FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Fotofolio – mobilní aplikace pro tvorbu portfolia
Student:	Kryštof Příhoda
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat mobilní aplikaci pro operační systém iOS, která bude sloužit profesionálním fotografům pro snadnou prezentaci jejich portfolia a nabízení služeb svým klientům. Aplikace umožňuje tvorbu samotných portfolií, včetně zaznamenání zpětné vazby od zákazníků. Zaměřte se především na klientskou část aplikace. Pokud budete potřebovat nějaká data proveďte jejich mocking vhodným způsobem. Při řešení postupujte dle následujících kroků.

1) Analyzujte existující řešení
2) Na základě analýzy specifikujte funkční požadavky aplikace
3) Proveďte návrh softwarového řešení
4) Navrhněte uživatelské rozhraní
5) Naimplementujte prototyp aplikace
6) Prototyp podrobte uživatelskému testování

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 18. listopadu 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Fotofolio – mobilní aplikace pro tvorbu portfolia

Kryštof Příhoda

Katedra softwarového inženýrství
Vedoucí práce: Ing. Filip Glazar

3. května 2022

Poděkování

Tímto bych chtěl upřímně poděkovat Ing. Filipu Glazarovi za vedení mé práce, věnovaný čas, úsilí a poskytnutí cenné zpětné vazby. Mé díky patří ale i všem mým přátelům a rodině, kteří mi byli oporou po celou dobu studia a tvorby této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 3. května 2022 . . . . . .

Odkaz na tuto práci

Příhoda, Kryštof. *Fotofolio – mobilní aplikace pro tvorbu portfolia.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Práce se zabývá analýzou, návrhem a tvorbou prototypu mobilní aplikace Fotofolio, který svou náplní pokrývá proces poptávky/nabídky fotografa. Fotografům platforma poskytuje možnost intuitivní tvorby portfolií a zákazníkům, kteří fotografa poptávají, tato portfolia prohlížet a na základě prací fotografů učinit volbu správného fotografa pro zákazníkovu událost. Aplikace je navržena pro mobilní telefony s operačním systémem iOS a prototyp je implementován v jazyku Swift s využitím frameworku SwiftUI. Výsledkem práce je platforma s minimalistickým uživatelským rozhraním, která žádné fotografy nediskriminuje a značně zjednodušuje proces navázání kontaktu zákazník-fotograf.

Klíčová slova fotograf, portfolio, mobilní aplikace, prototyp, iOS, SwiftUI, frontend

Abstract

This thesis is covering a process of analysis, design and prototype implementation of a mobile application called Fotofolio, which covers the process of photographer's service demand/supply. This platform provides photographers with the ability to intuitively create portfolios that customers can browse

through and make a decision to choose a suitable photographer for their occasion. The application is designed for mobile phones with operating system iOS and the prototype is implemented in Swift language with the use of SwiftUI framework. The result of this thesis is a platform with minimalistic user interface that does not discriminate any photographers and greatly simplifies the process of establishing a connection between customers and photographers.

Keywords photographer, portfolio, mobile app, prototype, iOS, SwiftUI, frontend

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Pokud člověk v současné době poptává profesionální fotografické služby, mobilní aplikaci v češtině, která by tento proces pokrývala, by hledal neúspěšně. Ač se jedná o poměrně častou a běžnou situaci, postup, jak vyhledat službu toho správného fotografa, není přímočarý tak, jak by mohl být.

Fotografové často působí na sociálních sítích a mohou být osloveni touto cestou, ale na sociálních sítí zákazníka v průběhu hledání rozptyluje mnoho rušivých elementů a příspěvků, které s procesem nemají nic společného.

Pokud je poptávající odpůrcem sociálních sítí, může si svého fotografa vyhledat na základě klíčových slov v internetovém vyhledávači. Z výsledků, které mu vyhledávač nabídne, se ale něco dozví až ve chvíli, kdy si je otevře do nových záložek prohlížeče a prokliká se jimi.

Tuto problematiku jsem se rozhodl vyřešit návrhem mobilní aplikace, která tento specifický proces řeší. Aplikace Fotofolio bude sloužit jako jednotná platforma nejen pro fotografy, kteří zde budou moci publikovat svá portfolia a další informace, ale i pro běžné uživatele, kteří fotografické služby poptávají. Pro navázání kontaktu zákazník-fotograf je mobilní telefon ideálním nástrojem, protože se stal nedílnou součástí našich životů a máme ho stále po ruce.

Uživatelské prostředí mobilní aplikace musí být navrženo tak, aby využilo malou plochu displeje na maximum a realizace prototypu aplikace je perfektní příležitostí pro prohloubení znalostí v oblasti vývoje mobilních aplikací pro iOS.

V práci se zabývám analýzou problematiky, vybranými technologiemi, návrhem aplikace, implementací a v neposlední řadě uživatelským testováním výsledného prototypu.

Kapitola 1

Cíl práce

Hlavním cílem práce je navrhnout, vytvořit a otestovat prototyp mobilní aplikace pro operační systém iOS, který svou funkčností pokryje proces navázání kontaktu uživatele poptávájícího služby fotografa a fotografům poskytne prostor, kde budou moci prezentovat svou práci. Grafické uživatelské rozhraní aplikace by mělo splňovat požadavky a zažité konvence moderních grafických rozhraní – vzhledově minimalistické a zároveň maximálně intuitivní a funkční. Ovládací prvky by měly být rozmístěny tak, jak jsou uživatelé operačního systému zvyklí, aby se uživatel v aplikaci zorientoval hned při prvním spuštění.

Aplikace by měla být vymyšlena a navržena tak, aby žádného fotografa nediskriminovala a v prvotní fázi výběru se zákazník rozhodoval čistě na základě finálního produktu fotografa – jeho fotografií. Jestli je autorem postarší pán, mladá slečna nebo mladý muž s pouhým rokem zkušeností by mělo být až druhotným faktorem při rozhodování zákazníka.

Kapitola 2

Analýza

Pro kvalitní návrh a následnou realizaci aplikace je nezbytné provedení důkladné analýzy problému a technologií. V této kapitole seznámím čtenáře s problematikou, představím současná řešení a specifikuji funkční a nefunkční požadavky aplikace.

2.1 Současné procesy hledání fotografa

Jak jsem již zmínil v úvodu práce, proces poptávání fotografa může mít hned několik podob. Pokud člověk nemá žádné zkušenosti s poptáváním fotografa, může uváznout hned v prvním bodě – kde vůbec začít. V této sekci rozeberu tři nejběžnější scénáře, se kterými jsem se setkal.

Poptávka fotografa na základě doporučení

Velmi častým způsobem volby fotografa je volba na základě osobní reference. Doporučení fotografa známým může být pro člověka jakousi zárukou kvality. Osobě, která je nám blízká, důvěřujeme a pokud byl známý se službou fotografa spokojený, zvyšuje se nám tím i důvěra ve fotografovy nabízené služby. Zde ale může dojít k problému, protože každý člověk má jiné představy a očekávání. Výsledná práce fotografa, se kterou může být spokojen náš známý, nemusí odpovídat kvalitativně nebo i kvantitativně tomu, co požadujeme my samotní.

Vyhledání webových stránek fotografa

Pokud člověk žádné doporučení od známého nemá, může se pokusit fotografa vyhledat pomocí svého preferovaného internetového vyhledávače. Nespornou výhodou této možnosti je pro fotografa to, že si může web nechat navrhnout dle svých představ a žádné kreativní meze se nekladou. Zákazníkovi může být prezentován působivý web fotografa, ale zastoupení fotografových stránek

mezi výsledky vyhledávání bude záviset nejen na tom, jak vhodná klíčová slova uživatel zadal, ale také jak kvalitně zvládnuté má fotograf / autor webu SEO[1]. Po rozkliknutí jednoho z výsledků a zobrazení stránky fotografa tento proces uživatel opakuje libovolně mnohokrát a je velmi pravděpodobné, že ho to po chvíli přestane bavit, nebo se ze změti nových záložek stane nepřehledný chaos. Kromě toho, že se fotografův web uživateli nemusí z různých důvodů mezi výsledky vůbec zobrazit, web fotografa nemusí být responzivní a optimalizovaný pro telefonní obrazovky. Dalo by se namítnout, že by měl mít v dnešní době fotograf svůj osobní web dobře zařízený, ale na kvalitní stránky nemusí mít pouze dostatečné prostředky nebo čas. Ve výsledku je rozhodující to, jak fotograf fotí, ne kvalita jeho webu.

Nalezení fotografa na sociálních sítích

Poslední ze tří vybraných možností, jak navázat kontakt s fotografem, je skrze sociální sítě. Fotografa můžeme na většině sociálních sítích poměrně snadno vyhledat pomocí tagů (klíčových slov) nebo pod jeho uživatelským či skutečným jménem. To ale za předpokladu, že jméno známe – to už se ale částečně vracím k prvnímu bodu doporučení na základě osobní reference. Sociální sítě jsou navrženy pro zcela odlišný účel, než je poptávání fotografů.

V první řadě mají uživatele v aplikaci udržet co nejdéle a nasytit ho co možná nejvíce monetizovaným obsahem za účelem výdělku. Kromě samotných prvků monetizace a reklamních bannerů zde uživatele rozptyluje i samotný obsah. Mísí se zde příspěvky profesionálního i neprofesionálního charakteru na základě toho, koho uživatel sleduje a často si ani neuvědomuje, kolik času na sítích tráví.

Často probíraným a důležitým tématem v kontextu sociálních sítí je ochrana osobních dat uživatele. I přesto, že se orgány snaží chránit osobní údaje a regulovat, jak firmy s daty nakládají, je kauza společnosti Meta – dříve známou pod jménem Facebook – příkladem toho, že ochrana může být mnohdy pouze zdánlivou. Celý tento spor z roku 2018, kdy firma Cambridge Analytica získala data desítek miliónů uživatelů sociální sítě Facebook pro účely volebních kampaní, trefně shrnula ve svém titulku a článku pro britský deník The Guardian Julia Carrie Wong [2] – „Kauza Cambridge Analytica změnila svět, ale nezměnila Facebook“.

[1]Search Engine Optimization, zkráceně SEO, je slovy Jerriho L. Ledforda [1] soubor technik a optimalizačních metod využívaných při návrhu webu za účelem maximalizace dosahu stránky a umístění mezi výsledky vyhledávače na předních pozicích.

2.2 Existující řešení

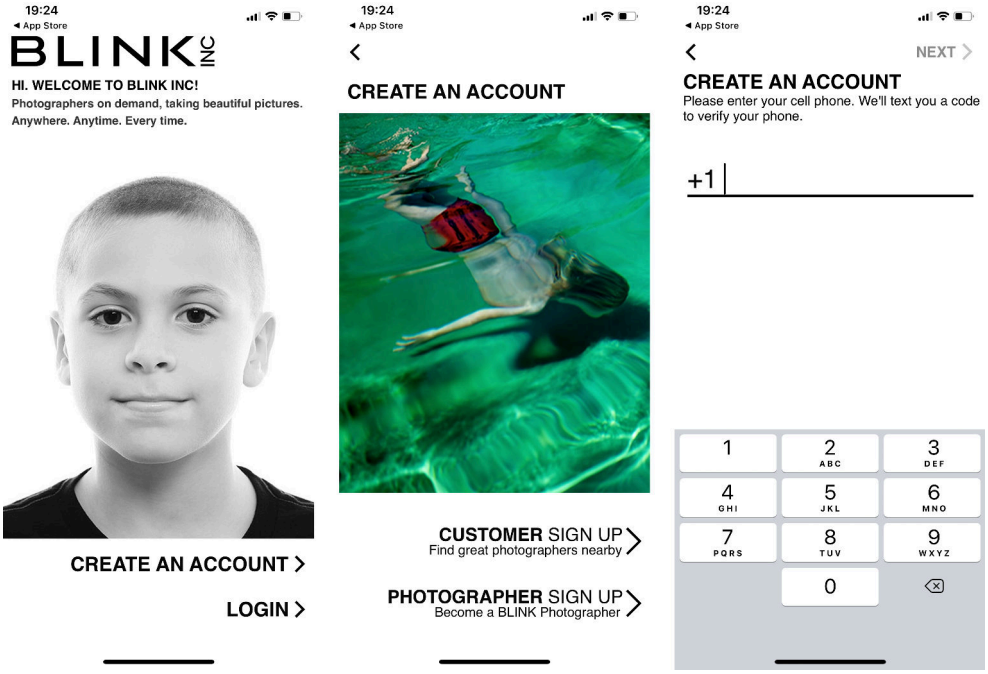
Kromě běžných procesů zmíněných v předchozí sekci se na trhu vyskytují aplikace, které se na proces nabídky/poptávky fotografa zaměřují, ale v offline ani online komunitě fotografů jsem se s nimi nikdy nesetkal. Na vině může být buď fakt, že žádná z aplikací není v českém jazyce, nebo nízký počet aktivních uživatelů.

Projekt Fotofolio bude navržen pro operační systém iOS a proto jsem se do analýzy rozhodl zařadit celkem 5 aplikací, které jsou v České republice volně stažitelné z oficiálního online obchodu aplikací pro iOS – App Store. Mezi výběr jsem zařadil i aplikace, které funkčností nedosahují nejvyšších kvalit a to kvůli jejich přínosu v podobě popisu funkcionalit. Kromě jejich specifikací v popisech navíc pomáhají sestavit celkový obrázek toho, jaké aplikace segment trhu opravdu tvoří. Aplikací je poměrně dost, ale pouze zlomek z nich je funkční a jen zlomek z nich je aktivně užíván. Při průzkumu se zaměřuji na nabízené funkce aplikací a jejich grafická uživatelská rozhraní[2] (v textu dále pod zkratkou GUI nebo UI).

Blink Inc

První verze této aplikace byla vydána v roce 2021 a dle popisku [4] v obchodě App Store se jedná o aplikaci, která dělá proces propojení fotografa se zákazníkem procesem jednoduchým. Zákazníka by měl být během jednoho kliknutí schopný propojit s fotografem a umožnit zamluvení fotografa v libovolný čas na jakékoliv lokaci. Během doby focení by měl mít zákazník možnost fotografie zobrazovat, kupovat a ukládat přímo do své mobilní galerie. Fotografovi je slibována platforma, kde může tvořit cokoliv se mu zachce a navíc na tom vydělat. Výběr fotografa by měl být možný i na základě fotografovy vystavené práce. Dalšími funkcionalitami vyzdvihovanými autory je možnost nákupu dárkových poukazů na focení, *booking* fotografa poblíž uživatelovy lokace a okamžité doručení fotografií.

[2]GUI, graphical user interface, česky grafické uživatelské rozhraní je systém interaktivních vizuálních komponent programu, který prezentuje uživateli informace a akce, které může provést. [3]



Obrázek 2.1: Ukázka grafického uživatelského rozhraní aplikace Blink Inc

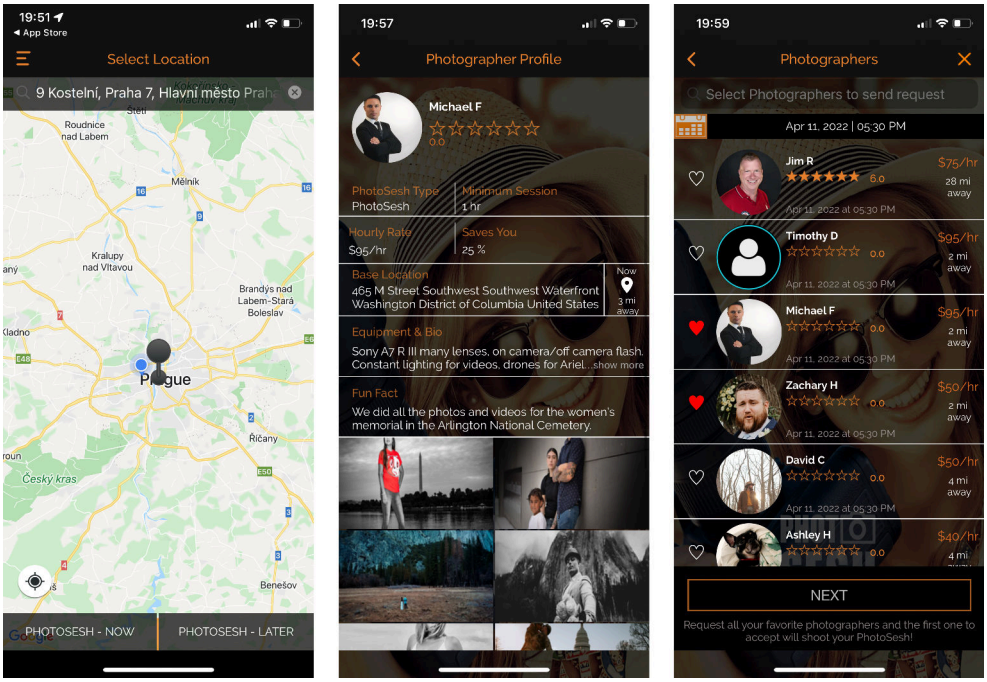
Po spuštění aplikace je nutné přihlášení nebo registrace. První krok registrace vyžaduje zadání telefonního čísla, které bohužel nepodporuje český formát. Zda aplikace splňuje slibovanou funkcionalitu tedy nemohu ověřit aniž bych si pořídil SIM kartu v jedné z podporovaných cizích zemí.

PhotoSesh – Find Photographers

Tato aplikace slibuje dle [5] vyhledávání a nalezení cenově dostupných fotografů v okolí uživatele za pomoci využití GPS. Pro domluvení možného termínu focení je využívána synchronizace kalendářů, dále je prezentováno 20 kategorií a typů focení, které může být fotografem nabízeno – od portrétů, sportu až po focení novorozenců. Zákazník si na fotografově profilu může zobrazit a zjistit hned několik informací:

- hodinovou sazbu
- minimální délku trvání focení
- polohu působení
- průměrné hodnocení na stupnici od 1 do 6
- fotografickou výbavu a popis fotografa

- fotografovu vystavenou práci v podobě fotografií



Obrázek 2.2: Ukázka grafického uživatelského rozhraní aplikace PhotoSesh – Find Photographers

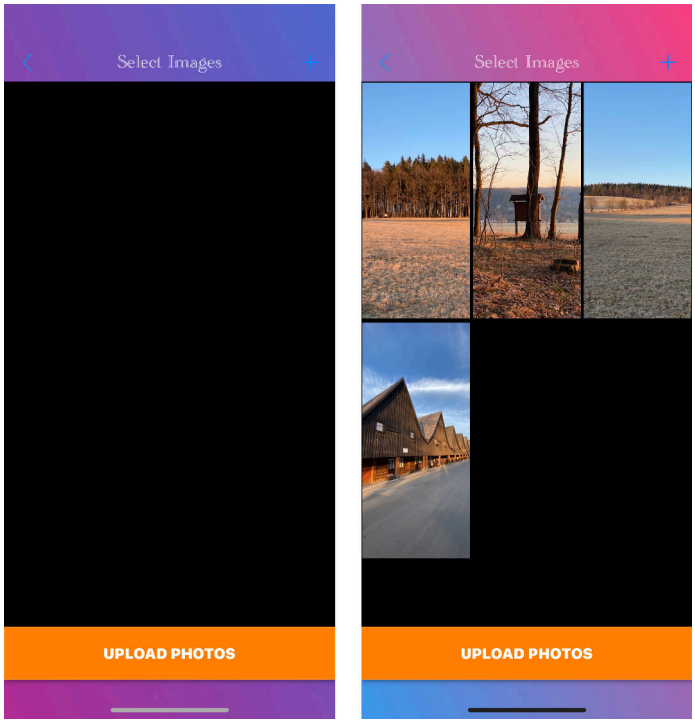
Po spuštění aplikace uživatel špendlíkem v mapě vyznačí lokaci, na které má focení proběhnout, stanoví čas focení a vybere typ focení ze dvou možností – „PhotoSesh LIGHT“ a „PhotoSesh“. První zmíněná možnost spočívá v rezervaci osoby, která pořídí fotografie na mobilní telefon – cenový odhad hodinové sazby je 15 až 25 dolarů. Druhá možnost je v rozsahu 30–95 dolarů za hodinu a slibuje služby profesionálního fotografa na volné noze. V dalším kroku uživatel specifikuje kategorii focení a dalším kliknutím se dostane na výsledný seznam fotografů, kteří odpovídají zadaným kritériím. Zde může vidět jejich profilový obrázek, jméno, průměrné hodnocení, možný termín focení, hodinovou sazbu fotografa a vzdálenost od zadané polohy. Po kliknutí na profilový obrázek se uživatel dostane na profil fotografa. Vedle profilového obrázku má každý fotograf tlačítko ve tvaru srdce. Po zmáčknutí srdce se fotograf dostane mezi vybrané fotografy, kteří budou zakázkou focení osloveni. Na další obrazovce na uživatele čeká shrnutí objednávky a pro její dokončení je nutné zadat platební kartu.

