$ je možné definovat i pomocné funkce. [82]

Pro třídy, které testují funkce pracující s databází Realm, byla vytvořena testovací třída $XCTestCase$. Tato třída se starala o konfiguraci instance Realmu, která byla pouze v paměti a tudíž testy probíhali stále stejně a neovlivňovali klasickou Realm databázi.

Příklad 6.1 zobrazuje obě dříve zmíněné funkce – $setUp$ vytvářející $ViewModel$, který bude podléhat testům a $tearDown$, která $ViewModel$ znovu dealokuje.

```
Test Suite 'All tests' passed at 2018-04-02 19:41:07.337.
      Executed 87 tests, with 0 failures (0 unexpected) in 1.302 (1.595) seconds
```

Obrázek 6.1: Výsledky testů

Následně jsou na ukázce zobrazeny dva testy.

První z nich testuje, zda jsou *Observeři* na *Signalu* informování o tom, že data z Realmu jsou připravena. U tohoto testu je použita dvojice *expectation* a *waitForExpectations*. *Expectation* je vyplněno právě pokud je na *Signal* odeslána hodnota indikující dokončení přípravy Realmu.

Druhý test také pracuje s instancí Realmu. Zde je ovšem klasická instance nahrazena instancí s konfigurací *in memory*, protože dochází k zápisu do Realmu. Testováno je zde pomocí *XCTAssertEqual*, zda funkce *numberOfRecordsInSection* správně reflektuje počet záznamů v databázi Realmu.

Výsledek testů je zobrazen na obrázku 6.1.

6.2 Heuristická analýza

Heuristická analýza byla provedena znovu poté, co byla aplikace implementována do stavu tzv. beta verze aplikace. I tato heuristická analýze byla prováděna podle pravidel Jakoba Nielsena a nyní již bylo zohledněno všech deset pravidel. Všechny nalezené chyby, stejně jako u předchozí heuristické analýzy, byly ohodnoceny na stupnici 1–5, kde číslo 1 identifikuje pouze drobný problém a číslo 5 velice závažný problém. U všech nalezených problémů je také uvedeno řešení, pomocí kterého byly chyby opraveny. Nakonec tabulka 6.1 zobrazuje u každého pravidla, zda bylo porušeno a jaká byla maximální závažnost porušení tohoto pravidla.

1. Viditelnost stavu systému –
   - Uživatel není explicitně informován o tom, že jsou záznamy filtrovány.
     - Závažnost problému – 4.
     - Řešení – přidání informující záložky, která umožňuje i zrušení filtrování.
   - Pokud neexistují žádné záznamy letů (ze simulátoru) nebo letadel, uživateli je zobrazena pouze prázdná tabulka.
     - Závažnost problému – 3.
     - Řešení – přidání zprávy, která bude informovat uživatele, že žádné záznamy (letů, ze simulátoru a letadel) neexistují.
   - Pokud parametrům filtrování neodpovídají žádné záznamy, je uživateli zobrazena prázdná tabulka.

Listing 6.1: Ukázka unit testů

```
@testable import FlightRecords
...
class RecordsViewModelTests: TestCaseBase {
    var viewModelUnderTest: RecordsViewModel!

    override func setUp() {
        super.setUp()
        viewModelUnderTest = RecordsViewModel()
    }

    override func tearDown() {
        viewModelUnderTest = nil
        super.tearDown()
    }

    func testRealmSetUpInformingOberversOnComplete() {
        let promise = expectation(description:
            "observers recieved notification")
        viewModelUnderTest.collectionChangedSignal
            .observeValues { changes in
            promise.fulfill()
        }
        waitForExpectations(timeout: 5, handler: nil)
    }

    func testNumberOfRecords() {
        viewModelUnderTest.realm = setUpRealm()
        try! viewModelUnderTest.realm.write {
            viewModelUnderTest.realm.add(Record())
            viewModelUnderTest.realm.add(Record())
        }
        viewModelUnderTest.realmInitCompleted()
        let numberOfRecords =
            viewModelUnderTest.numberOfRecordsInSection()
        XCTAssertEqual(numberOfRecords, 2,
            "Number of records is wrong.")
    }
    ...
}
```

  - Závažnost problému – 2.
  - Řešení – přidání zprávy informující uživatele o tom, že parametrům filtrování neodpovídají žádné záznamy.
- Uživatel není informován o úspěšném přidání záznamu/letadla.
  - Závažnost – 2.
  - Řešení – po přidání záznamu/letadla bude uživateli zobrazena hláška o úspěšném přidání.
- Zobrazení náhledu reportu a generování PDF může trvat delší dobu.
  - Závažnost – 3.
  - Řešení – uživatel je informován o tom, že aplikace pracuje pomocí tzv. spinneru.

