VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Jakub Lhotský

Bakalářská práce

Vedoucí práce: Ing. Pavel Dohnálek, Ph.D.

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/22/079095
vsboee220261E9

Zadání bakalářské práce

Student: **Jakub Lhotský**

Studijní program: B0613A140014 Informatika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: AstrumQ Interactive, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Dohnálek, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 07.11.2022 12:25:53

17. listopadu 2172/15
708 00 Ostrava-Poruba
Česká republika

spojovatelka: +420 597 321 111
epodatelna: epodatelna@vsb.cz
ID datové schránky: d3kj88v

IČ: 61989100
DIČ: CZ61989100

email: studijni.fei@vsb.cz
www.fei.vsb.cz

Abstrakt

Tato bakalářská práce je zaměřena na průběh mé odborné praxe, kterou jsem vykonával ve firmě AstrumQ na pozici mobilního vývojáře iOS aplikací. Mou hlavní pracovní náplní zde bylo navrhnout a implementovat kompletní řešení pro správu uživatelů v aplikaci MojeOSŽ, vyvíjenou pro Odborové Sdružení Železničářů. V hlavní části práce nejprve stručně popíši technologie, se kterými jsem v průběhu praxe pracoval, a následně se budu podrobně věnovat popisu, návrhu a implementaci jednotlivých zadaných úkolů. V závěru shrnuji mé celkové dojmy a zkušenosti získané během praxe.

Klíčová slova

iOS, Swift, mobilní vývoj

Abstract

This bachelor thesis is focused on my professional practice, which I did at AstrumQ as a mobile iOS application developer. My main job there was to design and implement a complete user management solution for the MyOSŽ app developed for the Railway Workers' Union. In the main part of the thesis, I will briefly describe the technologies I worked with during my internship, and then go into detail about the description, design, and implementation of each assigned task. Finally, I will summarize my overall impressions and experiences gained during the internship.

Keywords

iOS, Swift, mobile development

Poděkování

Rád bych zde poděkoval společnosti AstrumQ za poskytnutí příležitosti pro vykonání odborné praxe a za jejich vřelé přivítání do kolektivu. Dále bych rád poděkoval Tomáši Rozehnalovi, za jeho vedení a dohled ve firmě, Ing. Jakubovi Cięcialovi, za jeho trpělivost a vstřícnost při řešení úloh, a Ing. Radkovi Čepovi, za jeho cenné rady. V neposlední řadě bych také rád poděkoval Ing. Pavlovi Dohnálkovi, Ph.D. za pomoc a zpětnou vazbu při psaní této práce.

Obsah

Seznam použitých zkratek a symbolů

UI	–	User Interface
LLVM	–	Low Level Virtual Machine
BAAS	–	Back End As Service
SDK	–	Software Development Kit
FCM	–	Firebse Cloud Messaging
QR	–	Quick Response
HTTP	–	Hypertext Transfer Protocol

Seznam obrázků

Kapitola 1

Úvod

Tato bakalářská práce pojednává o průběhu mé odborné praxe ve firmě AstrumQ. Hlavní motivací pro absolvování praxe bylo ověření a uplatnění vědomostí, které jsem získal během studia na VŠB FEI a také získání cenných praktických zkušeností v oblasti vývoje software.

Má hlavní pracovní náplň se týkala projektu MojeOSŽ. Jedná se o aplikaci vyvíjenou pro Odborové Sdružení Železničářů, která má za cíl přinášet aktuální informace z dění OSŽ a železnic. Mimo jiné také obsahuje funkcionality jako přehled kontaktů na vedení svazu, okamžitou volbu na krizovou linku nebo možnost zasílat dotazy, náměty či připomínky bez použití e-mailu.[1] Mým úkolem bylo aplikaci rozšířit o autentizační systém určený pro členy sdružení.

V úvodu práce krátce představím firmu AstrumQ a popíšu proces mého pracovního zařazení do běhu firmy. V následující kapitole popíšu technologie a principy, které jsem během praxe použil. Hlavní část práce je pak zaměřena na všechny vykonané úkoly, které vedly k dosažení výše zmíněné funkcionality, a popíši způsob jejich implementace. V závěru nakonec zhodnotím mé dojmy z praxe a uvedu všechny nabyté zkušenosti.

1.1 AstrumQ Interactive, s.r.o.

1.1.1 O firmě

Společnost AstrumQ byla založena v roce 2012 a sídlí v Ostravě-Přívoze, a zaměřuje se převážně na vývoj mobilních a webových aplikací.

Společnost má několik projektů, kterými se může pyšnit. Zmínit lze třeba webovou a mobilní aplikaci Sportnect, která je zaměřena na sportovní aktivity a poskytování informací o sportovištích a sportovních událostech v dané oblasti. Firma se také zabývá integrací informačních systémů do průmyslových a logistických procesů. Jako příklad lze uvést aplikaci Driver od společnosti TrafinOil, která je určena pro řidiče, a slouží k prodeji čistého a sběru špinavého oleje z restaurací. Tento

projekt představuje inovativní řešení v oblasti environmentální odpovědnosti a přispívá k ochraně životního prostředí. [2]

Obrázek 1.1: Logo společnosti AstrumQ

1.1.2 Zařazení do firmy

Firmu jsem objevil skrze školní systém Katis. Před nastoupením na praxi jsem neměl žádné zkušenosti s vývojem mobilních aplikací ani s vývojem pro platformu Apple. Z tohoto důvodu mi proto firma měsíc před nastoupením na praxi poskytla notebook a online kurzy, ze kterých jsem se učil. Po nástupu mi byla zadána jednoduchá aplikace pro ověření mých znalostí a krátce na to mi byl již přiřazen první reálný úkol.

1.1.3 Nástroje používané ve firmě

- **Slack** - byl hlavním nástrojem pro interní komunikaci ve firmě. Tato platforma je určena pro spolupráci a komunikaci v týmech, a nabízí uživatelům možnosti textové a hlasové komunikace, sdílení souborů, skupiné hovory, nebo vytváření soukromých kanálů v rámci týmů.
- **Jira** - platforma sloužící k organizaci práce. Ve firmě byla práce na jednotlivých projektech organizovaná za pomocí Kanban board, kde jsou jednotlivé úkoly reprezentovány jako lístečky na virtuální nástěnce. Úkoly jsou na nástěnce rozděleny do různých fází podle toho, ve které části procesu implementace se právě nacházejí („To be done“, „In progress“, „Testing“, atd.).
- **Apiary** - nástroj pro popis a dokumentaci API.
- **Invision** - platforma sloužící ke sdílení grafických návrhů a interaktivních prototypů webových a mobilních aplikací.
- **BitBucket** - platforma pro verzování a správu kódu s využitím Gitu.

Kapitola 2

Úvod do technologií a principů

2.1 Swift

Swift je open-source programovací jazyk vyvinutý společností Apple. Poprvé byl představen v roce 2014 jako náhrada za jejich starší jazyk Objective-C a k dnešnímu datu byla vydaná již jeho pátá verze. Swift je stejně jako jeho předchůdce kompilován v LLVM kompilátoru, díky čemu jej lze kombinovat společně s Objective-C uvnitř jedné aplikace.

Mezi charakteristické vlastnosti Swiftu patří closures, což jsou ukazatele na funkce, které lze ukládat do proměnných a předávat jako parametry jiným funkcím, optionals, což je typ proměnné, který indikuje, že proměnná může obsahovat hodnotu, ale nemusí(může být null), nebo generika, což je způsob, jak vytvářet funkce nebo vlastní datové typy, které mohou pracovat s různými typy dat. [3]

2.2 XCode

XCode je integrované vývojové prostředí od společnosti Apple určené k vývoji aplikací pro platformy iOS, iPadOS, watchOS, a tvOS. Prvně bylo představeno v roce 2003 a jeho nejnovější dostupnou verzí je verze 14.3. Krom Swiftu a Objective-C také podporuje vývoj např. v jazycích C, C++, Java, Python nebo Ruby. XCode také obsahuje řadu užitečných nástrojů pro vývoj aplikací pro platformu Apple, jako je např. Interface Builder, sloužící ke konstrukci uživatelských rozhraní, nebo poskytnutí možnosti emulace vyvíjené aplikace na jakémkoliv Apple zařízení. [4]

2.3 CocoaPods

CocoaPods je manažer závislostí pro Swift a Objective-C. V současné době obsahuje přes 94 000 knihoven a využívá jej přes tři milióny aplikací. CocoaPods umožňuje developerům přidávat knihovny třetích stran do aplikací bez potřeby je manuálně stáhnout a nakonfigurovat. Všechny závislosti se

specifikují ve speciálním textovém souboru zvaném Podfile. Po zadání příkazu „pod install“ CocoaPods automaticky stáhne a přidá všechny uvedené závislosti v Podfile. [5]

2.4 SnapKit

Jedná se o knihovnu pro Swift sloužící k nastavení tzv. Auto Layout constraints. Auto layout je funkce iOS sloužící k tvorbě uživatelských rozhraní, která jsou schopná se adaptovat podle různých velikostí obrazovek nebo obsahu. SnapKit zjednodušuje proces tvorby Auto Layoutu tím, že poskytuje jednodušší a lépe čitelnou syntaxi oproti své nativní variantě. [6]

2.5 Swiftgen

Nástoj sloužící ke generování statického obsahu projektu jako jsou obrázky, lokalizované stringy, barvy apod. Zdrojové soubory, ze kterých se obsah generuje, se specifikují ve speciálním souboru. S využitím Swiftgen se kód stává přehlednějším a usnadňuje jakékoliv změny obsahu napříč aplikací. [7]

2.6 Tvorba uživatelského rozhraní

Při tvorbě uživatelských rozhraní ve frameworku UIKit mají vývojáři na výběr ze 2 způsobů - buďto tvořit UI v Interface Builderu nebo programaticky v kódu. Každý způsob přináší své vlastní výhody i nevýhody.