Aby bylo možné proces vyzkoušet, musel jsem cílovou polohu nastavit na New York – v Evropě mi žádné fotografy aplikace nevyhledala. Implementace

procesu je v pořádku, ale problém dle mého názoru nastává u kroku zadávání karty. První otázkou je, jakým způsobem jsou chráněny zadané údaje o bankovní kartě a další otázka je, jestli chce uživatel platit za službu, kdy pořádně ještě předem neví, jaký fotograf nabídku přijme a už vůbec neví, za jaké fotografie platí. UI neodpovídá dnešním standardům, ale opravdu spíš roku 2016, ve kterém byla aplikace vydána.

Phollio – Photos On-Demand

Další aplikací je Phollio, která podle popisu [6] slibuje jednoduché plánování focení kdekoliv a kdykoliv. Ovládáním je aplikace inspirovaná známou seznamovací aplikací Tinder[3] a využívá gest – *swipování*. V tomto případě je předmětem *swipu* portfolio fotografa a v případě, že uživatel táhne prstem doprava, má možnost s fotografem dohodnout termín a detaily focení, případně si ho pouze nechat v záloze do budoucna. Aplikace dále nabízí možnost vyhledávání tvůrců v okolí, zprávy pro komunikaci mezi uživateli a uživatelský profil, který na obrazovce zobrazuje mřížku fotografií tvůrce a informace jako je třeba hodinová sazba, hodnocení nebo věk.



Obrázek 2.3: Ukázka grafického uživatelského rozhraní aplikace Phollio – Photos On-Demand

[3] https://apps.apple.com/cz/app/tinder/id547702041?l=cs

Po spuštění aplikace se uživateli zobrazí jednoduchá obrazovka pro nahrání fotografií. Po navolení alespoň 4 fotografií a jejich potvrzení aplikace spadne. Obdobné ovládání, které používá zmíněný Tinder tedy není možné vyzkoušet. *Flagování*[4] portfolií je jinak princip, který je pro uživatele snadno pochopitelný a potenciálně tedy funkční a úspěšný.

Pozbee

Pozbee je aplikace, která dle [7] spojuje uživatele se zkušeným fotografem pro jakoukoliv příležitost od produktového focení až po svatby. Autoři uvádí tři možnosti, jak booking fotografa zařídit:

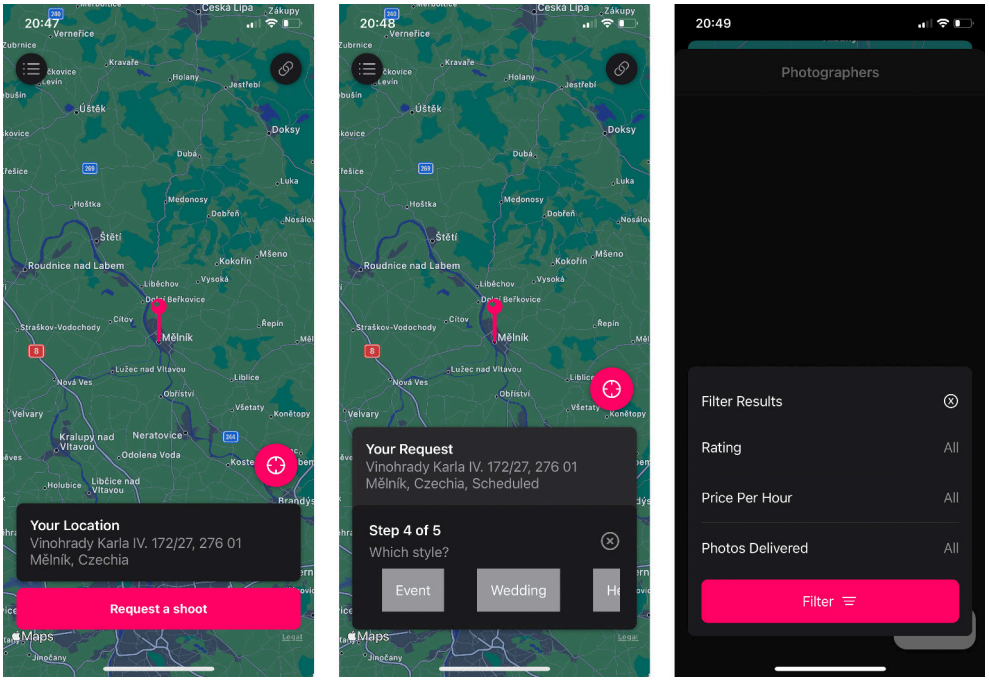
1. **Okamžité spojení** Uživatel na mapě vybere svou polohu a dá fotografům vědět, že má zájem o focení. Pozbee se postará o okamžité propojení s fotografem poblíž.

2. **Spojení pomocí unikátního kódu** Pokud uživatel poptává fotografa na koncertu, festivalu nebo podobné události, může si nechat vygenerovat unikátní kód pro okamžité spojení s fotografem.

3. **Naplánování focení** Vybráním data, polohy a typu focení Pozbee doporučí seznam fotografů. Uživatel si poté vybere svého favorita na základě fotografova portfolia, recenzí a ceny.

Pozbee dále vyzdvihuje cenovou přehlednost. Uživatel vždy předem ví, kolik za fotografie zaplatí. Na žádné skryté poplatky by neměl narazit. Fotografové v aplikaci jsou údajně pečlivě vybíráni, aby aplikace mohla zaručit tu nejvyšší kvalitu služeb. Fotografie jsou slovy Pozbee doručovány do 48 hodin, ale většinou ještě daleko dřív než je stanovený limit.

[4] Flagování je tvorba výběru obsahu, ke kterému má uživatel kladný vztah. Do výběru se obsah dostane po provedení akce typu dvojité klepnutí na displej nebo stisknutím tlačítka např. ve tvaru srdce.



Obrázek 2.4: Ukázka grafického uživatelského rozhraní aplikace Pozbee

Po spuštění aplikace se spustí proces registrace. Celý proces je jednoduchý a uživatelova autenticita je ověřena zasláním kódu na zadané telefonní číslo a emailovou adresu.

Po dokončení registrace uživatel vybere v mapě polohu focení a vyžádá si fotografa pro danou lokaci. Ve čtyřech krocích navolí dobu focení, typ (venkovní nebo vnitřní) a kategorii – na výběr je několik možností od portrétů až po focení nemovitostí a požadovaný počet fotografií. Po zadání kritérií by měl být výsledkem seznam fotografů, které požadavky splňují. Bohužel se mi v žádném státu nepodařilo nastavit kritéria taková, aby výsledek nebyl prázdný a to i s velmi obecnými a mírnými požadavky. Na obrazovce s výsledky se nachází tlačítko filtrování, pomocí kterého si uživatel může fotografy vyfiltrovat podle hodnocení, ceny a počtu doručených fotografií.

V aplikaci je možné přepnutí profilu z normálního uživatele na profil fotografa. Zadáním fotoaparátu, popisu fotografa, adresy působení a specializaci fotografa se odešle schvalovací formulář na server Pozbee. Pole formuláře nejsou nijak ošetřená a fotograf může spadat pouze do jediné kategorie, což považuji za velmi nepraktické.

Unikátní kód pro propojení s fotografem je možné vygenerovat, ale v okolí se žádní fotografové nenachází a proto se funkcionalita nedá vyzkoušet ani ověřit.

UI aplikace je minimalistické, je laděno do tmavších odstínů šedé s kon-

trastními růžovými prvky. V aplikaci není složité se zorientovat a i přes žádné nalezené fotografy ji osobně řadím ve výběru mezi ty lepší.

Zazzi

Dle mého názoru Zazzi dává svou prezentací v App Store na první pohled najevo, že je z dosavadních aplikací nejvyspělejší. Podle [8] byla první verze vydána na konci roku 2020 a od začátku je slibována platforma, která je navržena na míru fotografům, video tvůrcům i samotným zákazníkům. Hlavní myšlenkou je jednoduchá prezentace portfolií zákazníkům pro snadné vyhledání tvůrce a následné vytvoření rezervace tvůrce. Z každé platby za úspěšně dokončenou rezervaci a doručení obsahu si Zazzi účtuje 20 % z celkové částky.

Výčet funkcí v popisu služby se skládá z těchto bodů:

Foto a videoportfolia Umožnění tvorby portfolia a možnost volby úrovně pokročilosti a cenové kategorie pro dané portfolio.

Uživatelský profil Klienti si mohou své profily upravovat. Tvůrce si může u zaslané nabídky profil zobrazit ještě před tím, než nabídku akceptuje.

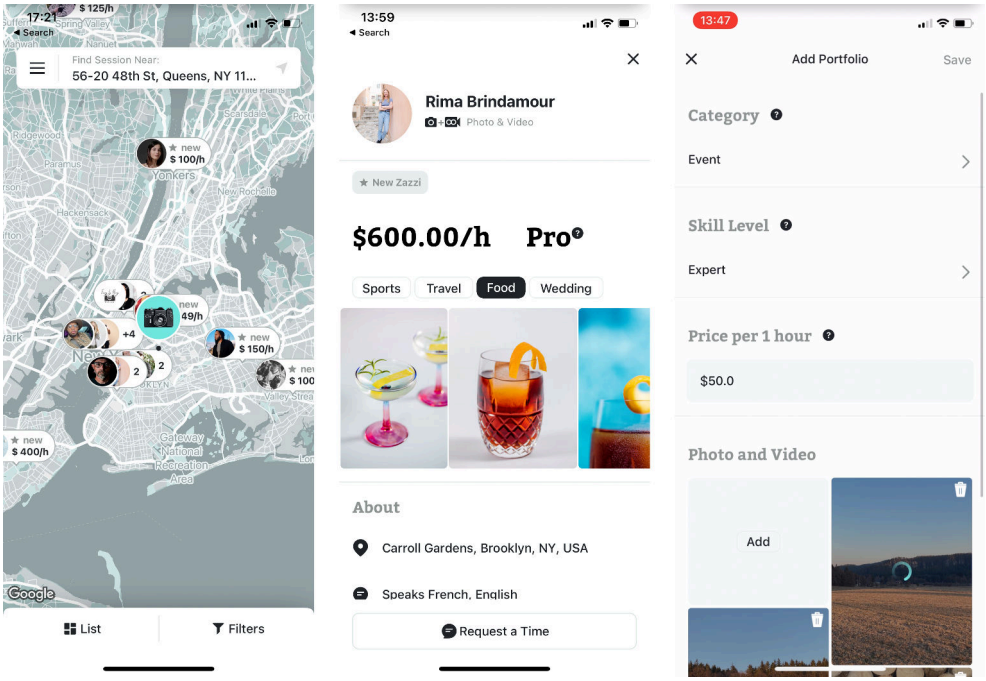
Rezervace přímo v aplikaci Zazzi umožňuje klientovi vytvořit rezervaci přímo v aplikaci. Po vytvoření rezervace může tvůrce nabídku potvrdit, zamítnout nebo se termín pokusit upravit ve zprávách se zákazníkem.

Hledání fotografa v okolí Aplikace umožňuje hledání tvůrce podle libovolné polohy.

Snadná komunikace Pro komunikaci mezi zákazníkem a tvůrcem je v aplikaci chat pro zasílání zpráv. Notifikace a upomínky k naplánovaným schůzkám focení uživateli připomínají aktuální a nadcházející události.

Sdílení fotografií Po skončení focení fotograf upraví, nahraje a pošle fotografie klientovi, který k nim bude moci přistoupit ve své mobilní galerii.

Bezpečné platby Pokud dojde k doručení fotografií a klient je v pořádku obdrží, Zazzi garantuje uhrazení platby fotografovi.



Obrázek 2.5: Ukázka grafického uživatelského rozhraní aplikace Zazzi

Po spuštění aplikace na uživatele čeká obrazovka s mapou, kde je nutné vybrat polohu focení. Polohu lze zadat do textového pole nebo využít špendlíku v mapě. Pokud je v okolí lokace dostupný tvůrce, zobrazí se společně s ostatními v přehledně udělané sekci, do které se uživatel dostane jedním kliknutím z obrazovky mapy. Sekce je příjemně minimalistická a kromě portfolií tvůrců zde můžeme dohledat informaci o hodinové sazbě a celé jméno. Výběr tvůrců v okolí se dá pomocí filtrů s parametry kategorie focení/natáčení, úroveň tvůrce, ceny a mnoho dalšího zúžit. Po kliknutí na jeden z dostupných profilů se uživatel dozví kompletní informace o tvůrci a uživatel má možnost zobrazení dalších portfolií tvůrce. Kromě informací a portfolií zde může dojít ke kontaktování tvůrce a následné domluvě termínu. Při testování jsem navíc objevil možnost hodnocení uživatelů, která v popisu aplikace není zmíněna.

Pokud se chce uživatel stát tvůrcem, po vyplnění nezbytných informací nahraje své první portfolio a zařadí se do fronty tvůrců, kteří čekají na schválení. Proces schvalování je zajímavou vlastností aplikace, která udržuje standard možných tvůrců. Nevýhodou tohoto procesu je samotné schvalování, které vyžaduje lidský faktor při registraci.

Aplikace je velmi pěkně zpracována a její UI je intuitivní a jednoduché. Navigace je dobře rozmyšlená a všechny funkce, které jsou implementovány podle mého názoru dávají smysl a tvoří kvalitní, funkční celek.

2.3 Požadavky

Kromě analýzy současných řešení na trhu, je neodmyslitelnou součástí počáteční fáze návrhu a analýzy aplikace stanovení požadavků. Požadavky můžeme rozdělit do dvou kategorií – funkční a nefunkční.

Než uvedu, jaké funkční požadavky bude aplikace Fotofolio splňovat, je třeba říci, co to funkční požadavky jsou. Přesnou definici jsem si vypůjčil z článku Anastasie Kompaniets [9]. Funkční požadavky jsou vlastnosti a specifikace aplikace, které musí být v aplikaci zahrnuty. Aby software dodržel specifikovanou funkčnost, musí tyto požadavky splňovat. Nefunkční požadavky aplikace stanovují, jak budou v systému funkční požadavky implementovány a systémem vykonávány.

2.3.1 Funkční požadavky

F01: Registrace uživatele

Uživatel se bude moci v aplikaci zaregistrovat ve dvou režimech:

- **Režim tvůrce** Tento režim bude fotografovi umožňovat navíc některé úkony, které v běžném režimu nebudou dostupné. Uživateli bude povolovat tvorbu portfolií a zákazník se na jeho profilu bude moci dozvědět více věcí než u běžné verze profilu.
- **Režim běžného uživatele** Tento režim bude oproti režimu tvůrce jednodušší. Uživatel nebude moci přidávat ani upravovat portfolia a na jeho profilu bude dostupných méně informací.

F02: Přihlášení a odhlášení uživatele

Registrovaný uživatel se bude moci do aplikace opětovaně přihlásit a stejně tak se z ní i odhlásit.

F03: Profil uživatele

Každý uživatel bude moci zobrazit libovolný profil jiného registrovaného uživatele, ze kterého se dozví základní informace o uživateli a zobrazí se mu profilová fotografie uživatele. Společnými údaji fotografa a zákazníka bude uživatelské jméno, křestní jméno a příjmení, přibližné místo působení a průměrné hodnocení. U profilu tvůrce se budou navíc zobrazovat jeho portfolia, jak dlouho je aktivním fotografem a krátký profilový popis.

F04: Úprava profilu uživatele

Svůj profil bude mít možnost uživatel upravit a to na základě toho, v jakém režimu bude zaregistrován. Pokud se uživatel v průběhu rozhodne akci přerušit,

profil se vrátí do původního stavu a nebude se tedy muset obávat, že by o nějaké informace mohl nenávratně přijít.

F05: Tvorba portfolia

Uživateli v režimu tvůrce bude umožněno vytvářet nová portfolia. Portfolio se bude skládat z názvu, fotografií, popisu a tagů – klíčových slov bez mezer.

F06: Feed portfolií

Feedem je myšlena obrazovka, ve které jakýkoliv uživatel přehledně uvidí všechna dostupná portfolia. Kromě samotných fotografií portfolia se bude zobrazovat uživatelské jméno autora a text daného portfolia.

F07: Filtrování a řazení portfolií

Zobrazovaná portfolia ve feedu bude možné seřadit podle času vytvoření portfolia a dále podle průměrného hodnocení autorů. Pro filtrování portfolií bude moci uživatel zadat libovolný počet tagů – pokud portfolio bude všechny specifikované tagy obsahovat, zobrazí se mezi výsledky.

F08: Přidání portfolia do výběru

Zmáčknutím tlačítka záložky vedle portfolia nebo dvojitým poklepáním na portfolio se portfolio přidá do výběru.

F09: Zobrazení vybraných portfolií

Výběr portfolií bude ukazovat uživateli více informací o autorovi portfolia. Krom informací, které vidí u portfolia i ve feedu, zde uvidí i celé jméno autora, místo působení fotografa, průměrné hodnocení a profilový text. Z obrazovky s výběry bude uživateli umožněno napsat autorovi portfolia nebo portfolio z výběru odebrat.

F10: Chat

Pro domluvu mezi uživateli bude sloužit obrazovka zpráv, kde uživatel přehledně uvidí jednotlivé chaty. Uživateli bude umožněno posílat zprávy libovolnému jinému uživateli.

F11: Vyhledávání uživatelů

Bude umožněno vyhledání libovolného uživatele na základě jeho polohy nebo uživatelského jména.

F12: Hodnocení profilu

Každý uživatel bude mít možnost ohodnocení cizího profilu. Proces hodnocení bude uživatelsky co nejjednodušší a promítne se do celkového průměru hodnocení, který se pro daný profil zobrazuje.

2.3.2 Nefunkční požadavky

N01: Podpora různých uhlopříček

Aplikace by měla být podporována všemi velikostmi iPhonů a její funkčnost by měla zůstat kompletně zachována i na menších úhlopříčkách.