2. Spojení systému a reálného světa –
   - Název „Limity“ je příliš obecný.
     - Závažnost – 1.
     - Řešení – přejmenování „Limity“ na „Limity nalétaných hodin“.
   - Popis „Typ“ při přidávání/upravě záznamu ze simulátoru nemusí jasně specifikovat, že se má jednat o typ simulátoru (zařízení).
     - Závažnost – 2.
     - Řešení – přejmenování popisu „Typ“ na „Typ zařízení“.
3. Uživatelská kontrola a volnost –
   - Smazání záznamu nebo letadla je definitivní.
     - Závažnost problému – 4.
     - Řešení – podpora funkce vrácení zpět (tzv. undo).
4. Konzistence a standardizace –
   - Nebyl nalezen problém.
5. Předcházení chyb –
   - Uživatel může omylem smazat záznam nebo letadlo.
     - Závažnost – 4.
     - Řešení – zobrazení dialogového okna s potvrzením uživatelovi akce.
6. Rozpoznání místo vzpomínání –
   - Nebyl nalezen problém.

7. Flexibilita a efektivita použití –

    - Nebyl nalezen problém.

8. Estetika a minimalismus –

    - Při nastavování filtrování se uživateli zbytečně zobrazují položky „Typ letadla“ a „Letadlo“ pokud uživatel vyhledává pouze záznamy ze simulátoru.
        - Závažnost – 3.
        - Řešení – skrývání zbytečných položek při specifickém nastavení filtrování.

9. Pomoci uživatelům rozpoznat, pochopit a vzpamatovat se z chyb –

    - Uživatel není informován o tom, že se nepodařilo přihlásit pomocí jeho iCloud účtu.
        - Závažnost – 4.
        - Řešení – zobrazení chybové hlášky informující uživatele o chybě a poskytující informace dostatečné k tomu, aby se uživatel z chyby vzpamatoval.

10. Nápověda a dokumentace –

    - Nebyl nalezen problém při zohlednění cílové skupiny uživatelů.

Bod	Rozpor	Max. závažnost
1	✓	4
2	✓	2
3	✓	4
4	✗	-
5	✓	4
6	✗	-
7	✗	-
8	✓	3
9	✓	4
10	✗	-

Tabulka 6.1: Závažnosti prorušení pravidel

6.3 Uživatelské testování

Uživatelské testování proběhlo ihned po heuristické analýze. Testováni byli celkem čtyři uživatelé, kteří byly rozděleni na dvě skupiny – ti, co mají znalosti v oboru letectví a ostatní. U všech uživatelů bylo podmínkou používání operačního systému iOS po dobu alespoň dvou let. Všichni uživatelé dostaly stejné otázky před začátkem testování, stejné scénáře a stejný závěrečný dotazník. Skupině uživatelů, která neměla znalosti v leteckém oboru, byly nejprve vysvětleny základní informace nutné k plynulému absolvování testů.

Všechny testy byly nahrávány z důvodu možné zpětné analýzy nedostatků aplikace. Všechny nahrávky testů společně s vyplněnými dotazníky jsou k nalezení v příloze této diplomové práce.

Před samotným testováním byli uživatelé seznámeni s testovacími scénáři (z důvodu vysvětlení nejasností) a odpovídali na několik otázek. Dále byli vyzváni ke komentování svých myšlenek v průběhu testování a také byli ujištěni o tom, že pokud cokoliv nepůjde, tak to není jejich chyba, ale chyba aplikace.

6.3.1 Základní otázky na testované uživatele

1. Jak dlouho používáte iOS?
2. Jaké zařízeni od Applu máte?
3. Jste právě teď přihlášeni do účtu iCloud?
4. Je toto Vaše první uživatelské testování?