Při tvorbě UI přímo v kódu může být snadnější udržovat kód čitelnější a lépe jej organizovat. Programátor má plnou kontrolu nad tím, jak se uživatelské rozhraní chová a vypadá, a může používat některé funkce, které nejsou k dispozici v Interface Builderu. To se hodí zejména u rozsáhlejších projektů, které mají složitější UI nebo potřebují mít kód dobře organizovaný.

Na druhé straně, Interface Builder poskytuje vizuální náhled rozhraní a lépe se v něm mohou tvořit constraints sloužící rozložení prvků na obrazovce. Tento přístup se doporučuje u menších projektů, kde se nachází spíše statický obsah.

2.7 Google Firebase

Jedná se o multifunkční BAAS službu od Googlu. Firebase poskytuje širokou škálu nástrojů, jako například:

- **Authentication** - služba pro správu a autentizaci uživatelů, která umožňuje přihlašování pomocí e-mailu a hesla, telefonního čísla, nebo jiných služeb jako jsou Twitter, Facebook atd.
- **Cloud messaging** - služba pro zasílání push notifikací na zařízení

- **Firestore** - dokumentová no-SQL databáze
- **Test lab** - infrastruktura sloužící pro vydávání a testování aplikací
- **Firebase Cloud Functions** - funkce, které umožňují vykonávat kód na serverové straně bez nutnosti provozu vlastního serveru

Součástí Firebase je také Firebase SDK, který zprostředkovává veškerou komunikaci s Firebase serverem. [8]

2.8 Návrhový vzor Delegate

Jeden z nejvíce používaných návrhových vzorů napříč vývojem pro iOS platformu. Delegace umožňuje jedné třídě přenechat část své funkcionality třídě jiné, která si ji může upravit nebo obohatit, a stává se tak jejím delegátem. [9]

Jako příklad mějme prvek textového pole UITextView. Pokud bychom chtěli text v poli při jeho zadávání nějakým způsobem formátovat, můžeme toho docílit implementací protokolu UITextViewDelegate. Jednou z metod, kterou tento protokol obsahuje, je textViewDidChange, která je volaná pokaždé, kdy se změní obsah uvnitř textového pole. Uvnitř této metody pak můžeme zavolat metodu pro naformátování textu.

```
extension MyViewController: UITextViewDelegate {
    func textViewDidChange(_ textView: UITextView) {
        textView.text = format(textView.text)
    }
}
```

Výpis 2.1: Využití návrhového vzoru Delegate

Kapitola 3

Práce na projektu MojeOSŽ

Následující kapitola bude detailně popisovat jednotlivé úkoly a způsob jejich implementace zadané autorovi práce na projektu MojeOSŽ, a krátce uvede čtenáře do problematiky reaktivního programování.

3.1 Využití reaktivního programování a frameworku ReactiveKit

Reaktivní programování je programovací paradigma, které se často používá v rámci vývoje webových a mobilních aplikací a je postaveno na návrhovém vzoru Observer. V reaktivním programování jsou veškerá data reprezentována jako proudy událostí. Strukturám, které tyto události generují, se říka Observables. Část aplikace, která pak na tyto události reaguje, se nazývá Observer. Operátory jsou nástrojem, který umožňuje transformaci těchto toků dat, jako je například filtrace, mapování nebo spojování. Díky nim lze jednoduše manipulovat s datovými toky a aplikace se tak stává flexibilnější a snadno udržovatelnější. [10]

Pro jazyk Swift exisuje více frameworků implementující toto paradigma. Pro projekt Moje OSŽ byl zvolen framework ReactiveKit. Většina projektu je napsána s jeho využitím, proto z důvodu zachování konzistence byla snaha jej i nadále při implementaci jednotlivých úkolů zahrnout.

3.1.1 Signály

V ReactiveKitu se tokům dat říká Signály. Při inicializaci nového signálu je třeba specifikovat 2 věci - jaký typ dat bude signál produkovat, a jaký typ chyby může vrátit. Signálu, který nemůže vrátit žádnou chybu, se říká SafeSignal.

Běžné signály mohou pro své observery vyprodukovat 2 typy událostí - `next` a `completion`. V prvním případě signál pouze propaguje další hodnotu v sekvenci. V druhém případě signál značí, že žádný z jeho observerů již od něj nemá očekávat další hodnoty a signál je terminován. Completion může být ze dvou druhů - `completed`, kdy je všechna akce uvnitř signálu úspešně ukončena a `failure`, který značí, že někde došlo k chybě. [11]

```
let cities = Signal<String, Never> { observer in
    observer.receive(.next( Paris ))
    observer.receive(.next( America ))
    observer.receive(.next( Germany ))

    observer(.completed)
}

cities.observeNext { city in
  print(city) // Vypíše  Paris ,   America  ,   Germany
}
```

Výpis 3.1: Ukázka signálu, který vyprodukuje 3 názvy měst a poté je ukončen

3.1.2 Operátory

Další důležitou součástí ReactiveKitu jsou tzv. operátory. Ty slouží k transformaci jednoho či více signálů na nový. Jako příklad lze uvést operátor `filter`, který jako parametr přijímá filtrační funkci. Tento operátor pak transformuje původní signál na nový, který propustí pouze ty hodnoty, pro které filtrační funkce vrátí hodnotu true. [11]

```
let cities = Signal<String, Never> { observer in
    observer.receive(.next( Paris ))
    observer.receive(.next( America ))
    observer.receive(.next( Germany ))
}

cities.filter { $0.hasPrefix( P ) }.observeNext { city in
  print(city) // Vypíše pouze  Paris
}
```

3.2 Úprava vyhledávání kontaktů

3.2.1 Specifikace zadání

Úkolem bylo upravit způsob vyhledávání kontaktů na vedení svazu v aplikaci. Kontakty šlo již vyhledávat na základě jména, identifikačního čísla osoby (IČO) a telefonního čísla, a cílem bylo vyhledaným kontaktům přidat textový atribut, který by specifikoval, na základě jakého parametru byly vyhledány.

3.2.2 Analýza a návrh

Protože základní verze databáze Firestore nebyla schopna udat na základě jakého atributu byl záznam vyhledán, bylo tuto funkcionalitu třeba implementovat na straně klienta.

3.2.3 Implementace

Při získaní listu struktur Contact se zjišťuje, která z hodnot obsahuje filtrovaný výraz, a poté je přemapována na pomocnou strukuru FilteredContact, která obsahuje enum FilterType, který udává, o jaký typ filtrace se jedná, a jaká část atributu by v rozhraní měla být zvýrazněna. Pro přemapování struktur Contact na FilteredContact je využit signálový operátor `compactMap`, který transformuje všechny hodnoty v toku na nové, a zahodí všechny prvky s hodnotou nil.

```
.compactMap { contacts, filter in
    contacts.map { contact in
        if let filter = filter {
            if let chairman = contact.chairman, chairman.contains(filter) {
                return FilteredContact(contact: contact, filterValue: chairman,
                    highlightedText: filter, filterType: .NAME)
            }
            if let ico = contact.ico, ico.contains(filter) {
                return FilteredContact(contact: contact, filterValue: ico,
                    highlightedText: filter, filterType: .ICO)
            }
        }
        return FilteredContact(contact: contact, filterValue: nil, highlightedText
            : nil, filterType: .NOTHING)
    }
}
```

Výpis 3.2: Ukázka využití signálového operátoru compactMap

3.3 Uživatelský profil

Po dokončení mého prvního úkolu mi byla přidělena práce na rozsáhlejším celku - rozšíření aplikace o možnost vytvářet a autentizovat uživatele. Pro splnění této funkcionality bylo zapotřebí naimplementovat následující celky:

1. Registrace uživatelů
2. Přihlašování uživatelů

3. Profil
4. Nastavení profilu
5. Zapomenuté heslo

Implementace každého z celků bude následně detailně popsána.