N02: Stabilita

Stabilita aplikace musí být na vysoké úrovni a nesmí docházet k žádným únikům paměti.

N03: Lokalizace

Aplikace by měla být kompletně lokalizována do českého jazyka, aby jejímu uživatelskému rozhraní porozuměl úplně každý bez výjimky.

N04: Dodržení zvyklostí iOS

Uživatelské rozhraní aplikace bude ctít tradiční zvyklosti a konvence rozmístění ovládacích prvků tak, jak jsou uživatelé operačního systému iOS zvyklí.

Kapitola 3

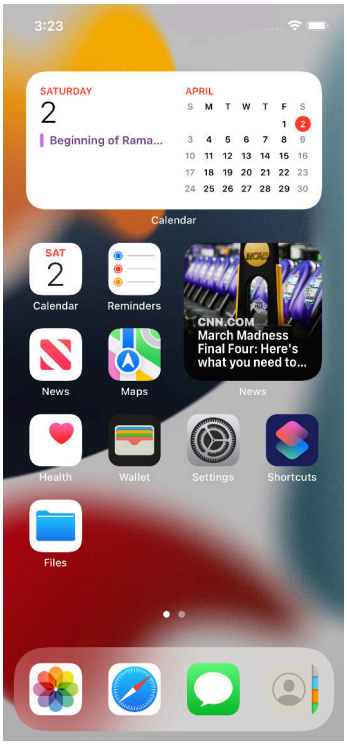
Technologie

S vhodným výběrem technologií úzce souvisí kvalita finálního řešení. V této kapitole stručně představím operační systém, pro který bude aplikace vyvinuta, zvolený software pro návrh uživatelského rozhraní, programovací jazyk Swift, vývojové prostředí Xcode a v neposlední řadě i framework SwiftUI.

3.1 iOS

Pro představení operačního systému čerpám z dokumentu Mobile Operating System: Analysis and Comparison of Android and iOS [10] a článku Edgara Cervantese [11]. První verze iOS byla vydána společností Apple v roce 2007 a od té doby systém prošel mnoha změnami. Tento OS je považován za jeden z nejbezpečnějších systémů pro chytré telefony a jeho uživatelé mohou počítat s výbornou odezvou systému a skvělou optimalizací. Uzavřenost Apple ekosystému umožňuje úzkou provázanost s hardwarem a díky odladění systému jsou mnohdy zařízení Applu responzivnější než konkurenční zařízení s operačním systémem Android s mnohem výkonnějším hardwarem. Systém je známý pro svou uzavřenost, jednoduchost, intuitivnost a funkčnost. Křivka učení je velmi strmá a to i díky tomu, že Apple díky svým Human Interface Guidelines [12] jasně stanovuje zásady a směrnice, kterými se vývojář může řídit, aby bylo UI maximálně funkční a dodržovalo konvence, na které je uživatel zvyklý.

Minimální podporovanou verzi systému jsem pro svou aplikaci zvolil číslo 15. Aktuálně obsahuje nejnovější bezpečnostní záplaty a dle oficiálního přehledu [13] je podporována stále i staršími modely jako třeba iPhone 6S, vydaný v roce 2015, a iPhone SE z roku 2016.

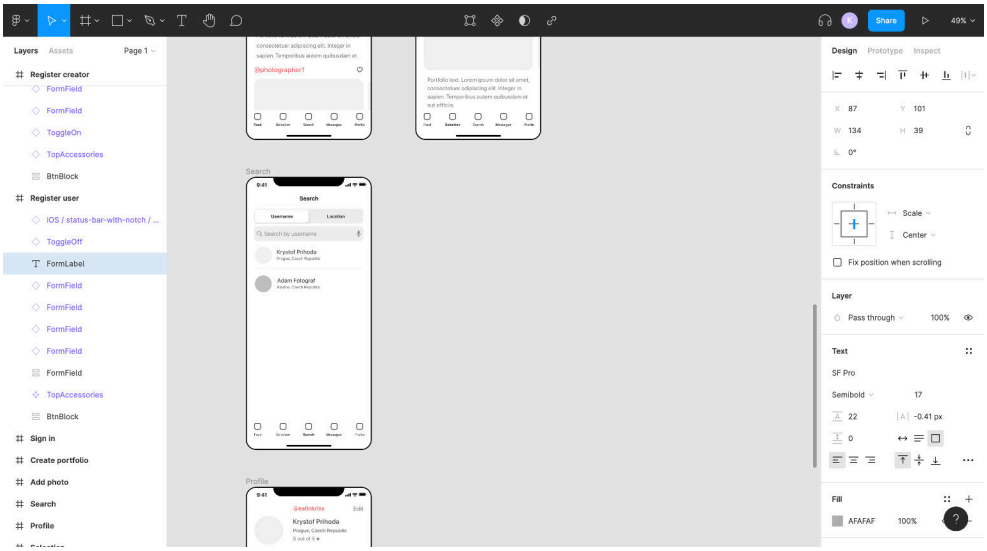


Obrázek 3.1: Ukázka domácí obrazovky operačního systému iOS

3.2 Figma

Figma je vektorový grafický nástroj určený pro návrh uživatelského rozhraní prototypů [14]. Služba je *cloud-based* – je plnohodnotně funkční i ve webovém prohlížeči a nenutí uživatele, aby měl pro návrh nainstalovanou desktopovou verzi aplikace. Tato vlastnost umožňuje běh na jakémkoliv OS ať už je to macOS, Windows, Linux anebo Chrome OS. Pro objektivní shrnutí dalších předností tohoto softwaru jsem čerpal z článku pana Kopfa [15] a zařadil jsem následující:

- starter plan umožňující plnohodnotnou funkcionalitu zdarma
- intuitivní a přímočaré uživatelské rozhraní
- snadná týmová spolupráce na projektu
- integrovaný verzovací systém
- jednoduché sdílení projektu napříč uživateli



Obrázek 3.2: Ukázka *cloud-based* služby Figma pro tvorbu wireframů

Hlavním rozhodovacím faktorem, proč jsem zvolil právě Figmu, je jednoduché rozhraní pro tvorbu wireframů v kombinaci s její snadnou přístupností přes webový prohlížeč a dostupností zdarma.

3.3 Programovací jazyk Swift

Swift můžeme kvalifikovat jako multi-paradigmatický[5], kompilovaný a *high-performance* jazyk, který ctí moderní teoretické koncepty programovacích jazyků [16]. Apple [17] slibuje čistou a jednoduchou syntaxi, snadné propojení s existujícími knihovnami a frameworky v Objective-C a C včetně vysoké paměťové bezpečnosti. Swift je *open source* – jeho zdrojový kód je veřejně dostupný a komunitou může být i nadále vylepšován. Základem jsou datové struktury, funkce a HOF[6], objekty, protokoly, closures a mnohé další. Aplikaci jsem se rozhodl napsat ve frameworku SwiftUI, takže byl jazyk Swift jasnou a jedinou volbou.

V následujících podsekcích stručně definuji pro Swift klíčové konstrukty, které jsou ve Swiftu hojně užívané nebo mimořádně silné. V podsekcích čerpám z oficiální dokumentace Swiftu [19].

[5]Umožňuje programátorovi více přístupů – OOP, funkcionální programování apod.

[6]Higher order function, česky funkce vyššího řádu je funkce, která má jako argument jednu nebo více funkcí anebo funkci vrací. [18]

Struktury a třídy

Struktury a třídy jsou univerzální konstrukty s flexibilní konstrukcí, které jsou díky svým vlastnostem a metodám základními stavebními kameny kódu. Definice tříd probíhá pomocí stejné syntaxe jako při definování primitivních proměnných (Integer, Float, Double, Bool, Character, String), konstant a funkcí. Struktury jsou hojně užívány ve frameworku SwiftUI – o tom ale až v sekci 3.5. Pro přehlednější srovnání těchto dvou typů jsem sestavil jednoduchou tabulku.

Tabulka 3.1: Tabulka srovnání vlastností struktur a tříd ve Swiftu

Vlastnost	Struktura	Třída
Definice metod	Ano	Ano
Definice vlastností	Ano	Ano
Definice inicializace	Ano	Ano
Reference type	Ne	Ano
Value type	Ano	Ne
Možnost rozšíření pro funkcionalitu navíc	Ano	Ano
Konformování k protokolu	Ano	Ano
Dědičnost	Ne	Ano
Type casting	Ne	Ano
Deinitializer	Ne	Ano

Optional

Optional je ve Swiftu speciální typ nad jiným datovým typem, který umožňuje proměnné nemít žádnou hodnotu – `nil`. Optional povoluje dva případy:

- Proměnná je `nil` a nenabývá žádné hodnoty.
- Proměnná má hodnotu a pro umožnění přístupu musí být rozbalena.

Ukázka často využívaných možností rozbalení *optionalu*

```
1  var strHello: String?
2
3  // ...
4
5  strHello = "Hello World!"
6
7  if strHello != nil {
8     print(strHello!)
9  } else {
10    print("*Silence*")
11 }
```

Listing 1: Forced unwrapping, vynucené rozbalení *optionalu*, je bezpečné použít pouze v případě, kdy si je člověk 100% jistý, že optional hodnotu má – jinak program spadne.

```
1  var strHello: String?
2
3  // ...
4
5  strHello = "Hello World!"
6
7  if let hello = strHello {
8     print(hello)
9  } else {
10    print("*Silence*")
11 }
```

Listing 2: Ukázka Optional Binding – konstrukce `if let`. Pokud optional obsahuje hodnotu, rozbalí se do dočasné proměnné (v ukázce `hello`), se kterou můžeme ve *scopu* na řádkách 7 až 9 pracovat jako s normální proměnnou.

```swift
var strHello: String?

// ...

strHello = "Hello World!"

guard let hello = strHello else { return }

print(hello)
```

Listing 3: Konstrukce `guard let` je velmi podobná konstrukci `if let` s tím rozdílem, že proměnnou s rozbalenou hodnotou můžeme využít kdekoliv ve zbytku kódu. Podmínkou této konstrukce je, že ve *scopu* za přiřazením do dočasné proměnné je nutné opustit funkci, podmínku nebo *loop* – smyčku pomocí `return`.

Protocol

Protokol je *interface* – rozhraní. Rozhraní definuje *blueprint* (modrotisk) metod, vlastností a požadavků, které zařizují požadovanou funkcionalitu. Protokol může být adoptován třídou nebo strukturou, aby poskytl skutečnou implementaci požadavků protokolu. Ve chvíli, kdy třída nebo struktura tyto vlastnosti naplňuje, říkáme o ní, že *konformuje* k protokolu. Protokoly se stejně jako struktury často využívají ve frameworku SwiftUI – sekce 3.5.

Enumeration

Enumeration, česky výčtový typ, zkráceně enum, je datový typ tvořený konečnou množinou programátorem pojmenovaných hodnot. Enum pak může nabývat právě jedné z definovaných hodnot a programátorovi umožňuje tvořit lépe čitelný kód. Enumy jsou implementovány i v jazycích jako je Java nebo C/C++. Specialitou Swift enumů jsou *associated values*, které umožňují přiřazení libovolného typu k pojmenované hodnotě. Výčty ve Swiftu navíc podporují incializátory pro nastavení počáteční hodnoty, mohou být rozšířeny pro přidání funkcionality a mohou konformovat k protokolům. Dále podporují *computed properties* – vlastnost, která je vypočítána spuštěním kódu, který uživatel pro danou vlastnost definuje.

```
enum Role {
    case guest(Int?)
    case registered(id: String)
}

// ...

let role: Role = .registered(id: registeredUser.id)
```

Listing 4: Ukázka inicializace enumu ve stavu *registered* s *associated value* v podobě id typu `String`.

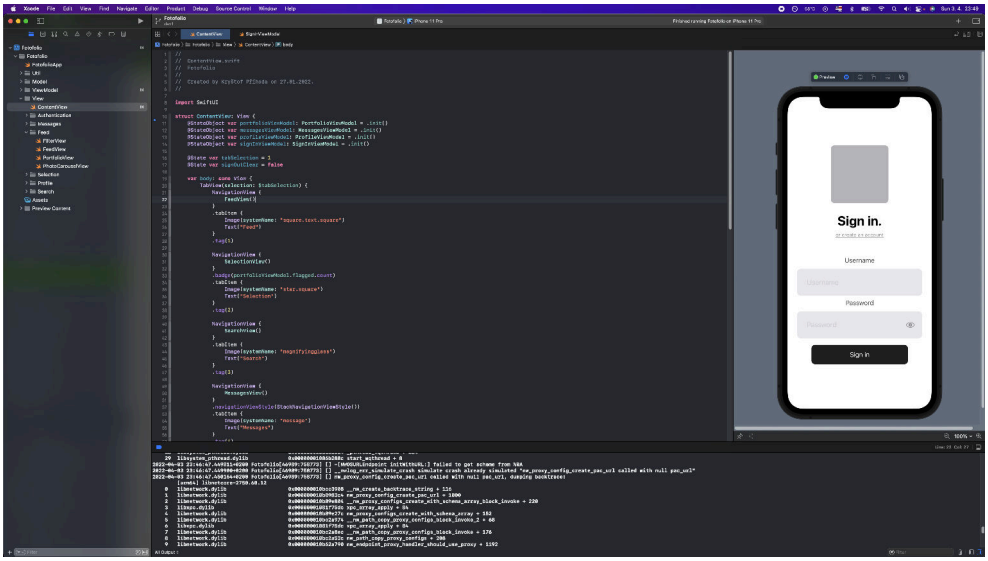
Extension

Extension, česky rozšíření, umožňuje přidání nové funkcionality k již existující třídě, struktuře, enumu nebo protokolu. Rozšířením můžeme typu dodat nový *initializer*, definovat metody nebo jiné vlastnosti a mnohem více. Důležitou vlastností rozšíření je pouze přidání funkcionality – ta původní zůstává zachována.

3.4 Vývojové prostředí Xcode

Xcode je oficiální sada nástrojů určených pro vývoj softwaru primárně pro Apple zařízení. Prostředí bylo představeno v roce 2003 a dle AppleInsider [20] bylo původně vyvinuto firmou NeXT pod názvem Project Builder. Od jeho představení software prošel velkým množstvím aktualizací a současnou verzí je 13.3. IDE[7] obsahuje kromě jazyku Swiftu další jazyky (nativně AppleScript, C, C++, Objective-C a další) a podporuje zvýrazňování syntaxe, *auto-completion*, vyznačování chyb v kódu včetně návrhů na jejich opravu a *debugovací* nástroje pro produkci čistého a čitelného kódu bez chyb. Xcode podporuje verzování, plně integruje Git repozitáře a pro svou plnou funkčnost vyžaduje balíček command line tools. Pro uživatele macOS je volně ke stažení a macOS je zároveň jediným operačním systémem, pro který je Xcode dostupný.

[7] Integrated Development Environment



Obrázek 3.3: Ukázka vývojového prostředí Xcode

3.5 Framework SwiftUI

SwiftUI je deklarativní framework představený v roce 2019 na konferenci WWDC, který nabízí řadu nástrojů pro rychlé sestavování moderních uživatelských rozhraní [21]. Framework je postavený na základech jazyku Swift a jeho deklarativní přístup výborně vysvětluje Robert Mejia v článku Declarative and Imperative Programming using SwiftUI and UIKit [22]. Tento přístup u SwiftUI představuje to, že data jsou propojena s prvky uživatelského rozhraní a aktualizace stavu v případě jejich změny není naší zodpovědností. Díky tomuto přístupu tedy sdělujeme aplikaci to, jak má vypadat a jak se má chovat při jednotlivých stavech, které mohou nastat, ale odpovědnost za přechody mezi jednotlivými stavy přebírá framework. Tento přístup je svou filozofií naprosto odlišný od toho, který Apple používal před představením SwiftUI ve svém starším frameworku UIKit.

SwiftUI je *cross-platform* a vývojář může navrhnout UI tak, aby fungovalo napříč platformami iOS, macOS, tvOS ale i watchOS. Skvělou funkcí je Preview[8], který po jednoduchém nastavení v reálném čase v simulátoru zobrazuje, jak UI vypadá. Pro zobrazení a základní otestování funkcionality UI nemusí vývojář zbytečně spouštět simulátor, ale postačí právě Preview. V dokumentaci [23] se můžeme dozvědět, že SwiftUI nezapomíná ani na starší frameworky Applu – AppKit, UIKit, WatchKit – a umožňuje integrovat

[8]Preview je vidět na obrázku 3.3 vpravo.

komponenty přímo do SwiftUI. Stejně tak funguje integrace i naopak – tedy ze SwiftUI do zmíněných tří frameworků.

Základem tvorby rozhraní je protokol `View` v jehož těle je možné využít mnoho stavebních prvků a komponent. Prvků je skutečně velké množství, Apple jím říká User Interface Elements a pro ukázku to je například `Button`, `Toggle`, `DatePicker`, `ProgressView`, `AsyncImage`, `TextField` nebo `Shape`. Jednotlivé *views* pak můžeme dále modifikovat pomocí *view modifiers* a uspořádávat po třech osách pomocí kontejnerů, které také konformují k protokolu `View` – představení a detailnější popis kontejnerů v podsekci 3.5. Kombinací a skládáním těchto primitivních konstrukcí lze snadno vytvářet komplexní obrazovky a *layouty*. Výhodou je, že se uživatelské rozhraní tvoří výhradně kódem, který se píše intuitivně a zároveň zůstává čitelným i při složitějších konstrukcích.

```
1  Rectangle()
2      .foregroundColor(.gray)
3      .cornerRadius(9)
4      .padding()
```

Listing 5: Ukázka aplikování modifikací na element reprezentující obdélník – modifikátor na 2. řádku nastavuje barvu na šedou, 3. řádek zakulacuje rohy obdélníku a poslední řádek vytváří okolo obdélníku volný prostor. Modifikátory se aplikují postupně – na pořadí záleží.

Rozdělení kontejnerů

Apple v oficiální dokumentaci SwiftUI [23], ze které v této podsekci čerpám, rozděluje kontejnery do tří velkých skupin:

Layout Containers Kontejnery této skupiny mají jasnou úlohu, kterou je uspořádání a organizace *views* v UI. Zásobníky a mřížky, které jsou pro tuto úlohu užívány se průběžně aktualizují a upravují pozici obalených *views* na základě změn v kontextu obrazovky. Kontejnery do sebe můžou být libovolně vnořovány pro úspěšnou realizaci složitějších *layoutů*.

- `HStack`/`VStack` jsou `View` struktury, kterými můžeme obalit *views* a výsledkem bude zobrazení obalených *views* horizontálně, respektive vertikálně za sebou. S těmito *views* úzce souvisí verze s prefixem Lazy. `LazyHStack` a `LazyVStack` se od svých běžných verzí liší tím, že vykreslení prvků, které obalují, proběhne právě tehdy, když jsou na obrazovce pozorovatelné.
- `ZStack` se chová stejně jako `VStack` nebo `HStack` s tím rozdílem, že se prvky zobrazují a rovnají ve směru osy z.

`LazyHGrid`/`LazyVGrid` organizují obalené *views* do mřížky, která roste horizontálně, respektive vertikálně a vykresluje prvek, pouze pokud je třeba.

`Spacer` je speciální struktura, která se svým obsahem roztáhne na maximální možný rozměr ve směru osy kontejneru, kterého je součástí. Při tvorbě UI je `Spacer` užíván k vyplnění místa v okolí jiných prvků, vytvořením prázdného prostoru nebo k vynucení specifického zarovnání ostatních prvků.