6.3.2 Scénáře

Testovací scénáře byly lehce upraveny a rozšířeny oproti scénářům použitým na otestování návrhu uživatelského rozhraní.

1. Právě jste dokončil(a) svůj první let z Brna do Prahy. Z Brna jste odlétal(a) v 7:00 a do Prahy jste přiletěl(a) v 8:45, neměl(a) jste žádné mezipřistání. Celou doby jste zastával(a) funkci vedoucího pilota a letěl(a) jste letadlem Boeing. Zaznamenejte svůj let.
2. Zjistil(a) jste, že jste u minulého záznamu chybně zaznamenal(a) čas. Upravte tedy čas příletu do Prahy na 7:45.
3. Chcete si přidat další záznam, tentokrát z výcvikového zařízení. Zadejte dnešní datum, čas na zařízení dvě hodiny.
4. Zadejte do aplikace další dva záznamy. Vyplňované hodnoty jsou na Vás, pouze datum specifikujte na minulý týden.
5. Pokuste se vyfiltrovat záznamy tak, aby se Vám zobrazovaly pouze záznamy letů z minulého týdne.

6. Jeden ze záznamů z minulého kroku smažte.
7. Vyplňte si osobní informace (nemusí být pravdivé).
8. U letadla Boeing z prvního kroku chcete změnit model. Proveďte tuto úpravu.
9. Za poslední tři týdny jste absolvoval(a) mnoho letů, zkontrolujte, zda jste nepřekročil(a) jeden z limitů.
10. Zjistěte kolik jste celkově nalétal(a) s Vaším letadlem Boeing.
11. Dnes Vám vydali zdravotní certifikát třídy jedna (Class 1). Vložte ho do aplikace.
12. Vytvořte report ze všech záznamů letů i záznamů z výcvikového zařízení za posledních čtrnáct dní a odešlete si tento report na email.

6.3.3 Závěrečný dotazník

Jako závěrečný dotazník byl zvolen SUS – „System Usability Scale“, který se používá pro měření uživatelského vnímání použitelnosti produktu. SUS obsahuje deset otázek:

1. „rád bych používal systém opakovaně,
2. systém je zbytečně složitý,
3. systém se snadno používá,
4. potřeboval bych pomoc od člověka z technické podpory, abych mohl systém používat,
5. různé funkce systému jsou dobře zakomponované,
6. systém je nekonzistentní,
7. myslím si, že se většina lidí se se systémem rychle naučí,
8. systém se těžko ovládá,
9. jsem si jistý(á) při používání tohoto systému,
10. potřebuji se naučit mnoho věcí, než začnu používat tento systém“ (překlad vlastní).

Ke každé otázce je odpověď s výběrem z pěti možností. Odpovědi jsou ohodnoceny 1–5, kde 1 představuje silný nesouhlas a 5 silný souhlas. [83]

	Uživatel 1	Uživatel 2	Uživatel 3	Uživatel 4
Úloha 1	2	2	1	2
Úloha 2	1	1	1	1
Úloha 3	1	1	1	1
Úloha 4	1	2	3	1
Úloha 5	1	2	1	1
Úloha 6	1	1	3	1
Úloha 7	1	1	1	1
Úloha 8	1	4	1	1
Úloha 9	1	1	1	1
Úloha 10	3	2	3	3
Úloha 11	1	1	1	1
Úloha 12	1	1	2	3

Tabulka 6.2: Hodnocení splnění jednotlivých scénářů

6.3.4 Výsledky testů

Tabulka 6.2 zobrazuje hodnocení splnění scénářů u jednotlivých uživatelů. Hodnocení bylo znovu v rozsahu 1 až 5, kde 1 představuje bezproblémové splnění a 5 nedokončení scénáře. Z tabulky je možné vidět, že všichni uživatelé nějakým způsobem splnily všechny zadané scénáře (nejhorší hodnocení je 4 a to se objevuje pouze jednou). Je také možné si všimnout, že pokud některé scénáře obsahovaly podobné kroky, jako scénáře již absolvované (např. filtrování nebo vkládání záznamu), uživatelé se s aplikací již seznámili a naučili a s dalšími obdobnými úlohami již neměli problém.