3.3.1 Návrh uživatelského rozhraní

Při řešení jednotlivých celků byl k dispozici grafický design, který specifikoval vzhled uživatelského rozhraní. Protože se ale grafický manuál aplikace od doby vzniku návrhu lehce změnil, bylo zapotřebí výsledný vzhled rozhraní vždy trochu upravit tak, aby korespondoval s aktuálním vzhledem aplikace.

3.4 Registrace uživatelů

Specifikace zadání

Registrační formulář je složen ze dvou částí - v první části uživatel zadává své osobní údaje, v druhé pak vybírá svého zaměstnavatele, organizaci a zadává své osobní číslo. Výběr zaměstnavatele a organizace je realizován prostřednictvím nového okna, které obsahuje seznam se vstupem pro vyhledávání, ze kterého uživatel vybírá. Po stisknutí tlačítka pro registraci se zobrazí načítací indikátor. Následně se zobrazí buďto profil nově vytvořeného uživatele, nebo v případě selhání registrace dialogové okno s informací, z jakého důvodu registrace neprošla.

Analýza zadání a návrh

Pro vytvoření nového uživatele je zapotřebí provést v návaznosti na sebe dva HTTP POST požadavky - jeden na end-point **„/user/register“** a druhý na **„/user/update“**. V prvním volání jsou do těla požadavku přiloženy atributy „name“, „email“ a „password“, a je jím vytvořen nový uživatel v databázi. V druhém volání je proveden update nově vytvořeného uživatele a jako parametry jsou zde přiloženy zbývající tři atributy z registračního formuláře - osobní číslo, identifikátor zvolené organizace a telefonní číslo. Po zaslání požadavku na server je ověřeno, zda byla nalezena v externí databázi shoda se zadanými daty. Pokud ano, uživateli je v databázi nastaven příznak „isInOrganization“ na hodnotu true, čímž je registrace úspěšně dokončena a uživatel se tak stává oprávněným. Pokud se ovšem shoda nenajde, nebo registrace selže z jiného důvodu, příznak „isInOrganization“ zůstává na hodnotě false a registrace je označena jako neúspěšná. Registrace tak může selhat z těchto důvodů:

- při volání end-pointu „register“ se již v databázi vyskytuje uživatel se stejnou e-mailovou adresou nebo stejným telefonním číslem

- při volání end-pointu „update“ v externí databázi nebyla nalezena shoda s daty zadanými uživatelem

V každém z uvedených případů je zapotřebí na neúspěšnou registraci zareagovat zobrazením dialogového okna s popisem, proč registrace selhala.

Implementace

Textové vstupy ve formuláři jsou realizovány jako vlastní podtřída UIView, která obsahuje malý nadpis, textové pole a neviditelnou chybovou hlášku, kterou lze zobrazit a skrýt metodou přijímající boolean hodnotu. Tyto vstupy jsou pod sebe rozloženy za pomocí vertikálního UIStackView.

Pro validaci formátu jednotlivých vstupů je pro každý z nich vytvořen validační signál. Tento signál při stisku tlačítka pro otevření dalšího kroku vezme současný obsah daného vstupu a transformuje jej na boolean indikující, zda je jeho formát validní či nikoliv. Na tento signál je pak napojena metoda pro zobrazení chybové hlášky v případě zadání chybného formátu.

```
lazy var isEmailValidSignal: SafeSignal<Bool> = {
    _nextStep
        .with(latestFrom: email)
        .map { $1.isValidEmail() }
        .share()
    }()

...

viewModel.isEmailValidSignal
    .bind(to: emailInput) { input, isValid in
        input.toggleError(!isValid)
    }
```

Výpis 3.3: Ukázka využití frameworku ReactiveKit pro validaci vstupů

Aby bylo zajištěno, že uživatel může přejít na další krok pouze v případě, že jsou jeho zadané vstupy validní, jsou všechny validační signály spojeny do jednoho za pomocí operátoru „zip“, který má jako výstup logický součin všech hodnot ze signálů. Na základě této výsledné hodnoty je pak rozhodnuto, zda lze přejít na další krok.

```
lazy var stepOneInputsValidSignal: SafeSignal<Void> = {
    zip(
        isNameValidSignal,
        isEmailValidSignal,
        isPasswordValidSignal
```

```
        ) { isNameValid, isEmailValid, isPasswordValid in
            return name && email && password
    }
        .filter { $0 }
        .eraseType()
        .share()
}()
```

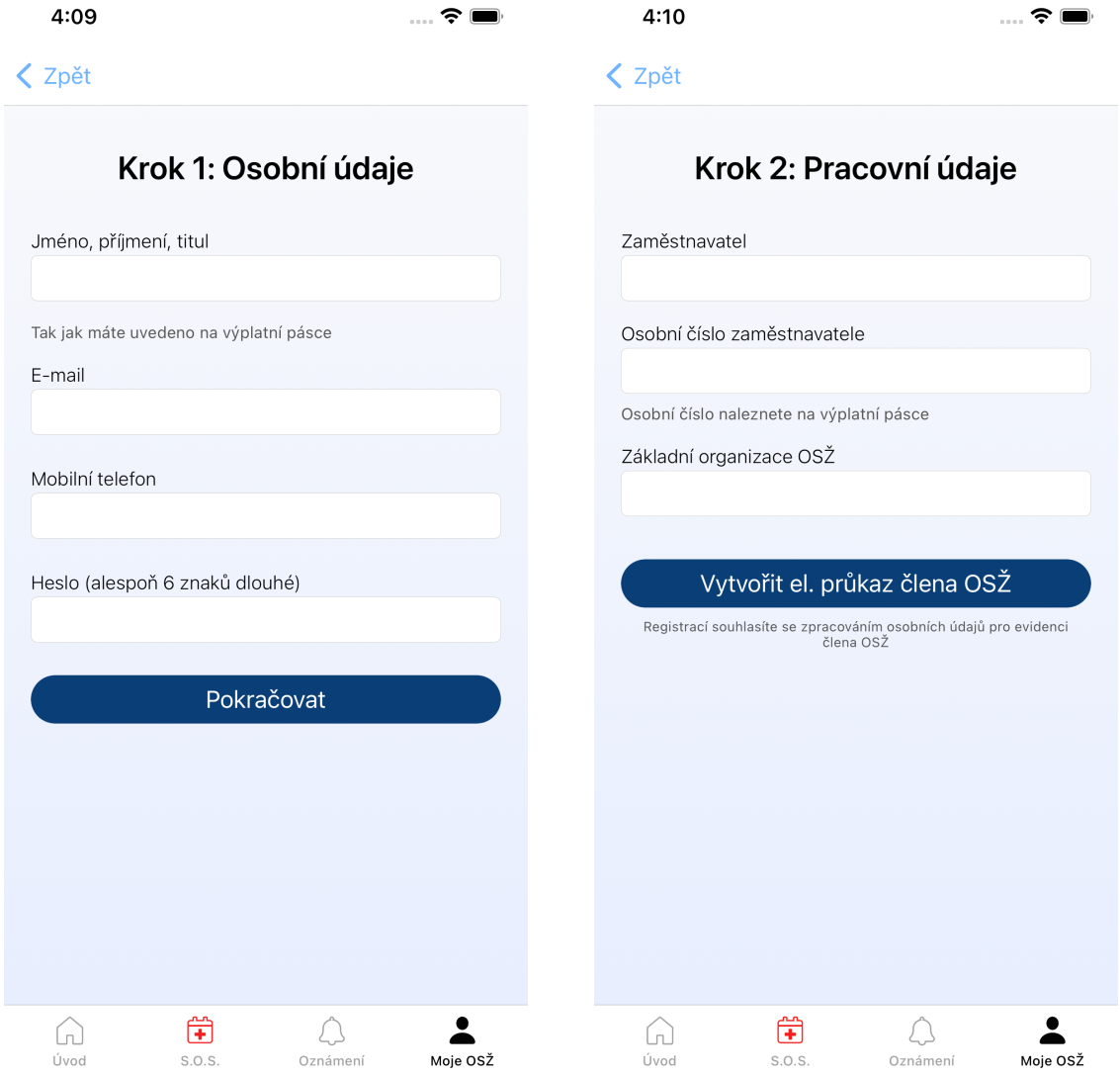
Výpis 3.4: Ukázka využití signálového operátoru „zip“