`Divider` je struktura, která tvoří přímku pro vizuální oddělení okolních prvků v rámci obrazovky.

```
1  var body: some View {
2      VStack {
3          HStack {
4              Text("Hello")
5
6              Spacer()
7
8              Text("World!")
9          }
10
11         Divider()
12     }
13 }
```

Listing 6: Ukázka vytvoření vlastního *view*, který vykreslí pozdrav Hello World! s prázdným horizontálním prostorem mezi slovy. Pod `HStackem` se vykreslí od okraje k okraji horizontální linka.

Collection Containers Kontejnery kolekcí jsou pokročilejší než Layout Containers a slouží k dynamickému seskupování prvků. Tyto kontejnery mají většinou výchozí chování a dají se snadno konfigurovat přidáním dalších vlastností jako je například přejetí prstem.

Níže uvedené kontejnery jsem vybral jako ukázku kontejnerů ze skupiny Collection Containers.

`List` je kontejner určený pro prezentaci dat ve sloupci. Tato struktura umožňuje snadné vybírání prvků, *pull to refresh*, mazání posunutím prvku doleva a další volitelné možnosti.

`ScrollView` je struktura umožňující scrollování nad jinou strukturou. ScrollView umožňuje nastavit scrollování horizontální, vertikální i kombinované, ale neumožňuje zvětšování. Kromě nastavení osy

umožňuje také skrytí ukazatele scrollování změnou vlastnosti `showsIndicators`.

`Group` umožňuje spojení více pohledů do jedné instance. První dojem z této struktury může být, že je zbytečná, ale její uplatnění najdeme například ve *views*, které by jinak měly více než 10 *subviews*. 10 je totiž maximální počet obalených *views* v jednom pohledu.

`ForEach` je struktura, která generuje *views* z kolekce, která je předána při incializaci. Elementy předané kolekce musí buď konformovat k protokolu `Identifiable`[9] anebo musí být při inicializaci poskytnut parametr `id`, který bude pro identifikaci použit.

```
1  private let names = ["Lumír", "Slavomír", "Břetislav"]
2
3  var body: some View {
4      VStack {
5          ForEach(names, id: \.self) { name in
6              Text(name)
7                  .font(.largeTitle)
8          }
9      }
10 }
```

Listing 7: Ukázka vytvoření vlastního *view*, který pod sebe vykreslí jména z proměnné `names`.

Presentation Containers Poslední ze skupin kontejnerů slouží k navigaci napříč hierarchií aplikace. Uživateli tato sada umožňuje průchod jednotlivými obrazovkami a navigační prvky jsou rozmístěny standardně tak, jak je uživatel iOS zvyklý. Níže popsané struktury slouží pro ukázku toho, jaké kontejnery můžeme v této skupině očekávat.

`TabView` je *view*, který umožňuje přepínání mezi více podřízenými pohledy. Toho dociluje pomocí vykreslení typické lišty s ikonami a názvy ve spodní části obrazovky telefonu.

`NavigationView` je struktura, pomocí které lze vytvořit zásobník pohledů, který reprezentuje cestu uživatele napříč navigační hierarchií pohledů. Přechod na novou obrazovku se realizuje pomocí `NavigationLink`. Po aktivování tohoto speciálního odkazu se obrazovka dostane na vrchol zásobníků a uživatel se zpět na původní obrazovku vrátí například tlačítkem zpět. Tlačítko je typicky umístěno v horním levém rohu obrazovky telefonu, ale způsobů přechodu

[9] Pro konformování k protokolu je třeba poskytnout id, které bude zaručeně unikátní.

mezi obrazovkami je více. Struktura `NavigationView` má mnoho modifikátorů jako je například `navigationBarBackButtonHidden`, `navigationTitle` a další.

`NavigationLink` je struktura sloužící jako odkaz, který po aktivování přidá *view* do navigačního zásobníku a umožní vykreslení daného pohledu. Pro správnou funkčnost je nutné, aby byl `NavigationLink` uvnitř `NavigationView`. Aktivace linku může proběhnout buď kliknutím na odkaz anebo provázáním s proměnnou typu `Bool` – odkaz se aktivuje v případě, že je hodnota nastavena na `true`.

```
1  struct TrainstationView: View {
2      var body: some View {
3          NavigationView {
4              VStack {
5                  Text("Nástupiště 9")
6
7                  Button(action: {}) {
8                      NavigationLink(destination: { Text("Nástupiště 9 a ¾")
                       ↪ }, label: {
9                          Text("Zed' mezi nástupištěmi")
10                     })
11                 }
12                 .padding()
13
14                 Text("Nástupiště 10")
15             }
16         }
17     }
18 }
```

Listing 8: Ukázka konstrukce jednoduché navigace. Tlačítko na řádku 7 odkazuje na novou obrazovku, na které bude vykreslen text Nástupiště 9 a ¾.

Rozložení komponent ale není vše, co lze ovlivnit. Framework umožňuje sledování událostí, jako jsou gesta, vícenásobná kliknutí, změna sledovaných stavů a mnoho dalších funkcí. Výhodou je i integrace animací, které se dají různě kombinovat a jejich zakomponování při změnách stavů není složité.

Property wrappers

Nezbytnou součástí SwiftUI jsou také *property wrappers* – dekorátory. Paul Hudson v článku o dekorátorech ve SwiftUI [24] tvrdí, že mohou zpočátku na leckoho působit svým zápisem nepřehledně, ale bez nich by kód nebyl

ani zdaleka tak dobře čitelný a udržitelný. Dekorátory vždy začínají symbolem @ a přidávají původnímu elementu funkcionalitu navíc. Nejběžnějším dekorátorem, se kterým se vývojář ve SwiftUI setká, je `@State`, díky kterému můžeme modifikovat proměnné ve strukturách. SwiftUI povoluje pouze jeden dekorátor nad elementem a jejich řetězení není možné. Vývojáři je umožněno definovat si i svůj vlastní dekorátor, který je v kódu možné použít.

Vysvětlení jednotlivých dekorátorů je výrazně jednodušší za pomoci ukázek funkčního kódu. Z tohoto důvodu se tomuto tématu budu hlouběji věnovat až v kapitole 5, kde jejich využití a důvod volby vysvětlím na ukázkách.

Nevýhody SwiftUI

První nevýhodou je dostupnost frameworku pouze pro macOS, což u Applu bohužel není nic neobvyklého. Dále bych mezi nevýhody SwiftUI zařadil samotnou dokumentaci, která je často velmi nekonzistentní co se informací týče. V některých případech je problém popsán detailně, ale velmi často je nezbytné čerpat z několika jiných zdrojů pro porozumění problematiky, protože je popis velmi stručný až mizivý. Tato skutečnost je pravděpodobně zapříčiněna tím, že je framework stále poměrně nový/mladý a věřím, že se do budoucna dokumentace zlepší.

Dále stojí za zmínku fakt, že některé komponenty, které byly běžnou součástí UIKitu, ve SwiftUI chybí a vývojář je nucen k importování komponenty ze staršího frameworku. Na tento problém jsem narazil u přidávání fotografií z galerie, kdy jsem byl nucen implementovat tuto funkcionalitu pomocí UIKitu.

Alternativy SwiftUI

SwiftUI má zcela jistě své nedostatky a menší pokrytí API než UIKit, ale i přesto pro mě byl od začátku první volbou z důvodu svého inovativního deklarativního přístupu. Chtěl jsem řešení přímo od Applu, čímž vypadla alternativa v podobě React Native, který je stejně jako SwiftUI deklarativní. UIKit je také framework od společnosti Apple, ale osobně mě odradil svým imperativním přístupem, který je komplikovanější a méně přímočarý. UIKitu se ale rozhodně nedá upřít jeho velká výhoda, kterou je jeho stáří. V případě, že narazíme na nějaký větší problém při tvorbě aplikace, je u staršího a osvědčeného frameworku výrazně jednodušší najít řešení problému. Je totiž daleko větší šance, že už se někdo s podobným problémem dříve setkal.

Kapitola 4

Návrh

V této kapitole se budu věnovat vhodné architektuře aplikace, rozeberu její jednotlivé části včetně data mockingu a v neposlední řadě se budu zabývat návrhem uživatelského rozhraní.

4.1 Architektura

„Architektura popisuje návrh z pohledu celé aplikace“ [25] a cílem vícevrstvé architektury je separovat odpovědnost do více vrstev. Jako vhodnou architekturu pro aplikaci jsem vybral dvouvrstvou architekturu klient-server, ve které budou klient a server vzájemně komunikovat přes počítačovou síť pomocí internetového protokolu HTTPS. Server se stará o persistenci, ověřování a aktuálnost dat a na základě požadavků klienta poskytuje data, která si klient vyžádal. Klient obdržená data prezentuje uživateli a umožňuje mu s nimi interagovat. V této práci se detailněji zaměřuji na klientskou část aplikace, která je důkladně zanalyzována v další podsekci.

4.1.1 Klientská část

Hlavní náplní této práce je klientská část aplikace a volba vhodné architektury klientské části je velmi důležitá hned z několika důvodů. Správně zvolený architektonický vzor klienta myslí na budoucnost aplikace, umožňuje její škálování a snadnější přidávání nových funkcionalit. Software se pak výrazně lépe udržuje, testuje a ve zdrojovém kódu je snadnější se zorientovat. Vzhledem ke zvolenému frameworku jsem se rozhodl pro architektonický vzor MVVM, který popisuji v další podsekci.

MVVM

Architektonický vzor MVVM je zkratka pro Model View ViewModel. Tento vzor je velmi podobný známému vzoru MVC – Model View Controller – který

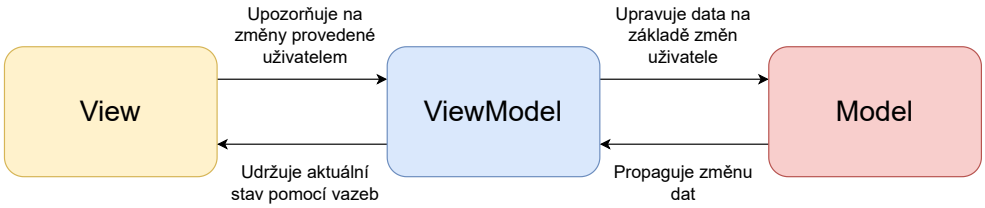
byl typickou volbou u UIKitu, a jeho cílem je oddělit logiku od prezentační vrstvy UI. Pro důkladné vysvětlení tohoto vzoru jsem čerpal z článku Vadima Bulavina [26], Aasifa Khana [27] a videa Paula Hegartyho pro Standford University [28]. Architektura MVVM se dá rozdělit do tří vrstev:

`View` zobrazuje prvky uživatelského rozhraní a stará se i o jejich případnou animaci. Dále zaznamenává aktivitu a vstupy uživatele, které pomocí obousměrných vazeb předává `ViewModelu`. Data k zobrazení „dostává“ od `ViewModelu` a o jejich změnách je informován. Po obdržení změn se view automaticky s aktuálními daty překreslí do správného stavu.

`ViewModel` obsahuje prezentační logiku a data, které `View` zobrazuje. Data převzatá z `Modelu` strukturuje tak, aby jejich zobrazení ve `View` bylo co možná nejjednodušší a nebylo třeba žádných konverzí nebo úprav. Zároveň se stará o udržování aktuálního stavu dat, synchronizaci a změny uživatele propaguje do `Modelu`. Tato komponenta je prostředníkem mezi `Modelem` a `View`.

`Model` představuje datovou část aplikace a stará se o zápis/čtení dat ze serveru nebo databáze – *backendu*. Získaná data ze serveru (například ve formě JSONu[10]) nebo databáze převádí na struktury, se kterými je aplikace schopna snadno pracovat.

MVVM se snaží minimalizovat rozhodovací logiku ve *views* a většinu stavů přesunout do *viewmodelů*, čímž se *views* stávají pasivními. Datový tok mezi všemi komponentami je vyobrazen na diagramu 4.1 níže.



Obrázek 4.1: Diagram architektury MVVM

4.1.2 Serverová část

Serverová část aplikace není hlavní náplní této práce, ale i přesto je potřeba ji alespoň částečně pokrýt, protože by s jejími vlastnostmi mělo korespondovat

[10] JavaScript Object Notation, zkráceně JSON je jednoduchý formát pro ukládání a přenos dat.

chování klientské části aplikace – prototypu. Server by měl splňovat požadavky REST API, mezi které David Bernhauer [29] řadí:

- Architektura klient-server pro separaci zodpovědností
- Vrstvená architektura serveru pro snadnou rozšířitelnost
- Uniformní rozhraní založené na zdrojích
- Bezestavovost – požadavek klienta musí obsahovat všechny informace
- *Cacheovatelnost* – možnost dočasného uložení často čtených dat

Server by měl být implementován pomocí vhodně zvolených technologií tak, aby byla doba mezi vyhodnocením požadavku klienta a zasláním odpovědi co nejkratší. Komunikace po síti by měla probíhat pomocí šifrovaného protokolu HTTPS a data by měla být zasílána ve formátu JSON, který se ve Swiftu velmi snadno dekóduje a *parsuje* pomocí JSONDecoder[11]. Dle Patniho Sanjaye [30] je JSON navíc oproti staršímu formátu XML lépe čitelný a datově úspornější.

4.2 Data Mocking

Aby prototyp pouze nezobrazoval statická data, která není možné uživatelem přepsat či upravit, bude využito datového mockingu. Pro představení této techniky čerpám z článku Rosie Hamilton [31] a Joãa Carreiry [32]. Mock data jsou uměle vložena do softwaru a jedná se o data, která nejsou skutečná. Jejich chování je kontrolované a díky tomu umožňují simulovat různé stavy aplikace, které mohou nastat, a vynutit i méně obvyklé stavy. Mocking serverové části aplikace přináší výhody jako je zaručená dostupnost *mockovaného* API, snadné manipulování samotných dat nebo nezávislost na rychlosti internetového připojení při načítání dat. Pro otestování případu nízkorychlostního připojení je možné simulovat asynchronním voláním libovolně dlouhé časové zpoždění.

V případě Fotofolia nebude využit žádný software pro generování dat a všechna testovací data v aplikaci budou vytvořena mnou. Většina generátorů dat nepodporuje český jazyk a vygenerovaná data navíc bývají velmi obecná. Proto jsem se rozhodl pro Fotofolio zformovat data, která budou autentická a navíc i v češtině – stejně jako UI prototypu – aby byl uživatelský zážitek při testování co nejlepší. Implementačním detailům mockování dat pro tento konkrétní projekt a simulaci zpoždění načítání se věnuji v podsekci 5.2.

[11] https://developer.apple.com/documentation/foundation/jsondecoder

4.3 Návrh uživatelského rozhraní

Aplikace může mít sebelepší funkce, ale pokud se v ní uživatel snadno nezorientuje a navigace napříč aplikací bude složitá, nebude ji používat. Špatně vymyšlené uživatelské rozhraní může být pro uživatele frustrující a může ho kompletně odradit od dalšího používání aplikace. Současná doba je velmi rychlá až hektická, lidé chtějí mít všechno hotové co nejrychleji a jsou zvyklí na pohodlí. Uživatel/zákazník je vždy na prvním místě a na vývojáři zůstává zodpovědnost, aby navrhl svou aplikaci jak nejlépe je to jen možné. Pro maximalizování uživatelského zážitku je dobré dodržovat zvyklosti cílové platformy a vycházet z principů, které jsou ověřené a funkční. Mezi tyto principy bezpochyby patří 10 heuristik použitelnosti Jakoba Nielsena [33], kterými se při návrhu můžeme řídit:

1. **Viditelnost stavu systému** – uživatel by měl mít neustále přehled o tom, co se v aplikaci děje.
2. **Shody mezi systémem a skutečným světem** – návrh by měl mluvit jazykem uživatele a využívat prvků reálného světa.
3. **Uživatelská kontrola a svoboda** – uživatel by měl mít stále pocit kontroly/jistoty a v případě, že udělá chybu, by měl mít možnost proces snadno přerušit.
4. **Konzistence a standardy** – uživatel by se neměl dostat do pozice, kdy přemýšlí nad významem slov nebo akcí.
5. **Prevence chyb** – dobrý design by měl chybám předcházet a snažit se je nepřipustit.
6. **Rozpoznání spíš než vzpomínání** – uživatel by si toho měl pamatovat co nejméně a informace by měly být snadno dostupné a viditelné.
7. **Flexibilita a efektivita použití** – uživatelům by měl design poskytovat prostředky pro zefektivnění používání aplikace a usnadnit akci, kterou potřebují provést.
8. **Estetický a minimalistický design** – UI by mělo zobrazovat jenom to, co je opravdu potřeba a nezahlcovat uživatele zbytečnostmi.
9. **Pomoct uživateli rozpoznat a zotavit se z chyb** – chybové hlášky by měly být srozumitelné a navrhovat řešení problému.
10. **Dokumentace a pomoc** – systém by měl být navržený tak, aby nepotřeboval žádné další vysvětlování, ale dokumentace nebo nápověda k úspěšnému provedení akce – krátké vysvětlivky, průvodce apod. – může uživateli přijít vhod.

Při vizualizaci uživatelského rozhraní pomocí *wireframů* – drátěných modelů, ve kterých jsou rozvrženy ovládací prvky obrazovky – se můžeme vydat dvěma směry, popsanými Nickem Babichem [34]. Low fidelity wireframe má nízkou podobnost s finální aplikací a jeho hlavní náplní je ukázat funkčnost a rámcové rozložení prvků. Důraz je tedy kladen na rychlost jeho vytvoření a pokrytí funkcionalit – estetická stránka věci není prioritou a řešena je minimálně. Opakem Low fidelity je High fidelity wireframing, který je velmi podobný finálnímu produktu a kromě samotné funkčnosti je zde kladen i velký důraz na estetickou stránku uživatelského rozhraní. Mezi dvěma extrémy leží ještě zlatá střední cesta – Medium fidelity [35]. U Fotofolia jsem se rozhodl pro cestu Medium fidelity, tedy velkou podobnost s výslednou implementací s prostorem pro změny a minimalističtějším zobrazením některých elementů.

Aplikace bude rozdělena na 5 hlavních sekcí, mezi kterými se bude přepínat spodní lištou, která je součástí `TabView`. Za 6. sekci můžeme považovat i přihlášení a registraci, která bude řešena formou překryvu celé obrazovky.

Feed portfolií

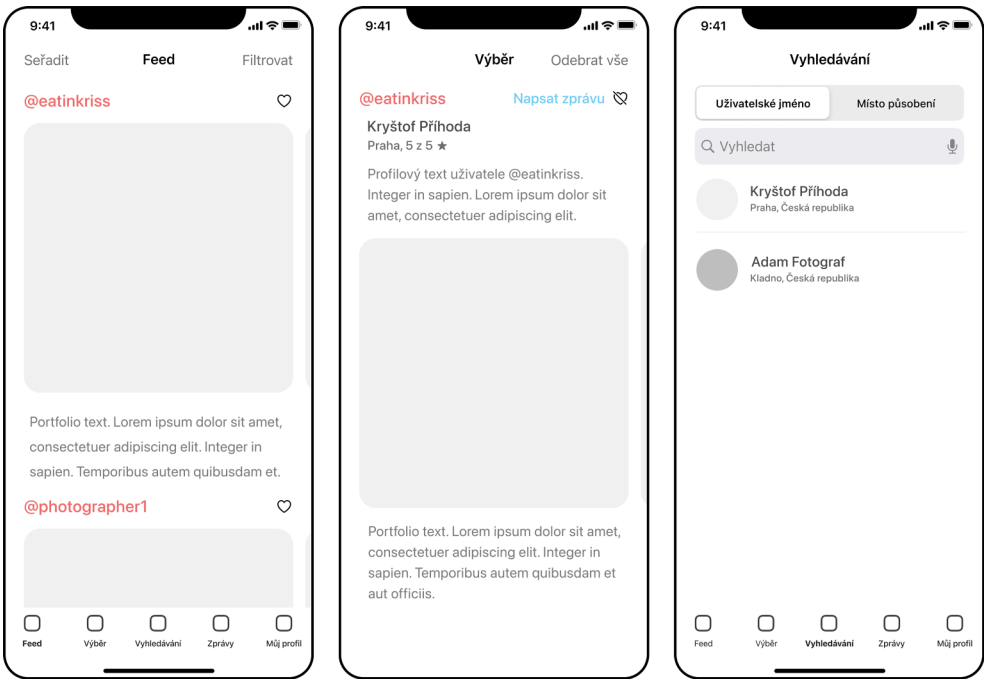
Výchozí obrazovka, která se zobrazí uživateli po přihlášení, bude feed portfolií. Hlavní doménou této obrazovky budou zobrazovaná portfolia. Každým portfoliem bude moci uživatel horizontálně scrollovat pro zobrazení všech fotografií. Podporovaný formát fotografií bude čtvercový. Společně s portfoliem uživatel uvidí také uživatelské jméno autora, popis portfolia a tlačítko, pomocí kterého může přidat portfolio do svého výběru portfolií. Záměrně se u portfolia bude zobrazovat pouze uživatelské jméno, popis a fotografie portfolia, aby se uživatel rozhodoval pouze na základě finálního produktu fotografa – portfolia. Kromě toho, že feed nediskriminuje, je i čistý a minimalistický. Po kliknutí na uživatelské jméno se uživatel dostane na profil autora portfolia – obrázek 4.4. V `NavigationBaru` se budou nacházet ovládací prvky pro řazení (vlevo nahoře) a filtrování portfolií (vpravo nahoře). Feed je vyobrazen na obrázku 4.2 vlevo.

Vybraná portfolia

S feedem portfolií úzce souvisí druhá sekce, kterou je obrazovka vybraných portfolií. Kromě uživatelského jména a popisu budou u portfolia zobrazeny i další informace o autorovi – celé jméno, místo působení, průměrné hodnocení na škále 1-5 a profilový text fotografa. Tato obrazovka totiž bude reprezentovat předfinální fázi procesu poptávky fotografa před jeho potenciálním oslovením. Uživateli bude umožněno přejít do přímých zpráv s autorem, kde může dojít k poptávce a další nezbytné komunikaci ohledně focení. I zde se po kliknutí na uživatelské jméno autora dostane uživatel na jeho profil. Ve fázi výběru bude možné odebrání konkrétního portfolia pro zúžení výběru, ale i odebrat všechna portfolia z výběru tlačítkem v pravém horním rohu. Obrazovka vybraných portfolií je znázorněna na obrázku 4.2 uprostřed.

Vyhledávání uživatelů

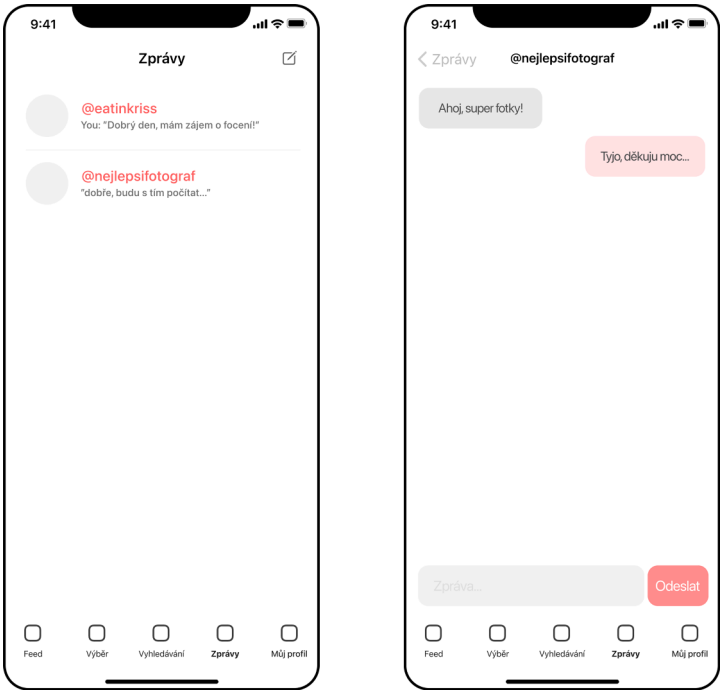
Pro snadné vyhledávání uživatelů bude sloužit třetí sekce na spodní liště. Uživatel bude moci vyhledávat napříč všemi uživateli podle jména a místa působení. Režim vyhledávání si uživatel zvolí pomocí jednoduchého přepínače. U jednotlivých výsledků se bude zobrazovat jméno uživatele, místo působení a profilová fotografie pro snazší identifikaci. Kliknutím na jeden z výsledků se bude uživatel moci dostat na daný profil.



Obrázek 4.2: Wireframe feedu dostupných portfolií (vlevo), uživatelem vybraných portfolií – předfinální fáze oslovení fotografa (uprostřed) a vyhledávání uživatelů (vpravo)

Zprávy

Každé focení má svá specifika, která se ne vždy dají kategorizovat a z tohoto důvodu je zde sekce zpráv. Ve zprávách bude moct zákazník s fotografem dohodnout všechny detaily týkající se focení. Obrazovka bude zobrazovat přehled všech aktuálních chatů s uživateli a po rozkliknutí se uživatel dostane do přímých zpráv s daným uživatelem, kde se mu zobrazí všechny zprávy a možnost novou zprávu vytvořit a odeslat. Pro odeslání zprávy novému uživateli bude sloužit tlačítko v pravém horním rohu `NavigationBaru`, které otevře novou obrazovku s vyhledáváním. Následným kliknutím na uživatelův profil mezi výsledky vyhledávání se otevře obrazovka přímých zpráv.

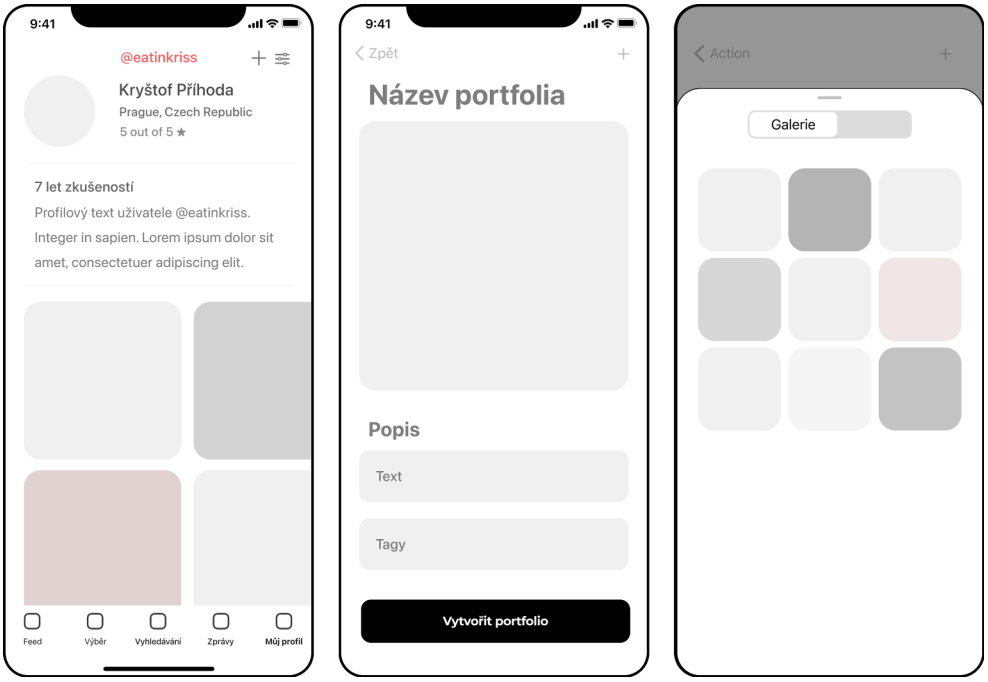


Obrázek 4.3: Wireframe zpráv a otevřeného chatu

Uživatelský profil

Poslední sekcí je uživatelův osobní profil. Pokud uživatel není zaregistrován v režimu tvůrce, budou se zde zobrazovat pouze základní informace. Režim tvůrce toho nabídne podstatně víc a kromě základních informací zde budou i uživatelova portfolia. Portfolia se budou zobrazovat horizontálně a bude možné fotografiemi horizontálně scrollovat. Tedy jeden řádek odpovídá jednomu portfoliu. Oproti běžnému profilu zde bude navíc i profilový popis a doba

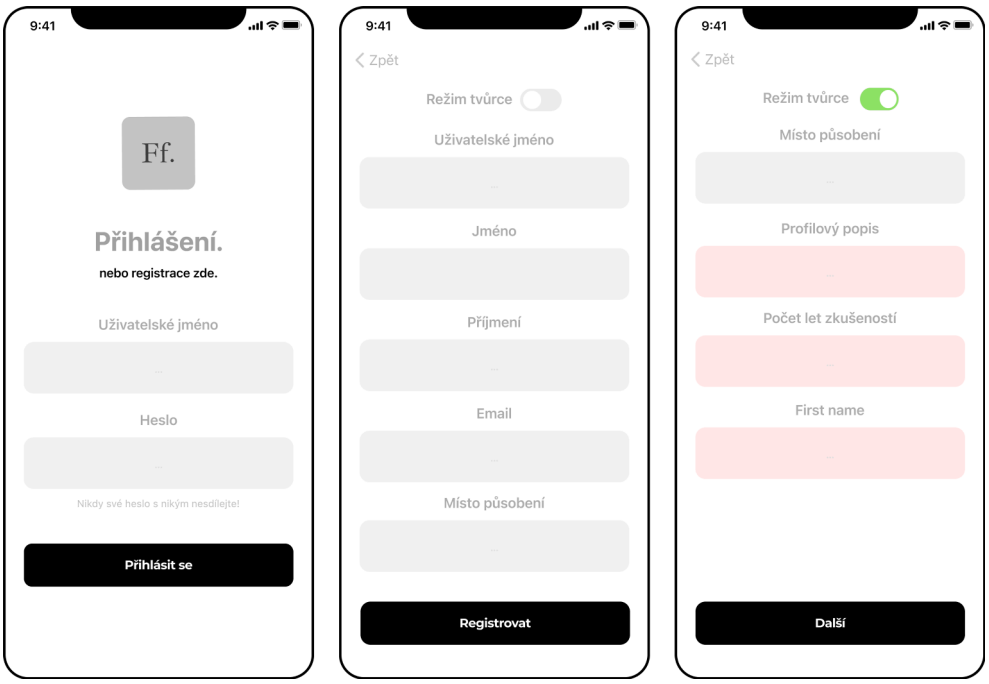
zkušeností s fotografováním. V `NavigationBaru` se uživatel dostane k nastavení svého profilu, kde bude moci upravovat nejen své osobní údaje a profilový obrázek, ale i portfolia. Úpravou portfolia je myšleno přidávání nových fotografií, změna popisu portfolia, úprava tagů nebo jeho úplné smazání. Počet tagů bude omezený na 5, aby se vždy jednalo pouze o co nejrelevantnější klíčová slova. Kromě úprav portfolií bude možné z této obrazovky i vytvoření nového portfolia. K vytvoření se uživatel bude moci prokliknout na novou obrazovku zmáčknutím tlačítka „plus“, kde bude možné přidání nových fotografií, popisu a tagů portfolia. Dále bude na profilu umožněno odhlášení z účtu pomocí vybrání této možnosti v `NavigationBaru`.



Obrázek 4.4: Wireframe uživatelského profilu a vytváření nového portfolia

Přihlášení a registrace

Za šestou sekci se dá považovat přihlašovací obrazovka, která uživatele uvítá hned po spuštění aplikace. Pokud je uživatel registrován, zadáním přihlašovacího jména a správného hesla obrazovka zmizí. Pokud registrován není, bude proveden jednoduchým procesem vyplnění údajů – uživatelské jméno, celé jméno, emailová adresa a místo působení. Zmíněné údaje jsou nezbytné k vytvoření obou typů účtu. Pokud se bude chtít uživatel registrovat v režimu tvůrce, pomocí ovládacího prvku `Toggle` se přepne do režimu tvůrce a navíc vyplní pole profilového popisu a dobu, po kterou je fotografem.



Obrázek 4.5: Wireframe přihlášení a registrace

Kapitola 5

Implementace

V této kapitole se budu zabývat implementačními detaily prototypu, frameworkem a jeho konkrétním využití ale i tím, jak je vyřešený data mocking a simulace časového zpoždění při načítání dat. Dále popíšu jednotlivé obrazovky, jejich funkcionality a zapracování architektonického vzoru MVVM.

5.1 Struktura projektu

Adresářová struktura projektu kopíruje vrstvy architektury MVVM a vypadá následovně:

- `Fotofolio`
  - `Model` ..... obsahuje soubory modelové vrstvy aplikace
  - `ViewModel` ..... zahrnuje soubory vrstvy ViewModelu
  - `View` ..... adresář jednotlivých obrazovek
    - `Util` ..... adresář sdílených souborů a deklarací
    - `Authentication` ..... adresář obrazovky přihlášení a registrace
    - `Feed` ..... adresář obrazovky feedu portfolií
    - `Selection` ..... adresář obrazovky vybraných portfolií
    - `Search` ..... adresář obrazovky vyhledávání
    - `Messages` ..... adresář obrazovky zpráv
    - `Profile` ..... adresář obrazovky uživatelského profilu

5.2 Model

Modelová vrstva pokrývá celkem 4 hlavní celky, které bylo nutné datově zareprezentovat – uživatel, fotografie/obrázek, portfolio a chat/zpráva. Kromě reprezentace dat bylo potřeba vyřešit i datový mocking kvůli absenci backendu. Testovací data a jejich chování jsem se rozhodl vyřešit bez použití externích knihoven pomocí `Extension` a voláním asynchronní metody `asyncAfter`, která

mi umožnila simulovat časové zpoždění, které je u načítání dat z *endpointů* API běžné.

Reprezentace obrázků

Pro reprezentaci obrázků bylo nutné vyřešit problematiku dvojího typu zdroje obrázku. V rámci prototypu jsou profilové obrázky a fotografie generovány službou PlaceIMG[12] a LoremFlickr[13], které nabízí volně dostupné aplikační rozhraní pro poskytování fotografií. Stačí specifikovat adresu ve tvaru https://placeimg.com/ŠÍŘKA/VÝŠKA/KATEGORIE a server vrátí jako odpověď náhodně vybranou fotografii zadaného rozměru a kategorie. Pokud ale uživatel přidává obrázek lokálně ze své galerie, fotografie může být datového typu `Image` anebo staršího typu z UIKitu – `UIImage`. Z tohoto důvodu jsem se rozhodl vyřešit implementaci výčtovým typem s *associated values*, zabalenou do struktury konformující k protokolu `Identifiable`. Enum rozlišuje dva stavy – remote s typem `String` a local s typem `Image`.

```
1  struct IImage: Identifiable {
2      let id = UUID()
3      var src: MyImageEnum
4  }
5
6  enum MyImageEnum {
7      case remote(String)
8      case local(Image)
9  }
```

Listing 9: Zdrojový kód reprezentace obrázků.

Uživatel

Oba typy uživatele jsou reprezentovány strukturou User, která konformuje k protokolu `Identifiable` pro snadnější iterování v prezentační vrstvě aplikace. Identifiable požaduje vlastnost `id`, pro kterou jsem vybral typ `UUID` – Universally Unique Identifier – u kterého je pravděpodobnost kolize v rámci UUID verze 4 velmi malá [36]. Tato struktura obsahuje vlastnosti reprezentující základní osobní údaje uložené pomocí typu `String`. V rámci prototypu je heslo uloženo jako hashValue řetězce typu `Int`. Heslo by nikdy nemělo být ukládáno jako plaintext/řetězec a tato hash by byla při produkčním nasazení nahrazena kvalitnější a bezpečnou hashovací funkcí jako je např. SHA-256 implementovanou v serverové části aplikace. Profilový obrázek uživatele je reprezentován strukturou IImage, která je popsána v předchozí podsekci 5.2. Hod-

[12] https://placeimg.com/
[13] https://loremflickr.com/

nocení uživatele reprezentuje *dictionary* typu `String` a `Int` – klíč a hodnota odpovídají uživatelskému jménu a celému kladnému číslu hodnocení. Kromě atributů/vlastností jsou součástí struktury dvě metody – `calculateRating` pro výpočet průměrného hodnocení a `isCreator`, která vrací `true` v případě, že je uživatel tvůrcem a naopak.

```
struct User: Identifiable {
    let id: UUID
    var username: String
    var fullName: String
    var password: Int
    var email: String
    var location: String
    var profilePicture: IImage?
    var ratings: [String:Int]
    var creator: Creator?

    func calculateRating() -> Double {
        if ratings.isEmpty { return 0 }

        let sum = ratings.values.reduce(0, +)
        return Double(sum) / Double(ratings.count)
    }

    func isCreator() -> Bool {
        return creator != nil
    }
}
```

Listing 10: Zdrojový kód struktury User.

Vlastnosti tvůrce jsou ve struktuře uživatele implementovány kompozicí – strukturou `Creator`, která obsahuje navíc profilový text a počet let zkušeností fotografa. Vše je uloženo pod typem `Optional`, který umožňuje uložení hodnoty `nil` v případě běžného uživatele. Volba tohoto návrhového vzoru počítá s budoucími rozšířeními u režimu tvůrce a umožňuje bezproblémové přidání nových funkcionalit a rozšíření.

```
struct Creator {
    var profileText: String
    var yearsOfExperience: Int
}
```

Listing 11: Zdrojový kód struktury Creator.

Testovací data jsou vytvořena pomocí rozšíření původních struktur. V rámci rozšíření jsou zadefinovány statické proměnné a pole všech fiktivních testovacích dat, se kterým se dále pracuje ve vrstvě `ViewModelu`.

```
let PROFILE_PIC = "https://placeimg.com/320/320/person"

extension User {
    static let sampleData: [User] = [dummy1, ..., dummyN]

    static var dummy1: User {
        User(id: UUID(), username: "vojtafoti", fullName: "Vojta Votruba",
             password: "heslovo".hashValue, email: "vojtavo@mail.com",
             location: "Praha",
             profilePicture: IImage(src: .remote(PROFILE_PIC)),
             ratings: ["ad.fotograf": 5, "michal1": 4],
             creator: Creator.dummy1)
    }

    // ...
}
```

Listing 12: Zdrojový kód rozšíření definující testovací data uživatele.

Portfolio

I Portfolio je struktura konformující k protokolu `Identifiable`, ale navíc konformuje i k `Equatable` pro snadnější možnost porovnávání portfolií v kódu – z tohoto důvodu je přetížen operátor `==`. Portfolio má dále atributy pro uživatelské jméno autora, jméno portfolia, popisek portfolia a pro tagy/klíčová slova pole typu `String`. Fotografie jsou reprezentovány jako pole vlastního datového typu `IImage` a poslední vlastností je časový údaj o vytvoření portfolia typu `Date`. Testovací data jsou opět vytvořena pomocí `Extension` a statického pole všech *dummy* portfolií.

```
struct Portfolio: Identifiable, Equatable {
    let id: UUID
    var authorUsername: String
    var name: String
    var photos: [IImage]
    var description: String
    var tags: [String]
    let timestamp: Date

    static func == (lhs: Portfolio, rhs: Portfolio) -> Bool {
        lhs.id == rhs.id && lhs.authorUsername == rhs.authorUsername
    }
}
```

Listing 13: Zdrojový kód struktury reprezentující portfolio.

Zprávy

Reprezentace zpráv je vyřešena strukturami `Message` a `Chat`. Struktura zprávy konformuje k `Identifiable` a `Equatable` ze stejných důvodů jako Portfolio. Kromě nutné vlastnosti `id` struktura zahrnuje informace o uživatelských jménech odesílatele a příjemce, těle zprávy a času odeslání zprávy ve formátu `Date`. Chat je struktura zaobalující jednotlivé zprávy, také konformuje k protokolu `Identifiable` a drží si informaci o obou svých vlastnících typu `User`. Testovací data jsou zadefinována v rozšíření struktur pomocí statických proměnných.

```
struct Message: Identifiable, Equatable {
    let id = UUID()
    var from: String
    var to: String
    var body: String
    let timestamp: Date
}

struct Chat: Identifiable {
    let id = UUID()
    var chatOwners: [User]
    var messages: [Message]
}
```

Listing 14: Zdrojový kód struktur implementujících zprávy.

5.3 ViewModel

Vrstva ViewModelu je implementována pomocí celkem 7 separátních tříd konformujících k protokolu `ObservableObject`. Tento protokol je ideální volbou pro jednotlivé ViewModely, protože umožňuje upozorňování na konkrétní změny dat v instanci třídy a publikování těchto změn do souvisejícího view. Aby proces publikování změn proběhl, všechny vlastnosti, které mají zapříčinit překreslení obrazovky, jsou v kódu v rámci třídy označeny dekorátorem `@Published`.

Pro simulaci zpoždění při načítání dat ze serveru využívám ve ViewModelech `DispatchQueue` a hlavní vlákno `main`, nad kterým volám metodu `asyncAfter`. Tato funkce má dva parametry – closure s kódem, který se má vykonat a doba zpoždění, po které se má předaná closure zavolat. Funkce je vždy vykonávána na hlavním vlákně, protože změny, které ovlivňují View, vždy musí být propagovány z hlavního vlákna aplikace.

```
1  class MessagesViewModel: ObservableObject {
2      @Published var chats: [Chat] = []
3
4      @Published var fetchingChats: Bool = true
5
6      func fetchChats(for username: String) {
7          DispatchQueue.main.asyncAfter(deadline: .now() + 0.43) {
8              self.chats = Chat.sampleData.filter({
9                  $0.chatOwners.contains(where: {
10                     $0.username == username
11                 })
12             })
13
14             self.fetchingChats = false
15         }
16     }
17 }
```

Listing 15: Ukázka implementace ViewModelu zpráv.

ViewModely jsou úzce provázány s jednotlivými obrazovkami a proto se v této sekci nezabývám rozborem jednotlivých tříd a upřednostnil jsem popis zajímavých sekcí a detailů této vrstvy v kontextu obrazovek v další sekci.

5.4 View

Hlavní kostra navigace napříč obrazovkami je vyřešena v `ContentView`. Každá sekce `TabView` je obalena do `NavigationView` pro přehlednou navigaci napříč

sekcí/obrazovkou. `TabView` vykresluje vespod displeje lištu, pomocí které uživatel přepíná mezi jednotlivými záložkami. Povinná vazba v parametru `TabView` je vyřešena výčtovým typem všech dostupných obrazovek pro přehlednější kód.

Kromě „navigační kostry“ a členění obrazovek je v `ContentView` inicializována většina ViewModelů, které využívají dekorátoru `@StateObject`. Ten zaručuje to, že při aktualizaci obrazovek instance nebude zničena, čímž je eliminováno potenciální spadnutí aplikace, které by hrozilo v případě použití `@ObservedObject`. Metoda `environmentObject` zajišťuje dostupnost instancí ViewModelů v rámci subhierarchií `ContentView` jako `EnvironmentObject`. `EnvironmentObject` je pro účely prototypu mimořádně praktický, protože odstraňuje nutnost předávání ViewModelu parametrem napříč hierarchií. Inicializace views je díky tomu čistší a propisování změn uživatele napříč aplikací je výrazně přímočařejší. Aplikace počítá s budoucím nasazením serverové části aplikace, se kterou zanikne nutnost propisování změn lokálně napříč ViewModely. Tím se proces značně zjednoduší, protože jediným zdrojem pravdy se stane server a data bude stačit pouze získat ze serveru vhodným požadavkem.

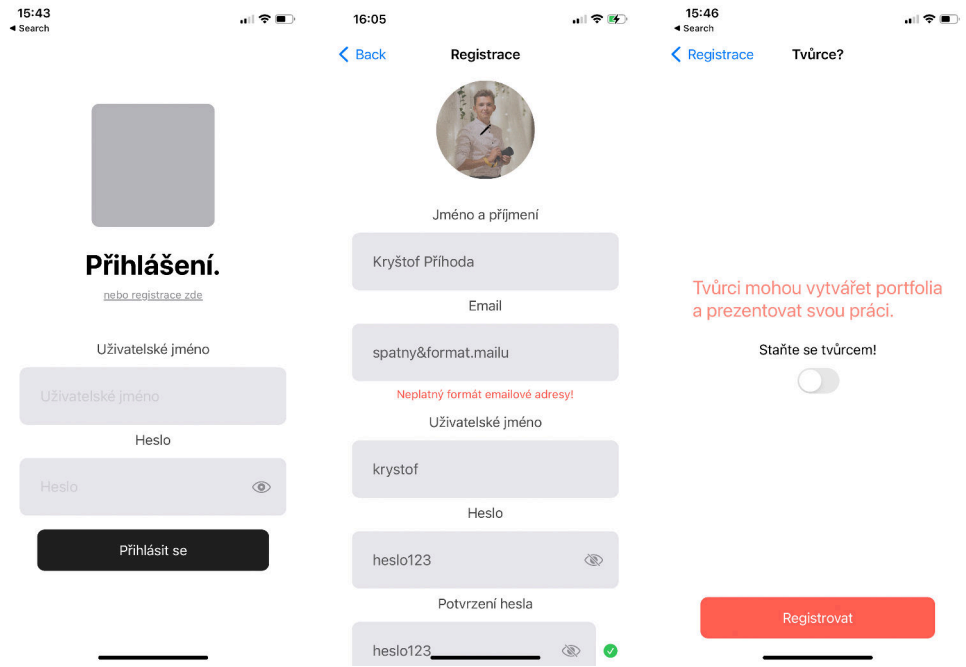
```
1  struct ContentView: View {
2      @StateObject var portfolioViewModel: PortfolioViewModel = .init()
3      // ...
4      @State var tabSelection = TabEnum.feed
5
6      var body: some View {
7          TabView(selection: $tabSelection) {
8              NavigationView {
9                  FeedView()
10             }
11             .tabItem {
12                 Image(systemName: "square.text.square")
13                 Text("Feed")
14             }
15             .tag(TabEnum.feed)
16
17             // ...
18         }
19         .environmentObject(portfolioViewModel)
20         .environmentObject(messagesViewModel)
21         // ...
22     }
23 }
```

Listing 16: Ukázka logiky aplikace a inicializace ViewModelů v ContentView.

Přihlášení a registrace

Přihlašovací obrazovka je implementována jako modální pohled `fullScreenCover` v `ContentView`, který je prezentován v případě, že uživatel není v třídě `signInViewModel` autorizován. Autorizace probíhá ve ViewModelu a kontroluje existenci uživatelského jména a správnost hesla – pokud je vše v pořádku, modální pohled se přestane prezentovat.

Registrační proces zahrnuje vyplnění základních povinných údajů. Vstupy textových polí jsou kontrolovány pro správnost povolených znaků a v případě uživatelského jména a e-mailu ošetřeny regulárními výrazy. Po dokončení registrace je uživatel vrácen na obrazovku přihlášení a je mu prezentována informace o potvrzení registrace na zadané e-mailové adrese. E-mailové potvrzení registrace není implementováno, ale po implementaci serverové části aplikace a jejím následném integrování bude nezbytnou součástí registračního procesu.



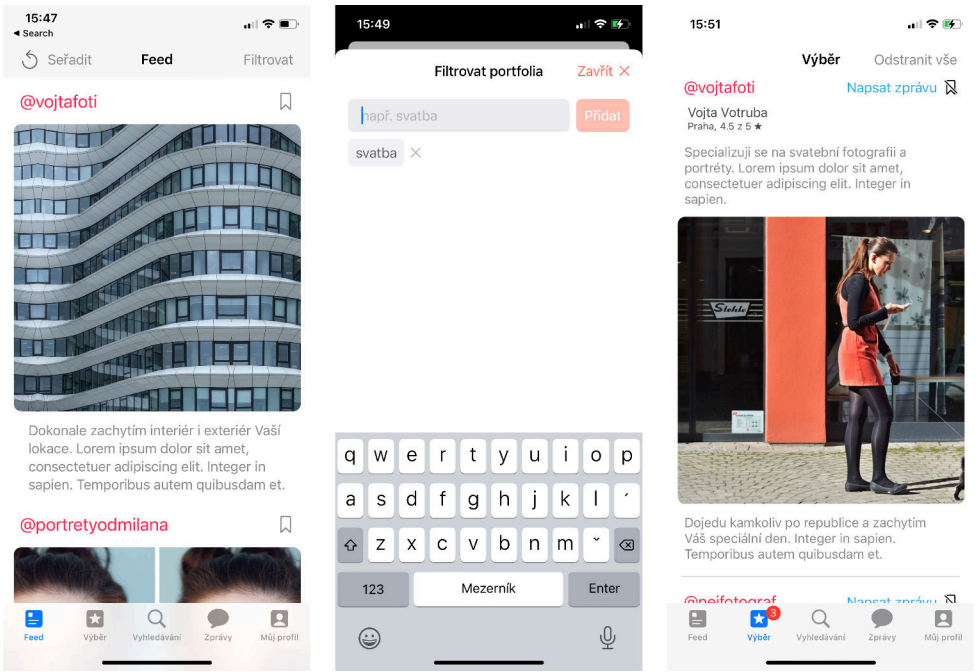
Obrázek 5.1: Snímky obrazovky přihlášení a registrace

Feed portfolií

Ve feedu je hlavní důraz kladen právě na samotná portfolia fotografů, která lze řadit podle kritérií a filtrovat pomocí tagů. Po kliknutí na ikonu záložky knihy se portfolio přidá do uživatelova výběru portfolií. Fotografie portfolií jsou implementovány pomocí `AsyncImage` a asynchronně načítají obrázky z API. Filtrování je vyřešeno pomocí `sheet`, který z většiny překrývá feed portfolií a na základě zadaných klíčových slov zobrazuje ve feedu pouze portfolia vyhovující specifikovaným tagům.

Vybraná portfolia

Obrazovka vybraných portfolií se do určité míry podobá feedu, ale uživatel už se zde může dozvědět detailnější informace o fotografovi jako je jeho průměrné hodnocení, místo působení, popis profilu a další. Tato obrazovka je předfinální fází oslovení fotografa a i zde jsou hlavním předmětem pozornosti horizontální `ScrollViews` s fotografiemi portfolií. Uživatel může vybraná portfolia z výběru odebírat, čímž se mu zužuje selekce potenciálně oslovených fotografů. Pro oslovení fotografa lze jedním kliknutím přejít do přímých zpráv s autorem portfolia nebo se kliknutím na uživatelské jméno přesunout na autorův profil, kde jsou dostupná i další fotografova portfolia.



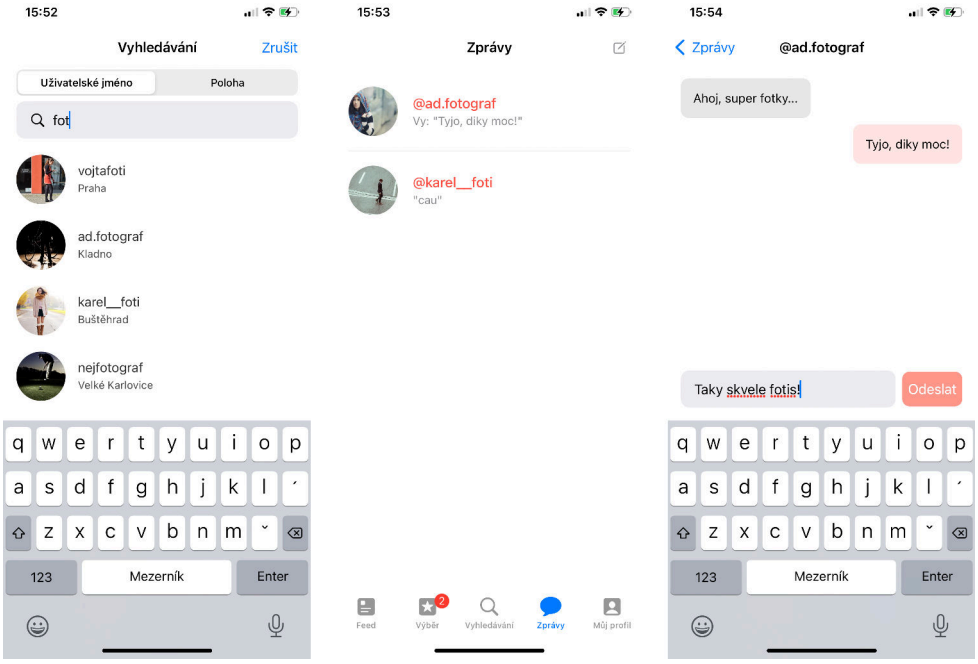
Obrázek 5.2: Snímky obrazovky feedu, filtrování a výběru portfolií

Vyhledávání

Vyhledávací obrazovka umožňuje vyhledávání konkrétních uživatelů podle uživatelského jména a polohy. Režim vyhledávání se přepíná pomocí prvku `Picker` a aktualizace výsledků vyhledávání už při psaní slova je vyřešena pomocí metody `onChange`, která je navázána na textové vyhledávací pole. Kliknutím na jeden z výsledků se uživatel dostane na daný profil – přechod je implementován pomocí `NavigationLink`.

Zprávy

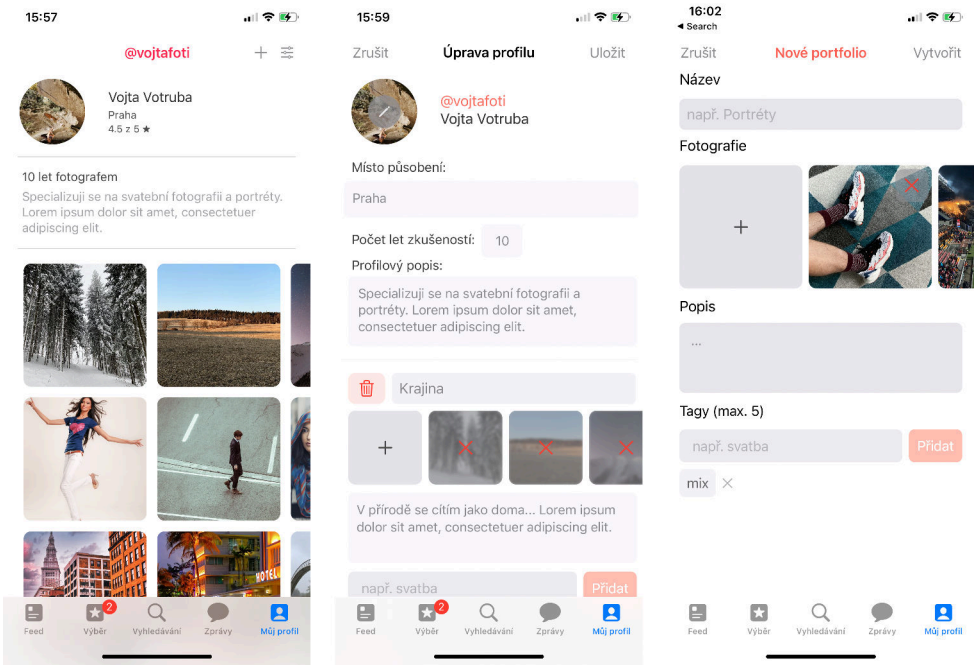
Obrazovka zpráv vykresluje chaty s ostatními uživateli a náhledy posledních zpráv. Pro vytvoření nového chatu slouží tlačítko v `NavigationBar`, které pomocí `NavigationLink` vykreslí novou obrazovku s vyhledáváním všech uživatelů, ze které se uživatel přesune do okna přímých zpráv s vybraným uživatelem.



Obrázek 5.3: Snímky obrazovky vyhledávání uživatelů a zpráv

Uživatelský profil

Uživatelský profil přehledně zobrazuje veškeré informace o uživateli včetně jeho profilové fotografie. V případě fotografa je zde navíc doba zkušeností s fotografováním, popis a hlavně mřížka portfolií, kde jeden řádek odpovídá jednomu portfoliu. Po kliknutí na libovolné portfolio se s animací zobrazí i zbývající detaily všech zobrazených portfolií profilu – názvy a popisky. Vlastní profil a portfolia je možné upravit a celý proces je bezeztrátový – veškeré změny se dají vrátit zpět. Kromě úprav je možné i vytvoření nového portfolia kliknutím na tlačítko plus. Pro přidávání fotografií z galerie bylo nutné integrovat `UIImagePickerController` pomocí `UIViewControllerRepresentable` ze staršího frameworku UIKit, protože SwiftUI tuto funkcionalitu stále neobsahuje. Místo tlačítek pro přidání portfolií a kontextového prvku `Menu` k úpravě profilu a odhlášení, se na cizím profilu zobrazuje tlačítko pro přechod do přímých zpráv a u základních informací uživatele i možnost ohodnocení profilu.



Obrázek 5.4: Snímky obrazovky uživatelského profilu, úpravy profilu a tvorby nového portfolia

Kapitola 6

Uživatelské testování

Uživatelské testování použitelnosti je klíčovou součástí vývoje aplikace, protože přímo reflektuje realitu a to, jak (ne)snadno se uživateli s aplikací interaguje. V poslední kapitole této práce stručně představím cílovou skupinu aplikace, vybrané testery a specifikuji testovací scénáře. Dále se budu věnovat průběhu testování, zaznamenané zpětné vazbě a výsledkům testování.

6.1 Cílová skupina

Cílová skupina aplikace by se dala na základě uživatelů rozdělit na dvě podskupiny. Obě podskupiny spojuje používaný operační systém telefonu, kterým je iOS. Na základě operačního systému můžeme předpokládat návyky na minimalistické uživatelské rozhraní s kladeným důrazem na funkčnost.