Při testování nebyly objeveny žádné kritické problémy, však díky pozorování a nahrávání uživatelů bylo objeveno několik nedostatků, které byly následně opraveny. Zde je jejich seznam.

- Při přidávání nebo úpravě záznamu letů již nemizí určité řádky tabulky (opraveno již po prvním testování).
- Certifikát, který není pojmenovaný se nyní v tabulce označuje jako „Nepojmenovaný“. Dříve nebyl označený nijak.
- U certifikátů s dlouhým názvem již není zobrazen celý název, ale pouze určitá část.
- U záložek, které obsahují *UISplitViewController* již funguje vrácení se na hlavní stránku záložky, při dvojím poklepání na danou záložku.
- Zaměření textového pole je nyní možné při kliknutí kamkoliv na řádek s tímto textovým polem.

- Řádky statických tabulek jsou nyní zvýrazňovány pouze pokud obsahují nějakou akci.
- Akce „Zrušit filtrování“ je potvrzována dialogovým oknem.
- U filtrování je nyní možné zrušit nastavené letadlo.
- Při odesílaní reportu na email je nyní uživatel informován pokud nemá nastaveny emailový klient.

Kapitola 7

Zhodnocení

Tato kapitola se zabývá zhodnocením postupů FRP – jejich výhodami a nevýhodami, a dále také zhodnocením časové a implementační náročnosti při vývoji aplikace pomocí FRP a architektury MVVM v porovnání se standardním přístupem architektury MCV.

7.1 Zhodnocení postupů FRP v aplikaci

Postupy funkcionálně reaktivní programování velice dobře vyhovují požadavkům, které jsou kladeny na mobilní aplikace – prezentování uživatelského rozhraní a neustálé reagování na změny, které mohou přijít jak od uživatele, tak i např. ze serveru. Díky FRP je možné všechny tyto změny pozorovat ve formě nezávislých datových toků a podle nich následně upravovat uživatelské rozhraní a to bez nutnosti složitého uchovávání stavů aplikace.

ReactiveCocoa se prokázala jako velice kvalitní framework pro tvorbu reaktivní iOS aplikace. Nejen, že tento framework poskytuje řadu základní reaktivní prvků, on také přidává komponentám uživatelského rozhraní reaktivní rozšíření, čímž umožňuje vytvořit plně reaktivní aplikaci.

Za zmínku zde stojí i Realm, který se dal velice jednoduše zkombinovat s frameworkem Reactive Cocoa. Kde pozorované změny např. na Realm kolekcích byly transformovány na *Signal*, na který následně reagovalo uživatelské rozhraní. Tímto postupem bylo dosaženo např. toho, že aplikace automaticky reagovala na každou změnu v Realmu, bez nutnosti jakékoliv uživatelovi iniciativy o znovu načtení dat.

Postupy FRP tedy přináší mnoho výhod, které ulehčují tvorbu mobilních aplikací. Je zde však nutné zmínit i pár nevýhod tohoto přístupu. Prvním drobným nedostatkem je to, že postupy FRP není možné zavést v rámci standardních Apple knihoven a je nutné použít neoficiální frameworky (tedy další knihovny/frameworky navíc). Další nevýhodou je to, že je nutné se seznámit se zcela novým programovacím stylem a tedy, i přizpůsobit to, jakým způsobem přemýšlíme nad problémy. Tato nevýhoda se projevuje hlavně na začátku, při

přechodu od klasického tzv. imperativního stylu programovaní k FRP, a je později zcela zastíněna tím, že dochází k tvorbě nezávislých komponent, které jsou schopné reagovat na danou změnu pomocí velice přehledného a elegantního kódu.

7.2 Zhodnocení časové a implementační náročnosti MVVM a FRP oproti MVC

Stejně jako postupy FRP, tak i architektura MVVM přinesla mnoho prospěšných aspektů. Tato architektura přidává novou vrstvu, která je zcela nezávislá na *View*. Tento fakt, může evokovat zbytečné zkomplikování struktury aplikace, však v tomto případě tomu tak jistě není. Nová vrstva *ViewModel* je nejčastěji použita jedna ku jedné se třídami *ViewController*ů a jedná se hlavně o oddělení logiky – té, co se zabývá *View* a ostatní. Díky tomuto rozdělení obsahují *ViewControllery* méně kódu a jsou daleko přehlednější.