Po ověření funkčnosti uživatelského rozhraní obou kroků bylo na řadě naimplementovat hlavní funkcionalitu formuláře - vytvoření nového uživatele. Pro jakékoli operace týkající se uživatelů v aplikace byla založená mikroslužba UserService, uvnitř které jsou implementovány metody pro registraci a update uživatele. Pro vykonání HTTP požadavků tyto metody využívají frameworku Alamofire. Jedná se o jednoduchého HTTP klienta, který navíc poskytuje funkce jako např. validace odpovědí nebo řetězení požadavků. [12]. Obě metody byly implementovány jako signály, aby na ně bylo možné aplikovat signálové operátory. Tyto metody musí být provedeny v návaznosti na sebe, proto byl využit signálový operátor `flatMapLatest`, který daný signál po dokončení transformuje na kompletně nový signál.

```
Alamofire.request(URL(string: self.firestoreURL + /user/register)!, method: .post,
     parameters: params)
    .validate()
    .responseJSON { response in
        guard let data = response.data else {
            observer.receive(completion: .failure(.noDataAvailable))
            return
        }
        let responseJSON = JSON(data)

        let userId = responseJSON[ uid ].stringValue
        observer.receive(userId)
    }
}
```

Výpis 3.5: Ukázka metody pro vytvoření nového uživatele s využitím frameworku Alamofire



(a) (b)

Obrázek 3.1: Výsledný vzhled registračního formuláře

3.5 Přihlašování uživatelů

Specifikace zadání

V přihlašovacím formuláři se nachází dva vstupy pro e-mailovou adresu a heslo. Pod těmito vstupy jsou umístěny dvě tlačítka, jedno pro přihlášení a druhé pro obnovení zapomenutého hesla. Po stisknutí tlačítka pro přihlášení dojde buď k zobrazení uživatelova profilu, nebo k zobrazení chybové hlášky s důvodem, proč přihlášení selhalo.

Analýza zadání a návrh

Pro projekt Moje OSŽ byla pro autentizaci uživatelů využita služba Firebase Authentication, která nabízí řadu autentizačních metod poskytnutých v rámci Firebase SDK. Tyto metody umožňují ověřit jakéhokoliv uživatele, který disponuje platnými přihlašovacími údaji a autentizovat jej v rámci Firebase SDK.

V databázi Firestore se pak nachází tabulka „user“, která obsahuje seznam všech registrovaných uživatelů. Každý uživatel má u svého záznamu příznak „isInOrganization“, který indikuje, zda se daný uživatel nachází v organizaci OSŽ. Tento příznak je prvně validován při registraci a poté automaticky každý měsíc, aby bylo ověřeno, zda je uživatel stále členem organizace. Aplikace je určena pouze pro členy organizace, proto každý přihlášený uživatel musí mít tento příznak nastaven na hodnotu true.

V průběhu přihlášení je také zapotřebí, aby byl na server zaslán FCM token zařízení, ze kterého bylo přihlášení provedeno. Jedná se tak z důvodu zasílání specifických notifikací danému uživateli, více o tomto později.

Jako finální krok celého procesu přihlášení musí být provedena synchronizace notifikací, které si uživatel přeje od aplikace dostávat.
V rámci procesu přihlášení tedy bylo třeba naimplementovat tyto části:

- autentizace uživatele
- odeslání FCM tokenu zařízení na server
- synchronizace odebírání notifikací

Aplikace doposud obsahovala pouze jednoduchou funkcionalitu, proto zde neexistovala žádná business vrstva a jakákoliv logika byla reliazována uvnitř jednotlivých ViewModelů. Protože celý proces přihlášení je již funkcionalita komplexnější, bylo třeba tuto vrstvu vytvořit. Pro její návrh byly využity principiy Clean Architecture, kde je všechna logika implementována do jednotlivých případů použití, které pak využívají ViewModely.

Implementace autentizace uživatelů

Autentizace musí být provedena na základě uživatelova příznaku „isInOrganization“, proto je třeba nejprve zjistit jeho hodnotu z databáze. Nicméně z důvodu bezpečnosti je databáze nastavena takovým způsobem, aby bylo možné provést dotaz nad záznamem daného uživatele pouze tehdy, je-li daný uživatel autentizován v rámci Firebase SDK. Autentizační pipeline tedy musela být realizována tímto způsobem:

1. přihlášení pomocí Firebase SDK funkce, díky které je získáno uživatelovo id a oprávnění provést dotaz nad jeho záznamem v databázi

2. získání uživatelova příznaku „isInOrganization"z databáze

3. pokud je hodnota false, provede se okamžité odhlášení z Firebase SDK a přihlášení končí neúspěchem

4. pokud je hodnota true, uživatel zůstává přihlášen a přihlášení je úspěšné

V případě úspěšného přihlášení si pak Firebase SDK na pozadí automaticky zachovává přihlášeného uživatele a není tak zapotřebí řešit jeho re-autentizaci mezi vypnutím a zapnutím aplikace.

Jak jsem již zmínil, uživatelův příznak „isInOrganization“ se může změnit na hodnotu false také v případě, že si přestane předplácet své měsíční členství v organizaci. A protože si aplikace uchovává přihlášeného uživatele do neurčita, mohl by tak nastat případ, kdy se uživatel přihlásí pouze jednou a později si přestane platit členství, ale stále bude v aplikaci považován za přihlášeného. Z tohoto důvodu bylo třeba ještě implementovat metodu, která se provede při každém spuštění aplikace a ověří, zda se současný uživatel stále nachází v organizaci, a případně jej z aplikace okamžitě odhlásí.

Implementace synchronizace notifikací

Dalším krokem bylo zesynchronizovat push notifikace, které si uživatel přeje od aplikace dostávat.

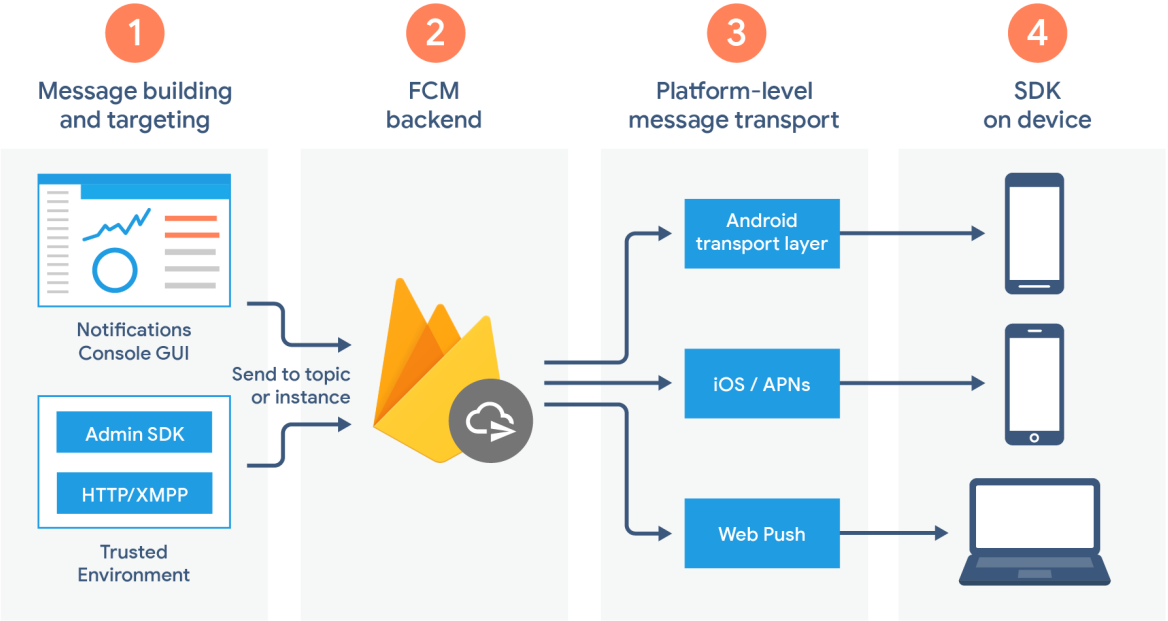
Notifikace z Firebase nejsou zasílány přímo do aplikace, ale nejprve na **Apple Push Notification service** (dále již APNS), a odtud jsou notifikace doručeny na dané mobilní zařízení. K umožnění komunikace mezi platformou Firebase a APNS je třeba v nastavení projektu na Firebase přidat APNS certifikát získaný z Apple Developer portálu. Další věcí potřebnou k tomu, aby mohl Firebase notifikaci na zařízení doručit, je **APNS token**. O tento token si žádá samotné zařízení, a většinou se tak děje při prvním spuštění aplikace, kdy je uživatel dotázán, zda si přeje dostávat push notifikace. Po získání uživatelova souhlasu pak zařízení získá APNS token z APNS serveru a zašle ho na Firebase, kde je mu výměnou poskytnut **FCM token**. Tento token je unikátní pro každé zařízení, a Firebase jej používá pro identifikaci všech zařízení. S pomocí Firebase lze zasílat notifikace dvěma způsoby:

- notifikace cílené na specifické zařízení
- notifikace cílené na téma, které dostanou pouze ta zařízení, která na daném tématu poslouchají

Témata si lze představit jako jakési kanály, na kterých jednotlivá zařízení mohou poslouchat, a dostávat tak specifické notifikace pro dané téma.