První podskupinou jsou fotografové ve věkovém rozsahu přibližně 18-65 let. Přestože fotografové pokrývají hned několik věkových kategorií, se jedná poměrně o specifickou cílovou skupinu, u které se dá obecně předpokládat nadstandardní technická zručnost. Při práci jsou fotografové zvyklí zacházet s pokročilou až (polo)profesionální technikou a rozdílnými uživatelskými rozhraními. Přichází do styku například s uživatelským rozhraním v podobě menu/nastavením fotoaparátu, preferovaným operačním systémem počítače/tabletu a komplexním softwarem při postprocesové úpravě fotografií. Tato cílová skupina se neustále učí novým věcem, technologiím a postupům.

Druhá podskupina je v kontrastu s tou první daleko širší a méně specifická. Touto skupinou je široká veřejnost, která buď jednorázově, nebo dlouhodobě poptává fotografa. Věkové ani technické předpoklady není možné objektivně stanovit a jediným společným rysem zůstává užívaný operační systém.

Z těchto dvou rozdílných světů jsem vybral celkem 5 respondentů, kteří odpovídají specifikovaným požadavkům cílových skupin. Dva fotografy rozdílných generací a tři reprezentanty široké veřejnosti poptávající fotografa.

6.2 Představení testerů

- Tester K
  - Fotograf, Digitální tvůrce – 24 let
  - iPhone 13 Pro (iOS 15.4.1)
  - Zkušený fotograf v oblasti svateb, portrétů, produktů, realit a krajiny
- Tester L
  - Fotograf, Systémový specialista – 46 let
  - iPhone 13 Mini (iOS 15.4.1)
  - Zkušený poloprofesionální fotograf se zaměřením na lifestyle fotografii
- Tester M
  - Magisterský student technického oboru – 23 let
  - iPhone 11 Pro (iOS 15.4.1)
  - Zkušenosti s poptáváním fotografa zatím nemá
- Tester N
  - Vývojová specialistka, Design – 31 let
  - iPhone SE 2020 (iOS 15.4.1)
  - Poptávala fotografa vždy na základě osobní zkušenosti
- Tester O
  - Architektka – 49 let
  - iPhone 11 (iOS 15.4.1)
  - S poptávkou fotografa má zkušenost pouze jednu – osobní reference – spokojena se službami, od té doby jiného fotografa nevyužívá

6.3 Testovací scénáře

Scénáře jsou navrženy tak, aby se co nejvíce přibližovaly reálné uživatelské posloupnosti akcí a byl v nich prostor pro odchycení chyb a nedostatků uživatelského rozhraní. Pro uživatele v režimu tvůrce – fotografy – jsou připraveny testovací scénáře 1 a 2, které pokrývají běžné procesy spojené s tvorbou portfolia a sledováním konkurence. Pro zákazníky poptávající fotografa jsou navrženy scénáře 3 až 5, které jsou zaměřeny převážně na poptávku fotografů.

Testovací scénář 1

1. Registruj se v režimu tvůrce
2. Přihlas se do svého účtu
3. Vytvoř své první portfolio s aspoň třemi fotografiemi
4. Podívej se na svůj profil a zkontroluj správnost informací portfolia
5. Změň popisek portfolia, odeber první fotografii a přidej novou
6. Odhlas se

Testovací scénář 2

1. Přihlas se pod uživatelským jménem „vojtafoti“, heslo „heslovo“
2. Podívej se na konkurenční portrétní portfolia
3. Zjisti jméno a lokaci nejlépe hodnoceného fotografa
4. Ohodnoť profil nejlepšího fotografa
5. U svého portfolia s názvem „Svatby“ přidej nové tagy
6. Smaž všechna portfolia kromě prvního
7. Zobraz si profil fotografa, který působí ve městě Velvary

Testovací scénář 3

1. Registruj se jako běžný uživatel
2. Přihlas se
3. Podívej se na svatební portfolia
4. Do užšího výběru přidej tři portfolia, která se ti nejvíc líbí
5. Zuž výběr na dvě portfolia
6. Podívej se na profil jednoho z autorů portfolií
7. Zobrazený uživatel ti nevyhovuje, odstraň ho z výběru portfolií
8. Kontaktuj autora portfolia, které ti ve výběru zbylo

Testovací scénář 4

1. Přihlas se do svého profilu z předchozího scénáře
2. Vyhledej fotografy ve městě Praha
3. Poptej prvního fotografa, kterého vyhledávač našel
4. Odhlas se

Testovací scénář 5

1. Přihlas se pod uživatelským jménem „vojtafoti“, heslo „heslovo“
2. Odepiš na nově příchozí zprávu
3. Načti si nová portfolia
4. Zobraz si portfolia pouze s tématikou architektury
5. Seřaď portfolia podle hodnocení autorů
6. Přejdi na profil autora portfolia, které se ti líbí ze všech nejvíc
7. Ohodnoť jeho profil

6.4 Průběh testování

Každý z testerů byl zasvěcen do kontextu aplikace – k čemu slouží, proč byla vytvořena a jak v jednoduchosti funguje. Ze dvou možných způsobů testování jsem se rozhodl pro testování na mém osobním zařízení (iPhone 11 Pro s verzí systému 15.4.1), protože se jedná o daleko intuitivnější variantu než je testování prototypu v simulátoru na počítači. Poslední verzi aplikace jsem sestavil a pomocí Xcode nahrál do telefonu. Toto je možné i bez Apple Developer Program[14], který po uhrazení umožňuje distribuci aplikace pomocí App Store – v případě, že Apple aplikaci schválí.

Průběh testování byl nahráván dvěma způsoby. Uživatel byl nahráván z předního pohledu, který pokrýval jeho obličej a ruku ovládající displej telefonu. Kromě obrazového záznamu byla nahrávána i zvuková stopa pro případ, že by uživatel ventiloval frustraci nebo nadšení slovně. Dalším zdrojem záznamu byl samotný displej jehož obrazovka byla po celou dobu scénáře nahrávána. Tyto stopy byly využity pro důkladnou zpětnou analýzu testování. Všemi testery byl vznesen požadavek na neuveřejňování záznamů, kterému jsem samozřejmě vyhověl, a všechny záznamy byly po dokončení analýzy odstraněny.

[14] https://developer.apple.com/programs/

Kromě samotných scénářů byly respondentům položeny doplňující otázky k průběhu testování a jejich subjektivním pocitům z prototypu aplikace. Otázky a odpovědi jsou zaznamenány na konci této sekce.

Testovací scénář 1 – Fotografové

S procesem registrace fotografové neměli žádný problém. Druhý tester při vyplňování číselných hodnot ocenil číselnou klávesnici namísto často využívaného posuvníku. Přihlášení bylo bezproblémové, ale u třetího kroku – vytvoření portfolia – byl vznesen požadavek na zvětšení tlačítka plus na uživatelském profilu, protože se testerovi L nedařilo tlačítko stisknout. Při tvorbě portfolia začal první testující zadávat do textového pole tagů více klíčových slov za sebou. Pole je navrženo pro přidávání tagů po jednom, ale i na tuto situaci je připraveno – mezery jsou před přidáním odstraněny a slova spojeny do jednoho – záměrem uživatele ale bylo přidat více tagů najednou. Zbývající kroky tvorby a úpravy portfolia proběhly tak, jak bylo zamýšleno. Pro zobrazení detailů portfolia na svém profilu testeři intuitivně klikli na mřížku portfolií, která s animací detaily zobrazila. Zbývající kroky proběhly bez komplikací.

Testovací scénář 2 – Fotografové

Druhý krok tohoto scénáře byl záměrně popsán méně specificky, aby testující sám musel odhalit, kde najít portfolia konkrétního žánru/kategorie – v tomto případě portréty. První respondent ve feedu pomocí tlačítka v `NavigationBar` intuitivně přešel přímo na obrazovku filtrování portfolií, ale druhý se nejprve přepnul na záložku Vyhledávání. Situaci nepomohlo ani to, že záložka nemá v `NavigationTitle` název Vyhledávání uživatelů, ale pouze Vyhledávání. Po chvíli se přesunul zpět do feedu portfolií a následně i do filtrování. Po zadání tagu na filtrovací obrazovce očekával druhý tester jakoukoliv informaci o aktualizaci stavu filtrování, která nepřišla, a chvíli čekal, jestli se výsledky nezobrazí na obrazovce zadávání filtrovacích tagů. Ve zbývajících krocích řazení portfolií, hodnocení uživatele, úprav portfolií a vyhledávání už žádný problém nenastal a vše proběhlo tak, jak bylo modelově zamýšleno.

Testovací scénář 3 – Poptávající

Při registraci poptávajících testerů došlo k zadání chybného tvaru emailové adresy bez tečky a potvrzení hesla. U emailu byl rozpoznán špatný tvar a u textového pole se správně zobrazil text upozorňující na nesprávné heslo, takže se vše rychle vyřešilo. Pro shodu hesel si respondentka N zobrazila znění hesel pomocí typické ikonky oka a problém vyřešila. U třetího kroku filtrování konkrétní kategorie portfolií svateb byla u většiny testerů pozorovatelná určitá nejistota po zadání tagu. Stejně jako tester K očekávali větší interaktivitu na

obrazovce filtrování a trvalo delší dobu než si uvědomili, že se výsledky zobrazují ve feedu a ne v překryvu obrazovky, kde zadali klíčové slovo. Přesun portfolií do vybraných a následná selekce proběhla v pořádku u všech respondentů. Ke kontaktování autora vybraného portfolia využili všichni tlačítka ve výběru až na respondentku M, která přešla do zpráv až z autorova profilu. Textu tlačítka Napsat zprávu si nevšimla, nepřečetla si ho a preferovala by variantu ikony zprávy. S odhlášením problém nenastal, všichni uživatelé ho správně vyhledali v záložce nastavení svého profilu.

Testovací scénář 4 – Poptávající

U čtvrtého scénáře vše proběhlo podle očekávání hladce. Jediný problém, který se vyskytl, byla velikost tlačítka zprávy na fotografově profilu. Stejně jako v případě testera L připadalo respondentům M a O nedostatečně velké pro snadné stisknutí.

Testovací scénář 5 – Poptávající

Po přihlášení bylo úkolem odepsat na poslední obdrženou zprávu. Všichni testeři se přepnuli na liště do záložky zpráv, ale nová příchozí zpráva nebyla zvýrazněna a odlišena od ostatních přečtených zpráv. Proto pár vteřin trvalo než si zprávy všimli. V dalším kroku scénáře – načtení nových portfolií – využili všichni testující šipky pro aktualizaci až na testera N, který místo toho zkoušel seřadit portfolia podle data. Při úmyslu kliknout na tlačítko šipky tester M kliknul omylem na řazení. Na vině byla opět zvolená menší velikost tlačítek v `NavigationBar` kvůli estetičtějšímu vzhledu, který ale potlačil funkčnost, která je na prvním místě. S filtrováním už měli respondenti zkušenost ze třetího scénáře a zobrazení portfolií s tématikou architektury proběhlo hladce a promptně. Zbývající kroky proběhly bez komplikací.

Shrnutí závěrečných otázek testování

- Jak hodnotíte průběh testování, narazili jste na nějaké problémy?
  - Všichni respondenti hodnotili testování velmi pozitivně, ale nezávisle na ostatních se shodli na dvou problémech, na které narazili. Prvním problémem byla obrazovka filtrování, kde 3 z 5 testujících očekávali větší interakci této obrazovky nebo informování o stavu hledání. Navíc nejsou tolik zvyklí na částečný překryv obrazovky, který pro ně byl matoucí. Druhým problémem, který byl vyřčen většinou testerů, byla velikost tlačítek v `NavigationBar`. Díky menší velikosti byly hůře stlačitelné a zároveň i hůře viditelné.

- Chyběly vám v aplikaci nějaké funkce?
  - Testeři L, M a N hodnotili aplikaci jako funkční a kompletní celek. Žádné funkce by nepřidávali a všechny potřebné funkce, které by v procesu poptávání/nabízení fotografických služeb využili, měli k dispozici. Testující M doporučila doimplementovat u profilů fotografů kolonku s cenovým odhadem / hodinovou sazbou – případně uvedení informace o ceně dohodou. Tester K v režimu tvůrce navrhl přidání inzerátů poptávek do feedu mezi portfolia, kde by se fotograf dozvěděl nezbytné informace o akci zákazníka a jeho nabídce.
- Jak byste ohodnotili navigaci napříč aplikací a její uživatelské rozhraní? Hodnoťte slovně a na stupnici 1-5 jako ve škole.
  - Uživatelé ocenili minimalistický vzhled UI, který popsali jako jednoduchý, přehledný a funkční. Na spodní lištu pro přepínání mezi obrazovkami jsou zvyklí a stejně tak i na rozmístění zbývajících navigačních prvků ve vrchní části obrazovky. Jedinou výtkou, která zazněla ve větší míře, byla slabá interaktivita obrazovky filtrování portfolií. Průměrné hodnocení na škále od 1 do 5 se zastavilo na hodnotě 1,5 většinově právě kvůli zmíněnému filtrování.
- Používali byste po téhle zkušenosti Fotofolio i v realitě?
  - Všichni respondenti odpověděli kladně a v případě, že by měla aplikace aktivní širokou uživatelskou základnu, by aplikaci využili pro proces nabídky/poptávky fotografických služeb.

6.5 Zhodnocení testování

Uživatelské testování se dle mého názoru vydařilo a ukázalo se jako mimořádně přínosné. Je ale třeba zmínit, že testování prototypu proběhlo na velmi malém vzorku lidí a všem náznakům a návrhům ke změně by tedy měla být kladena značná váha. Pokud je více lidmi z tohoto vzorku cílové skupiny vyřčena stejná výtka vůči prototypu a jeho uživatelskému rozhraní, je velká šance, že bude vyřčena i dalšími uživateli. Všechny poznámky a návrhy na změny jsem akceptoval, zaznamenal a před implementací serverové části a nasazením aplikace do produkce musí být slabší místa prototypu vylepšena. Aktuálně největší problém vidím v částečném překryvu feedu při filtrování pomocí `sheet`, který je pro mě osobně běžnou součástí UI v iOS, ale pro většinu testerů je tento přístup nepřehledný a nejsou na něj zcela zvyklí. Díky reakcím a zpětné vazbě respondentů vzniklo mnoho podnětů nejen na vylepšení aktuálního stavu prototypu, ale i na přidání funkcionalit do budoucna, jako je třeba zobrazování poptávek fotografa ve feedu. Pozitivním signálem je, že pocity testerů z prototypu jsou velmi kladné a aplikace plní svůj účel. Uživatelské rozhraní a jeho design je dle subjektivních pocitů testerů čisté, minimalistické a intuitivní. Plně

funkční aplikaci by pro navázání kontaktu s fotografem/zákazníkem využili i pro své osobní účely v realitě.

Závěr

Práce svým rozsahem pokrývá analýzu, návrh a implementaci prototypu mobilní aplikace Fotofolio, platformu pro snadné navázání kontaktu zákazníka s fotografem a vice versa. Fotografům je umožněno vytvářet portfolia a prezentovat tak svou práci zákazníkům, kteří mají o jejich služby zájem. Aplikace poskytuje uživateli minimalistické a funkční uživatelské rozhraní pro maximalizaci uživatelského zážitku. Podařilo se pokrýt všechny (ne)funkční požadavky specifikované v sekci 2.3, jednotlivé body zadání bakalářské práce a vytvořit prototyp aplikace, která žádné fotografy nediskriminuje. Těžištěm platformy je finální produkt fotografa v podobě portfolií a proces poptávání je navržen tak, aby se uživatel rozhodoval na základě vizuálního vjemu a až sekundárně podle fotografových osobních informací a údajů.

V první kapitole práce jsem důkladně zanalyzoval a představil problematiku společně se současnými řešeními a stanovil jsem funkční a nefunkční požadavky práce. Na základě těchto poznatků jsem zvolil vhodné technologie, představil je a navrhl architekturu aplikace se zaměřením na její klientskou část. Dále jsem navrhl uživatelské rozhraní aplikace a rozvedl implementační detaily prototypu. Finální verze prototypu aplikace byla podrobena uživatelskému testování osobami z cílové skupiny projektu, díky kterému vznikly relevantní návrhy na vylepšení aplikace. V kombinaci s nasazením serverové části aplikace, jeho integrací a volbou vhodného monetizačního modelu aplikace, bude poskytnuta široké veřejnosti platforma, která intuitivně řeší problematiku poptávání, respektive nabízení fotografických služeb.

Bibliografie

1. LEDFORD, Jerri L. *SEO: Search Engine Optimization Bible* [online]. Wiley, 2008 [cit. 2022-04-01]. ISBN 978-0470452646.
2. WONG, Julia Carrie. *The Cambridge Analytica scandal changed the world – but it didn't change Facebook* [online]. Guardian News a Media, 2019-03 [cit. 2022-03-27]. Dostupné z: `https://www.theguardian.com/technology/2019/mar/17/the-cambridge-analytica-scandal-changed-the-world-but-it-didnt-change-facebook`.
3. COMPUTER HOPE. *What is a GUI (graphical user interface)?* [Online]. 2021-04 [cit. 2022-03-28]. Dostupné z: `https://www.computerhope.com/jargon/g/gui.htm`.
4. BLINK INC. *Blink Inc* [online]. 2020-01 [cit. 2022-04-01]. Dostupné z: `https://apps.apple.com/us/app/blink-inc/id1475141049`.
5. PHOTOSESH. *PhotoSesh – find photographers on the app store* [online] [cit. 2022-04-01]. Dostupné z: `https://apps.apple.com/us/app/photosesh-find-photographers/id1101828097`.
6. RANSOM, Chris. *Phollio- Photoshoot On-Demand* [online]. 2019-02 [cit. 2022-04-01]. Dostupné z: `https://apps.apple.com/gb/app/phollio-photos-on-demand/id1451221338`.
7. POZBEE, LLC. *Pozbee - Photographer Hiring* [online]. 2021-08 [cit. 2022-04-01]. Dostupné z: `https://apps.apple.com/gb/app/pozbee-photographer-hiring/id1511196081`.
8. ZAZZI, LLC. *ZAZZI: Photography On-Demand* [online]. 2020-08 [cit. 2022-04-01]. Dostupné z: `https://apps.apple.com/us/app/zazzi-photography-on-demand/id1516438188`.
9. KOMPANIETS, Anastasia. *Functional vs. non-functional requirements: Why are both important?* [Online] [cit. 2022-04-01]. Dostupné z: `https://www.uptech.team/blog/functional-vs-non-functional-requirements`.

10. AHMAD, Mohd Shahdi; MUSA, Nur; NADARAJAH, Rathidevi; HASSAN, Rosilah; OTHMAN, Nor Effendy. Comparison between android and iOS Operating System in terms of security. In: [online]. 2013, s. 1–4 [cit. 2022-04-02]. ISBN 978-1-4799-1091-5. Dostupné z DOI: 10.1109/CITA.2013.6637558.
11. CERVANTES, Edgar. *10 things IOS does better than Android* [online]. 2022-02 [cit. 2022-04-02]. Dostupné z: https://www.androidauthority.com/ios-vs-android-1068950/.
12. APPLE INC. *iOS Design Themes* [online] [cit. 2022-04-02]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/.
13. APPLE INC. *iOS 15* [online] [cit. 2022-04-02]. Dostupné z: https://www.apple.com/cz/ios/ios-15/.
14. TEMPONE, Denise. *What Is Figma? A Quick Intro to the Collaborative Design Tool* [online]. DOMESTIKA, 2021 [cit. 2022-04-02]. Dostupné z: https://www.domestika.org/en/blog/9722-what-is-figma-a-quick-intro-to-the-collaborative-design-tool.
15. KOPF, Ben. *The power of Figma as a design tool* [online]. Toptal, 2018-07 [cit. 2022-04-02]. Dostupné z: https://www.toptal.com/designers/ui/figma-design-tool.
16. PANDEY, Amit. *Swift Basics* [online]. Medium, 2020-01 [cit. 2022-04-02]. Dostupné z: https://medium.com/@amit.3117/swift-basics-5ec6c86e0d29.
17. APPLE INC. *Apple/Swift: The Swift programming language* [online] [cit. 2022-04-03]. Dostupné z: https://github.com/apple/swift.
18. CTU FIT. *Programming Paradigms* [online] [cit. 2022-04-03]. Dostupné z: https://courses.fit.cvut.cz/.
19. APPLE INC. *The Swift Programming Language (Swift 5.6)* [online] [cit. 2022-04-03]. Dostupné z: https://docs.swift.org/swift-book/.
20. PETERSON, Mike; NEELY, Amber; OWEN, Malcolm; DILGER, Daniel Eran; HUGHES, Neil. *Xcode: Updates. features, languages* [online] [cit. 2022-04-04]. Dostupné z: https://appleinsider.com/inside/xcode.
21. TATIS, Mario. *What is SwiftUI?* [Online]. 2021-03 [cit. 2022-04-04]. Dostupné z: https://www.koombea.com/blog/what-is-swiftui/.
22. MEJIA, Robert. *Declarative and Imperative Programming using SwiftUI and UIKit* [online]. Medium, 2019-12 [cit. 2022-04-04]. Dostupné z: https://medium.com/@rmeji1/declarative-and-imperative-programming-using-swiftui-and-uikit-c91f1f104252.
23. APPLE INC. *SwiftUI Documentation* [online] [cit. 2022-04-05]. Dostupné z: https://developer.apple.com/documentation/swiftui.

24. HUDSON, Paul. *Understanding property wrappers in Swift and SwiftUI* [online]. Hacking with Swift, 2021-02 [cit. 2022-04-05]. Dostupné z: https://www.hackingwithswift.com/quick-start/swiftui/understanding-property-wrappers-in-swift-and-swiftui.

25. BERNHAUER, David. *Architektury webových aplikací* [přednáška] [cit. 2022-04-06]. Dostupné z: https://courses.fit.cvut.cz/BI-TWA/media/topics/t08-architecture.pdf.

26. BULAVIN, Vadim. *Modern MVVM iOS App Architecture with Combine and SwiftUI* [online]. Yet Another Swift Blog, 2020-03 [cit. 2022-04-06]. Dostupné z: https://www.vadimbulavin.com/modern-mvvm-ios-app-architecture-with-combine-and-swiftui/.

27. KHAN, Aasif. *MVVM and SwiftUI* [online]. 2021-12 [cit. 2022-04-06]. Dostupné z: https://www.appypie.com/mvvm-swiftui-how-to.

28. HEGARTY, Paul. *MVVM and the Swift Type System* [video]. YouTube, 2021-05 [cit. 2022-04-06]. Dostupné z: https://www.youtube.com/watch?v=--qKOhdgJAs&ab_channel=Stanford.

29. BERNHAUER, David. *REST* [přednáška] [cit. 2022-04-07]. Dostupné z: https://courses.fit.cvut.cz/BI-TWA/media/topics/t16-rest.pdf.

30. PATNI, Sanjay. *Pro RESTFUL APIS*. Introduction - XML, JSON [online]. APRESS, 2017 [cit. 2022-04-28]. ISBN 978-1-4842-2665-0. Dostupné z: https://doi.org/10.1007/978-1-4842-2665-0_3.

31. HAMILTON, Rosie. *Data mocking - A way to test the untestable* [online] [cit. 2022-04-28]. Dostupné z: https://blog.scottlogic.com/2016/02/08/data-mocking.html.

32. CARREIRA, João. *Mocking a remote API in IOS* [online]. The Startup, 2019-06 [cit. 2022-04-28]. Dostupné z: https://medium.com/swlh/mocking-a-remote-api-in-ios-4376b0cab962.

33. NIELSEN, Jakob. *10 Usability Heuristics for User Interface Design* [online] [cit. 2022-04-08]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/.

34. BABICH, Nick. *Prototyping 101: Low-fidelity vs high-fidelity: Adobe XD ideas* [online]. Adobe, 2019-10 [cit. 2022-04-08]. Dostupné z: https://xd.adobe.com/ideas/process/prototyping/low-fi-and-hi-fi-prototyping/.

35. DAM, Rikke Friis; SIANG, Teo Yu. *What Kind of Prototype Should You Create?* [Online] [cit. 2022-04-08]. Dostupné z: https://www.interaction-design.org/literature/article/what-kind-of-prototype-should-you-create.

36. CHUNG, Eddy. *UUIDs In Swift Tutorial* [online] [cit. 2022-04-11]. Dostupné z: `https://www.zerotoappstore.com/uuids-in-swift.html`.

Příloha **A**

Seznam použitých zkratek

SEO Search Engine Optimization

GPS Global Positioning Systém

UI User interface

GUI Graphical user interface

OS Operating system

HOF Higher Order Function

IDE Integrated Development Environment

WWDC Worldwide Developers Conference

API Application Programming Interface

MVC Model View Controller

MVVM Model View ViewModel

JSON JavaScript Object Notation

XML Extensible Markup Language

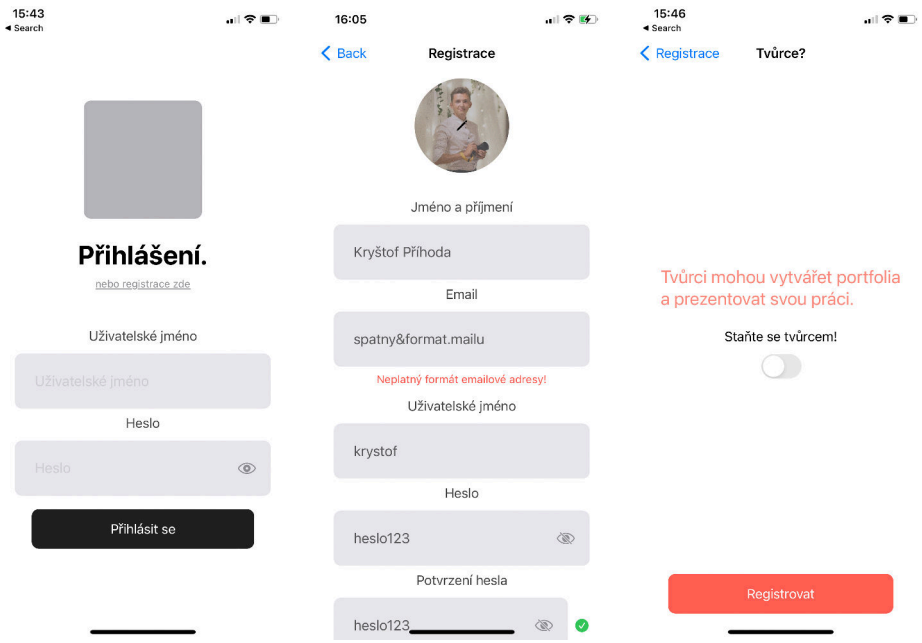
REST Representational state transfer

HTTPS HyperText Transfer Protocol Secure

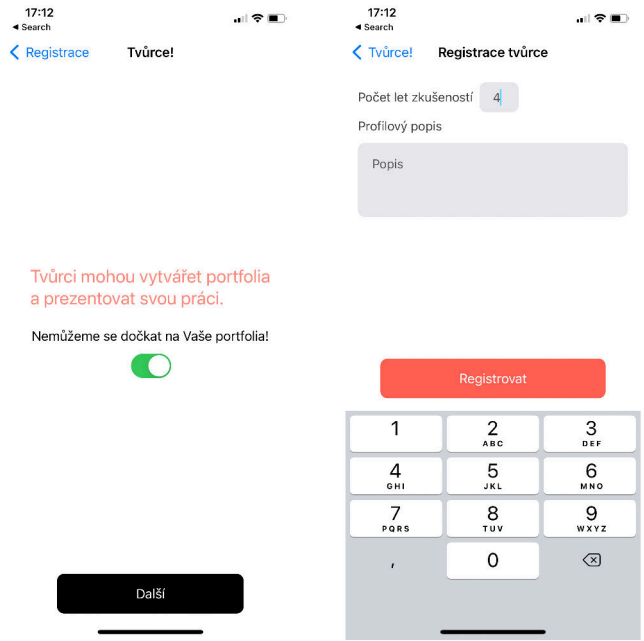
UUID Universally Unique Identifier

Příloha B

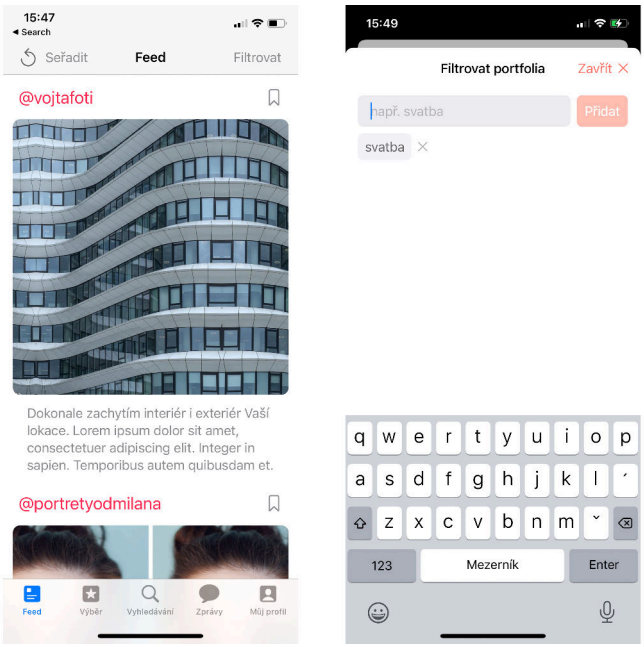
Snímky obrazovky prototypu



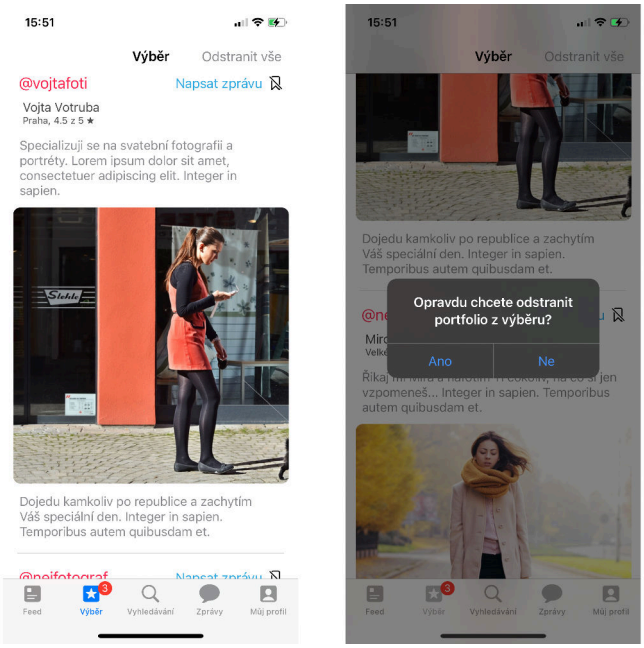
Obrázek B.1: Snímky obrazovky procesu přihlášení a registrace



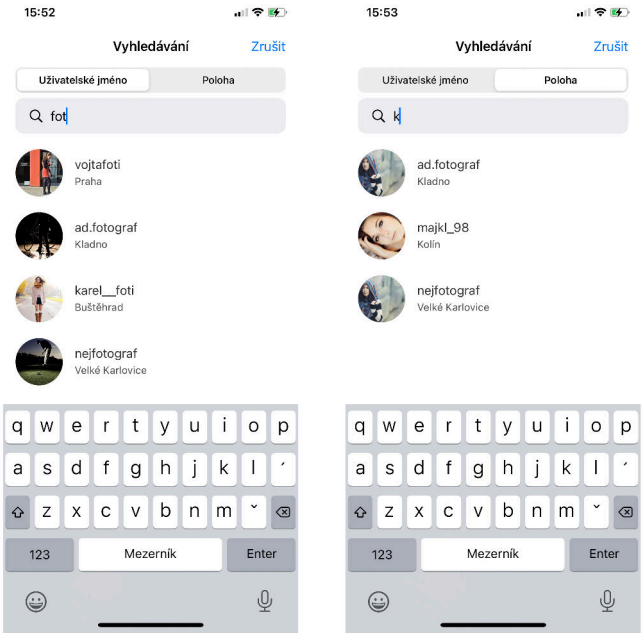
Obrázek B.2: Snímky obrazovky registrace jako tvůrce



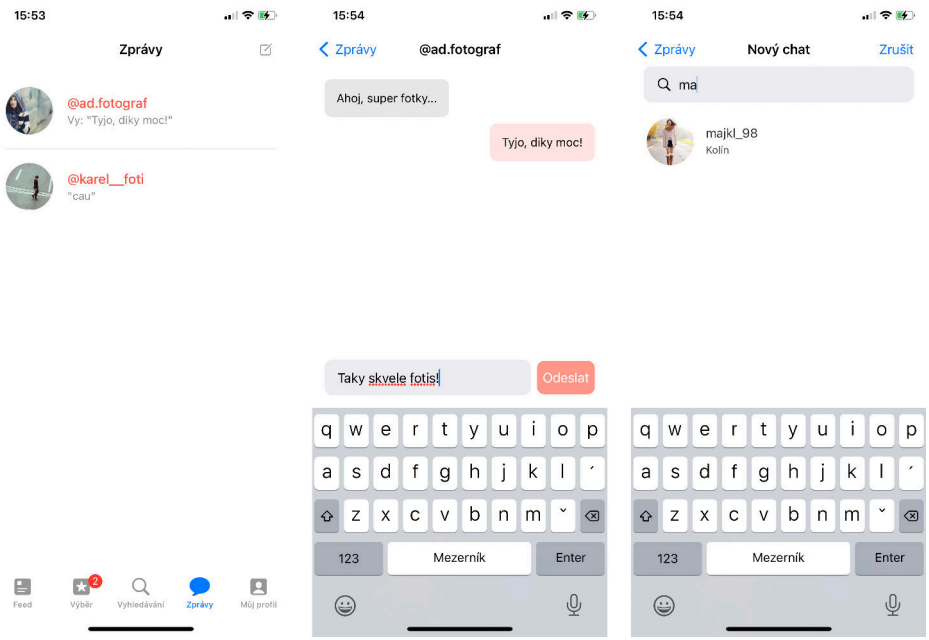
Obrázek B.3: Snímky obrazovky feedu a filtrování portfolií



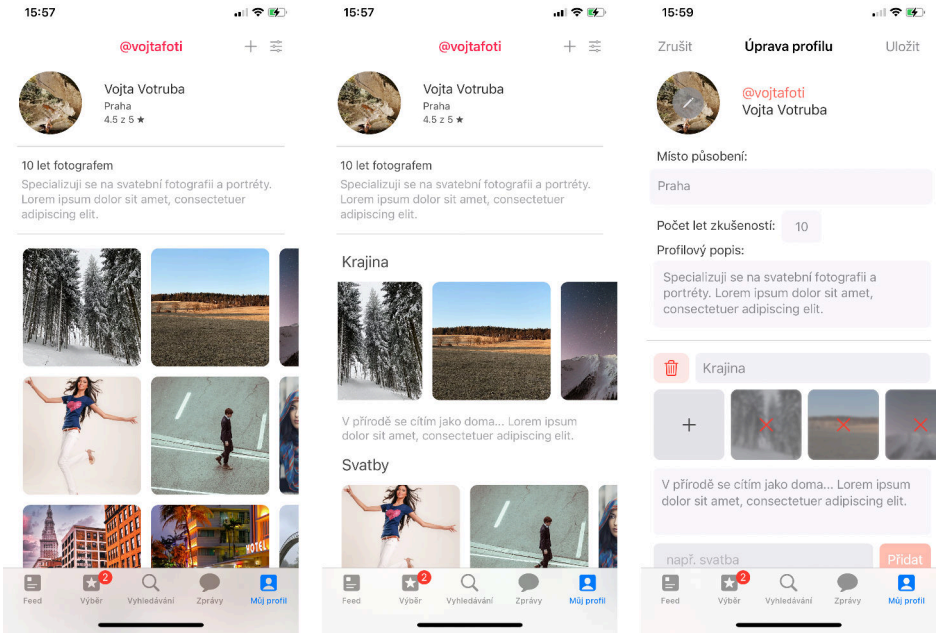
Obrázek B.4: Snímky obrazovky uživatelem vybraných portfolií



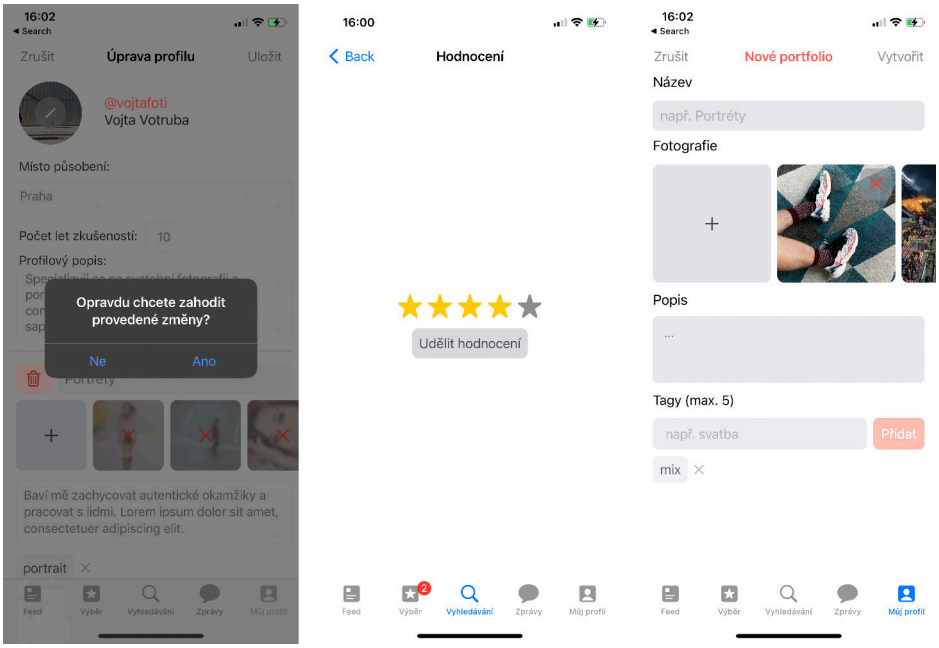
Obrázek B.5: Snímky obrazovky vyhledávání



Obrázek B.6: Snímky obrazovky zpráv



Obrázek B.7: Snímky obrazovky profilu a úpravy profilu



Obrázek B.8: Snímky obrazovky profilu, hodnocení a vytvoření nového portfolia

Příloha C

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu přiloženého média
- `text` ..... text práce
  - `thesis_fotofolio.pdf` ..... text práce ve formátu PDF
  - `thesis_fotofolio.zip` ..... zdrojová forma práce
- `impl` ..... zdrojové kódy implementace
- `screenshots` ..... snímky obrazovky prototypu aplikace