Tedy, i když je ke každému *ViewControlleru* vytvářen pokaždé soubor *ViewModelu* navíc, celkově je aplikace lépe upravitelná, rozšiřitelná a obsahuje komponenty, které splňují tzv. „single responsibility principle“ a jsou často znovupoužitelné.

Architektura MVVM skvěle funguje ve spojení s funkcionálně reaktivním programováním, které se dokáže postarat o jednoduché spojení vrstev *View* a *ViewModelu*.

Jak je již zmíněno dříve, FRP může být časově náročnější hlavně zpočátku při přechodu od klasického imperativního stylu. Toto je později zastíněno všemi výhodami, které FRP přináší (mezi něž patří i zjednodušení a urychlení implementace).

Klasické postupy společně s architekturou MVC se tedy hodí pouze pro velice malé a jednoduché aplikace. Jakmile začne být aplikace lehce komplexnější, architektura MVVM ve spojení s FRP se, jak z pohledu časové tak i implementační náročností, vyplatí.

Závěr

Literatura

[1] Mobile: Native Apps, Web Apps, and Hybrid Apps. *Nielsen Norman Group* [online]. United States of America: Nielsen Norman Group, © 1998-2017, [cit. 2017-09-05]. Dostupné z: https://www.nngroup.com/articles/mobile-native-apps/

[2] About Objective-C. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2014, [cit. 2017-09-05]. Dostupné z: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[3] Swift. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-05]. Dostupné z: https://developer.apple.com/swift/

[4] Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options. *Saleforce Developers* [online]. Suite 300, San Francisco, CA 94105, United States: Salesforce.com, inc., © 2000-2017, [cit. 2017-09-05]. Dostupné z: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

[5] Apache Cordova. *Apache Cordova* [online]. Forest Hill, Maryland, United States: The Apache Software Foundation, © 2015, [cit. 2017-09-05]. Dostupné z: https://cordova.apache.org/

[6] Should You Build a Hybrid Mobile App? *UpWork* [online]. Mountain View, CA, US.: Upwork Global Inc., © 2015 - 2017, [cit. 2017-09-05]. Dostupné z: https://www.upwork.com/hiring/mobile/should-you-build-a-hybrid-mobile-app/

[7] MVC Architecture. *MDN web docs* [online]. Mountain View, California, United States: Mozilla and individual contributors, © 2005-2017, [cit. 2017-08-29]. Dostupné z: https://developer.mozilla.org/cs/

[8] MVC Architecture. *Developer Chrome* [online]. Silicon Valley: Google, © 2017, [cit. 2017-08-29]. Dostupné z: https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture

[9] Orlov, B.: IOS Architecture Patterns. *Medium* [online], 2015, [cit. 2017-08-29]. Dostupné z: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52

[10] Model-View-Controller. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2015, [cit. 2017-08-29]. Dostupné z: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[11] Orlov, B.: Realistic Cocoa MVC. In: *Medium* [online], 2015, [cit. 2017-08-29]. Dostupné z: https://cdn-images-1.medium.com/max/800/1*PkWjDU0jqGJOB972cMsrnA.png

[12] The MVVM Pattern. *Microsoft Developer Network* [online]. Washington, U.S.: Microsoft, © 2017, [cit. 2017-08-29]. Dostupné z: https://msdn.microsoft.com/en-us/library/hh848246.aspx

[13] Morrison, J.; Schmidt, M.: IOS Design Patterns: MVC and MVVM. *CapTech*, 2014, [cit. 2017-08-29]. Dostupné z: https://www.captechconsulting.com/blogs/ios-design-patterns-mvc-and-mvvm

[14] Orlov, B.: MVVM. In: *Medium* [online], 2015, [cit. 2017-08-30]. Dostupné z: https://cdn-images-1.medium.com/max/800/1*uhPpTHYzTmHGrAZy8hiM7w.png

[15] Architecting iOS Apps with VIPER. *Objc* [online]. Berlin: Objc.io, 2013, [cit. 2017-08-30]. Dostupné z: https://www.objc.io/issues/13-architecture/viper/