V databázi jsou odběry notifikací uživatele realizovány jako pole jednotlivých témat, které se mohou měnit v závislosti na uživatelově nastavení. Aby byly notifikace správně synchronizovány, bylo nejprve nutné aplikaci odhlásit od všech témat, na kterých v daný moment poslouchá. Protože však Firebase SDK nenabízí žádnou možnost hromadného odhlášení od všech témat, musí být tato část provedena manuálně. Nejprve je získán seznam všech dostupných témat z databáze, a pro každé

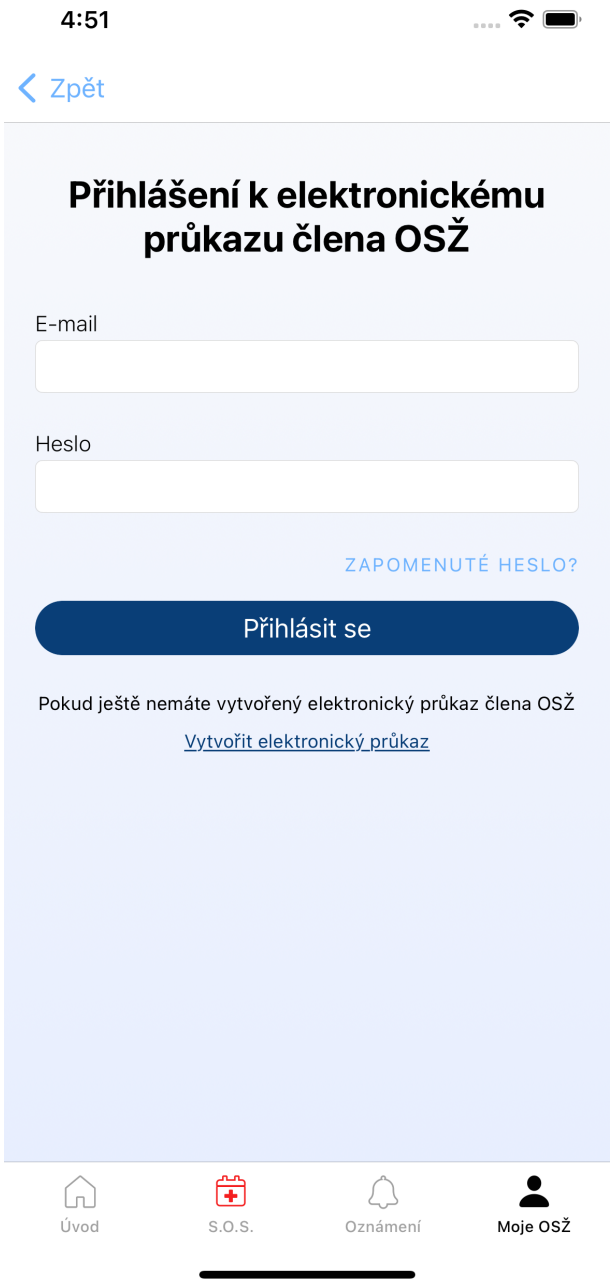
z nich je zavolána Firebase SDK metoda k odhlášení aplikace od daného tématu. Následně se získá seznam témat, ke kterým si uživatel přeje být přihlášen, a opět je použita Firebase SDK metoda k přihlášení aplikace ke všem tématům v tomto seznamu.



Obrázek 3.2: Diagram znázorňující zasílaní notifikací na platformě Firebase [8]

Implementace zasílání FCM Tokenu

Pro zaslání současného FCM tokenu zařízení byla v mikroslužbě UserService implementována metoda, která provádí HTTP požadavek na end-point „user/saveFCMToken“, kterým je token uložen do databáze.



Obrázek 3.3: Výsledný vzhled přihlašovacího formuláře

3.6 Uživatelský profil

Specifikace zadání

Obrazovka profilu je složena z těchto částí:

- bublina s iniciály uživatele
- tlačítko pro otevření nastavení
- dvě dlaždice - jedna obsahující základní organizaci uživatele, a druhá jeho počet odběrů notifikací
- název zaměstnavatele uživatele společně s jeho ikonkou
- osobní číslo uživatele
- karta obsahující miniaturu QR kódu generovaného z uživatelových dat

Při stisknutí dlaždice s počtem odběrů se otevře nové okno sloužící k nastavení uživatelových odběrů. Miniatura QR kódu je generována z uživatelova jména a osobního čísla, a po jeho stisknutí je uživatelovi zobrazeno nové okno obsahující jeho zvětšenou variantu.

Analýza a návrh

Jelikož se jedná o první případ, kdy je potřeba získat uživatelova data z databáze, bylo zapotřebí vhodně naimplementovat způsob, jak je co nejefektivněji získávat a sdílet napříč celou aplikací.

Samotný profil neobsahuje žádnou komplikovanou funkcionalitu. Každá je víceméně přímočará a skládá se pouze z vytažení dat z databáze a jejich následného zobrazení. Jediná věc, na kterou bylo třeba dát si pozor, byla aby se profil správně zobrazil i na menších zařízeních, jelikož obsahuje více prvků.

Implementace

Pro ikonku obsahující iniciály uživatele byla vytvořena nová podtřída UIView se jménem InitialsView, která na vstupu přijímá jméno a příjmení, a následně z nich vygeneruje view s uživatelovými iniciály. Pro dlaždice byla taktéž vytvořena vlastní podtřídá UIView. Rozložení celého rozhraní je pak dáno dohromady s využitím vertikálních a horizontálních UIStackView.

Pro reprezentaci uživatele v aplikace byla ve složce „models“ vytvořena nová struktura „User". K získání uživatele pak v mikroslužbe UserService slouží metoda „fetchCurrentUser“, která na základě současně přihlášeného uživatele získá jeho data z databáze a namapuje je na strukturu User.

K zamezení opětovného provádění dotazu na uživatele v případě, kdy je třeba jej získat na více místech v aplikaci, je využita LoadingProperty. Ta přijímá jako vstupní hodnotu signál. Pro jeho

prvního observera poté LoadingProperty tento signál provede a vrátí jeho hodnotu, kterou si uloží. Všem následujícím novým observerům pak LoadingProperty vrací již uloženou hodnotu. V případě, kdy bychom chtěli hodnotu načíst znovu, stačí zavolat metodu „reload“, která znovu provede daný signál a přepíše současnou uloženou hodnotu. To se může hodit např. v případech, kdy jsou změněny uživatelova data a tudíž je potřeba jej znovu načíst.

```
lazy var userLoadingProperty: LoadingProperty<User, UserServiceError> =
    LoadingProperty {
    self.fetchCurrentUser().toLoadingSignal()
}

lazy var user: LoadingSignal<User, UserServiceError> = {
    let user = self.userLoadingProperty
    let reloadUser = _reloadUser
        .flatMapLatest { _ in self.userLoadingProperty.reload() }
    return merge(user, reloadUser)
}()

func reloadUser() {
    _reloadUser.send()
}
```

Výpis 3.6: Ukázka využití LoadingPropety k získávaní dat uživatele z databáze

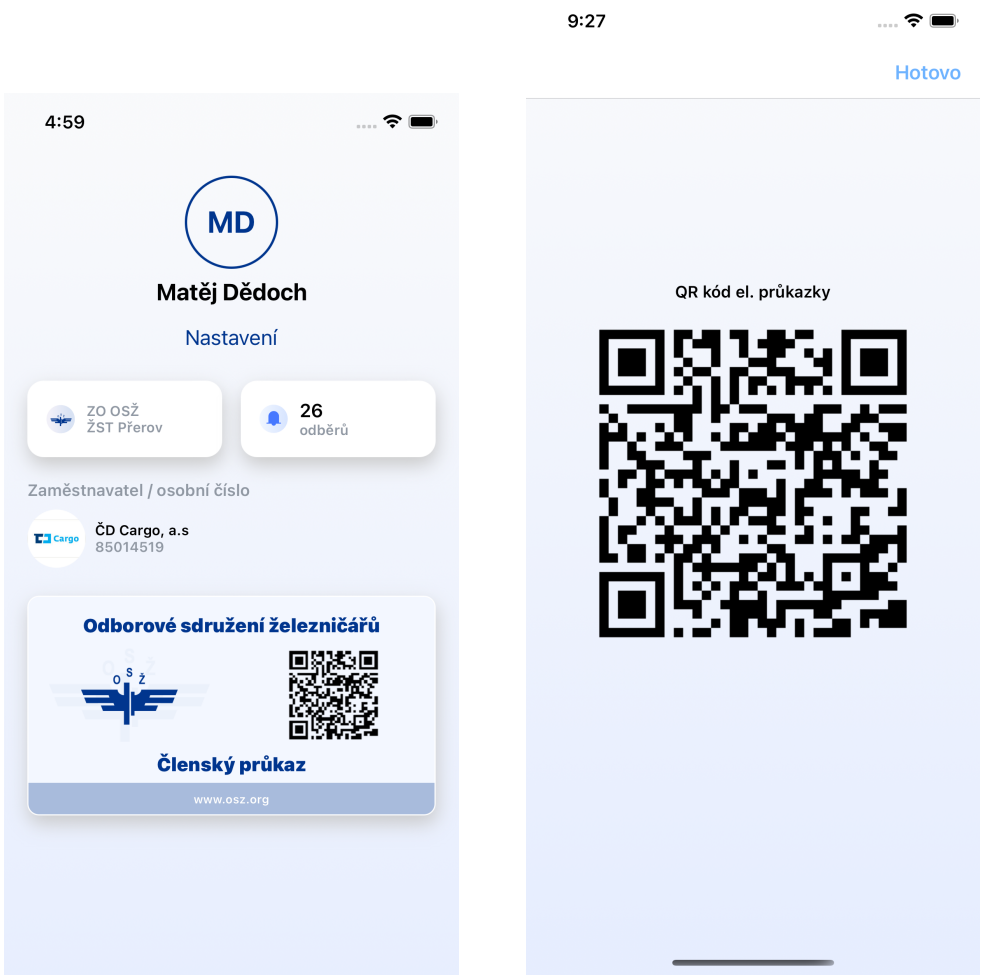
K vygenerování QR kódu byla použita třída CIFilter s názvem „CIQRCodeGenerator“, která je součástí Apple frameworku Core Image, určený k práci s různými typy médií. [4]

```
func generateQRCode(name: String, personalNumber: String, scale: CGFloat = 2.0) ->
     UIImage? {
    let string =  Jmeno  a prijmeni:    + name +   , osobni cislo:    +
        personalNumber
    let data = string.data(using: String.Encoding.utf8)
    if let QRFilter = CIFilter(name: CIQRCodeGenerator) {
        QRFilter.setValue(data, forKey:  inputMessage )
        guard let QRImage = QRFilter.outputImage else {return nil}
        let transformScale = CGAffineTransform(scaleX: scale, y: scale)
        let scaledQRImage = QRImage.transformed(by: transformScale)

        return UIImage(ciImage: scaledQRImage)
    } else {
```

```
        return nil
    }
}
```

Výpis 3.7: Metoda pro vytvoření QR kódu



(a) (b)

Obrázek 3.4: Výsledný vzhled profilu

3.7 Nastavení profilu

V nastavení profilu se nachází 4 položky:

- tlačítko pro odhlášení
- položka pro změnu e-mailu
- položka pro změnu telefonního čísla
- položka pro změnu konce data pojištění uživatele

3.7.1 Odhlášení uživatele

Specifikace zadání

Po stisku tlačítka pro odhlášení je uživatel odhlášen a vrácen na rozcestník pro nepřihlášené uživatele.

Analýza zadání a návrh

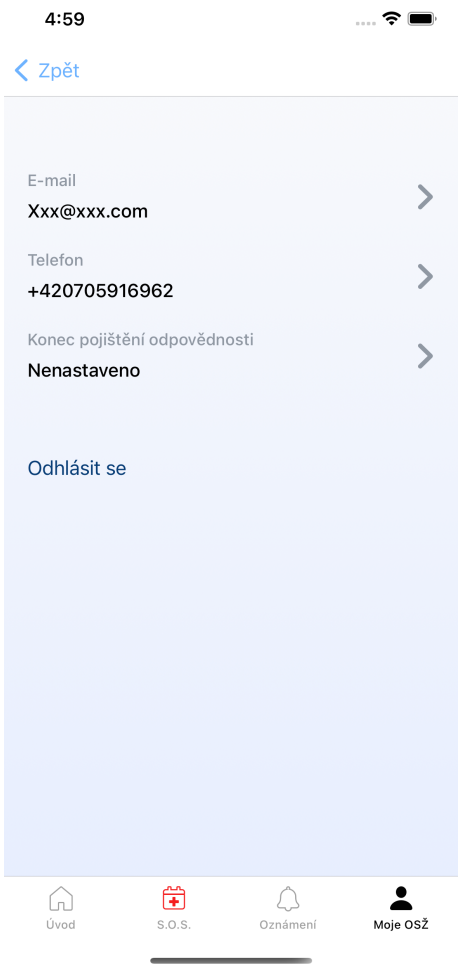
Firebase SDK již poskytuje metodu pro odhlášení současného uživatele v rámci SDK. Podobně jako u přihlášení je ale v návaznosti na odhlášení potřeba provést dva kroky.

Prvním z nich je smazání současného FCM tokenu zařízení. Tato akce je nutná, protože na zařízení přihlášeného uživatele jsou vázány speciální notifikace informující jej o blížícím se konci jeho nastaveného pojištění. Toto provázání se děje při přihlášení, kdy je na server zaslán současný FCM token zařízení. Aby se zabránilo zobrazování těchto notifikací i v případě, že se uživatel odhlásí, je nezbytné tento token smazat.

Dalším krokem bylo vytvořit nový záznam v databázi v tabulce „devices“. Tato tabulka slouží k nastavení odběru notifikací pro nepřihlášené uživatele.

Implementace

Podobně jako u přihlášení je celá odhlašovací pipeline implementována jako sled signálů navázaných na sebe operátorem `flatMapLatest`. Prvním provednou metodou v pořadí je „signOut", implementována v mikroslužbě AuthenticationService, která odhlásí současného uživatele z Firebase SDK. Při odhlášení uživatele z SDK je na pozadí automaticky provedeno anonymní přihlášení, sloužící k identifikaci nepřihlášených uživatelů. Další v pořadí se provádí metoda pro vytvoření nového záznamu v tabulce „devices“, která nejprve vytvoří záznam v tabulce s identifikátorem získaným z anonymního přihlášení, a poté získá seznam všech dostupných témat z databáze. Tyto témata jsou následně nastavena jako defaultní hodnota pro odebírané notifikace vytvořeného záznamu.



(a)

Obrázek 3.5: Výsledný vzhled menu pro nastavení profilu

3.7.2 Změna e-mailu a telefonního čísla

Specifikace zadání

Při stisku tlačítek pro změnu e-mailu a telefonního čísla se zobrazí formulář se dvěma vstupy, do prvního uživatel zadává novou hodnotu měněného atributu, a do druhého opisuje hodnotu z prvního vstupu pro validaci. Pokud je vstup validní a shoduje se s potvrzovacím vstupem, atribut je aktualizován a uživatel je vrácen do nastavení.

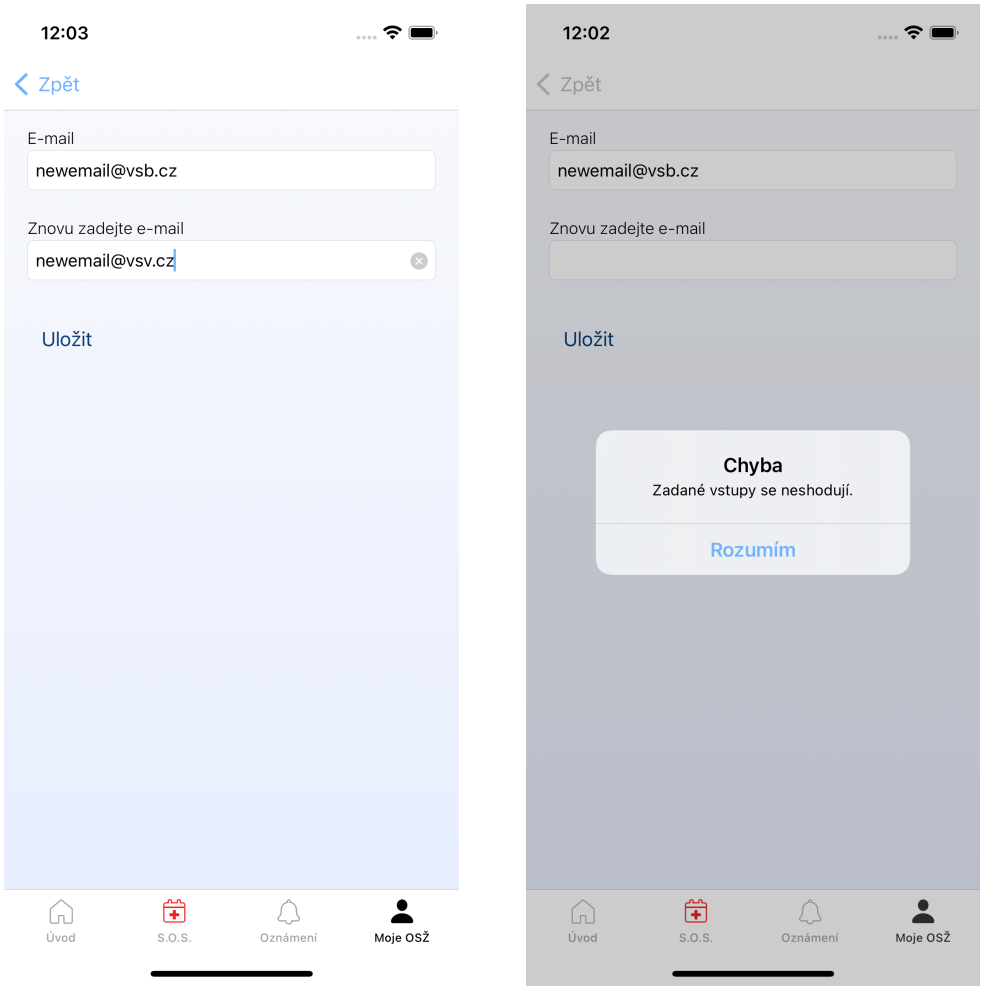
Analýza zadání a návrh

Protože je uživatelské rozhraní i funkcionalita dost podobná pro změnu e-mailu i telefonního čísla, lze oba případy implementovat v jednom ViewControlleru a ViewModelu. K rozlišení způsobu validace

vstupů a metod pro změnu atributu je využit speciální enum UserAttribute, který je předán jako parametr do ViewModelu při otevření okna.

Implementace

Ve ViewModelu na událost stisku tlačítka nejprve reaguje signál pro ověření formátu vstupu, ve kterém je validační metoda vybrána na základě přiloženého enumu. Z tohoto signálu pak vychází další signál, který ověří, zda je vstup shodný s potvrzovacím vstupem. Pokud jeden ze signálů vrátí hodnotu false, je zobrazeno dialogové okno s příslušnou chybovou hláškou. Pokud oba signály vrátí hodnotu true, je následně proveden samotný update atributu v databázi



(a) (b)

Obrázek 3.6: Výsledný vzhled formuláře pro změnu e-mailu

3.7.3 Změna data konce pojištění a notifikace o konci pojištění

Specifikace zadání

Po stisku tlačítka pro nastavení konce data pojištění se zobrazí kalendář, kde uživatel vybere datum a stiskne tlačítko uložit. Dva týdny před zvoleným datem je uživatelovi zaslána speciální notifikace upozorňující, že se blíží konec jeho pojištění.

Analýza zadání

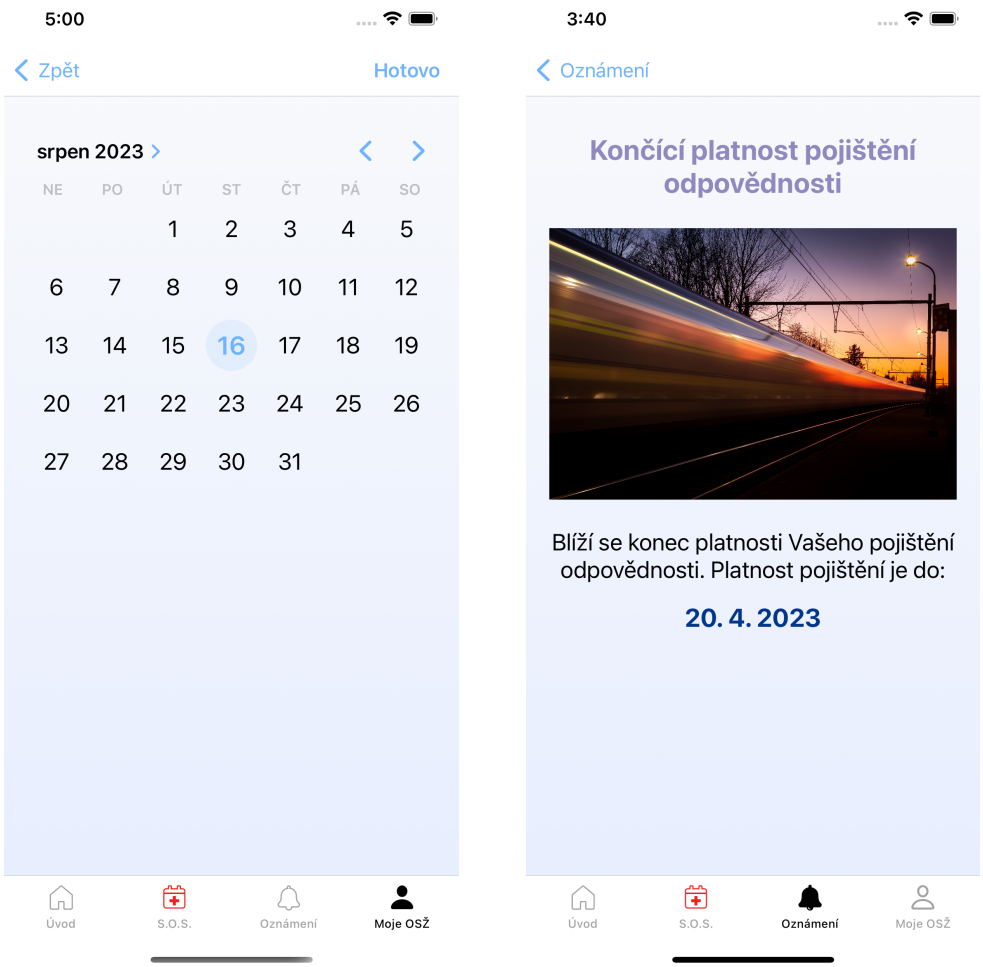
Pro kalendář byla využita již existující UIKit komponenta UIDatePicker. Důležité bylo ošetřit, aby uživatel nemohl zadat datum z minulosti. Na serveru poté periodicky probíhá kontrola zadaného data. Dva týdny před daným datem je zaslána push notifikace na FCM token zařízení, který je do databáze uložen při přihlášení.

Doposud se notifikace ve feedu při rozkliknutí zobrazovali v komponentě zvané UIWebView, která na vstupu přijímá webovou URL a následně zobrazí obsah dané webové stránky uvnitř aplikace. Notifikace o konci pojištění však URL neobsahuje, a její zobrazení bylo třeba naimplementovat nativně. Aby se daly tyto speciální notifikace rozlišit od ostatních, mají příznak „webView“ nastaven na hodnotu false.

Implementace

Nově založenému ViewControlleru byla přidána komponenta UIDatePicker, roztažená přes celou obrazovku s využitím frameworku SnapKit. Jako defaultní datum, na které je UIDatePicker nastaveno, je uživatelovo aktuálně vybrané datum, a v případě, že ještě vybráno nebylo, je zvoleno datum současné. Na navigační liště pak bylo přidáno tlačítko pro uložení nově vybraného data z kalendáře, které se objevuje a mizí na základě toho, zda uživatel vybral datum z minulosti nebo přítomnosti či budoucnosti. Po stisku tlačítka je volaná funkce, která provede HTTP požadavek na server a aktuálnímu přihlášenému uživateli nastaví datum konce pojištění na zvolenou hodnotu.

Pro zobrazení notifikace o konci pojištění byl založen nový ViewController a ViewModel, který při zobrazení zjistí datum konce pojištění současného přihlášeného uživatele a vypíše jej na obrazovku. Pro správné zobrazení bylo třeba upravit metodu, která se provede při kliknutí na notifikaci, a přidat ji novou podmínku, která tento nový ViewController otevře pouze v případě, že má daná notifikace nastavena příznak „webView“ na hodnotu false.



(a) (b)

Obrázek 3.7: Výsledný vzhled nastavení konce pojištění a notifikace

3.8 Zapomenuté heslo

Specifikace zadání

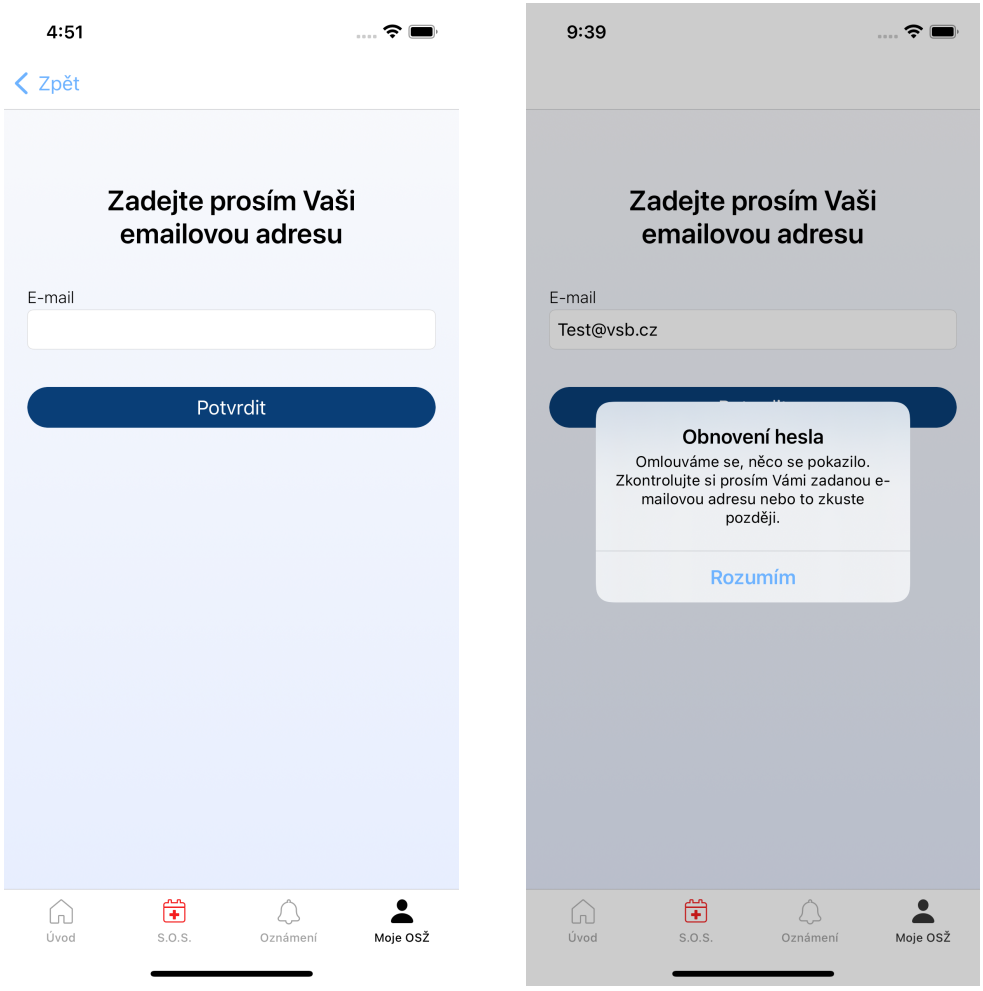
U přihlašovacího formuláře se nachází tlačítko pro změnu hesla pro případ, že jej uživatel zapomněl. Po stisknutí se otevře nová obrazovka obsahující jediný vstup, kde je uživatel dotázán zadat svou e-mailovou adresu, pod kterou je registrován. V případě, že adresa existuje, jsou uživateli na adresu zaslány instrukce pro obnovení hesla. V případě neexistující e-mailové adresy je zobrazena chybová hláška.

Analýza zadání a návrh

Firebase již poskytuje implementaci celého procesu pro změnu hesla, v aplikaci je pouze třeba tento proces započíst voláním speciální Firebase SDK metody „sendPasswordReset".

Implementace

Jako obvykle je volání metody „sendPasswordReset"implementováno jako signál. Tato metoda může vrátit chybu pouze v případě, kdy zadaná e-mailová adresa neexistuje. V tomto případě i daný signál vrátí chybu „.unknownError", a v uživatelském rozhraní je zobrazen dialog informující uživatele, že došlo k chybě.



(a) (b)

Obrázek 3.8: Výsledný vzhled formuláře pro zapomenuté heslo

Kapitola 4

Závěr

4.1 Využité znalosti získané v průběhu studia na VŠB-FEI

Jsem zastáncem názoru, že dobrý programátor není definován pouze jeho hloubkou znalosti konkrétního programovacího jazyka nebo počtem ovládaných programovacích jazyků, ale důležitým faktorem je také jeho způsob uvažování při řešení problémů, které se vyvíjí skrze čas strávený programováním bez ohledu na jazyk. Proto si myslím, že všechny předměty, které jsem měl možnost studovat na VŠB-FEI a které mi umožnily rozvíjet tento způsob uvažování pro mě byli v průběhu praxe přínosné.

Kdybych měl však jmenovat konkrétní předměty, které mi v průběhu praxe nejvíce pomohly, určitě bych zmínil Tvorbu mobilních aplikací I a II. Ačkoliv se tyto předměty zaměřují na vývoj pro platformu Android, mnoho klíčových konceptů lze aplikovat i na vývoj pro platformu iOS, a byly pro mě dobrým úvodem do problematiky vývoje mobilních aplikací. Dalšími užitečnými předměty byly Vývoj informačních systémů a Úvod do softwarového inženýrství, díky kterým jsem se naučil jak přistupovat k návrhu rozsáhlejších a komplexnějších aplikací.

4.2 Nově nabyté znalosti a zkušenosti z praxe

Během mé praxe jsem získal mnoho cenných obecných znalostí z oblasti vývoje software. Dostal jsem podrobný náhled do procesů a praktik, které tato oblast zahrnuje. Naučil jsem se komunikovat v týmu, rychle se adaptovat na náhlé změny ve vývoji, učit se novým technologiím za běhu a navrhovat svá vlastní řešení problémů.

V průběhu praxe jsem se také potýkal s technologiemi, se kterými jsem se na univerzitě nesetkal. Největší novinkou pro mě byl vývoj mobilních aplikací v ekosystému Apple. Ačkoliv jsem začínal s téměř žádnými znalostmi v této oblasti, postupem praxe jsem se v ní stával více sebejistý, a díky získaným zkušenostem z praxe je to nyní jedna z možností, kterou zvažuji jako kariérní cestu. Poprvé

jsem se také setkal s reaktivním programováním, které vyžadovalo změnu mého způsobu uvažování a poskytlo mi tak novou perspektivu na řešení problémů.

4.3 Shrnutí

Celkově hodnotím absolvování odborné praxe velmi pozitivně a jsem velmi vděčný společnosti AstrumQ, že mi tuto příležitost umožnila. Z praxe si odnáším mnoho nových znalostí a zkušeností, ať už ze specifických oblastí pro vývoj software, nebo obecných dovedností jako organizace práce či komunikace v rámci týmu. Praxe mi také pomohla získat jasnější představu, kterým směrem bych se chtěl v budoucnu profilovat, a ujistila mě v mých dovednostech jako softwarového vývojáře.

Literatura

1. *Organizace OSŽ* [online]. [cit. 2023-04-08]. Dostupné z: `https://www.osz.org/`.
2. *Webové stránky firmy AstrumQ* [online]. [cit. 2023-04-08]. Dostupné z: `https://www.astrumq.com`.
3. *Swift dokumentace* [online]. [cit. 2023-04-08]. Dostupné z: `https://www.swift.org`.
4. *Apple Developer* [online]. [cit. 2023-04-08]. Dostupné z: `https://developer.apple.com`.
5. *CocoaPods* [online]. [cit. 2023-04-08]. Dostupné z: `https://cocoapods.org`.
6. *Dokumentace frameworku SnapKit* [online]. [cit. 2023-04-08]. Dostupné z: `https://github.com/SnapKit/SnapKit`.
7. *Dokumentace frameworku SwiftGen* [online]. [cit. 2023-04-08]. Dostupné z: `https://github.com/SwiftGen/SwiftGen`.
8. *Google Firebase* [online]. [cit. 2023-04-08]. Dostupné z: `https://firebase.google.com`.
9. ROBSON, Eric Freeman Elisabeth. *Head First Design Patterns*. 2004. ISBN 9780596007126.
10. NURKIEWICZ, Tomasz; CHRISTENSEN, Ben. *Reactive Programming with RxJava*. 2016. ISBN 1491931655.
11. *Dokumentace frameworku ReactiveKit* [online]. [cit. 2023-04-08]. Dostupné z: `https://github.com/DeclarativeHub/ReactiveKit`.
12. *Dokumentace frameworku Alamofire* [online]. [cit. 2023-04-08]. Dostupné z: `https://github.com/Alamofire/Alamofire`.