Mendelova univerzita v Brně
Provozně ekonomická fakulta

Mobilní aplikace pro pomoc imigrantům

Bakalářská práce

Jan Janovec

Vedoucí práce:
Ing. David Procházka, Ph.D.

Brno 2023

Mendelova univerzita v Brně
Ústav informatiky

Provozně ekonomická fakulta
2022/2023

MENDELU
Provozně
ekonomická
fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	Jan Janovec
Studijní program:	Systémové inženýrství a informatika
Obor:	Ekonomická informatika
Specializace:	Mobilní a mapové aplikace
Vedoucí práce:	Ing. David Procházka, Ph.D.
Název práce:	**Mobilní aplikace pro pomoc imigrantům**
Jazyková varianta:	Čeština

Zásady pro vypracování:

1. Proveďte rešerši mobilních aplikací zaměřených na podporu imigrantů pro jednotlivé státy. Soustřeďte se jak na aplikace národního charakteru, tak regionální (např. Praguer). Sumarizujte funkcionalitu, kterou nabízí.
2. Prostudujte možnosti vývoje aplikací pro platformu iOS/iPadOS firmy Apple. Studium zaměřte zejména na aktuální způsob tvorby aplikací reaktivním způsobem, tedy technologie SwiftUI, Combine atp. Proveďte rešerši, jaké jsou možnosti vizualizace mapových podkladů v mobilní aplikaci a zohledněte i možnost fungování v off-line režimu.
3. Proveďte rešerši způsobů, jak se připojit na REST API webové služby. Tedy jak tento problém implementovat pomocí Swiftu a jaké existují pomocné knihovny.
4. Proveďte převod existujícího designu aplikace Smart Migration z platformy Android na platformu iOS. Snažte se maximálně dodržet designový jazyk aplikace, ale zohledněte obvyklé ovládací prvky na platformě iOS.
5. Implementujte mobilní aplikaci prostřednictvím vybraných knihoven v jazyce Swift s grafickým uživatelským rozhraním založeným na SwiftUI a Combine.
6. Proveďte uživatelské testování vytvořené aplikace. Vyhodnoťte zjištěné problémy a zapracujte příslušné změny. Navrhněte další rozvoj projektu zejména s ohledem na rešerši konkurenčních aplikací.

Rozsah práce: 1,5-3 AA

Literatura:

1. STRAWN, J. *Design Patterns by Tutorials (Third Edition): Learning Design Patterns in Swift.* USA: Razerware LLC, 2021. 392 s.
2. TAM, A. -- BAGBIE, C. *SwiftUI Apprentice (First Edition).* USA: Razeware LLC, 2021. 726 s.
3. SHAI, M. *Combine: Asynchronous Programming with Swift (Third Edition).* USA: Razeware LLC, 2021. 447 s. ISBN 978-1-95-032549-8.
4. TODOROV, M. *Modern Concurrency in Swift (First Edition): Introducing Async/Await, Task Groups & Actors.* USA: Razeware LLC, 2021. 273 s. ISBN 978-1-95-032553-5.

Datum zadání: říjen 2022

Datum odevzdání: květen 2023

Elektronicky schváleno: 11. 11. 2022
Ing. David Procházka, Ph.D.
Vedoucí práce

Elektronicky schváleno: 11. 11. 2022
Jan Janovec
Autor práce

Elektronicky schváleno: 11. 11. 2022
prof. Ing. Cyril Klimeš, CSc.
Vedoucí ústavu

Elektronicky schváleno: 23. 11. 2022
doc. Ing. Ivana Rábová, Ph.D.
Garantka studijního programu

Rád bych poděkoval svému vedoucímu za ochotu poskytnout mi pravidelné konzultace a rady, které se vážou k této práci. Dále mu děkuji za příležitost spolupracovat na tomto projektu. Dalším člověkem, kterému bych rád poděkoval, je Tomáš Čejka za odborné rady a pravidelné kontroly kódu. Speciální poděkování patří také členům mé rodiny a mým blízkým za jejich podporu a účast na testování.

Čestné prohlášení

Prohlašuji, že jsem práci *Mobilní aplikace pro pomoc imigrantům* vypracoval samostatně a veškeré použité prameny a informace uvádím v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů a v souladu s platnou Směrnicí o zveřejňování vysokoškolských závěrečných prací. Prohlašuji, že tištěná podoba závěrečné práce a elektronická podoba závěrečné práce zveřejněná v aplikaci Závěrečné práce v Univerzitním informačním systému je identická.

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 5. května 2023

.....

podpis

Abstract

Janovec, Jan. *Mobile application to help the immigrants*. Bachelor thesis. Brno : Mendel University in Brno, 2023.

This thesis deals with the development of a Smart Migration mobile application for the iOS platform. The application is primarily intended for foreigners coming to the Czech Republic for work. The purpose of the project is to allow people to easily navigate the legislation they have to comply with, but also in everyday life situations such as finding a school or kindergarten. The theoretical part of the thesis consists of a survey of current trends and technologies for creating apps for the iOS platform and analyzes the functionality of similar apps available on the market. The proposed app is based on the Swift language and the SwiftUI framework.

Key words

swift, iOS, swiftUI, mobile app development, mobile app, migration

Abstrakt

Janovec, Jan. *Mobilní aplikace pro pomoc imigrantům.* Bakalářská práce. Brno : Mendelova univerzita v Brně, 2023.

Tato práce se zabývá tvorbou mobilní aplikace Smart Migration pro platformu iOS. Aplikace je primárně určena pro cizince přicházející do České republiky za prací. Smyslem projektu je umožnit lidem snadno se zorientovat v legislativě, kterou musí dodržet, ale i v běžných životních situacích jako je hledání školy či školky. Teoretická část práce tvoří průzkum současných trendů a technologií pro tvorbu aplikací pro platformu iOS a popisuje také podobné aplikace dostupné na trhu. Navržená aplikace je založena na jazyce Swift a frameworku SwiftUI.

Klíčová slova

swift, iOS, swiftUI, vývoj mobilních aplikací, mobilní aplikace, migrace

Obsah

1 Úvod

1.1 Úvod do problematiky

Migrace je v lidské historii hluboce zakořeněná, už od dávných let lidé této planety migrovali, ať už kvůli nevhodnému podnebí, jídlu anebo nebezpečí styku se znepřáteleným kmenem. V dnešní době je migrace ve společnosti velice důležité téma. Lidé prchají před environmentálními změnami, hladem nebo válkou. Pro migranty může být často složité přizpůsobit se novému státu, kvůli jazyku, společenským standardům nebo náboženství. Na téma imigrace je často pohlíženo z mnoha úhlů, spousta lidí vidí v imigraci pouze negativní dopady, jsou tu však i pozitivní dopady, jako např. dopad na ekonomiku. Migranti často přichází z mnohem chudších států a jsou tak ochotni pracovat za nižší mzdy.

Mnoho států, stejně jako Česká republika, mají řadu zákonů a vyhlášek, ve kterých není jednoduché se zorientovat. Podmínky je nutno dodržet do posledního bodu a právě to může být pro nově příchozí často problém. Kvůli tomuto také vznikl projekt *Smart Migration*, který si dává za cíl pomoci migrantům s integrací do našeho státu a případnou úpravu legislativy. Pro zmíněný projekt již byla vytvořena aplikace pro platformu *Android*, která se zatím těší více než padesáti tisícům stažení a bezpochyby již pomohla velikému počtu cizinců. Mně byla nabídnuta práce na aplikaci *Smart Migration* pro platformu *iOS*, která pomůže dalším lidem se začleněním do našeho státu.

Tato práce se zabývá aktuálními trendy vývoje pro operační systém firmy *Apple*, tedy *iOS* a vývojem aplikace *Smart Migration*. Vývoj této aplikace je doplněn jejím testováním pomocí jednotkových testů, testů uživatelského rozhraní nebo uživatelským testováním.

1.2 Cíl práce

Cílem této práce je návrh řešení a následná implementace aplikace na podporu migrantů pro platformu *iOS*, která bude pokrývat funkcionality aplikace pro *Android*. Aplikace bude pokrývat tyto funkce a požadavky:

1. Při prvním spuštění aplikace představí aplikace své základní funkcionality uživateli.
2. Aplikace bude stahovat ze serveru seznamy tutoriálů, kontaktních míst a na ně navázaná data.
3. Seznamy kontaktních míst a tutoriálů budou ukládány do databáze, díky čemuž budou dostupné i bez připojení k internetu.
4. Dojde-li k aktualizaci dat na serveru, aplikace uložená data nahradí daty novými.
5. Aplikace bude na mapě zobrazovat kontaktní místa.
6. Aplikace bude podporovat tři jazyky – angličtina, ruština, ukrajinština.
7. Aplikace bude podporovat tmavý i světlý mód.
8. V aplikaci bude seznam úkolů, kam si uživatel může přidat vlastní úkoly nebo je importovat z řešených problémů.
9. Aplikace bude obsahovat chatbota.
10. Uživatelské rozhraní aplikace bude přizpůsobeno zařízením různých velikostí.

Abychom mohli aplikaci implementovat, musíme provést kroky, které povedou ke zvolení vhodných technologií, architektury anebo vhodných postupů řešení. V literární rešerši se seznámíme s již existujícími aplikacemi podobného charakteru, frameworky pro tvorbu uživatelského prostředí pro platformu *iOS* nebo technologiemi pro perzistenci dat, síťovou komunikaci, práci s mapou a chatbota. Seznámíme se také s metodami testování aplikace, které zajistí správné fungování aplikace.

2 Literární rešerše

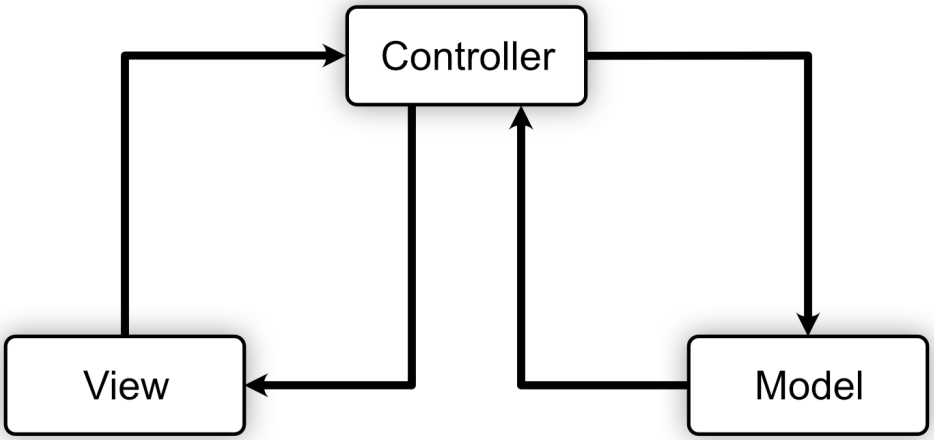
Tato kapitola se věnuje literární rešerši, která zkoumá aplikace podobného charakteru nebo architektury a technologie, využívané při vývoji aplikací pro platformu *iOS*.

2.1 Architektura aplikace

Jedním z prvních kroků vývojáře je stanovit si cíle aplikace a vhodnou architekturu. Vhodnost architektury může záviset, jak na rozsahu aplikace, tak i na operačním systému. Architekturu můžeme chápat jako sadu pravidel, které jednoznačně definují tok akcí a dat v aplikaci nebo strukturu jednotlivých částí. Vhodná architektura dokáže předejít chaosu v kódu nebo zaručit vhodnou míru škálovatelnosti aplikace. V této podkapitole se podíváme na tři z nejpoužívanějších architektur ve vývoji aplikací pro *iOS*.

MVC

MVC je architekturou, která je přímo doporučována společností *Apple* pro vývoj aplikací pro *iOS*. Tato architektura se skládá ze tří hlavních částí: *Model*, *View* a *Controller*. *Model* obsahuje data aplikace nebo jejich struktury. *View* reprezentuje jednotlivé komponenty, které jsou zobrazovány v rámci uživatelského rozhraní a reaguje na interakce uživatele. Komunikaci mezi těmito částmi zajišťuje *Controller*, který je upozorněn, kdykoli dojde ke změně dat a následně informuje *View*, na kterém se zobrazí již modifikovaná data. Komunikace funguje i obráceným směrem, kdy po interakci uživatele s obrazovkou upozorní *Model* na případnou změnu dat. Tato architektura je vhodná především pro tvorbu aplikací pomocí frameworku *UIKit*. (Dan Radu, 2021) Jedním z důvodů, proč se od architektury MVC upustilo je, že lidé měli tendenci psát veškeré funkce do *Controlleru*. Tímto přístupem vzniká těžko udržitelný a složitě strukturovaný kód. Opačným důvodem, proč je vhodné přemýšlet o volbě MVC architektury je její jednoduchost pro vývoj aplikací, které plní jednoduchou funkci a často obsahují pouze jednu obrazovku. (Pedro Alvarez, 2021)



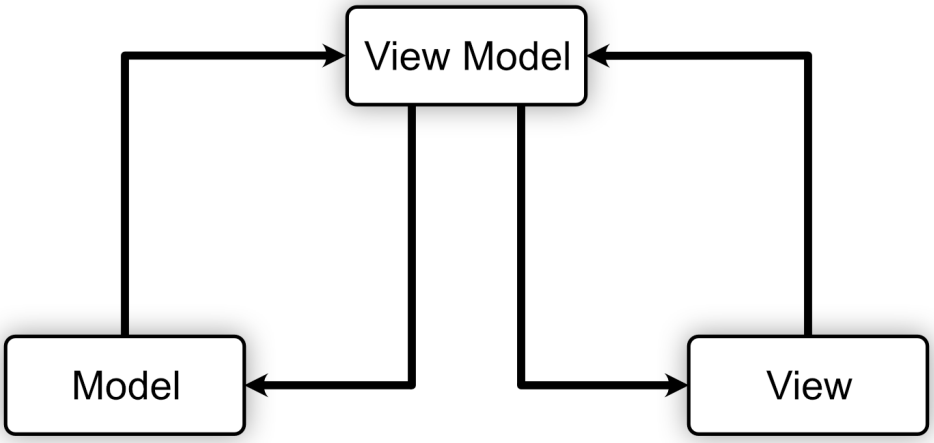
Obrázek 2.1
Schéma architektury MVC. Podle Apple (2018), upraveno.

MVVM

MVVM je novější architekturou než MVC a také se dělí na tři celky: *Model*, *View* a *View Model*. *Model* a *View* reprezentují to samé, co reprezentují v architektuře MVC. Novinkou v této architektuře je záměna *Controlleru* za *View Model*, který je opět prostředníkem mezi *View* a *Modelem*, ale funguje odlišně. *View Model* používá *binding*, což lze vnímat jako provázání mezi *Modelem* a *View*. Jsou-li data v *Modelu* změněny, pak jsou automaticky změněny na obrazovce. Komunikace funguje i opačným směrem a kdykoli je vyvolána změna dat na obrazovce, propíše se změna i do *Modelu*. Tato architektura je vhodnější k tvorbě *SwiftUI* aplikací, kde můžeme velice efektivně oddělit *View* od kódu, který *View* nepotřebuje. (Dan Radu, 2021)

VIPER

VIPER reprezentuje jednu ze složitějších architektur pro vývoj aplikací. Obsahuje celkově pět tříd a každá z nich je používána právě pro jeden účel. Tento přístup umožňuje vysokou míru škálovatelnosti, kdy je vývojář schopen vyměňovat moduly, aniž by to významně ovlivnilo provázání aplikace. *View* je jednou

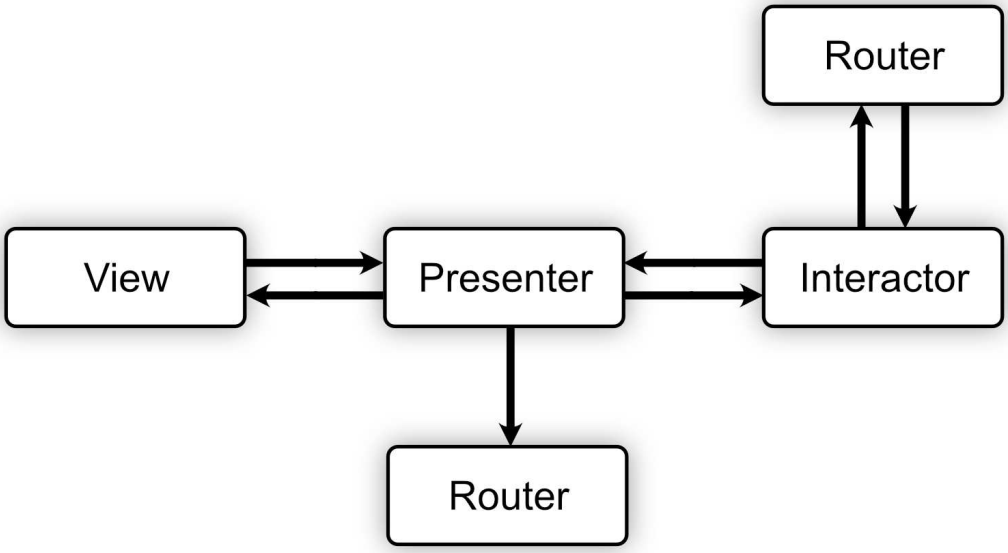


Obrázek 2.2
Schéma architektury MVVM. Podle Benoit Pasquier (2018), upraveno.

ze tříd, která již byla zmíněna v předchozích architekturách, stará se o zobrazení elementů uživatelského prostředí uživateli. Po interakci uživatele upozorní *Presenter*, třídu, která komunikuje s *View*, *Interactorem* a *Routerem*. *Presenter* komunikuje s *Routerem*, aby zjistil jaká obrazovka má být zobrazena a *Interactorem* za účelem zisku dat. Poslední třídou je *Entity*, obsahuje datové modely používaných objektů, s kterými komunikuje *Interactor*. (Sayed Mahmudul Alam, 2017) VIPER je architekturou, která díky svému rozdělení do více celků, dokáže usnadnit testování a lokalizaci chyb a problémů v jednotlivých částech aplikace. Je vhodnější pro větší projekty, kde máme více obrazovek, pravidel nebo modulů. (Michael Katz, 2020)

2.2 Aplikace pro podporu migrantů

Před vývojem jakékoli aplikace je důležité prozkoumat trh se zástupci podobných aplikací. Průzkum se provádí především z důvodu vyvarování se chyb, které udělali předchozí vývojáři nebo abychom se mohli inspirovat novými funkcionalitami pro námi zamýšlenou aplikaci. Vybral jsem celkem šest aplikací z celého světa, mezi nimi i jednu českou a právě průzkumu trhu s aplikacemi pro podporu imigrantů se bude věnovat tato podkapitola.

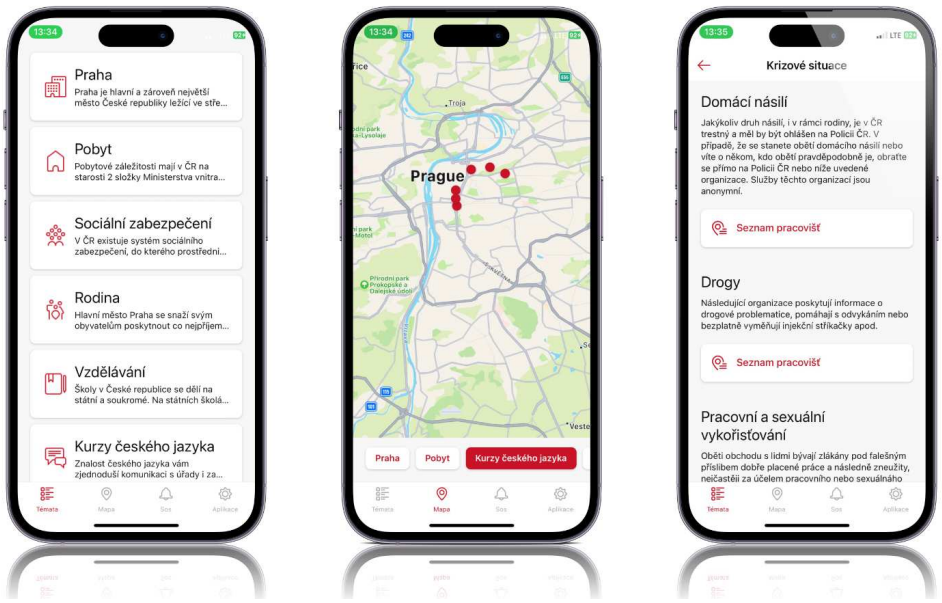


Obrázek 2.3
Schéma architektury VIPER. Podle Sayed Mahmudul Alam (2017), upraveno.

Praguer

Aplikace *Praguer* (Integrační centrum Praha, 2021) je aplikací, která pomáhá cizincům v Praze. V aplikaci nalezneme důležité informace o životě v Praze, rozdělené do různých kategorií, jako je např. sociální zabezpečení, pobyt, vzdělávání, zdravotnictví anebo informace o tom, jak si najít ve městě práci. K těmto tématům jsou navázány kontakty a jiné důležité informace, které mohou pomoci cizincům zorientovat se po Praze.

V aplikaci je možné se pohybovat pomocí seznamu kategorií anebo po mapě, kde jsou označena kontaktní místa. Místa je možno filtrovat podle zmíněných kategorií. Aplikace je dostupná nejen v angličtině, ale podporuje rovnou sedm světových jazyků. Aplikace má veliký nedostatek a to, že neukládá data přímo v zařízení, ale při každém spuštění aplikace si je musí stáhnout z online databáze. Toto může vést k dlouhému čekání na stažení dat a znemožňuje fungování bez připojení k internetu.

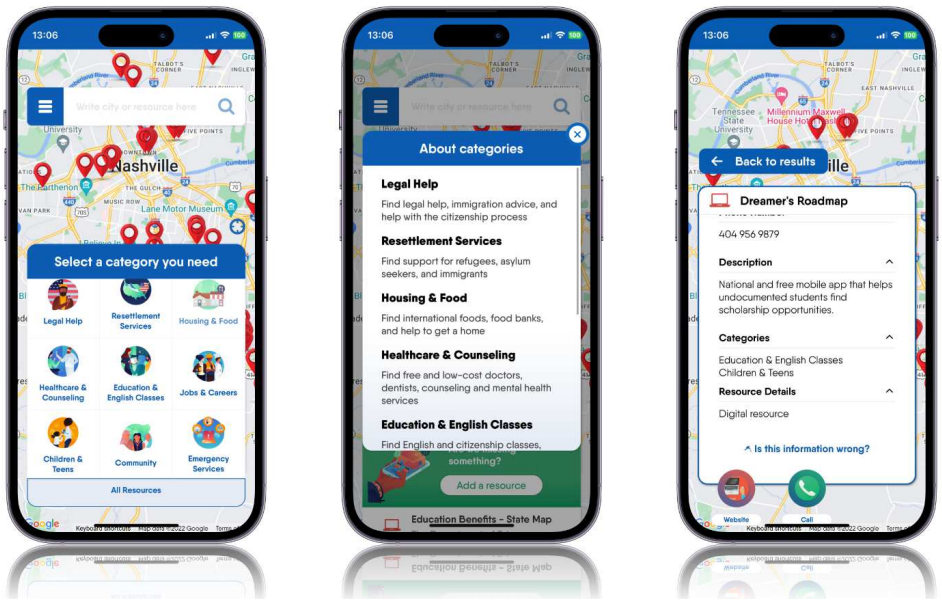


Obrázek 2.4
Mobilní aplikace Praguer pro iOS.

FindHello

Aplikace *FindHello* (USAHello, 2021) cílí na nové nebo budoucí rezidenty Spojených států amerických. Obsah aplikace se nijak neliší od aplikace *Praguer*, jediným rozdílem je skupina cílových uživatelů. *FindHello* umožňuje uživateli vyhledávat místa, kde mu mohou pomoci vyřešit problém např. se zabydlením v USA. Aplikace nenabízí pouze čtení informací, uživatel může informovat o místě, které v aplikaci není a může tak pomoci lidem, kteří hledají pomoc v neprozkoumané oblasti. Návrh pro vložení místa probíhá vyplněním jednoduchého formuláře, který poté putuje ke zpracování a ověření pověřenými lidmi.

Hlavní část uživatelského rozhraní zabírá mapa s vyznačenými místy. Místa je možné hledat pomocí textu, nebo pomocí kategorií, jako jsou např. práce, pohotovost, zdravotnictví, přesidlovací služby nebo vzdělávání. Uživatelé si při hledání míst mohou vybrat, zda hledají pomoc online, nebo fyzicky. Aplikace je dostupná ve více světových jazycích, a to konkrétně v šesti. Aplikace bohužel nepodporuje offline funkcionality.



Obrázek 2.5
Mobilní aplikace FindHello pro iOS.

RefAid

Aplikace *RefAid* (RefAid – Refugee Aid App, 2022) se nezaměřuje pouze na jednu zemi nebo region, ale pomáhá uprchlíkům po celé Evropě a USA. *RefAid* také obsahuje seznam problémových témat, se kterými se uprchlíci mohou potýkat a mapu, kde jsou kontaktní místa vyznačena. Aplikace v offline režimu ukazuje pouze místa, která jsou vzdálená v okruhu 150 km od aktuální pozice. Podle dat dostupných k 3. 12. 2022 to vypadá, že jedinými zeměmi ze střední Evropy, které se do projektu nezapojily jsou Česká republika a Slovenská republika. Ke všem místům jsou k dispozici i data o otevírací době a jiných detailech.

Nevýhodou aplikace je nutná registrace, proces je však velice rychlý. Uživatel může opět přispět místem, o kterém provozovatelé aplikace zatím nevědí, pomocí jednoduchého formuláře. Aplikace funguje i v režimu offline, což je obrovskou výhodou oproti předešlým aplikacím. Funkcionalita offline verze je omezená, ale i přesto dokáže posloužit svým účelům.

Ankommen

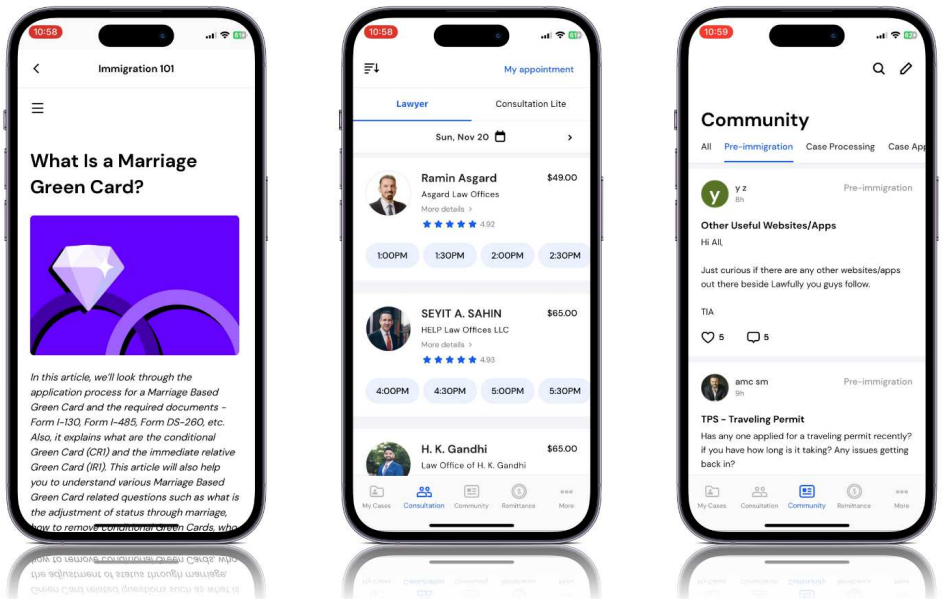
Ankommen (Bundesamt fuer Migration und Fluechtlinge, 2022) je, jak již název napovídá, určena pro přistěhovalce v Německu. Aplikace provede uživatele krok za krokem imigračními procesy, kurzy němčiny nebo procesem hledání práce. Uživatel se může také naučit o historii Německa a jeho kultuře, což může urychlit proces integrace mezi místní občany. Použitelnost aplikace se díky bezplatným kurzům němčiny může rozrůst i do jiných německy mluvících zemí, jako je např. Rakousko nebo Švýcarsko.

Navigace v aplikaci je zpočátku velice nepřehledná. Po prvním spuštění se zdá, že se aplikace načítá velice dlouho, děje se však to, že obrazovka čeká na dotek. I hlavní menu aplikace je poměrně nepřehledné. Menu je složeno ze tří obrázků, každý z nich vede k jiné sekci. Aplikace také bohužel nefunguje bez internetového připojení

Welcome app Germany

Tato aplikace je určená pro integraci lidí do Německa. I aplikace *Welcome app Germany* (Heinrich & Reuter Solutions, 2020) obsahuje seznam témat, která se nově příchozím mohou hodit, jako např. nouze, poradenství, práce nebo azyl. Tato aplikace, stejně jako *Ankommen* obsahuje část, která se stará o výuku jazyka. Výuka je však oproti předchozí aplikaci omezená pouze na pár frází. Aplikace se odlišuje seznamem konkrétních měst, u kterých je krátký popis a seznam kontaktních míst. *Welcome app Germany* také nabízí desítky videí, ve kterých je popsána historie a spousta dalších užitečných věcí o Německu. Ve videích se může uživatel dozvědět o různých tématech, jako jsou např. alkohol a drogy, práce, klima, oblečení, daně nebo životní prostředí.

Welcome app Germany mě zaujala špatným designem, aplikace vypadá velice staře a neprofesionálně. V aplikaci je na různých místech zobrazeno tlačítko s výchozím textem „Button“, které však nic nedělá. Dalším nedostatkem, který jsem objevil je, že po stažení informací o městě a jejich následném zobrazení po restartování aplikace se aplikace nečekaně ukončí.

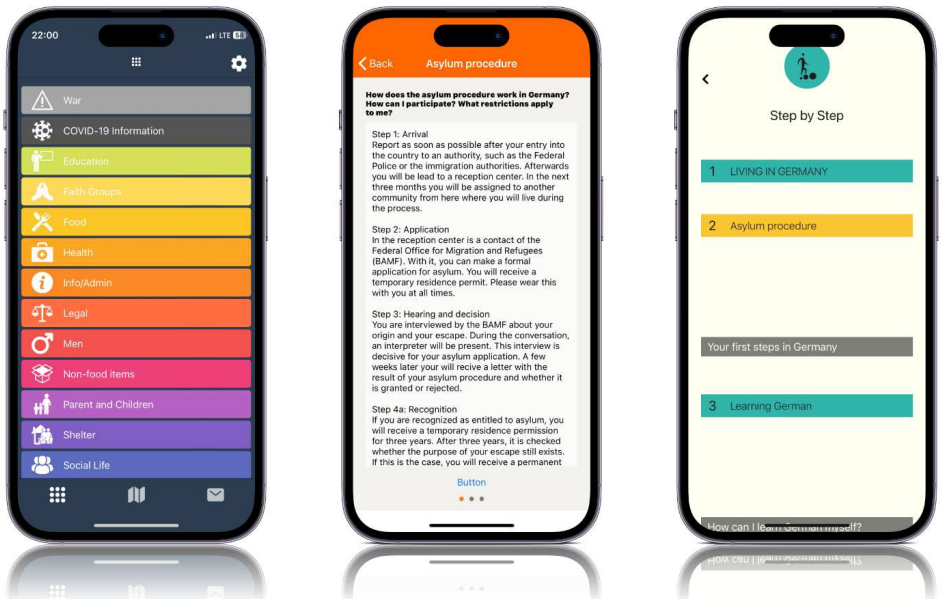


Obrázek 2.6
Mobilní aplikace Lawfully pro iOS.

Lawfully

Aplikace *Lawfully* (Lawfully, 2023) umožňuje sledování jednoho či více víz a informuje o průběžných krocích a jejího stavu pomocí notifikací. Právě notifikace jsou důležitou součástí aplikace, které předešlé zmíněné aplikace nenabízí. Uživatel si mimo sledování víz může domluvit konzultaci s právníkem přímo v aplikaci nebo komunikovat s komunitou přímo v aplikaci. Právní pomoc je k dispozici pomocí chatu nebo videohovoru za sazbu 20–100 USD/h. V komunitě je možné vyhledávat již zodpovězené otázky pomocí kategorií, jako např. finance, zelená karta nebo studentská víza. Již zodpovězené dotazy mohou pomoci migrantům urychlit proces žádosti a schválení víz. Aplikace dále nabízí různé tipy k vyřizování víz anebo denní testy, které jsou vázány na historii a celkové fungování USA. Tyto testy mohou pomoci imigrantům se získáním amerického občanství.

Aplikace mě zaujala pěkným designem i zajímavými funkcionalitami. Oproti předešlým aplikacím upozorňuje uživatele o stavu jejich požadavku a má zabudované videohovory.



Obrázek 2.7
Mobilní aplikace RefAid (vlevo), Welcome Germany (uprostřed), Ankommen (vpravo).

Srovnání aplikací

Díky srovnání aplikací jsme se mohli inspirovat funkcionalitami aplikace. Objevili jsme i pár chyb a nedostatků, kterých se při vývoji naší aplikace vyvarujeme. V tabulce 2.1 je dostupné srovnání různých aspektů jednotlivých aplikací.

Tabulka 2.1 Tabulka srovnání existujících aplikací.

	Praguer	FindHello	Ankommen	Welcome app Germany	Lawfully	RefAid
Offline mód	Ne	Ne	Ne	Ano	Ne	Ano, zobrazuje blízká kontaktní místa
Mapa	Ano	Ano	Ne	Ano, ale prázdná	Ne	Ano
Seznam tutoriálů	Ano	Ne	Ano	Ano	Ano	Ne
Výuka jazyku	Ne	Ne	Ano, interaktivní	Ano, pár frází	Ne	Ne
Komunikace s komunitou	Ne	Ne	Ne	Ne	Ano	Ne
Pravidelné aktualizace	Ano	Ano	Ne	Ne	Ano	Ano
Nedostatky ve funkcionalitách	Ne	Ne	U jednotlivých tutoriálů je špatný překlad	Ano, spousta	Ne	Ne
Zhodnocení UI	Skvělé	Nevšední, vypadá skvěle	Nepřehledné menu bez popisků	Nevkusné	Skvělé	Pomalá odezva uživatelského rozhraní

2.3 Technologie pro vývoj iOS

V této podkapitole se podíváme na srovnání technologií využívaných ve vývoji *iOS* aplikací. Dále na konkrétní technologie, které budou vhodné pro implementaci tohoto projektu, např. technologie pro práci s HTTP požadavky přes API, práci s mapou, persistenci dat nebo chatbota.

Xcode a Swift

Xcode je integrované vývojové prostředí vyvinuté společností *Apple* pro tvorbu softwaru pro platformy *macOS*, *iOS*, *iPadOS* a *watchOS*. *Xcode* poskytuje editor zdrojového kódu, nástroje pro tvorbu uživatelského rozhraní, ladicí nástroje a řadu dalších základních funkcí, které zjednodušují proces vývoje. Poskytuje také podporu jazyků jako je např. *Swift*, *C++*, *Java* nebo *Objective-C*, což vývojářům usnadňuje přepínání mezi těmito jazyky. *Xcode* je navržen tak, aby byl vysoce efektivní, uživatelsky přívětivý a poskytoval zjednodušený vývoj, který pomáhá vývojářům soustředit se na psaní kódu. *Xcode* navíc obsahuje řadu vestavěných šablon a nástrojů, které usnadňují začátek práce na nových projektech, a bezproblémově se integruje s dalšími nástroji společnosti *Apple*. (Apple, 2022; Wikipedia, 2023c)

Swift je relativně mladým programovacím jazykem, který byl představen v roce 2014 jako nástupce *Objective-C*, který byl vyvinut v 80. letech 20. století. Díky tomu, že jazyk vznikl později, může využít ponaučení z chyb, které byly nedostatkem jazyků, ze kterých čerpá. *Swift* se těší veliké popularitě díky jednodušší syntaxi a přímočarosti jazyka. (Wikipedia, 2023b)

Frameworky pro tvorbu uživatelského prostředí

Každý z frameworků pro tvorbu uživatelského prostředí přichází s výhodami a jistými nedostatky. Právě tímto se bude zabývat tato část textu, kde si představíme jednotlivé frameworky a jejich srovnání. Ve vývoji nativních aplikací se v dnešní době používají dva: *UIKit* a *SwiftUI*.

UIKit je framework, který byl představen společností *Apple* v roce 2008. Framework je určen na tvorbu aplikací pro platformy *iOS*, *iPadOS*, a *tvOS*. Základ *UIKitu* byl postaven na programovacím jazyce *Objective-C* a postupně byl pře-

psán na *Swift*. Vývojáři však stále mohou narazit na kód v *Objective-C*. Na tomto frameworku je založena většina existujících aplikací a *UIKit* stále nepřichází o oblibu a právě proto *Apple* stále *UIKit* inovuje. (Apple, 2023f; L. Jeroen, 2022)

SnapKit je populární framework třetí strany, který je používán k zjednodušení tvorby uživatelského rozhraní. Hlavním důvodem používání tohoto frameworku je *Auto Layout*, výkonný nástroj, který popisuje vztahy mezi různými *Views* aplikace. K tomuto frameworku přešli především vývojáři, kteří nedoceňovali tvorbu uživatelského rozhraní pomocí tzv. *Storyboardů*. *Storyboardy* byly složité například na verzování a přehlednost při návrhu UI. (Shai Mishali, 2019)

SwiftUI je moderní deklarativní framework pro vytváření uživatelských rozhraní pro platformy *Apple*. Poskytuje čistou, stručnou a efektivní syntaxi pro návrh prvků uživatelského rozhraní, což vývojářům umožňuje psát efektivnější kód. *SwiftUI* se bezproblémově integruje s dalšími frameworky společnosti *Apple*, což usnadňuje vytváření složitých a dynamických uživatelských rozhraní. Jednou z jeho klíčových funkcí je používání *Live Previews*, které vývojářům umožňují vidět prováděné změny v reálném čase, což činí proces návrhu intuitivnějším a efektivnějším. (Apple, 2023e)

Přestože je většina starších aplikací stále napsána a udržována v *UIKitu*, *Apple* doporučuje vývojářům přejít na novější *SwiftUI*. Pro tento projekt použijeme vývoj pomocí *SwiftUI*, nejen kvůli jeho jednoduchosti a doporučení, ale i kvůli problémům se kterými se *UIKit* potýká. Pro lepší představu je v útržcích zdrojového kódu 2.8 a 2.9 ukázka rozdílu mezi *UIKitem* a *SwiftUI*, které ukazují tvorbu tlačítka.

```
1  import UIKit
2
3  class ViewController: UIViewController {
4      override func viewDidLoad() {
5          super.viewDidLoad()
6
7          let button = UIButton(type: .system)
8          button.setTitle("Press here", for: .normal)
9          button.addTarget(self, action: #selector(buttonWasPressed),
10         for: .touchUpInside)
11         view.addSubview(button)
12     }
13
14     @objc func buttonWasPressed() {
15         print("Button was Pressed")
16     }
17 }
```

Zdrojový kód 2.8
Ukázka práce v UIKit.

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        Button(action: {
            print("Button was pressed")
        }) {
            Text("Press here")
        }
    }
}
```

Zdrojový kód 2.9
Ukázka práce ve SwftUI.

Síťová komunikace

Aby aplikace mohla komunikovat se vzdálenými uložišti dat, jako je server, musíme stanovit způsob síťové komunikace. Pro komunikaci se serverem existuje spousta knihoven a frameworků, každý z nich se však hodí na různé účely. Tato část literární rešerše se bude věnovat porovnání nejpoužívanějších technologií pro komunikaci se sítí.

Apple Foundation URL Loading System

V sadě nástrojů *Apple Foundation* můžeme nalézt sadu *URL Loading System*, která se stará o komunikaci pomocí jednoduchých URL požadavků a odpovědí. Komunikace je prováděná asynchronně, tudíž nijak nepozastaví činnost aplikace při čekání na odpověď serveru. Pomocí instance `URLSession` můžeme inicializovat komunikaci se serverem, `URLSession` může obsahovat jednu nebo více instancí `URLSessionTask`, které mohou přijímat nebo odesílat data na vzdálený server, který se poté postará o jejich zpracování. `URLSession` může běžet na pozadí, když je aplikace suspendována a může tak aktualizovat data i v tomto stavu. (Apple, 2023g)

```
do {
    let (data, _) = try await URLSession.shared.data(from: url!)
    if let response = try? JSONDecoder()
        .decode(Model.self, from: data) {
                result = response.self
    }
} catch {
    print("Invalid.")
}
```

Zdrojový kód 2.10
Ukázka dekódovaní JSON do modelu Model pomocí Apple Foundation.

Alamofire

Alamofire je síťová knihovna HTTP napsaná v programovacím jazyce *Swift*. Je široce používána pro vývoj *iOS* aplikací k provádění požadavků HTTP a práci s API. *Alamofire* zjednodušuje mnoho složitých a opakujících se aspektů práce v síti tím, že poskytuje přehledné a snadno použitelné rozhraní. Poskytuje také funkce, jako je dosažitelnost sítě, ověřování odpovědí a dekódování dat, což z něj činí ideální řešení pro komunikaci se sítí. (Alamofire, 2023)

```
AF.request(url).responseData { (response) in
    if let response = try? JSONDecoder()
        .decode(Model.self, from: response.data!) {
                result = response
    }
}
```

Zdrojový kód 2.11
Ukázka dekódovaní JSON do modelu Model pomocí knihovny Alamofire. Upraveno podle Alamofire (2023).

Perzistence dat

Aby uživatel neztratil při každém ukončení aplikace data, musíme data ukládat v nějakém uložišti. Může se jednat o bitové soubory, nebo databáze. V této podkapitole se podíváme na vybrané technologie, přístupy a jejich vzájemné srovnání.

UserDefaults

Tato technologie patří mezi jednodušší, co se týče uchovávání dat. Technologie není stavěná na uchovávání velikého množství dat, jako např. databáze. `UserDefaults`, jak již název napovídá, slouží spíše k ukládání uživatelských preferencí, např. nastavení aplikace, vzhled aplikace nebo informace o uživateli. Principem této technologie je ukládání dat v páru, který tvoří klíč a specifikovaná hodnota. (Paul Hudson, 2022b)

`UserDefaults` je jednoduchá na používání, jako většina technologií má však i své nedostatky, mezi které patří např. snadné přepsání hodnot nebo nezašifrovaná data. (florian, 2021)

V ukázce zdrojového kódu 2.12 je zobrazena práce s `UserDefaults`. V prvním řádku přiřadíme proměnné `firstRun` počáteční hodnotu `false`. Funkce `run()` poté nastaví hodnotu na `true`.

```
1  @AppStorage("firstRun") private var firstRun = true
2
3 func run() {
4     firstRun = false
5 }
```

Zdrojový kód 2.12
Ukázka práce s UserDefaults.

Ukládání souborů na disk

Některé technologie nepodporují ukládání souborů v různých formátech nebo o větších velikostech, z tohoto důvodu přichází ukládání souborů přímo na disk. Při ukládání na disk je nutné znát URL pro uložení nebo čtení souboru. Pro tento přístup vyžaduje *Apple* používání prefixu „file://“, který odliší URL pro lokální soubor od souboru, který je uložen a přístupný online.

S tímto přístupem přichází jisté výhody jako možnost ukládání objemných souborů, jednoduchost používání nebo možnost sdílení souborů mezi aplikacemi. Jednou z nevýhod ukládání souborů přímo na disk je rychlost čtení a zápisu. Adresa, na které je aplikace uložená se může kdykoli změnit a absolutní cesta k souboru již nebude použitelná. (FLORIAN, 2021)

```
1  func save(_ object: Model) {
2      do {
3          if let path = FileManager.default.urls(for: .documentDirectory,
4          in: .userDomainMask).first {
5              let data = try JSONEncoder().encode(object)
6              let url = path.appendingPathComponent("file")
7              try data.write(to: url)
8          }
9      } catch {
10         print(error)
11     }
12 }
```

Zdrojový kód 2.13
Ukázka uložení souboru na disk.

Core Data a CloudKit

Core Data je framework od firmy *Apple*, který je pro perzistenci dat při vývoji *iOS* doporučován. Framework umožňuje pomocí sad nástrojů jednoduše spravovat data v *SQLite* databázi bez nutnosti znalostí o dotazovacím jazyku *SQL*. Vývojové prostředí *Xcode* umožňuje uživatelsky přívětivou tvorbu modelu databáze ve vestavěném nástroji pro tvorbu entit. (APPLE, 2023b)

CloudKit je cloudová backendová služba společnosti *Apple* pro ukládání a načítání dat z cloudu. Nabízí jednoduché rozhraní API pro ukládání a načítání dat a funkce pro autentizaci nebo oznámení. *CloudKit* umožňuje vývojářům vytvářet cloudové aplikace a snadno spravovat data v cloudu pomocí ovládacího panelu poskytovaného společností *Apple*. (Apple 2017, 2023a)

Mezi výhody *Core Data* patří jejich jednoduchost nastavení a používání, výkon, rychlost, možnost synchronizace mezi zařízeními pomocí služby *CloudKit* nebo škála nástrojů pro práci s frameworkem přímo od společnosti *Apple*. Nevýhodou *CloudKitu* je nepřístupnost dat ze zařízení, která nejsou od značky *Apple*. (Apple, 2023a; florian, 2021)

```
1  func getContacts() {
2          do {
3              let request = Contact.FetchRequest()
4              request.sortDescriptors = [
5                  .init(key: "name", ascending: true)
6              ]
7              request.predicate = NSPredicate(
8                  format: "name == %@", "Jan"
9              )
10             contacts = try viewContext.fetch(request)
11         } catch {
12             Logger.log(error)
13         }
14     }
```

Zdrojový kód 2.14
Ukázka načtení dat s predikátem z Core Data.

```
1  func save() {
2          do {
3              let contact = Contact(context: viewContext)
4              contact.firstName = "Vincent"
5              contact.lastName = "van Gogh"
6
7              try viewContext.save()
8          } catch {
9              Logger.log("Cannot save to Core Data.")
10         }
11     }
```

Zdrojový kód 2.15
Ukázka uložení dat do Core Data.

Realm

Mezi alternativy *Core Data* patří open-source projekt *Realm*. *Realm* mimo podpory pro *iOS* zařízení podporuje i zařízení s *Android* nebo *wearables*. Tato databáze umožňuje synchronizaci dat mezi uživateli a zařízeními v reálném čase. (Missy Allan, 2017) Udává se, že *Realm* je první databáze svého druhu, která předčí nativní *Core Data*. (Josh Rondestvedt, 2020)

Mezi výhody této technologie patří rychlost komunikace s databází, synchronizace dat v reálném čase s online uložištěm nebo jednoduchost tvorby modelu databáze. (Shubham Singh, 2021)

Firestore

Firestore je databáze, která je jedním z nástrojů dostupných v rámci *Google Firebase*. Jedná se o *NoSQL* databázi, která je vhodná, jak pro mobilní aplikace, tak i pro webové aplikace. Data jsou ukládána v cloudovém uložišti, mohou však být používána i pokud je zařízení v offline stavu. Tato služba umožňuje sdílení a modifikaci dat v reálném čase, což je vhodné pro komunikaci mezi uživateli nebo mezi více zařízeními.

Jednou z hlavních výhod *Firestore* je synchronizace dat. Data mohou být modifikována offline a jakmile se mobilní zařízení připojí k internetu, data se sjednotí s cloudovým uložištěm. *Firestore* má dále výkonný dotazovací engine, který zajišťuje výkon a rychlost při komunikaci s databází. (Jessica Clark, 2021)

Práce s mapou

Aby se uživatelé aplikace *Smart Migration* mohli v České republice zorientovat, chceme vytvořit mapu, která je provede mezi kontaktními místy. Proto je nutné vybrat vhodnou technologii, která bude nejlépe splňovat naše požadavky a právě tomu se bude věnovat následující kapitola.

MapKit

MapKit je framework firmy *Apple*, který je určen k zobrazování mapy na zařízeních *iOS*, *iPadOS*, *macOS*, *tvOS* anebo *watchOS*. Framework umožňuje jednoduchou práci s mapami, vrstvami nebo anotacemi, díky nimž může být mapa interaktivnější.

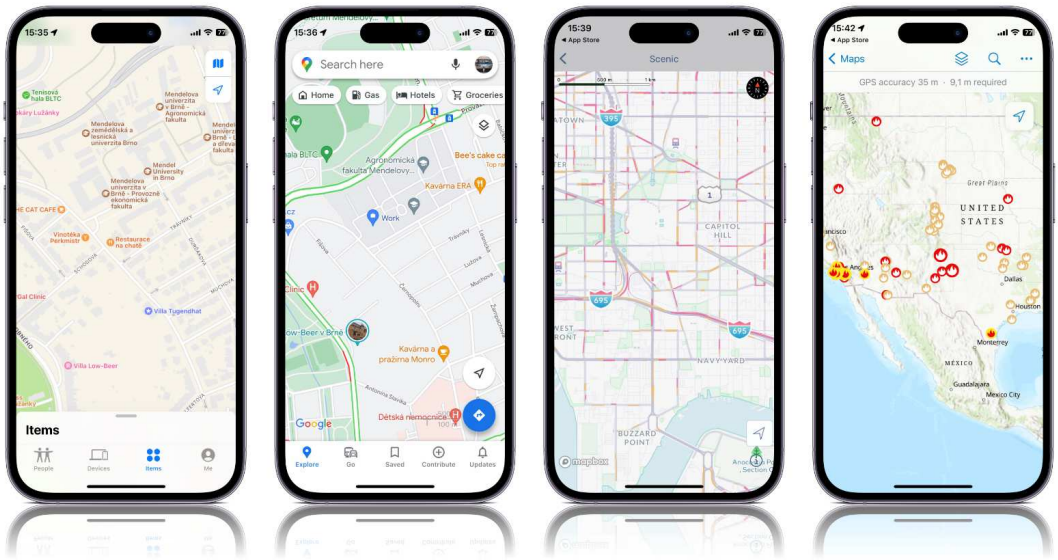
Výhodou *MapKitu* je především to, že je spravován firmou *Apple*, tudíž i doporučeným frameworkem pro práci s mapami na *iOS*. Nevýhodou *MapKitu* je především nutnost připojení k internetu nebo slabší pokrytí v menších městech České republiky, kde *Apple* zatím nedosahuje takové kvality jako např. *Seznam.cz* nebo *Google*. (Apple, 2023d)

ArcGIS SDK

Společnost *Esri* poskytuje sadu nástrojů pro práci s mapou *ArcGIS SDK*. *ArcGIS* se liší od ostatních technologií pro práci s mapou především v offline používání. Uživatel si může stáhnout část mapy a vrstvy pro práci v offline stavu. *ArcGIS* je nástrojem pro lidi, kteří se zajímají o GIS a proto *ArcGIS SDK* obsahuje pokročilé nástroje pro práci s geografickými daty a vrstvami. (ArcGIS, 2023)

Google Maps SDK

Google Maps SDK je sada nástrojů pro práci s mapou od firmy *Google*. Tato sada obsahuje nástroje pro zobrazování map firmy *Google*, vlastních map a anotací. *Google Maps SDK* poskytuje možnost práce se *Street View*. (Google, 2023) *Street View* je populární služba, která umožní procházet mapy ve 3D. Narozdíl od funkce *Apple Look Around* nabízí *Street View* nasnímané 3D mapy z celého světa. (AppCoda Editorial Team, 2017; Apple, 2023c)



Obrázek 2.16
Ukázka mapových frameworků v aplikacích. MapKit (vlevo), Google Maps SDK (uprostřed vlevo), Mapbox Maps SDK (uprostřed vpravo), ArcGIS SDK (vpravo).

Mapbox Maps SDK

Tato sada nástrojů od společnosti *Mapbox* nabízí nástroje pro práci s mapami. Vývojáři si mohou vybrat z velikého množství předdefinovaných map, nebo mohou vložit vlastní mapu, kterou mohou vytvořit v desktopové aplikaci *Mapbox Studio*. Vzhled map je dokonce možné měnit za běhu aplikace. (MapBox, 2023)

Testování

Mnoho vývojářů věří svému kódu natolik, že neprovádí testování, což může vést v mnoho případech k nepřesnostem v aplikaci. Psaní testů může být časově náročné, ale je důležitou součástí vývoje, kde se dozvíme, zda aplikace funguje podle našich představ a požadavků. Při vývoji a následnému zveřejnění je pak nutné, aby aplikace prošla více testy. K testování zdrojového kódu může vývojář využít nástrojů pro jednotkové testování a testování uživatelské přívětivosti, které jsou součástí vývojového prostředí *Xcode*. Po kontrole kódu následuje testování aplikace uživateli, kteří zkoumají aplikaci na základě zadání a pomocí

zpětné vazby dávají vývojářům najevo, zda jsou schopni s aplikací pracovat. Na základě zpětné vazby jsou vývojáři schopni provést změny v aplikaci, které zajistí vyšší míru uživatelské přívětivosti. Zmíněné testy jsou součástí zadání této práce a testování se bude věnovat samotná část textu v kapitole Metodika. Před samotným zveřejněním aplikace do *App Store* je nutné aby aplikace prošla řadou testů, které můžou zkoumat aplikaci z vnějšího pohledu a dojít k závěrům jaký bude mít aplikace dopad např. na telefon uživatele nebo na servery, se kterými komunikuje. Je vhodné aby aplikace nevyužívala příliš mnoho výkonu mobilního zařízení. Pokud má aplikace vyšší počet uživatelů, pak je vhodné zjistit zda servery budou schopny s jednotlivými instancemi aplikace rychle, bezpečně a korektně komunikovat.

Jednotkové testy jsou způsobem testování, při kterém testujeme konkrétní „jednotku“ kódu. V ukázce zdrojového kódu 2.17 si ukážeme testování metody `uppercasedFirst()`, která jako vstup přijme textový řetězec a jako výstup by měla vrátit stejný textový řetězec s velkým písmenem na začátku. Ověření funkcionality pak ověří testovací metoda `testUppercaseFirst()`. (Antoine Van der Lee, 2022)

```swift
extension String {
    func uppercasedFirst() -> String {
        let firstCharacter = prefix(1).capitalized
        let remainingCharacters = dropFirst().lowercased()
        return firstCharacter + remainingCharacters
    }
}
final class StringExtensionsTests: XCTestCase {
    func testUppercaseFirst() {
        let input = "antoine"
        let expectedOutput = "Antoine"
        XCTAssertEqual(input.uppercasedFirst(), expectedOutput)
    }
}
```

Zdrojový kód 2.17
Ukázka jednotkového testování. Zdroj: Antoine Van der Lee (2022).

K testování uživatelského rozhraní se používají tzv. UI testy. Testy uživatelského rozhraní slouží k testování jednotlivých elementů uživatelského rozhraní a jejich vzájemnou interakci s uživatelem. V tomto přístupu lokalizujeme konkrétní element uživatelského rozhraní a na tomto prvku simulujeme řadu interakcí. Je-li výsledek interakce stejný jako očekávaný, pak je test splněn. (Riyam Rudrank, 2021; David Piper, 2021)

Zajímavým přístupem k testování je tzv. *Monkey testing*. Tento přístup spočívá v náhodných dotecích, dlouhých stiscích nebo posouvání po obrazovce. K testování je možné vymezit část obrazovky, ve které budou náhodné interakce

```
1 func createEntry(app: XCUIApplication, note: String){
2         let textField = app.textFields["Enter a title"]
3         textField.tap()
4         textField.typeText(note)
5         let textView =  app.textViews["TextView"]
6         textView.tap()
7         textView.typeText(note)
8         app.navigationBars["Add Note"].buttons["Save"].tap()
9     }
```

Zdrojový kód 2.18
Ukázka testování uživatelského rozhraní. Zdroj: Chris Ching (2021).

prováděny nebo vymezit čas, ve kterém budou testy prováděny. Tímto přístupem může vývojář simulovat naprosto náhodné chování uživatele a zkoumat reakce aplikace na tyto akce. (Alexey Alter-Pesotskiy, 2023)

Chatbot

Chatbot je technologie určená k automatizované komunikaci s lidmi. První koncept chatbota přišel už v 50. letech 20. století, kdy Alan Turing navrhl tzv. Turingův test, který má za úkol prověřit, zda je technologie dostatečně inteligentní. S chatboty se v dnešní době můžeme setkat na různých místech, např. v zákaznické podpoře, při nákupu online nebo při *onboardingu* nových zaměstnanců. Na podzim roku 2022 zaznamenalo odvětví s chatboty významný pokrok, když společnost *OpenAI* představila *ChatGPT*. Tento produkt donutil veliké technologické společnosti, jako je např. *Google* reagovat na vzrůstající poptávku po umělé inteligenci, oznámením vlastního výzkumu umělé inteligence a představením asistenta *Bard*. (Wikipedia, 2023a)

ChatGPT je nástroj, který se těší veliké popularitě mezi širokou veřejností. Využití v něm najde téměř každý člověk. *ChatGPT* je schopný reagovat na různé požadavky uživatele a je schopen vytvořit např. skripty, aplikace, básně nebo novinové články. Společnost *Microsoft* do vývoje investovala 10 mld. USD, aby podpořila své produkty, jako je vyhledávač *Bing*, který nedosahuje takového počtu uživatelů jako např. *Google*. Pomocí integrace *ChatGPT* do vyhledávače *Bing* však může *Google* o místo nejoblíbenějšího vyhledávače přijít. (Forbes, 2023)

V následujícím citátu se nachází, báseň o Provozně-ekonomické fakultě Mendelovy univerzity v Brně, kterou vygeneroval ChatGPT na základě instrukce: „*Write a one verse poem about Mendelu faculty of business and economics*“. „*Where*

Brno's beauty meets business acumen, Mendelu's faculty of business and economics, a gem, Innovation and insight, a constant flow, Here, students' futures, we proudly sow." (ChatGPT, OpenAI).

Backend

K implementaci chatu v aplikaci použijeme již existujícího chatbota, který je založen na frameworku *Rasa*. *Rasa* je jedním z nejpoužívanějších nástrojů pro tvorbu chatbotů, kteří jsou schopni reagovat na text i hlas. V tomto odstavci se budeme soustředit na fungování tohoto nástroje. Mezi hlavní komponenty chatbota patří: *intenty*, entity, paměť chatbota a jeho odpovědi. *Intenty* můžeme chápat jako úmysly uživatele, to, čeho chce docílit. Entity chápeme, jako extrahovatelné kousky dat ze zprávy uživatele. Ze zprávy „Ahoj, jmenuji se Jan." může chatbot pochopit, že mým úmyslem je chatbota pozdravit a zároveň mu sdělit jméno, kterým mě může oslovovat. Kvůli návaznosti na předchozí zprávy musí mít chatbot paměť, do které může ukládat jak entity, tak i *intenty*. Chatbot bohužel sám nerozumí jazyku, který používáme ke komunikaci, lze ho však naučit některé fráze, nebo formáty různých entit. Například můžeme chatbotovi poskytnout informaci o formátu emailové adresy jako: „<some_text>@<some_text>.com", kterou poté může použít např. pro rozesílání emailů, které jsou součástí marketingové kampaně. V ukázce 2.19 si ukážeme, jak naučit chatbota rozeznat pozdrav a jaké kontaktní informace mu poskytujeme. (Aniruddha Karajgi, 2021)

```
nlu:
- intent: greet
  examples: |
    - hi
    - hello
- intent: supply_contact_info
  examples: |
    - My name is [John](name). email's [john@email.com](email)
    - name: [David](name) email: [david@email.com](email)
    - I'm [Barbara](name). My email is [barbara@email.com](email)
    - [Susan](name), [susan@email.com](email)
    - Sure. It's [Fred](name). My email is [fred@email.com](email).
```

Zdrojový kód 2.19
Ukázka učení chatbota pomocí frameworku Rasa. Zdroj: Aniruddha Karajgi (2021).

Frontend

Co se týče frontendu chatbota, musíme zobrazit komunikaci ve vhodné formě. *Apple* bohužel neposkytuje nativní framework pro zobrazování chatu. Z tohoto důvodu můžeme zobrazení zpráv navrhnout sami nebo se spolehnout na frameworky třetích stran, jako je např. *MessageKit*.

Za *MessageKitem* stojí komunita více než stovky vývojářů, kteří se snaží ulehčit tvorbu chatu v *iOS* aplikacích. Chat lze přizpůsobit potřebám vývojáře, který může měnit barvy bublin nebo jejich obsah. Framework nám také nabídne animaci, která naznačuje psaní uživatele na druhé straně chatu. (MessageKit Contributors, 2023)

Ostatní nástroje

SDWebImageSwiftUI a SDWebImageSVGCoder

V naší aplikaci budeme chtít zobrazovat obrázky stahované ze serveru a právě jedním z frameworků, který slouží k tomuto účelu je *SDWebImageSwftUI*. Pomocí tohoto frameworku můžeme asynchronně stahovat obrázky v různých formátech. Framework bohužel není ideálním pro zobrazování obrázku ve formátu *svg* a proto musíme přidat další balíček funkcí *SDWebImageSVGCoder*, který se postará o rozkódování a správné zobrazení obrázků ve formátu *svg*. Jednou z výhod *SDWebImageSwiftUI* je i *modifier* `placeholder` pomocí něhož lze určit, co se uživateli zobrazí v době, kdy se obrázek načítá. (SDWebImage, 2022)

R.Swift

Práci při používání barev, obrázků, fontů z katalogu *assetů* nám může ulehčit framework *R.Swift*. Při použití některé ze zmíněných věcí může dojít k chybě v podobě překlepu. V ukázce 2.20 je vidět rozdíl při použití obrázku standardním způsobem a pomocí *R.Swift*. *R.Swift* automaticky vygeneruje výčet *assetů*, díky němuž se vyhneme právě daným překlepům a aplikace nebude hlásit chybu, např. při nenalezení obrázku. (Mathijs Kadijk, 2023)

```swift
// Standardní přístup
let image = Image("SMLogoTransparent")
// Přístup pomocí R.Swift
let image = R.image.SMLogoTransparent()
```

Zdrojový kód 2.20
Ukázka použití frameworku R.Swift.
Zdroj: Repozitář R.swift na GitHub (2023).

SwiftLint

Znakem dobrého vývojáře je mimo jeho schopnosti tvorby aplikací i dodržování pravidel a konvencí při psaní kódu. Právě tomuto se věnuje *SwiftLint* od *Realm*. *SwiftLint* hlídá, zda je kód v souladu s doporučeními *Apple* pro psaní kódu a komunitou iOS vývojářů. Při každém spuštění aplikace ve vývojovém prostředí *Xcode* se spustí skript, který kód zkontroluje. Po vyhodnocení kódu se v *Xcode* mohou zobrazit varování nebo errory. Pravidla si může vývojář upravit podle svých preferencí nebo přidat vlastní v souboru ve formátu *yml*. V ukázce 2.21 je zobrazena tvorba vlastního pravidla, které zobrazí chybu, pokud komentář v kódu nebude odsazen alespoň jednou mezerou od „//“. (REALM, 2023)

```yaml
custom_rules:
   comments_space:
      name: "Space After Comment"
      regex: "(^ *//\w+)"
      message: "There should be a space after //"
      severity: error
```

Zdrojový kód 2.21
Ukázka tvorby SwiftLint pravidla Zdroj: Repozitář SwiftLint na GitHub (2023).

3 Metodika

3.1 Obsah a funkcionalita aplikace

Aplikace *Smart Migration* je aplikací, která poskytuje uživateli informace o vážných tématech a jejich řešení související s migrací do České republiky. Aplikace zobrazuje kontakty nebo tutoriály, které popisují konkrétní kroky řešení. Další funkcionalitou je chatbot, se kterým může uživatel komunikovat. Hlavním požadavkem jsou funkcionality fungující bez připojení k internetu, což vyžaduje perzistenci dat na zařízení. Primárním jazykem aplikace je angličtina, která je dnes brána jako univerzální jazyk. Aplikace bude však podporovat i dva další jazyky, a to ruštinu a ukrajinštinu. Tyto funkcionality jsou dány zadáním projektu.

Grafické uživatelské rozhraní aplikace je dáno již existující aplikací pro platformu *Android*. Jsou však jisté komponenty a konvence, které se pro platformu *iOS* implementují jinak. V současné době se uvažuje implementace aplikace pro Jihomoravský kraj a Ministerstvo práce a sociálních věcí. Vzhledem k tomu, že se jedná o dva jednotlivé subjekty, bude nutno dodržet odlišující prvek ve formě log, nebo např. barevného schématu. Tato aplikace je v současném stavu napojena na servery Jihomoravského kraje, tudíž i grafické uživatelské rozhraní bude v jejich barevném schématu.

3.2 Architektura a použité nástroje

Po dokončení rešerše, která se zabývá technologiemi používanými při vývoji mobilních aplikací pro *iOS*, jsem schopen dojít k závěru při volbě jednotlivých postupů nebo technologií. V této kapitole zvolím vhodné nástroje pro vývoj této aplikace.

Pro vývoj aplikace použiji programovací jazyk *Swift* a jako technologii pro tvorbu uživatelského rozhraní *SwiftUI*, jako nástupce staršího *UIKitu*. *SwiftUI* jsem zvolil, protože se jedná o novější a na syntax jednodušší framework. Dalším důvodem je, že mnoho firem přepisuje své aplikace do *SwiftUI* a díky tomuto výběru bude aplikace do budoucna snadněji udržovatelná.

Jako architekturu jsem zvolil MVVM, kvůli vhodnému rozdělení uživatelského rozhraní a kódu. MVVM je také robustním standardem ve vývoji nejen mobilních aplikací. Díky dobré škálovatelnosti, kterou architektura poskytuje, zajistíme opět jednodušší údržbu aplikace do budoucna.

Významná část dat je obsažena na serverech, se kterými bude aplikace komunikovat skrze *REST API*. Jelikož je ke komunikaci využíváno pouze jednoduchých HTTP požadavků a zpracovávání odpovědí ve formátu JSON, rozhodl jsem se pro nepoužití knihoven třetí strany a nástroje ke komunikaci si navrhnu sám. Konkrétně budu využívat nativní nástroje, které poskytuje *Swift* a *Xcode*.

Jedním z požadavků na aplikaci je také ukládání dat přímo v zařízení uživatele a jejich použitelnost bez připojení k internetu. Technologie pro perzistenci dat jsem zvolil dvě: *Core Data* a *UserDefaults*. *Core Data* bude aplikace používat k ukládání dat z *REST API*, tj. k ukládání kontaktů, tutoriálů a k nim navázaných dat, jako jsou např. tagy. *UserDefaults* jsem zvolil pro ukládání uživatelských preferencí a informacích o aplikaci. Do *UserDefaults* se bude ukládat např. uživatelem preferovaný jazyk, zda již uživatel prošel seznámením s aplikací nebo informace o oblíbených kontaktech.

Abychom ověřili správné fungování aplikace implementuji jednotkové testy a testy uživatelského rozhraní pro důležité komponenty naší aplikace. Pro zjištění, zda jsou uživatelé schopni pracovat s aplikací, provedu i uživatelské testování.

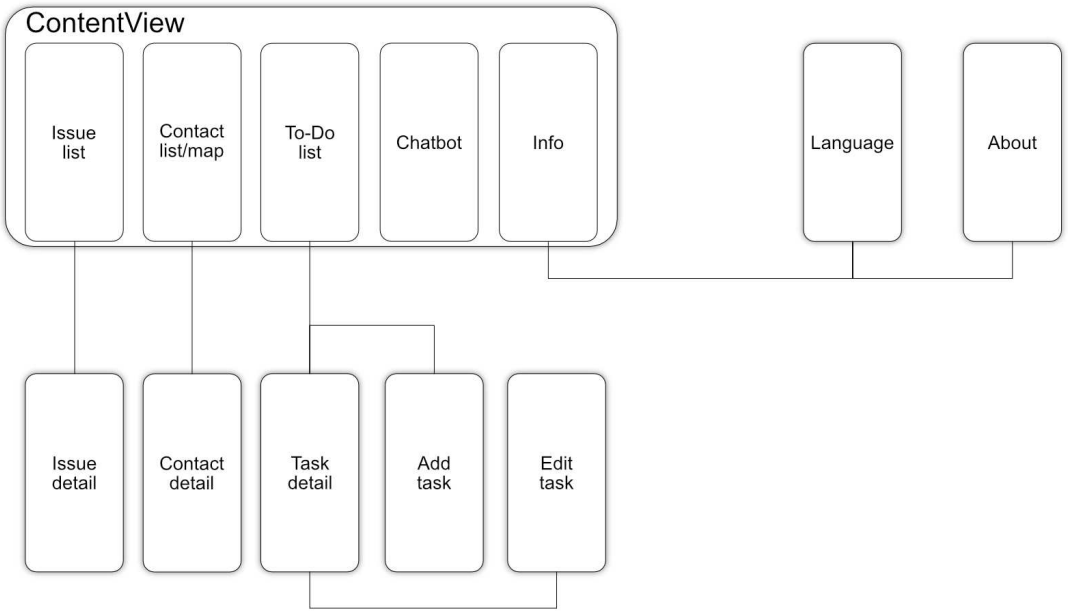
4 Implementace

4.1 Struktura aplikace

Před vývojem aplikace je nutno zmapovat zamýšlené obrazovky, z tohoto důvodu jsem si vytvořil jednoduchý diagram, který se nachází v obrázku 4.1.

Vhodná struktura složek může vývojáři ušetřit čas při navigaci mezi soubory. Takto vypadá složková struktura projektu této aplikace.

API – Zde uchovávám veškeré informace a nástroje k API.
API/Model – Tato složka obsahuje veškeré modely ke komunikaci s API.
API/Routers – Zde uchovávám jednotlivé Routers, tj. výčty, které určují jednotlivé endpointy a data požadavků nebo odpovědí, odpovídající jednotlivým entitám, které se na serveru nachází, např.: tagy, kontakty, tutoriály, verze dat.
API/Utils – Doplňkové nástroje pro práci s API.
App – Tato složka obsahuje jediný soubor. Jedná se o kořenový soubor aplikace.



Obrázek 4.1
Schéma aplikace.

CoreData – Složka s informacemi a nástroji, které slouží pro práci s *Core Data*. Obsahuje jednu podsložku a jeden soubor `CoreDataManager.swift`, který se stará o práci s *Core Data*.

CoreData/Model – Obsahuje `Model.xcdatamodel`, ve kterém jsou definovány entity pro ukládání do *Core Data*. Dále obsahuje jednotlivé extensions pro práci s danými modely.

Dependency Injection – Obsahuje soubor, který obsahuje implementaci *Dependency Injection*.

Extensions – Jednotlivé extensions pro datové typy, třídy a struktury, které poskytuje programovací jazyk *Swift*.

Model – Modely dat, se kterými aplikace pracuje. Jedná se např. o tutoriály, kontakty, varování, možnosti nebo zprávu.

Modifiers – Obsahuje *modifiers*, díky kterým můžeme upravovat chování jednotlivých objektů *SwiftUI*.

Resources – V této složce jsou uloženy obrázky, barvy, fonty nebo statické texty na základě lokalizace.

Scenes – Složka obsahující jednotlivé obrazovky naší aplikace. Mimo tyto podsložky obsahuje také `ContentView.swift`, ke kterému patří i jeho *View Model* v souboru `ViewModel.swift`.

Scenes/Chatbot – Složka obsahuje *View* a *View Model* pro chatbota.

Scenes/Contacts – Složka je rozdělena na dvě podsložky, které reprezentují obrazovku pro detail kontaktu a seznam kontaktů. Každá z těchto podsložek obsahuje *View* a *View Model*.

Scenes/Info – Obsahuje obrazovky pro zobrazení informací o aplikaci, nebo pro změnu jazyka aplikace.

Scenes/Issues – Složka je rozdělena na dvě podsložky, které reprezentují obrazovku pro detail tutoriálů a jejich seznam. Každá z těchto podsložek obsahuje *View* a *View Model*.

Scenes/ToDo – Složka, která obsahuje obrazovky pro práci s úkoly, jako: seznam, detail, vytvoření úkolu a jeho úprava.

Utils – Obsahuje nástroje, jako např. `Logger` nebo `NetworkManager`.

Views – Znovupoužitelné komponenty napříč obrazovkami.

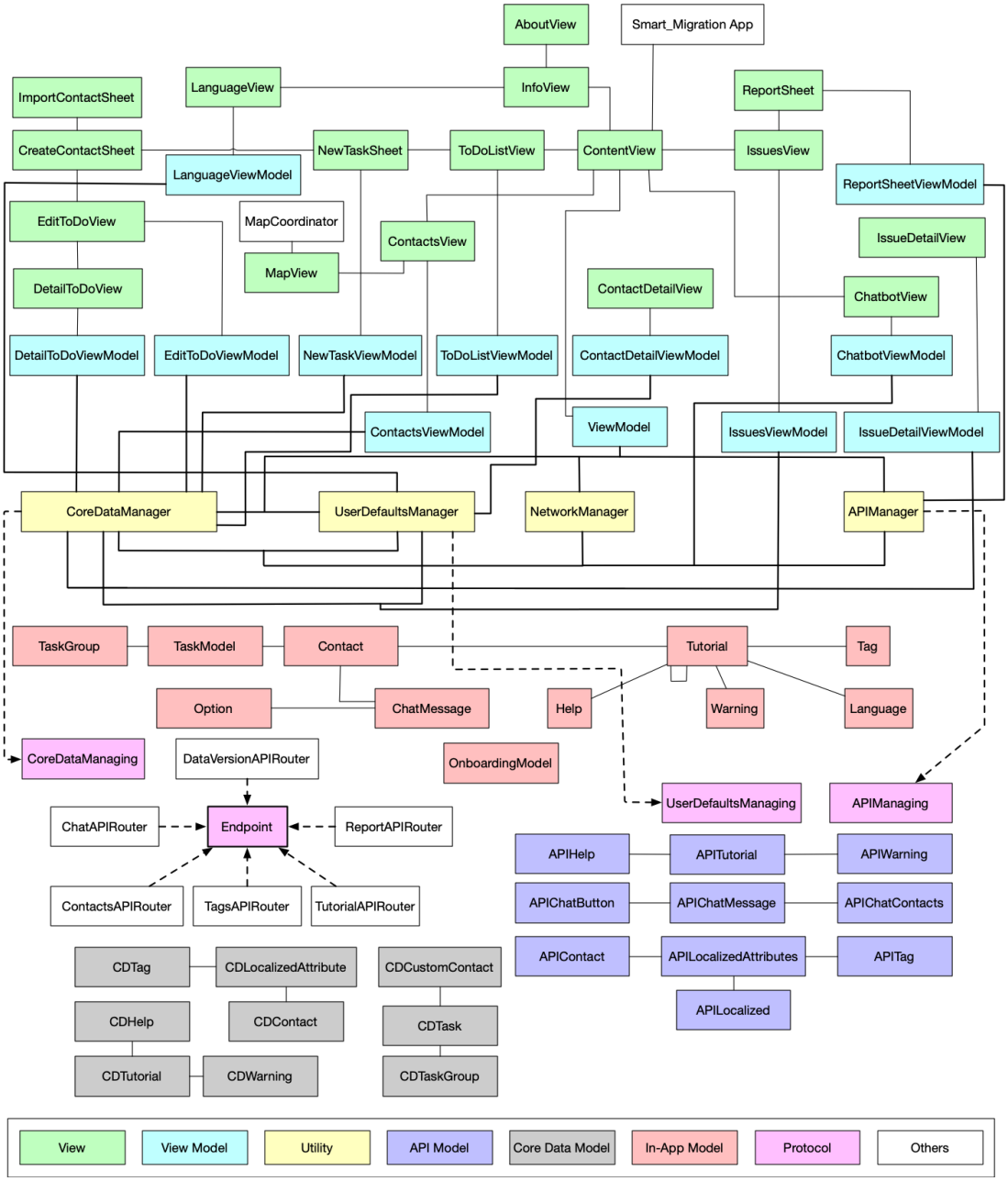
Složková struktura však nedokáže popsat samotnou strukturu aplikace ve smyslu provázanosti jednotlivých tříd, struktur a samostatných jednotek kódu. Z tohoto důvodu jsem vytvořil diagram, který zjednodušeným způsobem popisuje provázání jednotlivých objektů aplikace. Tento diagram je možné vidět na obrázku 4.2.

4.2 Architektura

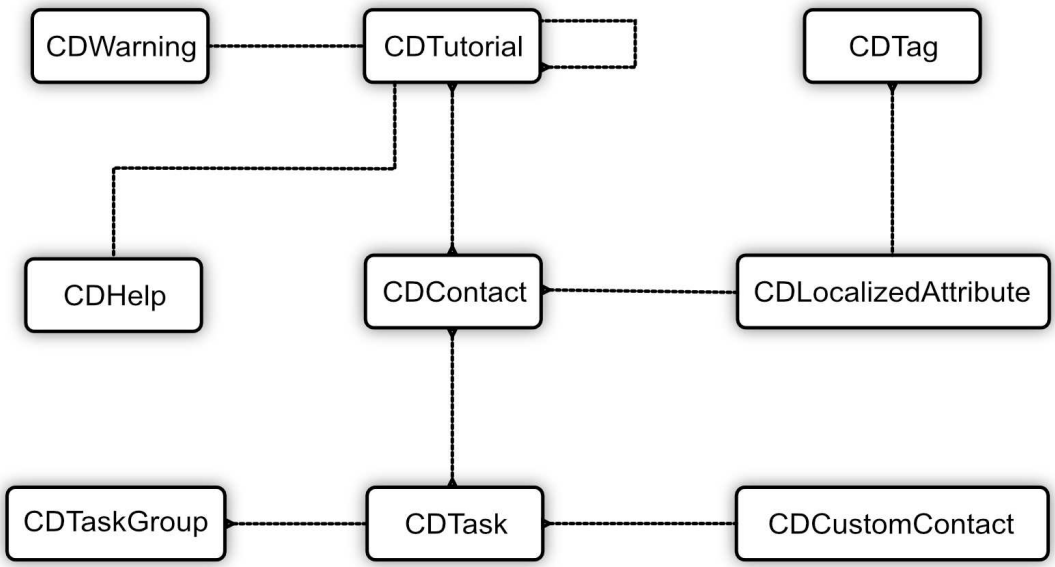
Názory, jestli je nutno mít ke každému *View* i *View Model* se rozcházejí. Z tohoto důvodu budu vytvářet *View Modely* pouze k *View*, kde to má smysl. V ukázce 4.3 je znázorněna tvorba *View Modelu* k *View*. *View Model* je anotován *property wrapperem* `@MainActor`, díky čemuž se zajistí, aby *View Model*, pomocí něhož aktualizujeme *View*, běžel na hlavním vlákně. (Paul Hudson, 2021) *View Model* je delegátem protokolu `ObservableObject`, díky čemuž jsou změny ve *View Modelu* ihned propsány do *View*, který *View Modelu* naslouchají pomocí anotace `@StateObject`. (Paul Hudson, 2022a) Ve *View Modelu* je následně možno vidět `apiManager`, který je zpřístupněn pomocí vlastního *property wrapperu* `@Injected`. Díky *Dependency Injection* dosáhneme v aplikaci lepší udržitelnosti nebo lepší možnosti testování.

```
1  @MainActor final class ViewModel: ObservableObject {
2      @Published var data: "Hello, world!"
3      @Injected var apiManager: APIManaging
4  }
5
6  struct ContentView: View {
7      @StateObject var viewModel = ViewModel()
8
9      var body: some View {
10             Text(viewModel.data)
11         }
12 }
```

Zdrojový kód 4.3
Ukázka přiřazení View Modelu k View.



Obrázek 4.2
Zjednodušený diagram popisující provázání jednotlivých částí aplikace.

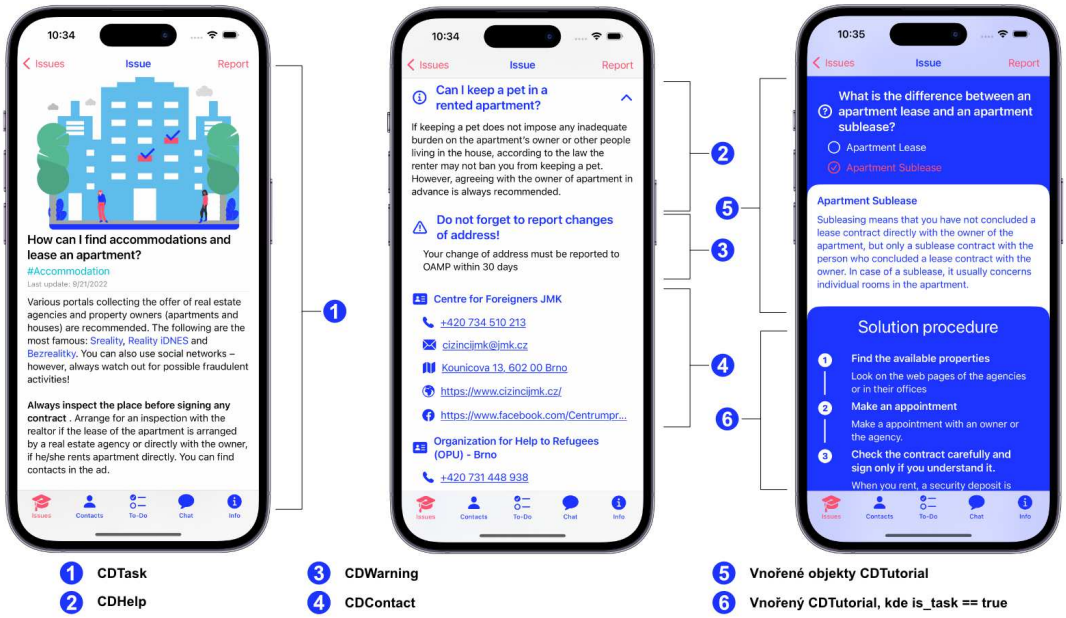


Obrázek 4.4
Schéma databáze.

4.3 Perzistence dat

V této podkapitole popíšu postup vytváření databáze *Core Data*. Vývojové prostředí *Xcode* má vestavěný editor právě pro tvorbu databáze *Core Data*. Pro vytvoření je nutno vytvořit speciální soubor, ve formátu *xcdatamodel*. V tomto souboru vytvořím jednotlivé entity, které budou obsahovat data k následnému offline použití. Mezi entitami je možné vytvořit vazby typů 1 : 1, 1 : M nebo N : M. V našem případě použijeme vazbu například mezi entitou `CDTutorial` a `CDWarning`. Jedná se o vazbu 1 : 1, kdy objekt `CDTutorial` může obsahovat pouze jeden objekt `CDWarning` a naopak. Celé schéma databáze je vyobrazeno v obrázku 4.4. Obrázek 4.5 pak znázorňuje detail tutoriálu , kde je vidět využití a provázanost jednotlivých entit.

Další technologií pro perzistenci dat je *UserDefaults*. V *UserDefaults* budeme ukládat např. informaci o dokončení *onboardingu* nebo o nastaveném jazyku aplikace. Po dokončení *onboardingu* se do *UserDefaults* pod klíčem `onboarding_done` uloží hodnota `true`, díky níž se při dalším startu aplikace *onboarding* nespustí a uživatel ho nemusí znovu absolvovat.



Obrázek 4.5
Ukázka použití entit z databáze v detailu tutoriálu.

4.4 Síťová komunikace

Jak již bylo zmíněno v úvodu do metodiky, ke komunikaci aplikace se serverem nepoužiji knihovnu třetí strany, kvůli jednoduchosti komunikace. Vytvoření vlastních nástrojů nám také umožní vědět a definovat, jak přesně komunikace funguje.

Základem komunikace je dotazování se jednotlivých endpointů, proto si navrhnu protokol `Endpoint`. Protokol určuje povinné vlastnosti a funkce tříd, kterým se říká delegát. Každý delegát musí obsahovat tyto vlastnosti a funkce. V útržku kódu 4.6 se nachází definice protokolu `Endpoint`. Delegáty tvoří jednotlivé *routery*, z nichž je každý určen pro zpracovávání jednotlivých endpointů.

Samotnou komunikaci pak zprostředkovává třída `APIManager`, díky které můžeme ze serveru získávat data pro naší aplikaci, zjistit jejich aktuálnost nebo komunikovat s chatbotem a to vše pomocí jednoduchých metod. Díky generickému programování jsme schopni použít univerzální metodu pro každou entitu, kterou ze serveru můžeme získat. V ukázce 4.7 je zobrazeno získání dat pomocí jedné z univerzálních metod.

```
protocol Endpoint {
    var path: String { get }
    var method: HTTPMethod { get }
    var urlParameters: [String: Any]? { get }
    var headers: [String: String]? { get }
    var postBodyParameters: Data? { get }

    func asRequest(chatting: Bool) throws -> URLRequest
}
```

Zdrojový kód 4.6
Ukázka protokolu Endpoint. Zdroj: Repozitář STRV na GitHub (2022).

```
@Injected private var apiManager: APIManaging

let apiResponse: [APITutorial] = await apiManager.fetchAPIEntity()
```

Zdrojový kód 4.7
Ukázka získání seznamu tutoriálů ze serveru.

4.5 Backend

Aplikaci budou poskytována data ze vzdáleného uložiště, na které je již napojena aplikace *Smart Migration* pro *Android*. V této podkapitole stručně popíšu napojení aplikace na server.

Aplikace využívá pro stahování dat *REST API*. Po odeslání požadavku GET z aplikace, server požadavek zpracuje a aplikaci vrátí odpověď ve formě JSON, viz. ukázka 4.8. Aplikace tímto způsobem stahuje informace různých entitách:

.../data-version?fromTimestamp=1680816549 – Endpoint, kde aplikace kontroluje data. Jako parametr zadává čas poslední kontroly v UNIX formátu. Odpovědí z endpointu tvoří pole s názvy tabulek, které byly změněny.

.../contacts/ – Tento endpoint požíváme pro stažení veškerých informací k jednotlivým kontaktům.

.../faq/tags/ – Tento endpoint vrací tagy, na které jsou následně napárovány tutoriály.

.../faq/items/?nested=0&withHtml=true – Zde aplikace získává tutoriály. V našem případě chceme, aby odpověď vracela plochou formu dat, proto specifikujeme parametr `nested` s hodnotou 0. Dalším parametrem, který musí být nastaven je `withHtml`, díky němuž budou data ze serveru chodit v HTML formátu.

.../faq/reports/ – Endpoint, na který odesíláme požadavek metody POST, kde specifikujeme tělo požadavku. Tento endpoint používáme pro nahlášení chyb v jednotlivých tutoriálech. Tělo požadavku tvoří popis chyby a identifikátor daného tutoriálu.

.../webhooks/myrest/webhook – Tento endpoint používáme k odeslání a přijímání zpráv pro chat. Jedná se opět o endpoint, se kterým komunikujeme pomocí metody POST. Tomuto endpointu se bude více věnovat samostatná podkapitola.

```
{
    "items": [
        {
            "id": 138503,
            "name": "Centre for Foreigners",
            "color": "#E91E63",
            "order": 8,
            "localizedAttributes": {
                "ru": {
                    "name": "Centr dlja inostrancev"
                },
                "uk": {
                    "name": "Centr dlja nozemcv"
                },
                "en": {
                    "name": "Centre for Foreigners"
                }
            }
        }, ...
    ]
}
```

Zdrojový kód 4.8
Ukázka odpovědi serveru (kontakt).

4.6 Lokalizace

Aby bylo možné poskytovat služby skrze naší aplikaci širšímu spektru uživatelů, musíme vyřešit možnost spuštění aplikace v jiných jazycích. V současné době se uvažují pouze tři jazyky, a to: angličtina, ukrajinština a ruština. Naším cílem je použít takové nástroje, aby aplikace byla snadno rozšiřitelná i na další jazykové mutace.

Lokalizování aplikace pro platformu *iOS* lze zajistit velice jednoduše. V první řadě je nutné si navolit podporované jazyky přímo ve vývojovém prostředí *Xcode*. V následujícím kroku je nutno vytvořit soubor, který obsahuje veškeré textové řetězce, které jsou v aplikaci použity a nejsou stahovány. Soubor se

v *Xcode* zobrazí ve více variantách, pro každý jazyk jinak. Textové řetězce jsou v souboru uloženy ve formě, která je ukázána ve výstřižku kódu 4.9, kde můžeme vidět i jeho následné použití v uživatelském rozhraní, kde k němu přistoupíme pomocí `LocalizedStringKey`.

V aplikaci je nutno zajistit i systémové hlášky, které budou odpovídat jazyku zařízení. Systémové hlášky aplikace zobrazuje, žádá-li například o přístup k datům uživatele. Z tohoto důvodu jsem musel vytvořit nový soubor s textovými řetězci, kde jsem napároval odpovídající hlášky ke klíči oprávnění.

```
1 "add_time_and_date" = "Add time and date";
2 "remove_time_and_date" = "Remove time and date";
3 ...
4 Text(LocalizedStringKey("add_time_and_date"))
```

Zdrojový kód 4.9
Ukázka lokalizace aplikace pro iOS.

4.7 Tutoriály

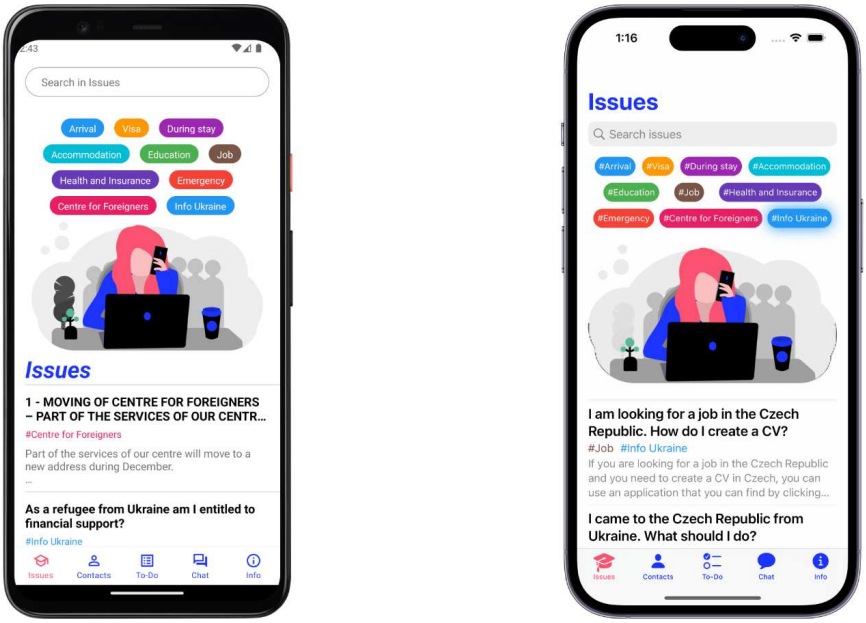
V této podkapitole se zaměříme na část, která je pro naší aplikaci esenciální, tj. část, která se zabývá tutoriály. Hlavním důvodem, proč si naší aplikaci přistěhovalci budou stahovat, bude především zájem o správné řešení různých problémů, se kterými se mohou potýkat. Tuto část aplikace budou tvořit dvě obrazovky: seznam a detail tutoriálu.

4.7.1 Seznam tutoriálů

Obrazovku tutoriálů tvoří dvě hlavní části. První částí je část pro filtrování tutoriálů. Uživatel může filtrovat tutoriály dle svých potřeb pomocí tagů nebo vyhledávacího textového políčka. Po definici filtrů se uživateli zobrazí v seznamu pouze tutoriály, o které má zájem.

Další část této obrazovky tvoří samotný seznam. Seznam je navrhnut v jednoduchém uspořádání prvků pod sebou, aby byla aplikace přehlednější a korespondovala s designem aplikace pro platformu *Android*. Obsah jednotlivých položek seznamu tvoří nadpis tutoriálu, k němu přiřazené tagy a text, který je pro přehlednost omezen na tři řádky. Celé toto políčko je obaleno do navigačního linku, díky čemuž se uživatel může prokliknout na detail daného tutoriálu.

V obrázku 4.10 je možno vidět porovnání seznamu tutoriálů na platformě *iOS* a *Android*.



Obrázek 4.10
Srovnání zobrazení seznamu tutoriálů mezi Android (vlevo) a iOS (vpravo).

4.7.2 Detail tutoriálu

V detailu tutoriálu může uživatel vidět informace k danému tutoriálu. Informace mohou být např. obecný popis, varování, pomoc, zdroj, kontakty nebo navazující tutoriály.

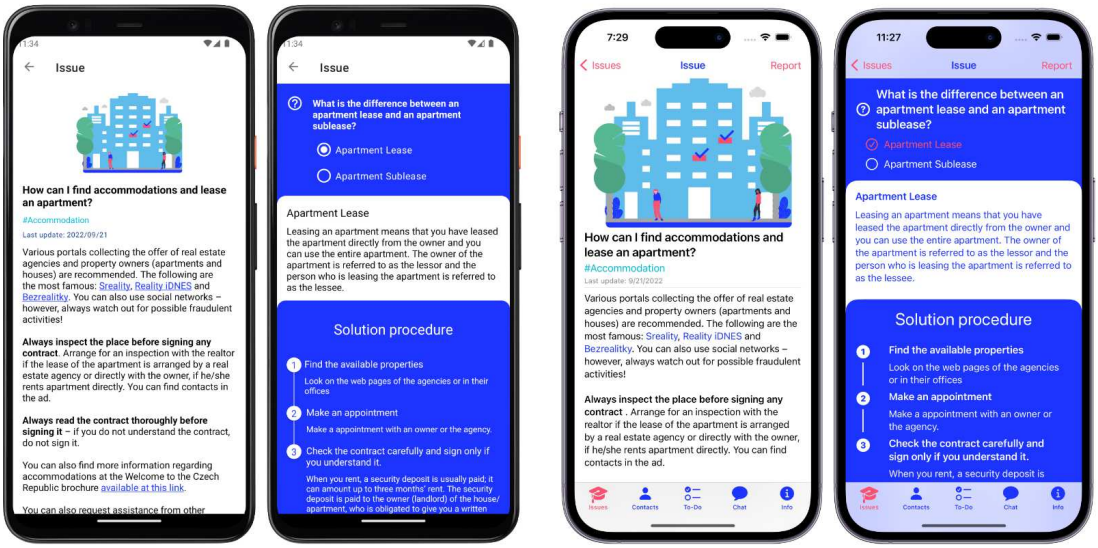
Uživatele uvítá hlavička, kterou skládá obrázek ve formátu *svg*, který je navázán na první napojený tag, nadpis, seznam tagů a informaci o aktuálnosti dat tohoto problému.

Po hlavičce následuje hlavní text, který ze serveru přichází jako kód ve formátu HTML. Pro zobrazení, které zadavatel dat zamýšlel, musíme tento text převést pomocí vhodných regulárních výrazů na tzv. `AttributedString`. `AttributedString` umožňuje modifikované zobrazení textu.

V útržku kódu 4.11 je možno vidět tvorbu tučného textu a odkazů za pomoci `AttributedString`.

```
1 Text("""
2      **Bold text**
3      [Google link](https://google.com)
4      """)
```

Zdrojový kód 4.11
Ukázka práce s Attributed String.



Obrázek 4.12
Srovnání zobrazení detailu tutoriálů mezi Android (vlevo) a iOS (vpravo).

Jednou z dalších zobrazovaných informací je seznam kontaktů. Jedná se o hlavičku se jménem kontaktu a k němu napárované informace. Tyto informace jsou zobrazovány ve formě odkazů. Po kliknutí na telefonní číslo se zobrazí dialogové okénko, které si žádá potvrzení volání na dané telefonní číslo. Odkaz s adresou otevře aplikaci *Apple Maps* s předdefinovaným cílem, který tvoří právě adresa. Další odkazy tvoří webová stránka a Facebook adresa kontaktu, které po prokliknutí otevřou webový prohlížeč.

Další část obrazovky tvoří navazující otázky. Každý z tutoriálů může obsahovat vnořené tutoriály. Má-li daný tutoriál navazující otázku, zobrazí se uživateli na výběr více možností. Po výběru odpovědi na navazující otázku se uživateli zobrazí kartička, která zobrazuje stejné informace jako v předchozích odstavcích. Každá z těchto kartiček reprezentuje samostatný tutoriál, jenž je potomkem hlavního tutoriálu.

Poslední částí obrazovky tvoří postup řešení. Jednotlivé kroky jsou také tutoriály, v tomto případě přímými potomky, kteří jsou označeni boolean hodnotou `isTask`. V každém z těchto kroků řešení mohou být opět zobrazeny informace jako např. varování, kontakty nebo pomoc. Veškeré tyto kroky je možné exportovat do úkolníčku pomocí tlačítka „Save tasks“.

Najde-li uživatel v detailu tutoriálu chybnou nebo zavádějící informaci, může tento problém ohlásit pomocí tlačítka „Report“, které je umístěno na horní liště. Po stisknutí se uživateli zobrazí formulář, jehož vyplnění a odeslání zajistí upozornění pověřených lidí, kteří se tomuto problému budou věnovat.

4.8 Kontakty

4.8.1 Seznam kontaktů

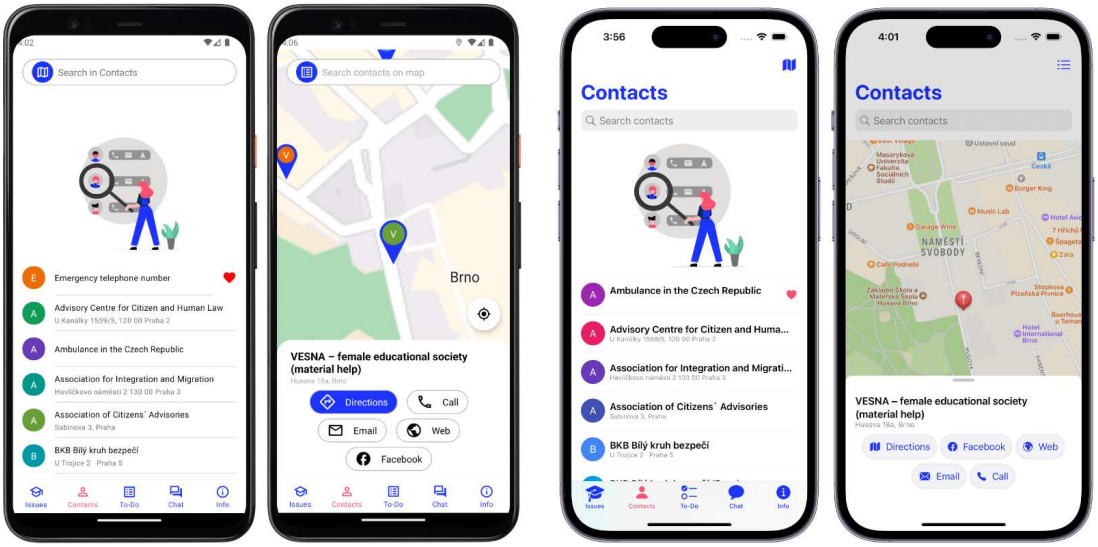
Seznam kontaktů má dvojí podobu, a to mapu a seznam. Mezi těmito módy zobrazení se může uživatel jednoduše přepnout pomocí tlačítka na horní liště. Jelikož se jedná o obrazovky se stejným cílem, tak mapa a seznam sdílí jeden společný *View Model*.

Seznam kontaktů je opět uspořádán k zobrazení kontaktů přímo pod sebou. Jednotlivé buňky seznamu tvoří grafické zdůraznění prvního písmenka názvu kontaktu, které je vloženo do barevného kolečka. Tento prvek byl do designu zařazen, aby se uživatel lépe orientoval mezi kontakty s jiným začínajícím písmenem. Mimo kolečka buňku doplňuje název kontaktu a jeho adresa. Řádek seznamu může také zobrazovat obrázek srdce, a to v případě, že je kontakt zařazen mezi oblíbené. Kontakty v seznamu mají dvě priority pro řazení, první je řazení oblíbených kontaktů jako první, následuje abecední řazení.

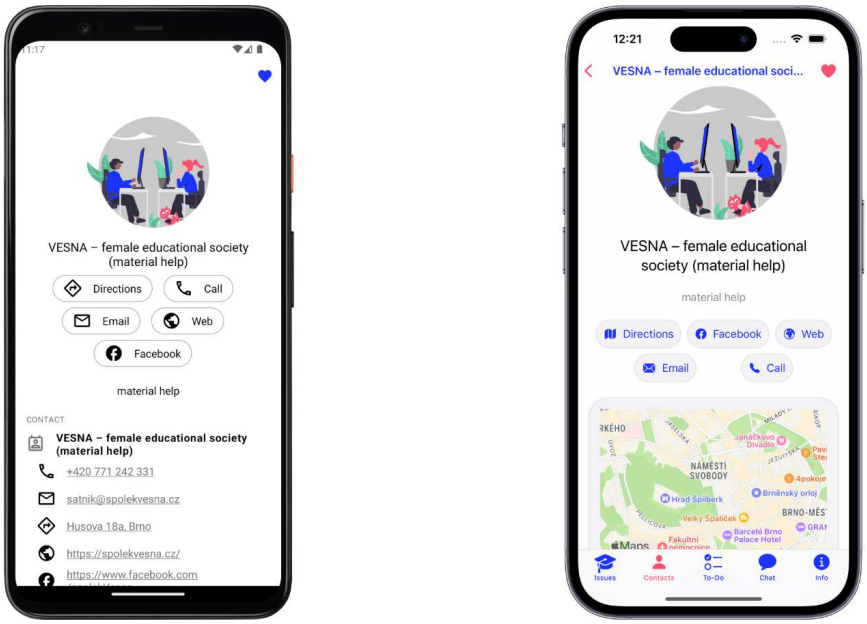
Zobrazení na mapě zajišťuje nativní mapový framework `MapKit`. Jednotlivé kontakty jsou na mapě zobrazovány pomocí anotací. Při inicializaci mapy je střed mapy nastaven, kvůli vysoké koncentraci kontaktních míst na Brno. Po kliknutí na mapovou anotaci kontaktního místa se uživateli vysune okénko kontaktu v podobě modulárního *sheetu*. Obsah *sheetu* tvoří základní informace o kontaktu, jako jeho adresa nebo název. Tyto informace doplňují tlačítka pro volání, přepnutí do *Apple Maps*, poslání emailu nebo otevření prohlížeče pomocí odkazů na webové stránky nebo Facebook. Uživatel si také pomocí tohoto *sheetu* může zobrazit i podrobnější detail tohoto kontaktu, který je popsán později. Aby uživatel docílil podrobnějšího zobrazení detailu kontaktu, musí vysunout *sheet* směrem nahoru.

4.8.2 Detail kontaktu

Samotný detail kontaktu je opět tvořen úvodním obrázkem ve formátu *svg*, který je v tomto případě pro všechny kontakty stejný. Hlavičku doplňuje název kontaktu a jeho případný popis. Následují ke kontaktu přiřazená tlačítka, splňující stejné funkce jako v modulárním *sheetu* u mapy kontaktů. Design je oproti *Android* aplikaci doplněn výstřižkem mapy, díky čemuž uživatel tuší, kde

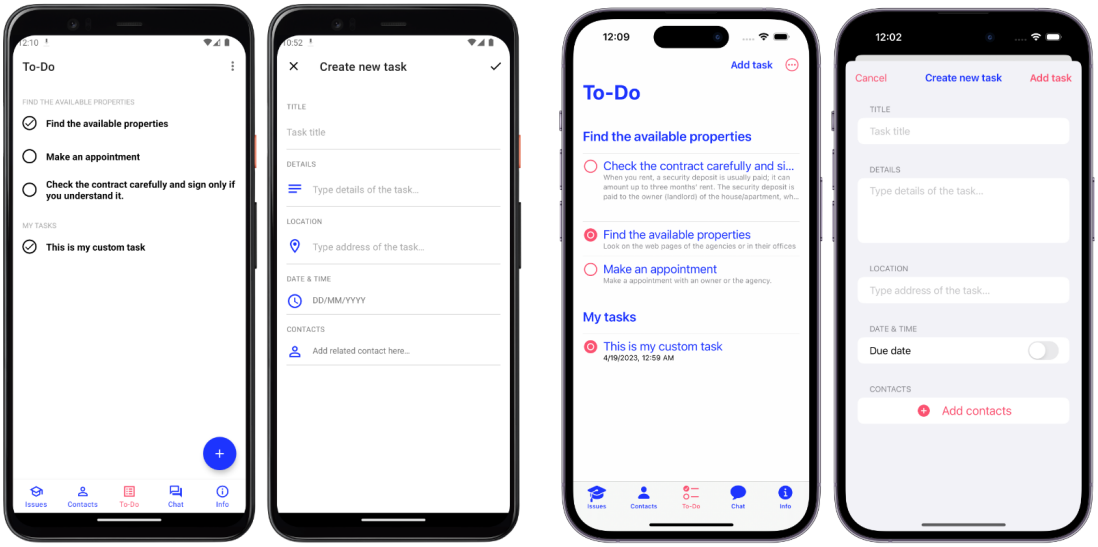


Obrázek 4.13
Srovnání zobrazení seznamu kontaktů mezi Android (vlevo) a iOS (vpravo).



Obrázek 4.14
Srovnání zobrazení detail kontaktu mezi Android (vlevo) a iOS (vpravo).

se místo nachází. Detail zakončuje výpis jednotlivých kontaktních informací. Uživatel může každý kontakt označit jako oblíbený pomocí tlačítka, které je umístěno na pravé straně navigační lišty.



Obrázek 4.15
Srovnání zobrazení seznamu úkolů a jejich vytváření mezi Android (vlevo) a iOS (vpravo).

4.9 Úkolníček

Aby uživatel na nic nezapomněl obsahuje aplikace také část, která se věnuje ukládání úkolů. Úkoly si může uživatel vytvořit vlastní anebo je přímo importovat z jednotlivých tutoriálů. Pro zobrazení jednotlivých úkolů využije uživatel `TabItem`, které nese název „To-Do“. Po přepnutí na tuto kartu se uživateli zobrazí seznam úkolů, které je možno filtrovat, podle jejich dokončení. Pro vytvoření vlastních úkolů využije uživatel tlačítko, které se nachází na horní liště karty, jenž vysune *sheet*, ve kterém může uživatel zadávat jednotlivé podrobnosti úkolu. Uživatel může přidat základní podrobnosti jako např. název úkolu, popis, lokaci nebo čas, dokdy má být úkol dokončen. K úkolu je možné taky navázat kontakty, které stahujeme ze serveru nebo si uživatel může přidat kontakty vlastní. Pro lepší přehled úkolů jsou zobrazovány podle skupin, které nesou název po tutoriálu, z něhož byly importovány.

Pro lepší zobrazování úkolů jsem oproti aplikaci pro *Android* připravil i detail jednotlivých úkolů, kde uživatel může vidět jeho podrobnosti. V této obrazovce může uživatel také úkol dokončit, pomocí jednoduchého tlačítka. Uživatel má nadále možnost úkol editovat ve formuláři, který je stejné struktury jako formulář pro vytváření.

V obrázku 4.15 je možné vidět srovnání obrazovek zobrazujících seznam úkolů a jejich tvorbu.

4.10 Chatbot

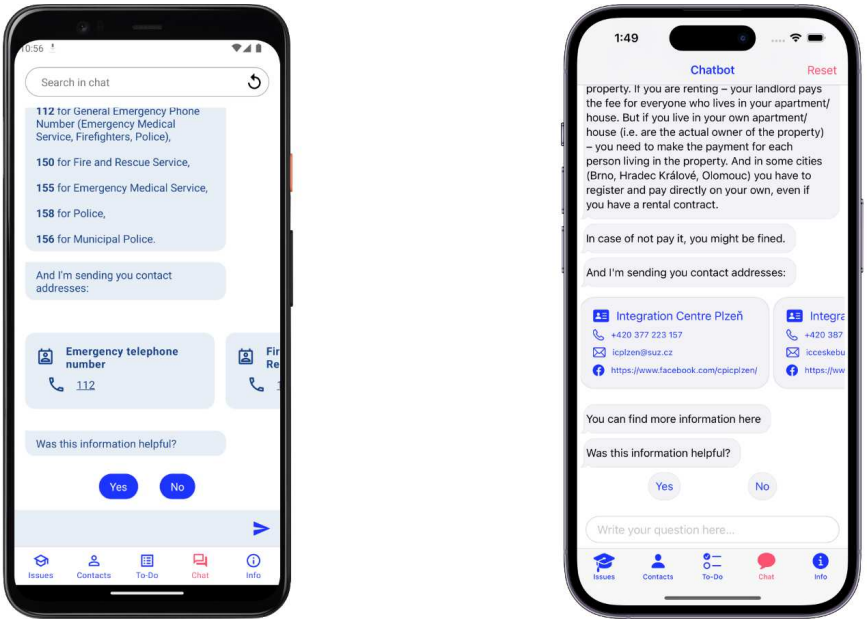
Nejzajímavější součástí aplikace se může jevit chatbot. Pro zobrazení chatbota používáme pouze jednu obrazovku, která obsahuje jednotlivé zprávy a políčko pro zadání vlastní zprávy. Zprávy mohou být trojího typu: textové zprávy, skupina kontaktů nebo výběrová tlačítka. Doporučená komunikace s chatbotem je právě pomocí tlačítek, které reprezentují zprávy, na které je chatbot schopen odpovědět. Uživatel může také využít textového pole ve spodní části obrazovky, je však možné, že chatbot na tuto zprávu nebude znát odpověď. Každá z textových zpráv je zobrazena ve tvaru bubliny, která byla navržena pomocí Bézierovy křivky. Tyto zprávy jsou narozdíl od ostatních typů zobrazovány v každé části konverzace. Ostatní z typů zpráv jsou zobrazovány, pouze jedná-li se o poslední zprávu.

Komunikační bot založený na frameworku RASA byl již navržen a implementován, není tedy součástí řešení této práce. Komunikace probíhá pomocí POST požadavků, kde se v těle požadavku definují následující informace: jazyk komunikace, zpráva uživatele a identifikátor uživatele. Identifikátor uživatele je prozatím tvořen identifikačním číslem telefonu, které je unikátní. Po odeslání požadavku na server se v odpovědi vrátí pole zpráv. Každá z těchto zpráv může obsahovat pole identifikátorů kontaktů nebo pole tlačítek, které představují možnosti odpovědí. V útržku kódu 4.16 je vidět odpověď chatbota, kde nám přijdou dvě zprávy. Každá zpráva má `recipient_id` reprezentující identifikátor uživatele, text, který je zobrazen v aplikaci. Poslední zpráva obsahuje také dvě tlačítka, kde `title` reprezentuje text tlačítka a `payload` hodnotu, která se odesílá na server po stisknutí tlačítka.

Zahájení chatu probíhá skrze speciální zprávu, která je odeslána ihned po zobrazení chatu v aplikaci. Uživatel může celou konverzaci obnovit na začátek pomocí tlačítka v horní liště.

```
[
    {
        "recipient_id": "132232234",
        "text": "Thanks for this feedback"
    },
    {
        "recipient_id": "132232234",
        "text": "Any other problem or task?",
        "buttons": [
            {
                "title": "Yes",
                "payload": "\/affirm"
            },
            {
                "title": "No",
                "payload": "\/deny"
            }
        ]
    }
]
```

Zdrojový kód 4.16
Ukázka odpovědi chatbota.



Obrázek 4.17
Srovnání zobrazení chatu mezi Android (vlevo) a iOS (vpravo).

5 Testování

V této kapitole se zaměříme na testování aplikace pro kontrolu její funkčnosti a také z hlediska uživatelské přívětivosti.

5.1 Jednotkové testy a testy uživatelského rozhraní

K jednotkovému testování jsem použil knihovnu, která je součástí vývojového prostředí *Xcode*, `XCTest`. Otestoval jsem nástroje, které jsem navrhl pro komunikaci se serverem nebo získávání dat z databáze *Core Data*.

K testování uživatelského rozhraní jsem opět použil knihovnu `XCTest`. Jako subjekty testování jsem použil všechny z pěti částí této aplikace. Pro tutoriály jsem testoval fungování *sheetu* pro odesílání reportů, reakci na odpověď na navazující otázku nebo filtrování pomocí tagů a textu. Pro kontakty jsem otestoval přepínání mezi listem a mapou pro zobrazení kontaktů nebo navigaci na detail kontaktu. V úkolníčku jsem otestoval přidání úkolu a jeho následné zobrazení v seznamu úkolů. V části aplikace pro chatbota jsem otestoval komunikaci se serverem, obnovení konverzace nebo komunikaci pomocí tlačítek pro výběr odpovědi. Dalším testem byl test pro kontrolu změny jazyka.

```
1 func testFilteringByTags() {
2         app.buttons["#Job"].firstMatch.tap()
3         XCTAssert(!app.staticTexts[
4             "Postponement of Compulsory School Attendance"
5         ].firstMatch.exists)
6     }
```

Zdrojový kód 5.1
Ukázka testu pro filtrování tutoriálů pomocí tagů.

Ne všechny tyto testy proběhly podle očekávání a já musel nedostatky do aplikace zapracovat. V současné době funguje vše, jak má a testy končí s pozitivním výsledkem.

Do následující tabulky jsem vypsal všechny provedené testy a jejich popis.

Tabulka 5.1 Tabulka popisující jednotkové testy a testy uživatelského rozhraní.

Název testu	**Popis testu**	**Typ testu**
testCoreDataFetch()	Testuje získávání objektů z *Core Data*.	Jednotkový test
testDataCheck()	Testuje validitu dat z endpointu pro kontrolu aktuálnosti dat.	Jednotkový test
testDataFromAPI()	Testuje zisk dat z API a jejich následné přetypování.	Jednotkový test
testChat()	Testuje, zda chatbot vrací validní data.	Jednotkový test
testLanguageChange()	Testuje, zda funguje přepínání jazyka aplikace	Test uživatelského rozhraní
testReset()	Testuje obnovení konverzace na začátek.	Test uživatelského rozhraní
testAddingTask()	Testuje přidání vlastního úkolu.	Test uživatelského rozhraní
testModeSwitch()	Testuje přepnutí mezi listem kontaktů a mapou.	Test uživatelského rozhraní
testNavigateToDetail()	Testuje navigaci do detailu kontaktu.	Test uživatelského rozhraní
testIssueReport()	Testuje zobrazení *sheetu* pro odeslání reportu.	Test uživatelského rozhraní
testFilteringByTags()	Testuje filtrování seznamu tutoriálů pomocí tagů.	Test uživatelského rozhraní
testFilteringBySearch()	Testuje filtrování seznamu tutoriálů pomocí textu.	Test uživatelského rozhraní
testFollowUpQuestions()	Testuje zobrazení vnořeného tutoriálu po jeho výběru.	Test uživatelského rozhraní

5.2 Uživatelské testy

Pro uživatelské testování jsem vybral skupinu osmi testovacích subjektů u nichž se předpokládá pokročilá znalost angličtiny, nebo jednoho z aplikací nabízených jazyků. Tento předpoklad je důležitý, aby byl uživatel schopen aplikaci porozumět. Každý z těchto subjektů byl opatřen úkoly, kterých pomocí aplikace musí dosáhnout. Interakce s aplikací byla zaznamenána pomocí nahrávání obrazovky a díky hlasovému komentáři, kde subjekt testování popisuje tok myšlenek při používání aplikace. Zde je seznam úkolů pro uživatelské testování a k nim krátké shrnutí výsledků.

Úkol č. 1

Představte si, že se přestěhujete do České republiky, bydlení v hotelech vás již neuspokojuje a chcete si najít pronájem. Nalezněte informace o pronájmech bytů.

Očekávaným výsledkem tohoto úkolu byla navigace do konkrétního tutoriálu, který informuje uživatele, jak si v České republice najít ubytování. Možností, jak tento tutoriál najít je více, uživatel má na výběr z filtrování pomocí tagů nebo pomocí vyhledávacího políčka. Po nalezení tutoriálu jsou informace o nájmech zobrazeny v doplňujících otázkách.

Většina uživatelů zvolila filtrování pomocí tagu „Accomodation“, který shrnuje témata související s ubytováním. Jeden ze subjektů zvolil vyhledávání pomocí textu, kde vybral intuitivně slovíčko „Lease“, které ho dovedlo ke správnému tutoriálu. Odtud již neměl žádný ze subjektů testování problém najít informace o pronájmu bytu.

Úkol č. 2

Angličtina není vaším primárním jazykem a zjistili jste, že instrukcím v aplikaci naprosto nerozumíte. Rozhodnete se zjistit, zda aplikace podporuje více jazyků a jeden z nich si vyberete.

Očekávaným výstupem tohoto úkolu je schopnost zorientovat se v aplikaci a změnit její jazyk. Tester se musí dostat do části aplikace, která obsahuje informace o ní a také nabízí možnost změny jazyka. Po výběru jazyka musí být uživatel schopen aplikaci restartovat, aby viděl změnu textů v aplikaci do vybraného jazyka.

Všichni z testujících se bez problému v aplikaci zorientovali a byli schopni změnit jazyk aplikace. Ve dvou případech došlo k nedorozumění při restartování aplikace, kde testeři nerestartovali aplikaci, ale pouze přešli na domovskou obrazovku a běžící aplikaci opět otevřeli. Tento postup do aplikace nepromítl žádné změny v jazyce.

Úkol č. 3

Vraťte se na téma, které se týká hledání ubytování. Rozhodli jste se, že chcete postupovat podle bodů, které jsou v tutoriálu uvedeny. Tyto body si uložte, abyste se k nim mohli vrátit později.

Od testera bylo očekáváno, opětovné nalezení konkrétního problému a uložení jednotlivých kroků řešení do databáze pomocí tlačítka, které se nachází za posledním krokem.

Zde jsem si všiml, že všichni z testerů preferovali filtrování pomocí tagů, které je podle nich přívětivější a rychlejší než vyhledávání pomocí textu. Při ukládání kroků řešení všichni až na jednoho testera použili tlačítko pro uložení. Jeden z testerů použil namísto tlačítka snímek obrazovky. Podle testovacího subjektu je pro něj jednodušší najít kroky řešení v galerii, kam je zvyklý odkládat si podobné věci.

Úkol č. 4

Potřebujete zjistit informace o zdravotním pojištění a o jeho případném zřízení. K vyhledání informací nepoužívejte seznam tutoriálů, ale zkuste informace zjistit pomocí chatbota.

Výstupem tohoto úkolu je navigace do chatbota a následné vyřešení úkolu pomocí chatbota.

Žádný ze subjektů neměl problém s nalezením informací, jak získat zdravotní pojištění. Všem uživatelům se líbila možnost komunikace pomocí tlačítek, nimiž úkol vyřešili. Připomínkou k tomuto úkolu byla nutnost posouvání chatu k zobrazení nejnovějších zpráv.

Úkol č. 5

V příštím týdnu máte schůzku s jistou organizací, kterou nemůžete zapomenout. Rozhodnete si vytvořit v aplikaci úkol. K úkolu zkuste přiřadit kontakt ze seznamu nebo si vlastní kontakt vytvořit.

Od uživatele se očekává vytvoření nového úkolu, na který bude navázán kontakt.

Žádný ze subjektů neměl s tímto úkolem problém a všem se podařilo úkol dokončit. Jedinou připomínkou uživatelů byla nemožnost importovat kontakt přímo ze seznamu kontaktů, který mají uložený přímo v kontaktu.

Úkol č. 6

Přestěhovali jste se z Ukrajiny do Letohradu. Hledáte nejbližší místo, kde by vám mohli pomoci. Nalezněte kontakt, který by vám mohl pomoci vyřešit tento problém. Zařízení simuluje aktuální lokaci uživatele právě na toto město.

Výstupem úkolu bylo nalezení nejbližšího místa, které se stará o přistěhovalce z Ukrajiny v okolí Letohradu.

Většina z uživatelů se rozhodla pro hledání kontaktu na mapě. Těmto uživatelům chybělo tlačítko, které by zobrazilo jejich aktuální pozici, podle níž by nejbližší místo nalezli. Tři uživatelé se rozhodli pro filtrování kontaktů pomocí vy-

hledávacího políčka, kam napsali jméno města. Výsledkem hledání bylo prázdné pole kontaktů a uživatelé kontakt nenašli. Rozhodli se tedy využít mapu, kde kontakt nalezli.

Všem kontaktům chybělo tlačítko na mapě, které by zobrazilo jejich polohu. Těm, kteří vyhledávali kontakty pomocí textu chybělo vyhledávání pomocí adresy.

Shrnutí výsledků

Při uživatelském testování jsem si všiml, že lidé mají tendenci používat filtry, než aby výsledky filtrovali pomocí psaného textu. Při druhém úkolu jsem došel k závěru, že všichni testeři souhlasí s umístěním nabídky pro změnu jazyka a je přesně tam, kde čekali. Jako nevýhodu uvedli nutnost restartovat aplikaci, aby viděli změnu v jazyku aplikace. U třetího úkolu mě překvapilo uložení snímku obrazovky do galerie, namísto uložení přímo v aplikaci. Po vysvětlení jsme došli k závěru, že by bylo přívětivé, kdyby aplikace měla možnost exportu úkolu do aplikace Připomínky, která je nativní aplikací zařízení *Apple* pro správu úkolů. U úkolníčku to nebyla jediná připomínka, dalším problémem, který uživatelé zmínili je importování kontaktů ze zařízení.

Testeři těmito úkoly s testováním neskončili a poskytli mi další zpětnou vazbu. Jako velice pozitivní hodnotí zvolené barevné schéma aplikace, které je přizpůsobeno schématu Jihomoravského kraje. V aplikaci však našli další nedostatky, všimli si např. že stahování dat aplikace trvá při prvním spuštění déle než jim je příjemné čekat. Dále objevili poměrně vysoké množství chyb v anglickém jazyce při pročítání textů.

6 Diskuse

Kdybych začínal s aplikací od začátku, rozhodně bych si nejdříve připravil grafický návrh aplikace např. v nástroji *Figma*. Pomocí tohoto nástroje bych přesně věděl, jaká komponenta má mít jaký design a nemusel to optimalizovat ke konci vývoje.

Aplikace by do budoucna určitě mohla mít mapu, která je dostupná i bez připojení k internetu, např. skrze balíček *MapBox*. Od tohoto řešení bylo upuštěno, protože tento balíček by narozdíl od *MapKit* představoval vyšší režii na údržbu kódu nebo na uložiště mobilního zařízení. Dalším důvodem je, že *MapKit* jakožto nativní framework je optimalizován pro zobrazování map na platformě *iOS*. Z důvodů ceny mobilních telefonů společnosti *Apple* také předpokládám, že většina uživatelů má dostupný balíček mobilních dat, jejichž cena je stále dostupnější a uživatelé nemají problém s vyšší spotřebou dat.

Dalším vylepšením aplikace by dle mého uvážení mohlo být přiřazení tagů ke kontaktům. Díky tomu by mohli uživatelé mnohem rychleji nalézt kontakt, který přímo vyhovuje jejich požadavkům.

Vzhledem k vážnosti a charakteru této aplikace nelze navrhnout např. získávání odznaků za splněné úkoly nebo hravé uživatelské prostředí. Z průzkumu trhu s podobnými aplikacemi se mi však velice zalíbila část aplikace, která se věnuje výuce jazyku. Aplikace *Smart Migration* by mohla např. disponovat malým slovníkem s pár základními frázemi, se kterými se člověk setkává na každodenní bázi. Dále mě zaujala aplikace *Lawfully*, konkrétně dvě věci. Velice se mi líbí možnost sdílet své problémy s komunitou. Díky tomuto se může uživatel dočkat často rychlejší odpovědi přímo od jiného uživatele, který má s daným problémem již osobní zkušenost. Sledování procesu vydávání víz a jiných právních procesů by mohlo uživatelům ušetřit čas s komunikací s úřady. Z tohoto důvodu by nebylo špatné se nad přidáním této funkce také zamyslet.

Na výsledky uživatelského testování jsem zareagoval změnami v aplikaci. V aplikaci jsem umožnil uživatelům importování kontaktů, které mají v zařízení nebo exportování úkolů z úkolníčku do aplikace Připomínky. Dalším vylepšením aplikace, které jsem provedl je zobrazení polohy zařízení na mapě s kontakty, díky čemuž jsou uživatelé schopni rychleji vyhledat blízká místa.

7 Závěr

Tato práce obsahuje literární rešerši, která se zaměřuje především na dvě věci: průzkum existujících aplikací podobného typu jako je *Smart Migration* a technologie, které jsou ve vývoji aplikací pro *iOS* esenciální.

Hlavním cílem této práce bylo navrhnout a následně implementovat aplikaci *Smart Migration* pro platformu *iOS*, podle funkcionalit stejnojmenné aplikace pro *Android*. Aplikaci pro *Android* jsem podrobně prozkoumal a mohu říct, že se mi povedlo udělat aplikaci, která tyto funkcionality splňuje.

Aplikace *Smart Migration* při prvním spuštění zobrazí uživateli *onboardingové* obrazovky, kde uživateli představí své základní funkcionality. Tyto obrazovky se za celý chod aplikace zobrazí pouze jednou. Aplikace má lokalizované texty a obrázky ve třech světových jazycích, a to: angličtina, ruština, ukrajinština. Veškerá data aplikace, kromě statických textů, jsou stahována ze serveru Jihomoravského kraje, lze však jednoduchým přepsáním konstanty docílit, aby aplikace stahovala data z Ministerstva práce a sociálních věcí, pro které je aplikace v současnosti také uvažována. Tato data jsou po stažení uložena v databázi *Core Data*, díky čemuž může aplikace bezproblémově fungovat i bez připojení k internetu. Aplikace kontroluje verzi těchto dat, pokud se data změnila, dojde k aktualizaci a stará data jsou nahrazena za nová. Uživatel si může do aplikace také ukládat úkoly, které mohou být importovány z data dostupných na serveru, nebo si vytvořit vlastní. Aplikace dále umožňuje komunikaci s chatbotem, se kterým může uživatel komunikovat pomocí předdefinovaných odpovědí nebo pomocí vlastních zpráv.

Graficky jsem se snažil *Android* aplikaci neduplikovat, ale navrhl jsem designové změny, které aplikaci přizpůsobí konvencím pro *iOS*. Aplikace je také navrhnuta, aby více kopírovala barevné schéma Jihomoravského kraje, které tvoří tmavě modrá a růžová barva.

Po implementaci aplikace jsem navrhl jednotkové testy a testy uživatelského rozhraní. Výsledky testů jsem zhodnotil a do aplikace nedostatky zapracoval. Následovalo také uživatelské testování, kde jsem vybral osm testovacích subjektů. Výsledky uživatelského testování jsem zhodnotil a pro aplikaci navrhl změny. Z rešerše aplikací podobného typu jsem navrhl změny pro tento projekt a zahrnul je v diskusní části této práce.

Literatura

Alam, Sayed Mahmudul. VIPER Design Pattern in Swift for iOS Application Development. [on-line]. In *Medium.* 2017 [cit. 2022-01-29]. Dostupné na: https://medium.com/@smalam119/viper-design-pattern-for-ios-application-development-7a9703902af6.

Alamofire. Alamofire [on-line]. In *GitHub.* 2023 [cit. 2023-01-28]. Dostupné na: https://github.com/Alamofire/Alamofire.

Allan, Missy. Intro to Realm: Getting Started with the Realm Mobile Platform [on-line]. In *Medium.* 2017 [cit. 2023-01-29]. Dostupné na: https://medium.com/@missyalienn/intro-to-realm-getting-started-with-the-realm-mobile-platform-e60ddbc24413.

Alter-Pesotskiy, Alexey. Stress testing on iOS with xcmonkey [on-line]. In *Testableapple.* 2023 [cit. 2023-02-10]. Dostupné na: https://testableapple.com/stress-testing-on-ios-with-xcmonkey/.

Alvarez, Pedro. iOS Architectures Explained: Which One Best Fits My Project? [on-line]. In *Medium.* 2021 [cit. 2022-01-28]. Dostupné na: https://betterprogramming.pub/ios-architectures-explained-which-one-best-fits-my-project-94b4ffaad16.

AppCoda Editorial Team. How to Integrate Google Street View in iOS Apps [on-line]. In *AppCoda.* 2017 [cit. 2023-02-03]. Dostupné na: https://www.appcoda.com/google-street-view-ios/.

Apple. CloudKit Quick Start [on-line]. In *Apple Developer Documentation.* 2017 [cit. 2023-01-28]. Dostupné na: https://developer.apple.com/library/archive/documentation/DataManagement/Conceptual/CloudKitQuickStart/Introduction/Introduction.html.

Apple. Model-View-Controller [on-line]. In *Apple Developer Documentation.* 2018 [cit. 2022-01-28]. Dostupné na: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

Apple. Xcode 14 Overview [on-line]. In *Apple Developer Documentation.* 2022 [cit. 2022-01-12]. Dostupné na: https://developer.apple.com/xcode/.

Apple. Build Apps Using CloudKit [on-line]. In *Apple Developer Documentation.* 2023a [cit. 2023-01-28]. Dostupné na: https://developer.apple.com/icloud/cloudkit/.

Apple. Core Data [on-line]. In *Apple Developer Documentation.* 2023b [cit. 2023-01-28]. Dostupné na: https://developer.apple.com/documentation/coredata.

Apple. Explore a location with a highly detailed map and Look Around [on-line]. In *Apple Developer Documentation.* 2023c [cit. 2023-02-04]. Dostupné na: https://developer.apple.com/documentation/mapkit/explore_a_location_with_a_highly_detailed_map_and_look_around.

Apple. MapKit [on-line]. In *Apple Developer Documentation.* 2023d [cit. 2023-02-03]. Dostupné na: https://developer.apple.com/documentation/mapkit/.

Apple. SwiftUI [on-line]. In *Apple Developer Documentation.* 2023e [cit. 2023-01-20]. Dostupné na: https://developer.apple.com/documentation/swiftui.

Apple. UIKit [on-line]. In *Apple Developer Documentation.* 2023f [cit. 2023-01-20]. Dostupné na: https://developer.apple.com/documentation/uikit.

Apple. URL Loading System [on-line]. In *Apple Developer Documentation.* 2023g [cit. 2023-01-27]. Dostupné na: https://developer.apple.com/documentation/foundation/url_loading_system.

ArcGIS. ArcGIS Runtime API for iOS [on-line]. In *ArcGIS Developers.* 2023 [cit. 2023-02-03]. Dostupné na: https://developers.arcgis.com/ios/key-features/.

Bundesamt fuer Migration und Fluechtlinge. *Ankommen* [on-line]. Ver. 1.7.5. 2022. [cit. 2023-01-07]. Dostupné na: https://apps.apple.com/cz/app/ankommen/id1066804488.

Ching, Chris. Xcode UI Testing in Swift – Code Examples [on-line]. In *Code with Chris.* 2021 [cit. 2023-02-10]. Dostupné na: https://codewithchris.com/xcode-ui-testing-swift/.

Clark, Jessica. Firebase vs. Firestore | What are the differences? [on-line]. In *back4app.* 2021 [cit. 2023-02-01]. Dostupné na: https://blog.back4app.com/firebase-vs-firestore/.

Forbes contributor. Microsoft Confirms Its $10 Billion Investment Into ChatGPT, Changing How Microsoft Competes With Google, Apple And Other Tech Giants [on-line]. In *Forbes.* 2023 [cit. 2023-02-20]. Dostupné na: https://www.forbes.com/sites/qai/2023/01/27/microsoft-confirms-its-10-billion-investment-into-chatgpt-changing-how-microsoft-competes-with-google-apple-and-other-tech-giants/.

Google. Maps SDK for iOS Overview [on-line]. In *Google Maps Platform.* 2023 [cit. 2023-02-04]. Dostupné na: https://developers.google.com/maps/documentation/ios-sdk/overview.

Heinrich & Reuter Solutions. *Welcome app Germany* [on-line]. Ver. 2.1.0. 2020. [cit. 2023-01-07]. Dostupné na: https://apps.apple.com/cz/app/welcome-app-germany/id1047174574.

Hudson, Paul. How to use @MainActor to run code on the main queue [on-line]. In *Hacking with Swift.* 2021 [cit. 2023-03-16]. Dostupné na: https://www.hackingwithswift.com/quick-start/concurrency/how-to-use-mainactor-to-run-code-on-the-main-queue.

Hudson, Paul. Introducing MVVM into your SwiftUI project [on-line]. In *Hacking with Swift.* 2022a [cit. 2023-03-16]. Dostupné na: https://www.hackingwithswift.com/books/ios-swiftui/introducing-mvvm-into-your-swiftui-project.

Hudson, Paul. Storing user settings with UserDefaults [on-line]. In *Hacking with Swift.* 2022b [cit. 2023-01-28]. Dostupné na: https://www.hackingwithswift.com/books/ios-swiftui/storing-user-settings-with-userdefaults.

Integrační centrum Praha. *Praguer* [on-line]. Ver. 2.0. 2021. [cit. 2023-01-08]. Dostupné na: https://apps.apple.com/cz/app/praguer/id1312865177.

florian [pseud.]. iOS Data Persistence in Swift [on-line]. In *iOS App Templates.* 2021 [cit. 2023-01-28]. Dostupné na: https://iosapptemplates.com/blog/ios-development/data-persistence-ios-swift.

Jeroen, L. UIKit vs. SwiftUI: How to Choose the Right Framework for Your App [on-line]. In *Stream.* 2022 [cit. 2023-01-20]. Dostupné na: https://getstream.io/blog/uikit-vs-swiftui/.

Kadijk, Mathijs. R.swift [on-line]. In *GitHub.* 2023 [cit. 2023-02-25]. Dostupné na: https://github.com/mac-cain13/R.swift.

Karajgi, Aniruddha. Building a Chatbot with Rasa [on-line]. In *Medium.* 2021 [cit. 2023-02-20]. Dostupné na: https://towardsdatascience.com/building-a-chatbot-with-rasa-3f03ecc5b324.

Katz, Michael. Getting Started with the VIPER Architecture Pattern [on-line]. In *Kodeco.* 2020 [cit. 2022-01-29]. Dostupné na: https://www.kodeco.com/8440907-getting-started-with-the-viper-architecture-pattern.

Lawfully. *Lawfully* [on-line]. Ver. 4.2.70. 2023. [cit. 2023-01-09]. Dostupné na: https://apps.apple.com/cz/app/lawfully-uscis-case-tracker/id1435063223.

MapBox. Maps SDK for iOS [on-line]. In *MapBox Docs.* 2023 [cit. 2023-02-04]. Dostupné na: https://docs.mapbox.com/ios/maps/guides/.

MessageKit Contributors. MessageKit [on-line]. In *GitHub.* 2023 [cit. 2023-02-20]. Dostupné na: https://github.com/MessageKit/MessageKit.

Mishali, Shai. SnapKit for iOS: Constraints in a Snap [on-line]. In *Kodeco.* 2019 [cit. 2023-01-20]. Dostupné na: https://www.kodeco.com/3225401-snapkit-for-ios-constraints-in-a-snap.

Pasquier, Benoit. How to implement MVVM pattern in Swift from scratch [on-line]. In *Benoit Pasquier.* 2018 [cit. 2022-01-28]. Dostupné na: https://benoitpasquier.com/ios-swift-mvvm-pattern/.

Piper, David. iOS Unit Testing and UI Testing Tutorial [on-line]. In *Kodeco.* 2021 [cit. 2023-02-10]. Dostupné na: https://www.kodeco.com/21020457-ios-unit-testing-and-ui-testing-tutorial.

Radu, Dan. Battle of the iOS Architecture Patterns: A Look at Model-View-ViewModel (MVVM) [on-line]. In *Medium.* 2021 [cit. 2022-01-28]. Dostupné na: https://betterprogramming.pub/battle-of-the-ios-architecture-patterns-a-look-at-model-view-viewmodel-mvvm-bdfd07d9395e.

Radu, Dan. Battle of the iOS Architecture Patterns: Model View Controller (MVC) [on-line]. In *Medium.* 2021 [cit. 2022-01-28]. Dostupné na: https://betterprogramming.pub/battle-of-the-ios-architecture-patterns-model-view-controller-mvc-442241b447f6.

realm. SwiftLint [on-line]. In *GitHub.* 2023 [cit. 2023-02-25]. Dostupné na: https://github.com/realm/SwiftLint.

Rondestvedt, Josh. An Intro to Realm for iOS [on-line]. In *Medium.* 2020 [cit. 2023-01-29]. Dostupné na: https://betterprogramming.pub/an-intro-to-realm-for-ios-2633162952f1.

Rudrank, Riyam. UI Testing in Swift [on-line]. In *Semaphore.* 2021 [cit. 2023-02-10]. Dostupné na: https://semaphoreci.com/blog/ui-testing-swift.

SDWebImage. SDWebImageSwiftUI [on-line]. In *GitHub.* 2022 [cit. 2023-02-25]. Dostupné na: https://github.com/SDWebImage/SDWebImageSwiftUI.

Singh, Shubham. Simplify Data Store & Operations in Your iOS Projects with Realm [on-line]. In *Medium.* 2021 [cit. 2023-01-29]. Dostupné na: https://medium.com/engineering-dr/simplify-data-store-operations-in-your-ios-projects-with-realm-dfec443f11d.

STRV. Endpoint.swift [on-line]. In *GitHub.* 2022 [cit. 2023-01-28]. Dostupné na: https://github.com/strvcom/ios-course-mendelu-2022-autumn/blob/develop/SpriteKit2/Final/RickAndMorty/API/Endpoint.swift.

trellyz. *RefAid - Refugee Aid App* [on-line]. Ver. 4.3.0. 2022. [cit. 2023-01-08]. Dostupné na: https://apps.apple.com/cz/app/refaid-refugee-aid-app/id1080936380.

USAHello. *FindHello: Immigrant Help* [on-line]. Ver. 1.1.8. 2021. [cit. 2023-01-09]. Dostupné na: https://usahello.org/findhello/.

Van der Lee, Antoine. Getting started with Unit Tests in Swift [on-line]. In *SwiftLee.* 2022 [cit. 2023-02-10]. Dostupné na: https://www.avanderlee.com/swift/unit-tests-best-practices/.

Wikipedia contributors. Chatbot [on-line]. In *Wikipedia.* 2023a [cit. 2023-02-20]. Dostupné na: https://en.wikipedia.org/wiki/Chatbot.

Wikipedia contributors. Swift (programming language) [on-line]. In *Wikipedia.* 2023b [cit. 2022-01-12]. Dostupné na: https://en.wikipedia.org/wiki/Swift_(programming_language).

Wikipedia contributors. Xcode [on-line]. In *Wikipedia.* 2023c [cit. 2022-01-12]. Dostupné na: https://en.wikipedia.org/wiki/Xcode.

Seznam tabulek

Seznam obrázků

Seznam zdrojových kódů