[16] Orlov, B.: VIPER. In: *Medium* [online], 2015, [cit. 2017-08-30]. Dostupné z: https://cdn-images-1.medium.com/max/800/1*0pN3BNTXfwKbf08lhwutag.png

[17] Bello, A.: IOS User Interfaces: Storyboards vs. NIBs vs. Custom Code. *Toptal Developers* [online]. Toptal, © 2010 - 2018, [cit. 2018-02-18]. Dostupné z: https://www.toptal.com/ios/ios-user-interfaces-storyboards-vs-nibs-vs-custom-code

[18] Storyboard. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-18]. Dostupné z: https://developer.apple.com/library/content/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

[19] Veselý, D.: Programování uživatelského rozhraní na iOS v Ackee (Storyboards vs. Xib vs. kód). *Ackee* [online]. Praha: Ackee s. r. o., © 2018, [cit. 2018-02-18]. Dostupné z: https://www.ackee.cz/blog/programovani-uzivatelskeho-rozhrani-na-ios-v-ackee-storyboards-vs-xib-vs-kod/

[20] Nayebi, F.: *Swift 3 Functional Programming*. Packt Publishing, 2016, ISBN 9781785881619, 200–201 s. Dostupné z: https://books.google.cz/books?id=LP9vDQAAQBAJ

[21] Staltz, A.: The introduction to Reactive Programming you've been missing. In *GitHubGist*, California, US.: GitHub, 2014, [cit. 2017-09-01]. Dostupné z: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

[22] Blackheath, S.; Jones, A.: *Functional reactive programming*. United States: Manning Publications, 2016, ISBN 978-163-3430-105.

[23] ReactiveSwift. *GitHub* [online]. California, US.: GitHub Inc., © 2017, [cit. 2017-09-01]. Dostupné z: https://github.com/ReactiveCocoa/ReactiveSwift

[24] ReactiveCocoa. *GitHub* [online]. California, US.: GitHub Inc., © 2017, [cit. 2017-09-01]. Dostupné z: https://github.com/ReactiveCocoa/ReactiveCocoa

[25] RxSwift: ReactiveX for Swift. *GitHub* [online]. California, US.: GitHub Inc., © 2017, [cit. 2017-09-04]. Dostupné z: https://github.com/ReactiveX/RxSwift

[26] Reactive Extensions. *GitHub* [online]. California, US.: GitHub Inc., © 2017, [cit. 2017-09-04]. Dostupné z: https://github.com/Reactive-Extensions/Rx.NET

[27] ReactiveX. *ReactiveX* [online], [cit. 2017-09-04]. Dostupné z: http://reactivex.io/intro.html

[28] ReactiveX. *ReactiveX* [online], [cit. 2017-09-04]. Dostupné z: http://reactivex.io/

[29] What Is Core Data? *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-16]. Dostupné z: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html

[30] Persistent Store Types and Behaviors. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-16]. Dostupné z: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/PersistentStoreFeatures.html

[31] Creating a Managed Object Model. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-16]. Dostupné z: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/KeyConcepts.html

[32] iCloud. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-16]. Dostupné z: https://developer.apple.com/icloud/

[33] CloudKit. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-16]. Dostupné z: https://developer.apple.com/documentation/cloudkit

[34] CloudKit. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-16]. Dostupné z: https://developer.apple.com/icloud/cloudkit/

[35] Capabilities Available to Developers. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-16]. Dostupné z: https://developer.apple.com/support/app-capabilities/

[36] Purchase and Activation. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-16]. Dostupné z: https://developer.apple.com/support/purchase-activation/

[37] The Realm Mobile Platform. *Realm* [online]. Realm, © 2014-2017, [cit. 2017-09-16]. Dostupné z: https://realm.io/docs/get-started/overview/

[38] European Aviation Safety Agency (EASA). *European Union* [online], [cit. 2017-09-14]. Dostupné z: https://europa.eu/european-union/about-eu/agencies/easa_en

[39] Regulations. *European Aviation Safety Agency* [online], © 2017, [cit. 2017-09-14]. Dostupné z: https://www.easa.europa.eu/regulations

[40] PART-FCL. London, United Kingdom: EASA, June 2016. Dostupné z: https://www.easa.europa.eu/system/files/dfu/Part-FCL.pdf

[41] Official Journal of the European Union. London, United Kingdom: EASA, 2014. Dostupné z: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2014:028:0017:0029:EN:PDF

[42] EU Part-MED medical certificates for EU licences. *Civil Aviation Authority* [online]. London, United Kingdom: Civil Aviation Authority, © 2015, [cit. 2017-09-14]. Dostupné z: https://www.caa.co.uk/General-aviation/Pilot-licences/EASA-requirements/Medical/EASA-Part-MED-requirements/

[43] LogTen Pro X. *ITunes* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-08]. Dostupné z: https://itunes.apple.com/us/app/logten-pro-x/id837274884?mt=8

[44] Logbook Pro Aviation Flight Log for Pilots. *ITunes* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-08]. Dostupné z: https://itunes.apple.com/us/app/logbook-pro-aviation-flight-log-for-pilots/id410773111?mt=8

[45] Safelog Pilot Logbook. *ITunes* [online]. California, U.S.: Apple Inc., © 2017, [cit. 2017-09-08]. Dostupné z: https://itunes.apple.com/us/app/safelog-pilot-logbook/id411409786

[46] FlyLogio - Pilot Logbook. *Google Play* [online]. Silicon Valley: Google, ©2017, [cit. 2017-09-08]. Dostupné z: https://play.google.com/store/apps/details?id=com.flylogio&hl=en

[47] Smart Logbook. *Google Play* [online]. Silicon Valley: Google, ©2017, [cit. 2017-09-08]. Dostupné z: https://play.google.com/store/apps/details?id=com.kviation.logbook&hl=en

[48] Logbook Pro Desktop. *NC Software* [online]. Richmond, Virginia: NC Software, Inc., © 2017, [cit. 2017-09-08]. Dostupné z: http://www.nc-software.com/Logbook-Pro-Flight-Log-Software-for-Pilots

[49] Wireframing and Prototyping. *Nielsen Norman Group* [online]. California, United States: Nielsen Norman Group, © 1998-2017, [cit. 2017-09-26]. Dostupné z: https://www.nngroup.com/courses/wireframing-and-prototyping/

[50] What is wireframing? *Experienceux* [online]. Bournemouth: Experience UX, © 2017, [cit. 2017-09-26]. Dostupné z: http://www.experienceux.co.uk/faqs/what-is-wireframing/

[51] Nielsen, J.: How to Conduct a Heuristic Evaluation. In: *Nielsen Norman Group* [online]. California, United States: Nielsen Norman Group, © 1998-2017, [cit. 2017-09-26]. Dostupné z: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/

[52] Nielsen, J.: 10 Usability Heuristics for User Interface Design. In: *Nielsen Norman Group* [online]. California, United States: Nielsen Norman Group, © 1998-2017, [cit. 2017-09-26]. Dostupné z: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/

[53] Nielsen, J.: Curve showing the proportion of usability problems in an interface found by heuristic evaluation using various numbers of evaluators. In: *Nielsen Norman Group* [online], © 1998-2017, [cit. 2017-09-26].

Dostupné z: https://media.nngroup.com/media/editor/2012/10/30/heur_eval_finding_curve.gif

[54] Nielsen, J.: How Many Test Users in a Usability Study? In: *Nielsen Norman Group* [online]. California, United States: Nielsen Norman Group, © 1998-2017, [cit. 2017-09-27]. Dostupné z: https://www.nngroup.com/articles/how-many-test-users/

[55] Welcome to Swift.org. *Swift* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-01]. Dostupné z: https://swift.org/

[56] The Basics. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-16]. Dostupné z: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

[57] Extensions. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-16]. Dostupné z: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/Extensions.html

[58] Statements. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-16]. Dostupné z: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/Statements.html

[59] Protocols. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-17]. Dostupné z: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html

[60] Automatic Reference Counting. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-17]. Dostupné z: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/AutomaticReferenceCounting.html

[61] UIKit. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-22]. Dostupné z: https://developer.apple.com/documentation/uikit

[62] Foundation. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-22]. Dostupné z: https://developer.apple.com/documentation/foundation

[63] Signal. *ReactiveSwift Docs* [online]. ReactiveCocoa, © 2018, [cit. 2018-02-20]. Dostupné z: http://reactivecocoa.io/reactiveswift/docs/latest/Classes/Signal.html

[64] SignalProducer. *ReactiveSwift Docs* [online]. ReactiveCocoa, © 2018, [cit. 2018-02-20]. Dostupné z: http://reactivecocoa.io/reactiveswift/docs/latest/Structs/SignalProducer.html

[65] MutableProperty. *ReactiveSwift Docs* [online]. ReactiveCocoa, © 2018, [cit. 2018-02-20]. Dostupné z: http://reactivecocoa.io/reactiveswift/docs/latest/Classes/MutableProperty.html

[66] Basic Operators. *GitHub* [online]. California, US.: GitHub Inc., © 2018, [cit. 2018-02-20]. Dostupné z: https://github.com/ReactiveCocoa/ReactiveSwift/blob/master/Documentation/BasicOperators.md

[67] The Realm Mobile Platform. *Realm* [online]. Realm, © 2014-2018, [cit. 2018-02-21]. Dostupné z: https://realm.io/docs/swift/latest/

[68] NSPredicate. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-21]. Dostupné z: https://developer.apple.com/documentation/foundation/nspredicate

[69] Kreutzberger, S.: UX: iOS Onboarding without Signup Screens. *Medium* [online]. Medium. Dostupné z: https://medium.com/@skreutzb/ios-onboarding-without-signup-screens-cb7a76d01d6e

[70] Foundation. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-22]. Dostupné z: https://developer.apple.com/documentation/cloudkit/ckcontainer

[71] Realm Object Server Documentation. *Realm* [online]. Realm, © 2014-2018, [cit. 2018-02-22]. Dostupné z: https://realm.io/docs/realm-object-server/latest/

[72] Interface Builder Built-In. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-03-01]. Dostupné z: https://developer.apple.com/xcode/interface-builder/

[73] Interface Builder Built-In. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-03-01]. Dostupné z: https://developer.apple.com/library/content/featuredarticles/ViewControllerPGforiPhoneOS/TheAdaptiveModel.html

[74] UISplitViewController. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-03-03]. Dostupné z: https://developer.apple.com/documentation/uikit/uisplitviewcontroller

[75] Build Apps for the World. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-03-01]. Dostupné z: https://developer.apple.com/xcode/ide/

[76] Xcode. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-28]. Dostupné z: https://developer.apple.com/xcode/ide/

[77] Git. *Git* [online]., [cit. 2018-02-28]. Dostupné z: https://git-scm.com/

[78] Carthage. *GitHub* [online]. California, US.: GitHub Inc., © 2018, [cit. 2018-02-28]. Dostupné z: https://github.com/Carthage/Carthage

[79] Download Realm Studio. *Realm* [online]. Realm, © 2014-2018, [cit. 2018-02-28. Dostupné z: https://realm.io/products/realm-studio/

[80] Unit Testing. *Software Testing Fundamentals* [online]. STF, 2018, [cit. 2018-02-22]. Dostupné z: http://softwaretestingfundamentals.com/unit-testing/

[81] Foundation. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-22]. Dostupné z: https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/01-introduction.html

[82] XCTestCase. *Apple Developer* [online]. California, U.S.: Apple Inc., © 2018, [cit. 2018-02-22]. Dostupné z: https://developer.apple.com/documentation/xctest/xctestcase

[83] Measuring Usability With The System Usability Scale (SUS). *MeasuringU* [online]. Measuring Usability LLC, © 2004-2017, [cit. 2018-02-12]. Dostupné z: https://measuringu.com/sus/

Příloha **A**

Seznam použitých zkratek

FRP Funkcionálně Reaktivní Programování

MVC Model View Controller

MVVM Model View ViewModel

VIPER View Interactor Presenter Entity Router

PC Personal Computer

EASA European Aviation Safety Agency

PIC Pilot-In-Command

SE Single Engine

ME Multi Engine

SPIC Student PIC

PICUS PIC Under Supervision

FSTD Flight Simulation Training Devices

FI Flight Instructor

FE Flight Examiner

LAPL Light Aircraft Pilot Licence

SPL Sailplane Pilot Licence

BPL Balloon Pilot Licence

PPL Private Pilot Licence

SUS System Usability Scale

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS