FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	iOS aplikace bezpečného cestování
Student:	Bc. Matyáš Procházka
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Při přesunech do zaměstnání či obecném pohybu mimo domov vznikají rizikové situace, které lze správným přístupem odvrátit, či alespoň přispět k jejich minimalizaci. Pokud již na ně dojde je třeba korektně reagovat. S touto problematikou mohou napomoci i mobilní zařízení uživatele. Cílem této práce je navrhnout a realizovat iOS aplikaci, která by zvýšila bezpečnost osoby při cestování (přesun z práce, apod).

Postupujte v těchto krocích:
Pomocí metod softwarového inženýrství analyzujte potřeby budoucích uživatelů aplikace. Také řádně analyzujte možnosti, kterými lze zvýšit bezpečnost osoby při cestování za použití smartphonu.
Proveďte důkladnou rešerši konkurence.
Na základě analýzy a rešerše navrhněte vhodnou funkcionalitu aplikace.
Proveďte návrh aplikace.
Dle návrhu implementujte funkční aplikaci.
Aplikaci řádně otestujte, optimálně i v terénu s reálnými uživateli.
Pokuste se zajistit řádné nasazení pro běžné uživatele.
Zhodnoťte výsledky a navrhněte možná vylepšení.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 8. února 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

iOS aplikace bezpečného cestování

Bc. Matyáš Procházka

Katedra webové a softwarové inženýrství, zaměření Webové inženýrství

Vedoucí práce: Ing. Jiří Hunka

3. května 2022

Poděkování

Rád bych poděkoval Ing. Jiřímu Hunkovi za vedení mé diplomové práce, za jeho ochotu a vstřícnost. Své rodině, především mamce, za to, že vždy stála při mně a podporovala mě. Své slečně za krásné momenty, jež jsem při psaní tolik potřeboval. A finálně všem kamarádům, bez kterých by to studium nebylo tak zábavné.

Prohlášení

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

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

Praha dne 3. května 2022 . . . . . .

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Procházka, Matyáš. *iOS aplikace bezpečného cestování*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato diplomová práce se zabývá vývojem mobilní aplikace pro bezpečné cestování. I v dnešní době stále dochází k mnoha rizikovým situacím a velká část lidí se necítí bezpečně při pohybu mimo domov. Práce se zaměřuje na analýzu, návrh a následnou realizaci aplikace pro iOS. Implemetovány byly tři hlavní funkcionality – SOS tlačítko, žádost o checkin a sdílení cesty. Aplikace poté byla podrobena uživatelskému testování, na jehož základě byly navrženy úpravy stávajících funkcionalit a nástin plánu budoucího vývoje.

Klíčová slova mobilní aplikace, analýza, návrh, implementace, uživatelské testování, iOS

Abstract

This diploma thesis covers the development of a mobile application for safe travel. Even today, there are still many risky situations taking place and many people do not feel safe when travelling away from home. This thesis focuses on the analysis, design and subsequent implementation of an application for

iOS. Three main functionalities were implemented – SOS button, checkin request and trip sharing. The application was then subjected to user testing, based on which modifications to existing functions and an outline of a future development plan were proposed.

Keywords mobile app, analysis, design, implementation, user testing, iOS

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam tabulek

Úvod

Pocit bezpečí při cestování je neustále probírané téma. Asi většina z nás zaznamenala nějaký incident ve svém okolí nebo se ocitla v situaci, ve které se necítila bezpečně. Ať je to nepříjemně vypadající skupinka, pokřikování na ulici, cestování v neznámém městě nebo přímo pronásledování. Těchto situací v každém případě nastává velmi mnoho.

Britská firma YouGov zabývající se výzkumem trhu a analýzou dat provedla v roce 2021 průzkum, jak často se Britové necítí bezpečně napříč různými situacemi. 63 % z dotázaných žen uvedlo, že se během chůze v noci o samotě cítí ohroženě, a 41 % odpovědělo, že pravidelně podnikají kroky k tomu, aby se ochránily před sexuálním napadením.[22]

V této práci si tedy kladu za cíl prozkoumat možnosti použití mobilních zařízení na zvýšení pocitu bezpečí při cestování a následně vyvinout vlastní aplikaci, jež by sama k tomuto dojmu přispívala a zároveň se podílela na řešení krizových situací, případně sloužila jako podklad informací pro vyšetřování. Vyvinutá aplikace by měla mít úlohu proof of concept (zkušební provedení) pro budoucí tvorbu v této oblasti.

Samotná práce začíná průzkumem aktuální situace, chování a potřeb potenciálních uživatelů a definicí person, poté následuje analýza již existujících řešení. Na základě nabytých vědomostí z těchto částí jsou definovány požadavky zamýšlené aplikace. Dle těchto požadavků je vytvořen návrh serverové části a samotné mobilní aplikace a zároveň provedena rešerše technických možností nabízejících se na platformě od Apple[24]. Následně je vyvinuta mobilní aplikace, jež je podrobena uživatelskému testování, podle kterého jsou zformulovány návrhy na zlepšení.

Kapitola 1

Analýza

V první kapitole této diplomové práce se zaměřím na analytickou část. Nejdříve se budu zabývat chováním a potřebami cílové skupiny. Dalším krokem bude seznámení se s existujícími řešeními a prozkoumání jejich silných a slabých stránek. Poté definuji funkční a nefunkční požadavky a finálně popíši hlavní případy užití.

1.1 Cílová skupina

V rámci této sekce se kouknu na možnosti provádění výzkumu ohledně chování a potřeb lidí a potenciálních uživatelů, poté zvolím vhodnou variantu a sám jeden z výzkumů provedu.

Metody výzkumu lze rozdělit na dvě základní – kvantitativní a kvalitativní.

„***Kvantitativní výzkum*** *se také označuje jako tradiční, pozitivistický, experimentální nebo empiricko-analytický. Zaměřuje se na hledání vztahů mezi dvěma či více proměnnými. Jeho hlavním cílem je ověřování platnosti teorií pomocí testování z těchto teorií vyvozených hypotéz.*

Kvantitativní výzkumník chápe sociální realitu jako existující nezávisle na jeho osobnosti. Udržuje si od ní odstup nezúčastněného pozorovatele a snaží se o její popis. Lidské chování považuje za determinované a tedy měřitelné a předpověditelné.“[25]

„***Kvalitativní výzkum*** *bývá také nazýván konstruktivistickým, naturalistickým, interpretativním nebo reflexivním. Disman jej charakterizuje jako nenumerické šetření a interpretaci sociální reality. Kvalitativní přístup klade důraz na důkladné (hloubkové) poznání zkoumaného sociálního jevu (události, fenoménu). Snaží se o vytvoření komplexního, holistického obrazu zkoumaného*

problému, o porozumění lidem v různých sociálních situacích a jejich interpretacím těchto situací.

Kvalitativní výzkumník se domnívá, že sociální realita by měla být spíše interpretována než popisována, a to na základě toho, jak ji prožívají konkrétní jednotlivci. Lidské chování je nahlíženo jako výsledek svobodných rozhodnutí.“[25]

Rozdíly ve sběru dat obou metod jsou přehledně vidět v tabulce 1.1:

Kvantitativní sběr dat	**Kvalitativní sběr dat**
Vzorek respondentů je velký	Stačí málo respondentů
Provádí se pomocí dotazníkových šetření	Provádí se pomocí osobních rozhovorů
Snaží se kvantifikovat výskyt problému	Zkoumá problémy do hloubky
Časově nenáročný	Časově náročný
Dedukce z výsledků	Indukce z výsledků
Statistické zpracování dat	Nestatistické zpracování dat

Tabulka 1.1: Kvantitativní vs kvalitativní sběr dat[21]

V této diplomové práci jsem se rozhodl osobně provést pouze kvalitativní výzkum, neboť vede k hlubšímu porozumění uživatelových potřeb a zároveň kvantitativní výzkumy na toto téma jsou dobře dohledatelné a byla jich provedena již celá řada. Následující podsekce 1.1.1 a 1.1.2 obsahují shrnutí získaných informací.

1.1.1 Kvantitativní výzkum

V kvantitativním výzkumu by měla být zodpovězena základní otázka, zdali se opravdu lidé v určitých situacích cítí nekomfortně. K jejich nalezení jsem použil vyhledávač Google[26] a klíčovou frázi „*do women feel safe walking home alone at night*“. Z dostupných článků a prací na první stránce jsem nakonec zvolil výzkum z roku 2021 od britské společnosti YouGov[22], jenž byl dokonce citován i na dalších odkazech. YouGov je mezinárodní skupinou provádějící sběr a následnou analýzu dat a nejcitovanějším zdrojem v oblasti výzkumu trhu ve Spojeném Království.[27]

Základní informace o provedeném výzkumu shrnuji v tabulce 1.2.

V rámci tohoto výzkumu bylo pokládáno několik otázek a tázaní odpovídali, jak často se v daných situacích necítí bezpečně. Tabulka 1.3 obsahuje procentuální zastoupení lidí, kteří odpověděli „vždy“ nebo „často“. Důležitou informací je fakt, že 19 % dotázaných žen vůbec v noci o samotě nechodí.

Velikost vzorku	1667
Ženy	857
Muži	810
Region výzkumu	Velká Británie

Tabulka 1.2: Základní informace o výzkumu ohledně pocitu nebezpečí společnosti YouGov[22]

Situace	Ženy	Muži
Chození v noci o samotě	63	15
Chození alejí, uličkou, průchodem (anglické „alley“) o samotě	63	17
První schůzka, rande (anglické „date“)	28	4
Návštěva domova cizího člověka	31	9

Tabulka 1.3: Procento lidí, kteří se v daných situacích necítí bezpečně („vždy“ nebo „často“)[22]

1.1.2 Kvalitativní výzkum

Kvalitativní výzkum jsem v rámci této práce provedl osobně. Vytipoval jsem si tři osoby ve svém blízkém okolí, které mají zkušenosti s chozením domů v noci o samotě a provedl s nimi 30-60 minutový rozhovor. Konverzace neměla striktní strukturu, nýbrž přirozeně plynoucí atmosféru. Cílem rozhovorů bylo zjistit následující body:

- obecný pohled na toto téma,
- způsob, jakým se v určitých situacích samy chovají,
- jaké prostředky využívají pro informování ostatních (zdali vůbec nějaké),
- co by jim usnadnilo jejich situace či nějakým způsobem pomohlo.

Finálně jsem také vyhledal zkušenosti lidí na internetu.

1.1.2.1 Rozhovory

Rozhovory proběhly se třemi respondentkami ve věku od 19 do 26 let.

Všechny tři respondentky se shodly, že při cestě o samotě pociťují strach, a tudíž podnikají určité kroky – převážně kontaktují své blízké, že na cestu vyrážejí a během ní o sobě dávají občas vědět. Problém vidí v situacích, kdy nikdo z nejbližších není dostupný – například již všichni spí. V tuto chvíli je informování o své poloze a situaci minimální. Dvě z dotázaných navíc s nikým během cesty nesdílí svojí polohu pomocí dalších aplikací. Pokud by nastala kritická situace, není zde žádná přibližující informace na jaké části trasy se

daná věc udála. Všem třem se tedy líbil nápad ohledně možnosti u každé zprávy mít informaci o poloze, ze které byla odeslána.

Jedna z respondentek sama zmínila určitou nevhodnost kontaktování přes klasické aplikace pro psaní zpráv. Je-li v roli toho, který je doma a strachuje se o druhého, zdráhá se mu v této situaci psát – druhá strana může být naprosto v pořádku a bavit se a považovat zprávu za otravnou. Zároveň vyslovila myšlenku, že někteří lidé automaticky notifikace z těchto aplikací ignorují nebo jim nepřidávají takovou váhu, neboť jim jich chodí velké množství. Myslí si tedy, že pokud by přišlo upozornění z aplikace dedikované na bezpečnost, kterou by druhý pouze rychle odkliknul, a dal tak o svém bezpečí vědět, měli by lidé menší tendenci je přecházet. I pro ni samotnou by to znamenalo, že se necítí „*vlezle*“. Zajímavou zmínkou respondentky taktéž bylo, že by notifikace z aplikace vyhraněné k tomuto tématu ji samotnou podnítila k tomu, aby se kolem sebe rozhlédla a zanalyzovala situaci – například při jízdě v tramvaji.

Další z respondentek vyprávěla příběh o rodinné kamarádce, jež byla napadena na zastávce autobusu, kde následně v bezvědomí ležela, dokud nepřivolal pomoc kolemjdoucí starší pár. Nešel-li by tento pár okolo, bůh ví jak dlouho by na místě byla, pokud by již nebylo pozdě. V tento moment by bylo důležité, aby někdo věděl informace o její lokaci a že se do nějaké doby neozvala. Jelikož se jednalo o již dospělou ženu, nikdo ji přímo nekontroloval a nečekal, zdali dorazí v pořádku domů. Například pouhé odkliknutí každých 10-30 minut by mohlo následně vyvolat u někoho z blízkých další kroky ke zjištění stavu.

Základní činnosti, které samy všechny respondentky dělají, by se daly shrnout do těchto bodů:

- co nejméně na sebe upozorňovat,
- informovat své okolí na parametry cesty (otázky odkud, kudy, kam, kdy),
- pravidelně někoho kontaktovat,
- dávat pozor, co se v okolí děje, a co nejméně se nechat rozptylovat.

1.1.2.2 Sociální sítě a internet

Jedním z míst, kde se téma bezpečí velmi často diskutuje, jsou sociální sítě. Ačkoliv to není plně věrohodný zdroj, může dokreslit obrázek o situaci.

Zaujal mě příspěvek na sociální síti Twitter[28], který bych zde rád uvedl: „*Ženy: RT (retweet, poznámka autora) pokud jste někdy šly přes parkoviště s klíči mezi prsty nebo jste předstíraly, že telefonujete, protože jste se necítily bezpečně.*“[29] (přeloženo autorem) Tento příspěvek měl dne 6. 3. 2022 skoro 77 tisíc oblíbení a 64 tisíc retweetů (přesdílení) a mnoho uživatelů pod ním psalo obdobně znějící komentáře, že drží klíče mezi prsty, vyhýbá se určitým místům, předstírá hovory či jsou stále s někým v kontaktu.

1.1.3 Shrnutí

Během analýzy cílové skupiny jsem si potvrdil, že téma bezpečnosti cestování je velmi relevantním a že velká část lidí se i v dnešní době necítí bezpečně. Dále jsem si v kvalitativním výzkumu přiblížil základní činnosti, které během cest oslovené respondentky dělají a získal i nápady na funkcionality, které by jim jejich situace ulehčily či zpříjemnily. Samotné funkcionality jsou rozebrány až v kapitole 1.4.2 Funkční požadavky.

1.2 Persony

„Persona je fiktivní postava využitelná při návrhu aplikace, webu či jiné technologie. Persona představuje fiktivního uživatele testované aplikace, který má svou osobnost, potřeby, vlastnosti, specifické přístupy a názory.“[13] (přeloženo autorem)

V této části vytvořím tři různé persony – jedna bude reprezentovat typického uživatele, druhá občasného a třetí takzvanou antipersonu, tedy člověka, který vyvíjený systém používat zcela jistě nebude (toto rozdělení vychází z předmětu NI-NUR – Návrh uživatelského rozhraní[13]). Persony se hodí pro uvědomění si, jaký typ lidí by mělo aplikaci využívat, a při výběru účastníků do testování. Pro tvorbu person jsem využil znalostí z kapitoly 1.1 Cílová skupina a vlastní představy o zamýšlené aplikaci.

1.2.1 Persona A – Typický uživatel

Jméno: Radka Adamová
Věk: 18
Pohlaví: Žena
Záliby: Tancování, sociální sítě, kamarádi
Typický den:

Radka vstává kolem půl sedmé, jde se rychle nasnídat, provést ranní hygienu a následně připravit do školy. Ve škole je od osmi hodin ráno do zhruba tří odpoledne, záleží na konkrétním dni. Pokud má hodně úkolů a dalších věcí, jde přímo domů, jinak tráví odpoledne s kamarády venku. V pondělí a čtvrtek chodí na lekce tancování.

Přes víkend nemá nastavený budík a vstává nepravidelně. Den tráví s kamarády nebo poleháváním v posteli, večer chodí na akce či do hospody a vrací se v pozdních nočních hodinách. Její matka často zůstává vzhůru, dokud se nevrátí.

Krátká charakterizace:

Radka je společenská, komunikativní slečna, která se ráda hýbe, zároveň dokáže celý den proležet. Tancuje od svých osmi let a aktuálně studuje gymnázium v Praze.

1.2.2 Persona B – Občasný uživatel

Jméno: Vanda Matyášová
Věk: 34
Pohlaví: Žena
Záliby: Její dvě malé děti, čtení, procházky, kolo
Typický den:

Vanda je již rok po rodičovské a aktuálně chodí do práce, kde dělá vedoucí marketingového oddělení. Vstává v šest, aby se stihla sama připravit a následně připravit děti do školky. V práci je zhruba do půl sedmé a poté se vrací rovnou domů. Zřídka kdy zajde po pracovní době posedět na skleničku s kolegy z práce. O víkendu tráví čas s manželem a dětmi.

Krátká charakterizace:

Vanda pochází z Moravy, kde vystudovala základní i střední školu ve Zlíně. Je disciplinovaná, inteligentní a cílevědomá. Na vysokou školu se odebrala do Brna, kde poznala i svého partnera. Po škole se vydala rovnou pracovat do marketingového oddělení, kde se časem vypracovala až na vedoucí pozici.

1.2.3 Persona C – Antipersona

Jméno: Ludvík Tkáč
Věk: 60
Pohlaví: Muž
Záliby: Motorky, stará auta, zahrada
Typický den:

Ludvík pracuje v chemickém závodě, kam přichází již na sedmou hodinu, a vstává proto velmi brzy. Pracovní doba mu končí ve tři odpoledne a následně čas tráví buď na zahradě nebo v garáži. Ačkoliv není vystudovaný automechanik, má v motorech velkou zálibu. Občas zajde večer s kamarády na pivo.

Krátká charakterizace:

Ludvík není příliš společenský a má rád svůj klid. Vystudoval Vysokou školu chemicko-technologickou v Praze a po studiích se usadil zpět ve svém rodném městě. Nemá partnerku ani děti.

1.3 Existující řešení

V třetí sekci analytické části se zaměřím na alternativy, které aktuálně lidé mohou využívat. Nejdříve se v podsekci 1.3.1 budu krátce věnovat nepřímé konkurenci, za kterou považuji vše kromě mobilních aplikací. V druhé podsekci 1.3.2 se následně zaměřím na řešení na mobilních platformách.

1.3.1 Nepřímá konkurence

Mezi nepřímou konkurenci lze zařadit předměty jako pepřový sprej, nůž či jiná zbraň. Člověk si také může udělat kurz sebeobrany či existuje mnoho webových stránek a článků o tom, jak se při určitých situacích chovat. Tuto podsekci však nebudu detailně rozebírat, neboť mobilní aplikace nemá těmto věcem konkurovat, nýbrž je doplňovat.

1.3.2 Mobilní aplikace

Konkrétní mobilní aplikace, které jsou součástí hodnocení, jsem vyhledával na třech různých místech:

- iOS aplikace na App Store[30],
- Android aplikace na Google Play[31],
- pro obě platformy ve vyhledávači Google[26].

Platformu Android jsem do této části zahrnul, neboť i na ní lze nalézt aplikace, které mohou být pro rešerši zajímavé.

Pro vyhledávání jsem používal klíčová slova „*home*“, „*safe*“, „*safely*“, „*protect*“, „*emergency*“, ze kterých jsem tvořil určité fráze. Aplikace jsem vybíral primárně podle popisu funkcionalit. Druhotným faktorem byl počet stažení a hodnocení. U hodnocení jsem však považoval za důležité spíše kvantitu nežli samotnou hodnotu, jelikož přímo z psaných recenzí uživatelů lze mnohdy vypozorovat zajímavé skutečnosti, i když jsou negativní.

Pro samotné hodnocení aplikací jsem použil Nielsenovu heuristiku použitelnosti uživatelského rozhraní s textovým komentářem a bodovým ohodnocením na škále od 1 (nejlepší) do 5 (nejhorší). Jako další faktor jsem při vyhodnocování bral užitečnost samotné funkcionality právě s ohledem na uživatelská hodnocení, neboť některé aplikace lze testovat pouze za specifických situací, a některé dokonce nebylo možné stáhnout na českém trhu.

1.3.2.1 Nielsenova heuristika[32]

Nielsenova heuristika se skládá z 10 principů pro hodnocení uživatelského rozhraní. Nejsou to konkrétní pokyny či pravidla, nýbrž oblasti a zásady, které vedou ke zlepšení uživatelova zážitku.

1. **Viditelnost stavu systému**
   Uživatel by stále měl mít přehled o stavu a činnostech, které probíhají, pomocí zpětné vazby.

2. **Shoda systému s reálným světem**
   Design by měl být navržen a text napsán tak, aby odrážel uživatelovu představu z reálného světa – například odstraňování souborů do „koše“ s odpovídající ikonou.

3. **Uživatelova kontrola a svoboda**
   Uživatel často provede činnost omylem, a měl by tak mít možnost jednoduše svou akci odvolat či se bez zbytečných překážek dostat do počátečního stavu.

4. **Dodržování konzistence a standardů**
   Systém by měl dodržovat platné standardy platformy a stejné nebo obdobné akce a prvky by měly znamenat a dělat na vícero místech to samé.

5. **Prevence chyb**
   Zejména před kritickou akcí, jež by mohla způsobit škody, by měl být uživatel informován a systém by měl chtít dodatečné potvrzení – například mazání.

6. **Jasnost místo zapamatování**
   Uživatel by neměl být nucen zapamatovávat si zdlouhavé věci. Možnosti, které aktuálně nemůže využít, by měly být odlišeny či schovány úplně. Na konkrétních místech, kde je to potřeba, by měly být popisky.

7. **Flexibilita pro náročné**
   Pokročilejší uživatelé by měli mít možnost provádět akce rychleji – třeba pomocí klávesových zkratek.

8. **Estetika a minimalita**
   Rozhraní by nemělo obsahovat prvky a elementy, jež jsou irelevantní. Znepřehledňují a zesložitují situaci.

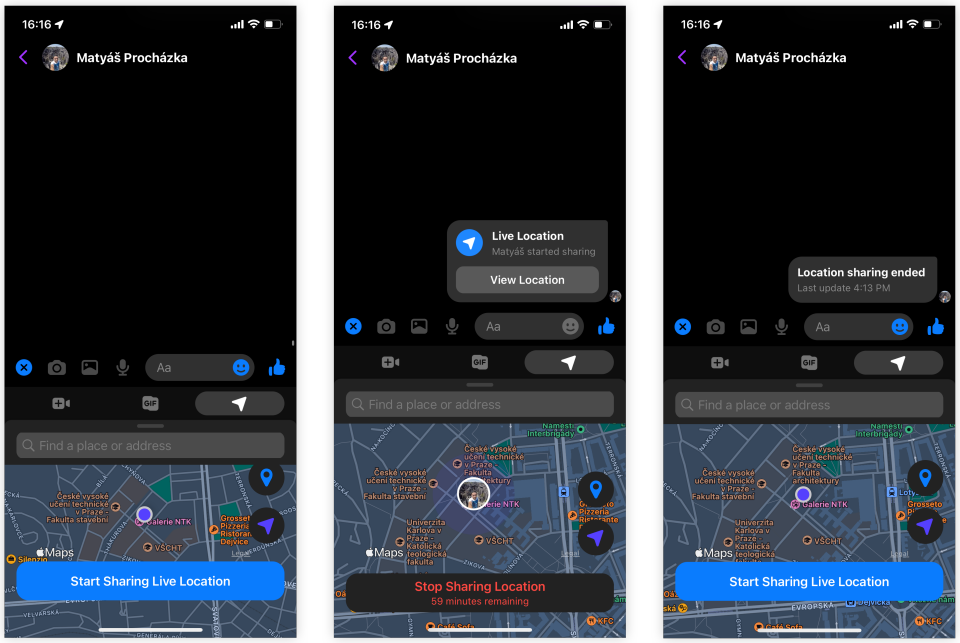
9. **Srozumitelnost problémů a jejich řešení**
   Chybové hlášky by měly být psány srozumitelným jazykem, jasně popisovat problém a nabídnout konstruktivní řešení.

10. **Dokumentace a nápověda**
    V nejlepším případě by systém neměl potřebovat dodatečné vysvětlení. Aby uživatelé s aplikací snadněji pracovali, může však někdy být potřeba vytvořit dokumentaci.

1.3.2.2 Messenger od Meta[33]

Velmi populárním nástrojem pro komunikaci je celosvětově známá a používaná aplikace Messenger. Umožňuje psaní zpráv v reálném čase a zároveň nabízí funkcionalitu na sdílení polohy. Rozhraní lze vidět na obrázku 1.1 a jeho použití je velmi jednoduché. Pro kontaktování více lidí je zde možnost vytváření skupin.

Popis nebude zahrnovat další funkcionality, které se bezpečného cestování a komunikace netýkají.



Obrázek 1.1: Messenger – Sdílení polohy

Výhody

Hlavní výhodou je psaní a sdílení polohy v rámci jedné aplikace. Dalšími klady jsou pokročilá funkcionalita ohledně viditelnosti doručení a zobrazení zpráv a možnosti jako nahrávat hlasovou zprávu, přiložit fotografii nebo video či dostupnost služby na dalších zařízeních včetně počítačů.

Nevýhody

Nevýhodou je předem definovaný interval 60 minut sdílení, který nelze nijak upravit, pouze ručně vypnout. Samotná zpráva o sdílení, přes kterou se mapa rozklikává, se časem ztratí v historii komunikace a je nutnost na ní ručně odscrollovat. I samotnou historii uživatelova pohybu nelze dohledat. Pro kon-

taktování více lidí může být nepříjemné vytváření skupin či obepisování jednotlivě.

Jak již bylo zmíněno v kvalitativním výzkumu (kapitola 1.1.2), Messenger má navíc určitou konotaci a notifikace i zprávy z něj mohou uživatelé ignorovat a nečíst. Při závažné situaci se také druhá strana nedozví, že nastal problém, pokud se sama nebude aktivně zajímat. Usne-li například, může se začít starat až další den ráno.

Evaluace heuristiky

1. **Viditelnost stavu systému** (3 body)
   V systému je lehké se zorientovat, uživatel vidí stav poslané zprávy či jestli již ukončil sdílení polohy. Avšak informaci o tom, že někomu sdílí polohu si musí sám zapamatovat, neboť je součástí zpráv a není pro ni dedikováno samostatné fixní místo na obrazovce.

2. **Shoda systému s reálným světem** (1 bod)
   U prvků je na první pohled zřejmá jejich funkcionalita.

3. **Uživatelova kontrola a svoboda** (1 bod)
   Uživatel může poměrně jednoduše smazat poslanou zprávu či přerušit sdílení polohy.

4. **Dodržování konzistence a standardů** (1 bod)
   Aplikace dodržuje standardy platformy.

5. **Prevence chyb** (3 body)
   Uživatelovi se může lehce stát, že přestane svou polohu sdílet a nemusí si toho všimnout. Opětovné zapnutí je však velmi jednoduché.

6. **Jasnost místo zapamatování** (3 body)
   Pokud se odkaz na mapu se sdílením dostane do historie, uživatel ho musí zbytečně hledat. Jinak je vše přehledné a jasné.

7. **Flexibilita pro náročné** (1 bod)
   Aplikace je velmi jednoduchá a nevyžaduje speciální funkce pro náročné.

8. **Estetika a minimalita** (3 body)
   Samotné rozhraní je velmi jednoduché, avšak pro využití při cestování obsahuje přebytečnou a nevyužitelnou funkcionalitu.

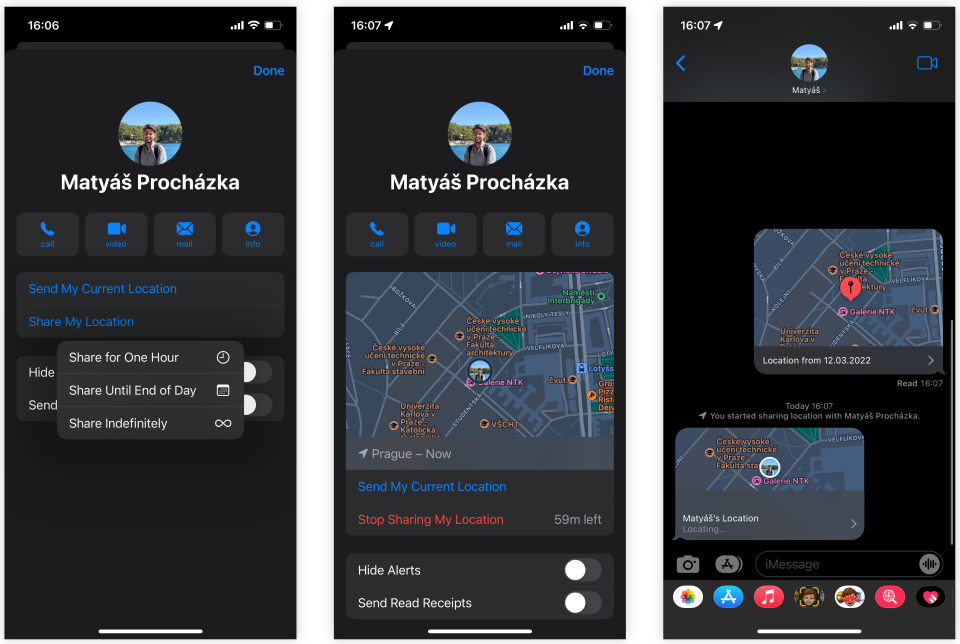
9. **Srozumitelnost problémů a jejich řešení** (4 body)
   Chybové hlášky nejsou vždy srozumitelné. Občas se stane, že poslaná příloha (například fotografie) nelze zobrazit, avšak není nabídnuto žádné řešení této situace.

10. **Dokumentace a nápověda** (1 bod)
    Aplikace nevyžaduje dodatečnou dokumentaci.

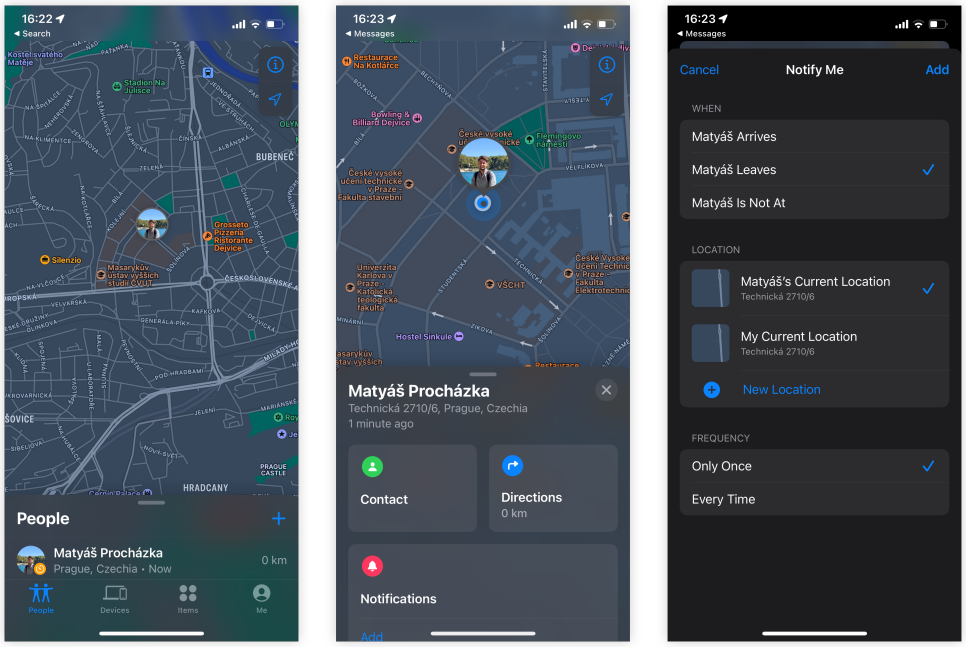
1.3.2.3 iMessage a Find My[34, 35]

Obdobně jako Messenger lze využívat i nativní komunikátor platformy od Apple iMessage. Aplikace primárně slouží k psaní SMS zpráv, avšak má-li druhá strana taktéž zařízení od Apple, může využít dalších funkcí jako je posílání zpráv přes internet, sdílení polohy či dokonce hraní her. Pro sdílení polohy stačí rozkliknout profil uživatele, zmáčknout tlačítko na sdílení, načež se ho systém zeptá, zdali tak chce provést na hodinu, do konce dne či do neurčita. Vypnutí se provádí stejným postupem. Navíc je zde možnost jednoduše poslat aktuální polohu jednorázově. Sdílení se samo propaguje i do aplikace Find My, která slouží pro sdílení polohy mezi lidmi a zjištění polohy dalších zařízení. I proto je tato sekce věnována oběma aplikacím zároveň.

Sdílení polohy lze zapnout i přímo v aplikace Find My, kde uživatel vybere člověka z adresáře kontaktů a opět zvolí, na jak dlouho chce sdílení zapnout. Find My má navíc funkcionalitu na posílání notifikací, když uživatel opustí určité místo nebo naopak do něj vejde. Ukázky aplikací je možné vidět na obrázcích 1.2 a 1.3.



Obrázek 1.2: iMessage – Sdílení polohy



Obrázek 1.3: Find My

Výhody

Výhodou je automatické propojení s aplikací Find My, kdy se druhá osoba ihned ukazuje v obou aplikacích. Stejně jako u Messengeru je pozitivní možnost psaní zpráv a sdílení polohy v rámci jedné aplikace, dále nahrávání hlasových zpráv a přidávání fotografií či videí. Intuitivní je i rozkliknutí profilu uživatele, kde je hned k dispozici aktuální poloha, a uživatel tak nemusí hledat v historii, což je velkým kladem oproti aplikaci Messenger. Člověk, který má stolní zařízení od Apple, může celou komunikaci provádět přes něj. Užitečnou funkcionalitou je zapnutí upozornění při dosažení určité lokace.

Jelikož jsou obě aplikace vyvíjeny přímo Applem, je poloha velmi přesná i v případě, když je má uživatel vypnuté. Standardní program by k údajům o poloze v takovém případě přístup neměl.

Nevýhody

Největší nevýhodou je nemožnost sledování historie polohy a nutnost vytvoření skupiny pro kontaktování více lidí, popřípadě je obepisovat jednotlivě. Velmi neohrabané je také potenciální přidávání notifikací pro dosažení více nežli jednoho místa. Finálně se opět při závažné situaci druhá strana automaticky nic nedozví.

Evaluace heuristiky

1. **Viditelnost stavu systému** (1 bod)
   Obě aplikace jsou přehledné a umožňují jednoduše vidět stav sdílení.

2. **Shoda systému s reálným světem** (1 bod)
   Použití ikon a textů je intuitivní.

3. **Uživatelova kontrola a svoboda** (1 bod)
   Uživatel má plnou kontrolu a možnost vrátit se o krok zpět je velmi jednoduchá.

4. **Dodržování konzistence a standardů** (1 bod)
   Jelikož jsou aplikace vyvíjeny přímo samotnou platformou, dodržování standardů a konzistence je takřka bezchybné.

5. **Prevence chyb** (2 body)
   V rámci iMessage je jednoduché omylem přestat sdílet polohu. Opětovné zapnutí je velmi snadné.

6. **Jasnost místo zapamatování** (1 bod)
   Oba dva systémy jsou velmi přímočaré a funkcionalita prvků je na první pohled zřejmá.

7. **Flexibilita pro náročné** (1 bod)
   Aplikace nevyžadují zkratky pro náročné uživatele.

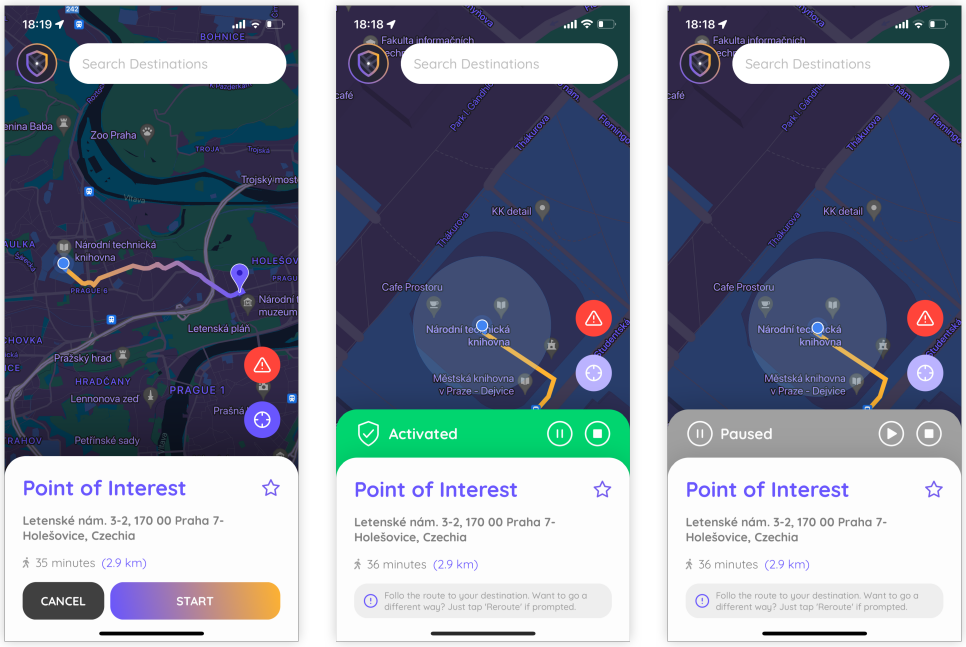
8. **Estetika a minimalita** (2 body)
   Rozhraní je estetické a snadné na používání.

9. **Srozumitelnost problémů a jejich řešení** (1 bod)
   Systémy jsou odladěné a nenabízí takřka možnost dostat se do potíží.

10. **Dokumentace a nápověda** (1 bod)
    Intuitivnost aplikací nevyžaduje dokumentaci či nápovědu.

1.3.2.4 Follo - Walking Safety App[36]

Follo má pouze jeden, avšak velmi hezky zpracovaný, účel a tím je pasivní kontrola. Uživatel zadá cílovou destinaci a aplikace následně sleduje jeho polohu, podle které detekuje kritické situace. V případě dlouhého klidu a nehýbání se systém informuje nouzové kontakty. Tato spojení mají u sebe pouze jméno a telefonní číslo, a nemusí si tedy aplikaci stahovat. Neprobíhá žádné sdílení živé polohy a aplikace ani neumožňuje průběžné kontaktování ohledně aktuálního stavu. Rozhraní lze pozorovat na obrázku 1.4.



Obrázek 1.4: Follo – Walking Safety App

Výhody

Hlavní a prakticky jedinou výhodou je pasivní kontrola, kdy uživatel nemusí sám nic provádět a může se soustředit na cestu.

Nevýhody

Follo je dělané pouze na chození, a není jej tedy možné využívat při transportu veřejnou dopravou či taxi. Jelikož se zaměřuje pouze na velmi konkrétní funkcionalitu, není zde žádná možnost o průběžném kontaktování či sdílení průběhu cesty a zobrazení její historie.

Evaluace heuristiky

1. **Viditelnost stavu systému** (3 body)
   Stav cesty je indikován textem i barevně a velmi přehledně. Avšak v případě přidávání oblíbených míst se sice zobrazí načítací kolečko, ale celá aplikace zamrzne – tato chyba ale může být lehce opravena v další verzi.

2. **Shoda systému s reálným světem** (1 bod)
   Aplikace používá ikony a další prvky s předpokládaným významem.

3. **Uživatelova kontrola a svoboda** (2 body)
   Uživatel má snadnou cestu k přerušení cesty či změně nouzových kontaktů a oblíbených destinací.

4. **Dodržování konzistence a standardů** (3 body)
   Aplikace využívá svých vlastních vizuálních prvků a elementů namísto systémových, avšak jejich princip zůstává stejný, a tudíž nejsou nikterak matoucí.

5. **Prevence chyb** (1 bod)
   Aplikace před mazáním prezentuje konfirmační okno.

6. **Jasnost místo zapamatování** (2 body)
   Používání je snadné, avšak díky nedodržování standardů nemusí být pro někoho zcela zřejmé.

7. **Flexibilita pro náročné** (1 body)
   Jednoduchost aplikace nevyžaduje funkce pro náročné.

8. **Estetika a minimalita** (3 body)
   Rozhraní je přehledné, avšak místy křiklavé.

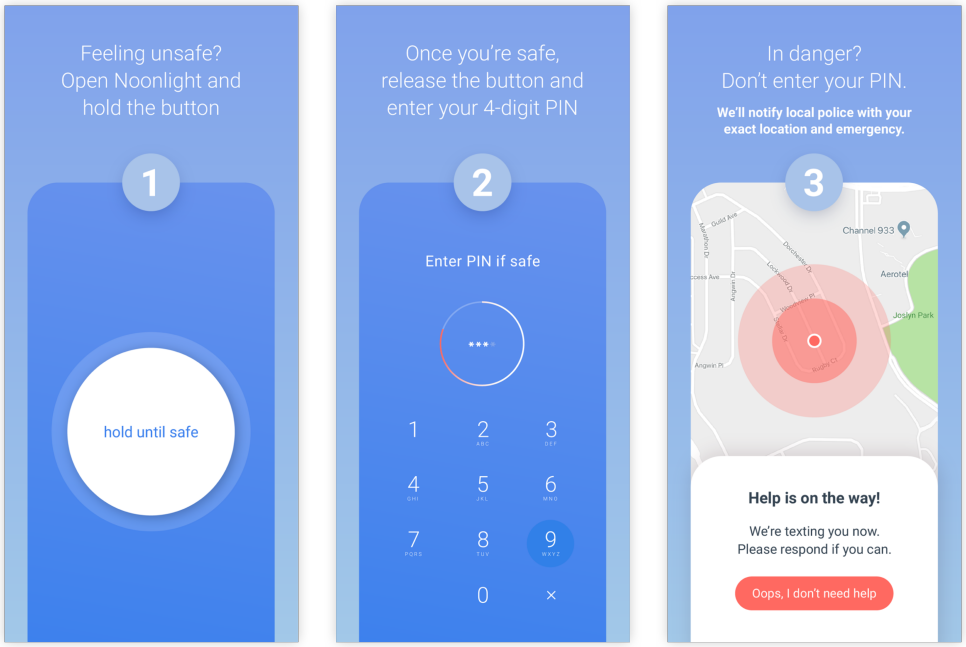
9. **Srozumitelnost problémů a jejich řešení** (5 bodů)
   Občas se nepodařilo přidat oblíbené místo, neboť aplikace zamrzla na načítacím kolečku a ani po minutě nezobrazila žádnou chybovou hlášku a možnost jak pokračovat dále. Stala se tedy v tento moment nepoužitelnou a bylo potřeba ji restartovat.

10. **Dokumentace a nápověda** (1 bod)
    Systém obsahoval v úvodu 10 obrazovek, kde uživateli sdělil základní informace. Následně již nebyla potřeba žádná dokumentace.

1.3.2.5 Noonlight: Feel Protected 24/7[37]

Noonlight je aplikací dostupnou pouze na trhu Spojených států amerických, hodnocení se tak odvíjí především z popisu služby, uživatelských recenzí a snímků dostupných na jejich webu a v obchodech pro mobilní platformy. Primární funkcí této služby je jednoduché tlačítko, které uživatel stiskne necítí-li se bezpečně. V momentě, kdy ho pustí, musí zadat svůj unikátní pin. V případě jeho nezadání se vyvolá akce na informování policejních složek. Celý proces lze vidět na obrázku 1.5.

Aplikace má další funkcionality jako požádání jiného uživatele o checkin či časovou osu, která je užitečná k přidání informací ohledně plánovaných schůzek. Na druhou zmiňovanou funkcionalitu navíc nabízí spolupráci se seznamovací sítí Tinder na rychlé připojení profilu.



Obrázek 1.5: Noonlight: Feel Protected 24/7

Výhody

Výraznou výhodou této aplikace je přímé kontaktování policie, ke které se zároveň donese informace o přesné lokaci uživatele. Použití je velmi jednoduché, což si chválí hlavně uživatelé v hodnoceních – aplikace má k 12. 3. 2022 přes 15 tisíc hodnocení a 4,7 hvězdičky na App Storu[37]. Také nabízí propojení s aplikacemi Uber a Lyft, ze kterých lze při mimořádné události získat důležité informace a předat je nouzovým kontaktům.

Nevýhody

Hlavní nevýhodou je zaměření pouze na americký trh, tudíž i další zajímavé funkcionality, které nejsou závislé na lokální policii, nejsou vůbec využitelné.

Evaluace heuristiky

U aplikace Noonlight jsem evaluaci vynechal, neboť nedošlo k jejímu fyzickému testování.

1.3.3 Shrnutí Nielsenovy heuristiky

V tabulce 1.4 je shrnutí Nielsenovy heuristiky pomocí bodového ohodnocení. Hodnotil jsem pouze samotné rozhraní a jeho používání, nikoliv nedostatky či

pozitiva v rámci funkcionalit. Jak již bylo zmíněno na začátku této kapitoly, škála pro hodnocení je od 1 (nejlepší) do 5 (nejhorší).

Heuristika	Messenger	iMessage, Find My	Follo
Viditelnost stavu systému	3	1	3
Shoda systému s reálným světem	1	1	1
Uživatelova kontrola a svoboda	1	1	2
Dodržování konzistence a standardů	1	1	3
Prevence chyb	3	2	1
Jasnost místo zapamatování	3	1	2
Flexibilita pro náročné	1	1	1
Estetika a minimalita	3	2	3
Srozumitelnost problémů a jejich řešení	4	1	5
Dokumentace a nápověda	1	1	1
Součet	21	12 (nejlepší)	22 (nejhorší)

Tabulka 1.4: Shrnutí Nielsenovy heuristiky (1 – nejlepší, 5 – nejhorší)

1.3.4 Shrnutí

Analýza ukázala, že ačkoli jsou mnohé aplikace vyspělé ve svém provedení, je zde stále velký prostor, který není zaplněn. Jednak absencí funkcionalit či pouhou nepřítomností na určitých trzích.

Mezi hlavními nedostatky napříč funkcionalitou vnímám nemožnost kontaktování a předávání informací s časovou a polohovou značkou. Některá ze zmíněných řešení nabízí sdílení aktuální polohy a zároveň možnost psaní, ovšem nelze nijak dohledat historii přemisťování, natož s možností zobrazení konkrétní informace na mapě. Uživatel může dané místo popsat slovně, avšak

tento způsob je zdlouhavý – kliknout na zprávu či jinou informaci a mít možnost si ji ihned zobrazit na mapě zkrátka chybí.

Taktéž žádná z aplikací uživateli neusnadňuje předávání jiných než polohových informací (mimo kritické momenty, kdy uživatele například spojí s operátorem policejních složek). Ten tak nemá možnost třeba sdělit, že nastupuje do tramvaje, aniž by tuto informaci ručně napsal – a opět musí sám dodat na jakém místě se tak děje.

1.4 Požadavky

V kapitole Požadavky se zabývám vymezením hranic vytvářené aplikace. Cílem je definování funkcionalit systému a dalších důležitých nefunkčních požadavků jako například podporovaná platforma a verze operačního systému či jazykové zaměření. V první podsekci se věnuji teorii, následuje samotná definice požadavků a finálně jsem vytvořil scénáře, jež usnadňují pochopení komplexnějšího fungování aplikace. Požadavky vzešly z rozhovorů s cílovou skupinou, rešerší konkurence a mé vlastní iniciativy.

1.4.1 Jak správně definovat požadavky a metoda FURPS[38, 39]

Požadavky lze rozdělit na dvě základní podskupiny:

- **Funkční požadavky** vymezují prvky a funkce zamýšleného produktu a popisují chování systému za určitých podmínek.
- **Nefunkční požadavky** určují omezení kladená na systém.

Samotné požadavky obsahují určité informace, které je popisují:

- název,
- zkratka (pro jednoduché odkazování),
- popis,
- priorita (vysoká, střední, nízká),
- složitost (vysoká, střední, nízká),
- kategorie (využil jsem kategorizace FURPS).

1.4.1.1 FURPS[40]

Mezi nejznámější metody kategorizace požadavků patří FURPS. FURPS je zkratkou pro Functionality, Usability, Reliability, Performance a Supportability a prvotně se tento model objevil v americké společnosti Hewlett-Packard[41], odkud se rozšířil do obecného povědomí. Tato metoda definuje pět tříd, do kterých lze požadavky klasifikovat.

1. **Funkčnost** (Functionality)
   Tyto požadavky popisují hlavní funkcionality a funkce systému. Odpovídají na otázku, co systém pro uživatele bude dělat?

2. **Použitelnost** (Usability)
   Definují estetiku a přístupnost, zabývají se uživatelovým zážitkem z používání. Odpovídají na otázku, jakým způsobem člověk k rozhraní přistupuje a jak s ním interaguje?

3. **Spolehlivost** (Reliability)
   Detaily ohledně spolehlivosti, robustnosti a odolnosti proti selhání sytému. Požadavky odpovídají na otázku, jak důležitá je systémová dostupnost?

4. **Výkon** (Performance)
   Požadavky zabývající se rychlostí či škálovatelností. Odpovídají na otázky, jak rychle má systém doručit uživateli odpověď a s jakou přesností?

5. **Podporovatelnost / Rozšiřitelnost** (Supportability)
   Náležitosti ohledně udržitelnosti systému s ohledem na jeho architekturu, převážně modularitu a jednoduchost vylepšení a oprav systému. Požadavky odpovídají na otázky ohledně četnosti úprav či jak dlouho systém může být ve stavu údržby?

1.4.2 Funkční požadavky

V rámci této sekce nadefinuji funkční požadavky, ke kterým zároveň přidám informaci o prioritě a základní odhad složitosti. Požadavky jsou také označeny zkratkou pro usnadnění odkazování.

1.4.2.1 F1 - Přihlášení přes externí služby

Priorita: Vysoká
Složitost: Střední

Uživatel bude moci místo ručního vyplnění registračního formuláře využít přihlášení přes existující služby. Konkrétně bude moci využít *Sign in with Google/Facebook/Apple*.

1.4.2.2 F2 - Profil uživatele

Priorita: Střední
Složitost: Nízká

Základním požadavkem je možnost vytvoření profilu a vyplnění dodatečných informací, které mohou usnadnit případnou identifikaci. Některá data lze získat již při registraci přes sociální sítě. Údaje, jež budou ukládány, jsou:

- uživatelské jméno (povinné),
- email (povinné),
- profilová fotografie (nepovinné).

1.4.2.3 F3 - Nouzové kontakty

Priorita: Vysoká
Složitost: Střední

Uživatel si bude moci přidat nouzové kontakty (guardians), které budou dostávat informace v případě důležitých událostí. Tito lidé si budou muset taktéž stáhnout aplikaci, neboť komunikace bude probíhat pomocí notifikací. V případě kritické situace se dá totiž vynutit hlasité upozornění i přes vypnutý zvuk a režim nerušit. Přidávání bude probíhat pomocí uživatelského jména.

1.4.2.4 F4 - Unikátní PIN

Priorita: Vysoká
Složitost: Střední

Pro zajištění, že aplikaci ovládá sám uživatel, bude fungovat unikátní PIN nastavený přímo v aplikaci. Systém tedy nebude využívat biometrické přihlášení, neboť to se dá velmi lehce zneužít. Tento PIN bude uživatel zadávat v kritické momenty, kdy je nutnost ověření důležitá.

1.4.2.5 F5 - Checkin

Priorita: Vysoká
Složitost: Vysoká

Jedním z hlavních cílů aplikace je, aby druhý věděl, že jsem aktuálně v pořádku. K tomu slouží funkce checkin, v rámci které jeden uživatel vyšle požadavek o přihlášení druhému, tedy potvrzení, že je v pořádku. Checkin se dále dělí na dva typy.

F5.1 - Jednorázový checkin

První možností bude jednorázový checkin, který bude ručně vyvolán konkrétní osobou. Tato funkcionalita vzniká na základě rozhovorů v kvalitativním výzkumu (kapitola 1.1.2), kdy se dotazující zdráhají psát pomocí klasických komunikačních aplikací. Uživateli, kterému je tato žádost odeslána, vyskočí notifikace ohledně požadavku, který následně bude moci v aplikace odsouhlasit. Poté přijde upozornění i druhému ohledně potvrzení.

F5.2 - Pravidelný checkin

Druhým způsobem bude možnost vyvolání pravidelného checkinu v určitém intervalu. Uživateli se následně notifikace ohledně stavu bude zobrazovat automaticky, aniž by ji někdo vyvolal manuálně. Při žádosti bude třeba vyplnit čas od a do a následně délku intervalu. Potřeba této funkcionality opět vzešla z rozhovorů a navazuje na jednorázový checkin.

1.4.2.6 F6 - SOS tlačítko

Priorita: Vysoká
Složitost: Vysoká

Tlačítko SOS bude fungovat na podobném principu jako prozkoumané v aplikaci Noonlight. Po jeho stisknutí, které musí trvat určitou minimální dobu (3 sekundy), obrazovka ztmavne. Uživatel nyní musí stále držet prst na obrazovce, ale již na libovolném místě, a až po puštění aplikace požádá uživatele o zadání jeho unikátního PINu. Nezadá-li ho do určitého časového intervalu, aplikace vyšle nouzový signál kontaktům zároveň s informacemi o poloze.

1.4.2.7 F7 - Trip

Priorita: Vysoká
Složitost: Vysoká

Aktuální stav uživatele bude zaznamenáván v rámci komplexní funkcionality s názvem Trip. Uživatel započne cestu, během které bude různými způsoby přidávat užitečné informace – těmi může být fotografie či textová zpráva. K těmto údajům bude zaznamenávána poloha a budou sloužit pro případnou rekonstrukci cesty. Všechny možnosti jsou rozebrány v následujících podsekcích.

F7.1 - Cílová destinace a oblíbené destinace

Uživatel před samotnou cestou jako první zadá cílovou destinaci. Bude mít však možnost tento krok přeskočit. Cestuje-li do určitých destinací opakovaně, bude si je moci uložit do seznamu oblíbených.

F7.2 - Zaznamenávání polohy a mapa

Aplikace bude pravidelně zaznamenávat uživatelovu polohu. Lokace bude taktéž ukládána k dodatečným informacím typu zpráva či fotografie. Následně bude sloužit pro rekonstrukci cesty a jejího zobrazení na mapě.

F7.3 - Textová zpráva a fotografie

Jednou z možností, jak zaznamenat situaci okolo sebe, bude napsání textové zprávy, ke které půjde taktéž přiložit fotografii. Obrázek bude moci být odeslán i s prázdnou zprávou.

F7.4 - Rychlé akce

Aby aplikace šetřila člověku čas a co nejméně ho rozptylovala od sledování okolí, budou k dispozici takzvané „rychlé akce“. Rychlá akce bude mít formu tlačítka, které po stisknutí zaznamená například nástup do tramvaje, blízkost zvláštní skupinky či vystoupení z taxi.

Jelikož nelze dopředu s přesností predikovat uživatelovu cestu, bude mít kromě pár určitých fixních tlačítek možnost před samotnou cestou definovat plán své trasy pomocí výběru z předem definovaných akcí. Může tedy navolit, že se jeho cesta skládá z přemístění tramvají a následnou pěší chůzí. Nebo naopak, že celou cestu plánuje využít taxi služeb.

F7.5 - SOS tlačítko

Tlačítko, které bylo definováno v rámci sekce 1.4.2.6, bude neustále přítomno na obrazovce i během aktivní cesty pro rychlou přístupnost.

F7.6 - Sdílení s kontakty

Nouzové kontakty budou mít přístup k aktuálním informacím ohledně cesty a zároveň jim budou chodit notifikace ohledně aktivit, které během ní uživatel udělal.

1.4.2.8 F8 - Obejití vypnutí telefonu

Priorita: Vysoká
Složitost: Střední

Aby nedocházelo k obejití informování nouzových kontaktů, například přímým vypnutím mobilního zařízení, bude server detekovat neaktivitu v rámci příchozích požadavků z klienta. Jelikož i data o aktivní cestě budou uložena na backendové části, nouzové kontakty budou mít přístup ke všem relevantním informacím.

1.4.3 Nefunkční požadavky

U nefunkčních požadavků bude evidován popis, informace o prioritě a odhad složitosti, avšak oproti funkčním bude navíc přiřazena i kategorie.

1.4.3.1 N1 - Platforma a operační systém

Priorita: Vysoká
Složitost: Nízká
Kategorie: Podporovatelnost / Rozšiřitelnost (Supportability)

Počáteční vývoj aplikace proběhne pouze pro iOS.

Hlavním důvodem dání přednosti iOS nad operačním systémem Android, které dohromady dominují evropskému a americkému trhu, je rychlost vývoje a testování. Ačkoliv si Android drží celkově větší zastoupení na trhu, má velkou roztříštěnost napříč verzemi, kdy k 27. 3. 2022 poslední verze Android 12 vydaná počátkem října 2021 měla podíl mezi 7 až 8 procenty v rámci všech verzí Androidu, což znesnadňuje vývoj a testování a zároveň s tím souvisí i složitější využití možností, které nové operační systémy a mobilní zařízení přinášejí.[42, 43]

Touto strategií bych rád vyvinul MVP (minimum viable product), tedy verzi aplikace, která obsahuje základní funkcionality tak, aby šla otestovat v reálném nasazení. A teprve následně by se vývoj posunul i na platformu Android.

Možnost použití technologií pro vývoj cross-platform aplikací jsem zamítl ihned, neboť s nimi je velmi složité přistupovat k nejaktuálnějším technickým možnostem platforem, popřípadě je nutnost si psát vlastní bridge (most), což opět zpomaluje vývoj.

1.4.3.2 N2 - Verze operačního systému

Priorita: Vysoká
Složitost: Nízká
Kategorie: Podporovatelnost / Rozšiřitelnost (Supportability)

Dostupnost aplikace bude omezená na iOS 15 a vyšší.

Na stránkách Apple[44] lze dohledat aktuální tržní podíl jednotlivých verzí iOS, který pro zařízení představená v posledních 4 letech k 27. 3. 2022 činí:

- iOS 15 – 72 %
- iOS 14 - 26 %
- Dřívější verze - 2 %

Pro všechna zařízení nehledě na stáří je rozložení následující:

- iOS 15 – 63 %
- iOS 14 - 30 %
- Dřívější verze - 7 %

Z tohoto důvodu byla vyvíjená mobilní aplikace omezena na verzi iOS 15 a vyšší, neboť zahrnuje většinu tržního podílu a zpětná podpora by znamenala v začátcích zpomalení vývoje kvůli nutnosti implementování vlastních řešení pro nové funkcionality.

1.4.3.3 N3 - Jazyk

Priorita: Střední
Složitost: Nízká
Kategorie: Použitelnost (Usability)

Aplikace bude v základu dostupná pouze v anglickém jazyce.

1.4.3.4 N4 - Rychlost notifikací

Priorita: Střední
Složitost: Nízká
Kategorie: Výkon (Performance)

Pro důležité notifikace by rychlost doručení měla splňovat požadavky zapsané v tabulce 1.5. Čas se počítá od vyvolání akce až po doručení.

Percentil	Rychlost doručení
99	15 sekund
95	10 sekund
50	5 sekund

Tabulka 1.5: Rychlost doručení důležitých notifikací

1.4.4 Případy užití a scénáře

Pro lepší pochopení některých funkčních požadavků jsem vytvořil případy užití a doplnil je scénáři. Ačkoli by se tato část dala kvalifikovat spíše jako návrh, zařadil jsem ji přímo za požadavky, neboť případy užití a jejich scénáře na ně logicky navazují a ujasňují princip fungování.

1.4.4.1 Jak správně popsat případ užití a scénář[45]

Případ užití popisuje, jakým způsobem budou uživatelé provádět jednotlivé úkoly v rámci systému, a je reprezentován pomocí sledu jednoduchých kroků.

Pomáhají tak přiblížit chování systému a zároveň při jejich tvorbě přispívají k uvědomění si, jaké problémy mohou nastat s jejich plněním.

Případ užití běžně obsahuje následující prvky:

- aktéry,
- cíl uživatele,
- spouštěče,
- prerekvizity,
- kroky k dosažení cíle,
- hlavní scénář,
- alternativní scénáře.

1.4.4.2 UC1 - Jednorázový checkin

Aktéři: Uživatel a jeho kontakt
Prerekvizity: Přidán alespoň jeden kontakt
Popis:

V případě, že chce uživatel zjistit, zdali je další osoba v pořádku, ale nechce využít konvenční cesty (viz kapitola kvalitativní výzkum 1.1.2), může požádat dotyčného o takzvaný checkin, tedy přihlášení, že je v pořádku.

Hlavní scénář:

1. Uživatel rozklikne stránku s žádostí o checkin.
2. Vybere kontakt.
3. Klikne na tlačítko požádat o checkin.
4. Systém pošle kontaktu notifikaci.
5. Kontakt rozklikne notifikaci a potvrdí, že je v pořádku.
6. Systém informuje notifikací žadatele, že jeho kontakt potvrdil checkin.

1.4.4.3 UC2 - Pravidelný checkin

Aktéři: Uživatel a jeho kontakt
Prerekvizity: Přidán alespoň jeden kontakt
Popis:

Obdobně jako u jednorázového checkinu může uživatel chtít po svém kontaktu, aby mu pravidelně od něj přicházelo, že je v pořádku.

Hlavní scénář:

1. Uživatel rozklikne stránku s žádostí o checkin.
2. Vybere kontakt.
3. Zvolí časový interval a periodu.

4. Systém pravidelně posílá kontaktu notifikace.
5. Kontakt potvrdí, že je v pořádku.
6. Systém notifikuje žadatele, že jeho kontakt potvrdil dílčí checkin.

1.4.4.4 UC3 - SOS Tlačítko

Aktéři: Uživatel
Prerekvizity: Přidán alespoň jeden nouzový kontakt
Popis:

V případě, že se člověk necítí v aktuální situaci komfortně, zmáčkne tlačítko, po jehož puštění buď zadá kód či budou notifikování nouzové kontakty.

Hlavní scénář:

1. Uživatel klikne na tlačítko SOS a drží ho.
2. Systém tři vteřiny čeká a následně zobrazí uživateli černou stránku (informace o spuštění se pošle na server).
3. Uživatel stále drží prst na obrazovce.
4. V momentě, kdy uživatel pustí prst z obrazovky, systém prezentuje komponentu na zadání uživatelova unikátního kódu (informace o puštění se pošle na server).
5. Zadá-li kód, obrazovka zmizí a uživatel může pokračovat v používání aplikace (informace o zadání kódu se pošle na server).
6. Nezadá-li kód v dostatečné době, systém rozpošle upozornění nouzovým kontaktům.

1.4.4.5 UC4 - Trip

Aktéři: Uživatel
Prerekvizity: Přidán alespoň jeden nouzový kontakt
Popis:

Uživatel chce zaznamenávat události při pohybu z bodu A do bodu B.

Hlavní scénář:

1. Uživatel klikne na zahájení cesty.
2. Nepovinně zvolí cílovou destinaci a předpokládaný čas příchodu.
3. Nepovinně naplánuje svou cestu pomocí rychlých akcí.
4. Započne cestu.
5. Následně zaznamenává informace pomocí rychlých akcí, fotografií či textu.
6. Bod 5 se opakuje, dokud uživatel nedá ukončit cestu.
7. Aplikace vyzve uživatele k zadání jeho unikátního PINu.
8. Uživatel zadá PIN a cesta se ukončí.

1.4.4.6 Ostatní případy užití

- UC5 - poslání žádosti o přidání kontaktu,
- UC6 - potvrzení žádosti o přidání kontaktu,
- UC7 - změna uživatelského jména,
- UC8 - změna PINu,
- UC9 - změna profilové fotografie,
- UC10 - zobrazení historie SOS,
- UC11 - zobrazení historie checkinů,
- UC12 - zobrazení historie cest,
- UC13 - zobrazení detailu SOS,
- UC14 - zobrazení detailu checkinu,
- UC15 - zobrazení detailu cesty.

Kapitola 2

Metodiky vývoje

Metodiky jsou standardizací pracovních procesů a přinášejí zjednodušení řízení projektu. V rámci vývoje softwaru jich existuje celá řada a liší se svým přístupem k organizaci, psaní samotného kódu či dokumentace. Jejich hlavním přínosem je přehlednost, opakovatelnost, snadnější kontrola a odhadování. Mezi sebou mají celou řadu odlišností, a proto je nutné zvažovat výběr vhodné metodiky na základě velikosti týmu, typu vyvíjeného produktu či pouze na subjektivních pocitech členů vývojové skupiny, při kterém přístupu jsou nejvíce efektivní.

Mezi nejznámější patří:

- Waterfall
- Iterativní
- Prototyping
- Spiral
- Rapid application development (RAD)
- SCRUM
- Extrémní programování

2.1 Klasické vs agilní metodiky

Metodiky vývoje lze dělit na dvě základní skupiny – klasické a agilní.

Klasické metodiky vývoje kladou větší důraz na tvorbu a udržování dokumentace. Dbají na počáteční přípravu a analýzu. Bývají tedy propracovanější, s čímž ovšem narůstá prvotní složitost, avšak tato složitost je investicí do budoucna při dalších úpravách a vylepšeních či začlenění nového kolegy.

Agilní metodiky se naopak snaží eliminovat nadbytečné procesy a nástroje, preferují spolupráci se zákazníkem, fungující software nad rozsáhlou dokumentací či reakci na změnu namísto držení se předem definovaného plánu.

Samotné metodiky nemusí striktně dodržovat jeden z těchto dvou přístupů, avšak popis tohoto dělení slouží k uvědomění si, jakými směry se mohou vydávat.

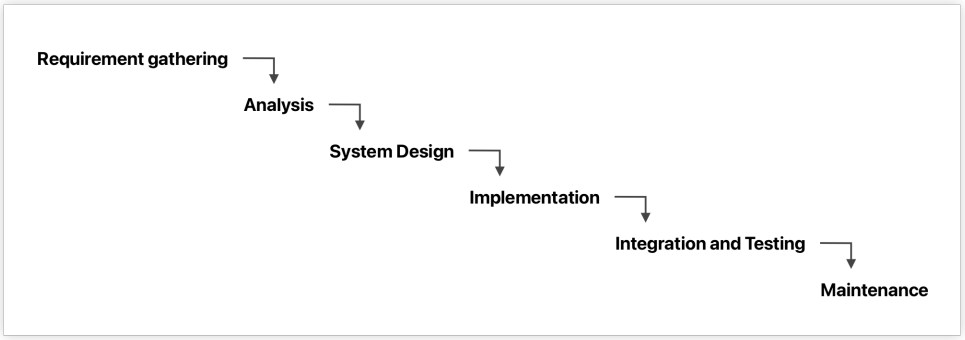
2.2 Waterfall[1]

Vodopádový model se skládá z několika fází, kdy před začátkem jedné musí být předešlé plně dokončené. Klade důraz na analytickou a plánovací část – vytváření dokumentace, diagramů, analýzu požadavků.

Jednotlivé fáze se podle různých zdrojů liší, avšak princip zůstává prakticky stejný. Já jsem je vizualizoval na obrázku 2.1 a shrnul v následujícím seznamu:

1. počáteční sběr informací a požadavků,
2. analýza,
3. návrh,
4. implementace,
5. nasazení a testování,
6. údržba.

Výhodou vodopádového modelu je jeho jednoduchost, důsledná kontrola výstupů a zpracování podpůrné dokumentace. **Nevýhodou** pak absence reagování na změny a nemožnost průběžného testování.



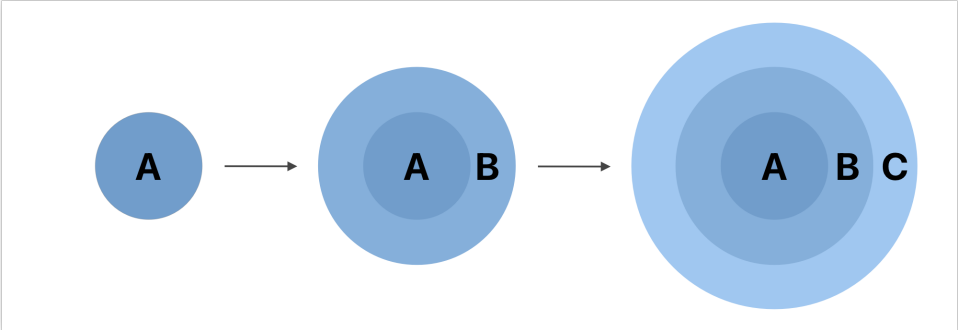
Obrázek 2.1: Waterfall (vodopád)

2.3 Iterativní a inkrementální[2, 3]

Iterativní a inkrementální způsoby vývoje rozšiřují vodopádový model a zvyšují jeho použitelnost v praktickém světě. Vývoj se rozdělí na několik menších částí, u kterých se následně pokračuje již známým sekvenčním přístupem. Iterativní a inkrementální přístupy se liší v několika detailech:

- **Iterativní** nelpí na dokončení určité části, ale zpětná vazba může přicházet přímo při vývoji konkrétní funkcionality, která se tak vypiluje.
- **Inkrementální** se zaměřuje na vyvinutí nutného funkčního minima a až je potřeba, tak se vyvíjí další funkcionality (obrázek 2.2).

Výhodou iterativního a inkrementálního přístupu oproti klasickému vodopádu je jeho větší flexibilita a možnost reakce na změny a nově zjištěné skutečnosti během vývoje. Tento model navíc umožňuje paralelní vývoj pro komplexnější systémy, u čehož je však nutné zkorigování, aby nedošlo k nejasnostem v rámci následujícího propojení.



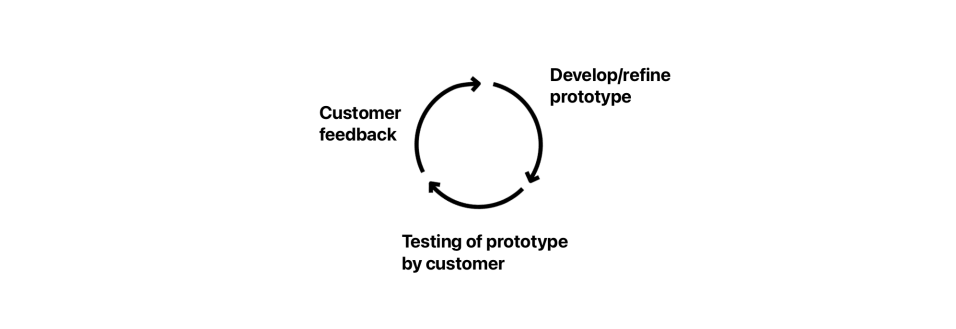
Obrázek 2.2: Inkrementální metodika

2.4 Prototyping[4]

Prototyping model je iterativní proces, při kterém se vytváří funkční prototyp systému, který nabízí v malém měřítku kopii koncového produktu. Tento prototyp se využívá k zjištění zpětné vazby zákazníka, jenž sám často předem neví přesné požadavky. Iterativním přístupem se společnými silami tým dostane k finální specifikaci produktu s možností využít již hotové naimplementované části do finálního systému. Vizuální přiblížení lze pozorovat na obrázku 2.3.

Výhodou prototypingu je aktivní účast zákazníka, která zvyšuje jeho spokojenost s finálním produktem, velká flexibilita ve zpracování změn a nových požadavků a včasné odchycení problémů při testování.

Naopak **nevýhodou** jsou velké náklady, kdy se celý proces může protáhnout díky zapojení zákazníka, nedůsledně vedená dokumentace a pro samotné vývojáře může nastat chaos v zadávání požadavků.



Obrázek 2.3: Prototyping

2.5 Spiral[5]

Spirální model je iterativní model soustřeďující se na analýzu rizik. V grafické reprezentaci (obrázek 2.4) znázorňuje spirálu rozdělenou na čtyři kvadranty. Každá smyčka reprezentuje jednu fázi sestávající právě ze čtyř částí, kterými jsou:

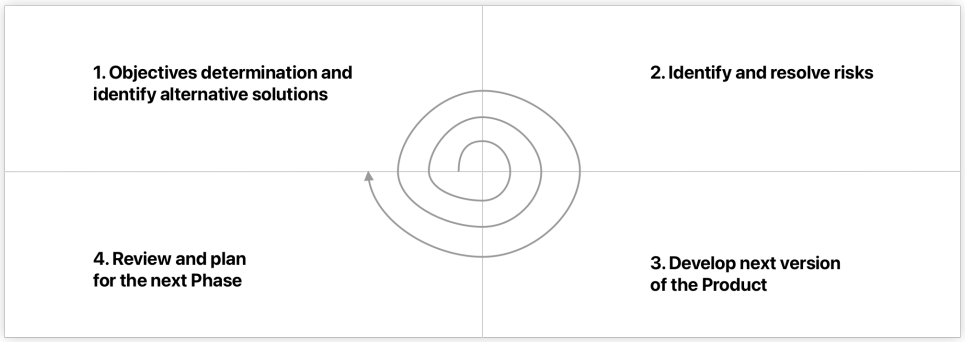
1. sběr požadavků, jejich analýza a nalezení alternativních cest,
2. identifikace a analýza rizik,
3. vývoj nové verze produktu a testování,
4. zhodnocení zákazníkem a plánování další fáze.

Výhodou tohoto modelu je jeho zaměření na rizika, která z velké části nelze předem odhadnout. Z iterativního charakteru je zde opět flexibilita v reagování na změny, a zákazník tak může být plně spokojen.

Hlavní **nevýhodou** tohoto modelu je jeho komplexita, jež se nehodí na menší projekty. Samotná analýza rizik je náročná jak časově, tak často i finančně.

2.6 Rapid application development (RAD)[6, 7]

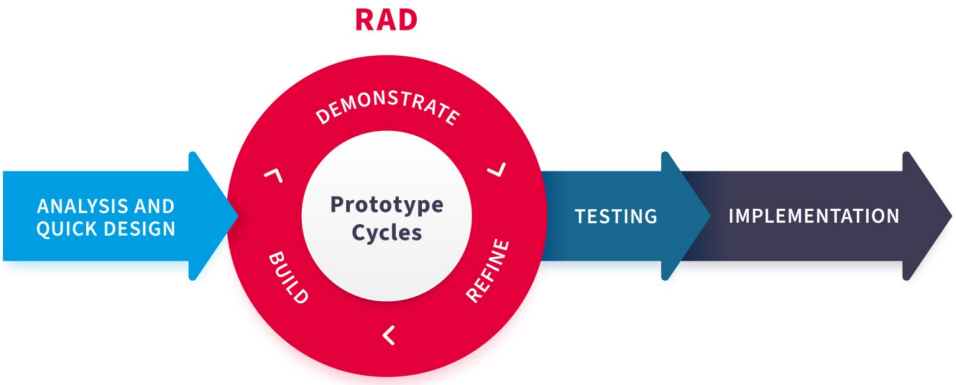
Rapid application development je modelem kladoucím důraz na rychlost vývoje za použití iterativního přístupu. Rozděluje celý projekt na moduly, které lze vyvíjet paralelně, a využívá nástrojů pro automatické generování kódu. Skládá se ze čtyř hlavních fází, kde fáze 2 a 3 se iterativně opakují:



Obrázek 2.4: Spiral

1. **Requirements planning** – obnáší analýzu úkolů, tvoření scénářů, případů užití, brainstorming.
2. **User description** – během této fáze se získává uživatelova zpětná vazba, prototyp se upravuje, aby splňoval jejich potřeby.
3. **Construction** – samotný vývoj, kterého uživatel může být stále součástí a navrhovat změny, použití automatizovaných nástrojů.
4. **Cutover** – testování, integrace a přijetí.

Výhodou je rychlost vývoje, důsledkem čehož klesá i samotná cena. Model včasně reaguje na změny a nová zjištění. **Nevýhodou** je složitější organizace a synchronizace týmů při paralelním vývoji a rychlost vývoje se může projevit na kvalitě kódu.



Obrázek 2.5: Rapid application development (RAD)[15]

2.7 SCRUM[8, 9, 10]

Scrum je agilní metodikou využívající iterativní proces, jehož základem jsou sprinty trvající určitou dobu, většinou v rozmezí jednoho týdne až měsíce. Priorita je na týmu jako celku, který se může sám organizovat. Pro jeho pochopení je nutné vysvětlit pár pojmů a vizualizace je na obrázku 2.6:

- **Product backlog** je seznam všech požadavků, jež se od výsledného systému očekávají.
- **Srum master** kontroluje dodržování celého procesu.
- **Product owner** odpovídá za chod projektu a prioritizuje požadavky v product backlogu.
- **Sprint** je iterací, na jejímž konci je nová verze produktu.
- **Sprint backlog** je seznam požadavků pro jeden konkrétní sprint.

Samotný sprint má následující části:

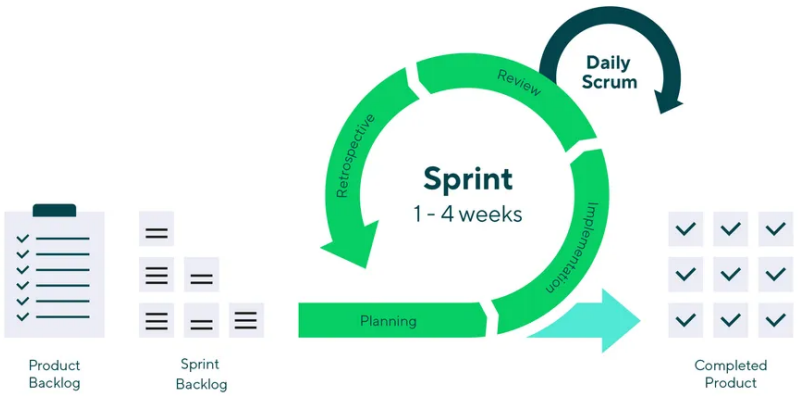
- **Sprint planning** je počáteční plánovací schůzka, během které se upřesňuje sprint backlog.
- **Scrum** je jednodenní iterací uvnitř sprintu začínající krátkou úvodní schůzkou, kde by se měl celý tým sesynchronizovat.
- **Sprint review** je konečné zhodnocení výstupů sprintu a jejich prezentování.
- **Sprint retrospective** je zhodnocení samotného fungování sprintu a hledání možností, jak zefektivnit a zkvalitnit práci.

Výhodou přístupu Scrum je jeho důraz na zákazníkovu spokojenost, pořádek ve vedení a sesynchronizování týmu, který následně může být mnohem efektivnější. Změny v požadavcích nemohou být provedeny v rámci začatého sprintu, což je nevýhodou pro zákazníka, avšak samotný tým nemá následně v práci chaos. Scrum popisuje spíš organizaci, nežli samotný vývoj a je vhodné ho zkombinovat například s praktikami z extrémního programování.

2.8 Extrémní programování[11]

Extrémní programování je agilní metodikou, která si klade za cíl vzít osvědčené postupy a posunout je na extrémní úroveň. Příklady takových principů jsou:

- **Párové programování a kontrola** – extrémní programování doporučuje párové programování, kdy jeden na druhého dohlíží a po určitém časovém intervalu se střídají. Také pomocí kontroly kódu lze odhalit velké množství problémů.
- **Testování** – časté testování, kdy mnohdy jsou samotné testy psané dříve než funkcionalita.



Obrázek 2.6: Scrum[16]

- **Integrační testování** – psaná funkcionalita by se měla testovat pravidelně v rámci celého systému a to klidně několikrát denně.
- **Komunikace** – velmi důležitá je komunikace týmu se zákazníkem tak, aby byl výsledně maximálně spokojen.
- **Jednoduchost** – funkcionality se píšou až v momentě, kdy jsou potřeba, požadavky se rychle mění a práce navíc je zbytečná.
- **Odvaha** – odvaha zahodit stávající části implementace či dělat velká rozhodnutí.

Výhodou extrémního programování je jeho důraz na kvalitu a funkčnost, nová funkcionalita je otestována a nerozbíjí stávající, komunikace se zákazníkem předchází nedorozuměním, nevzniká technický dluh do budoucna, avšak mnohdy je nedostatečná dokumentace. **Nevýhodou** je náročnost, kterou s sebou přináší častá komunikace se zákazníkem a obecně častá komunikace celého týmu a dodatečná práce spojená s testováním a párovým programováním, což může protahovat celý vývoj.

2.9 Test driven development[12]

Programování řízené testy je způsob, v jehož prvním kroku po specifikování funkcionalit není jejich programování, nýbrž napsání testů, které ověřují jejich funkčnost. Až následně se píše samotný kód, jenž je při nesplnění testů upravován.

2.10 Shrnutí a výběr pro vývoj aplikace Safie

V této kapitole jsem popsal nejznámější metodiky vývoje a uvedl jejich výhody a nevýhody. V praktickém použití se nutně nevylučují a není nezbytné se držet jejich přesných hranic.

Pro výběr metodiky na vývoj aplikace v rámci této diplomové práce jsem bral v potaz důležitou skutečnost, že systém vyvíjím sám, z čehož mi vyšlo několik bodů:

- Není potřeba pořádat schůzky a synchronizovat se v týmu.
- Ačkoliv se dokumentace bude hodit do budoucna, není nutné na ni klást přílišný důraz na úkor samotného vývoje, neboť se vyznám v jednotlivých částech systému a tedy je lehčí se vyvarovat problémům při jejich propojení.
- V rámci tvorby zastávám roli vývojového týmu i zákazníka, tudíž je zde absence nutnosti vytváření prototypů a získávání zpětné vazby od klienta.

Z těchto důvodů jsem se rozhodl vývoj vést iterativním přístupem – tedy vyvinout jednu konkrétní funkcionalitou a až následně se věnovat další.

Kapitola 3

Serverová část

V této kapitole se zabývám realizací serverové části, čemuž předchází výběr technologií, návrh, analýza a tvorba podpůrné dokumentace ve formě diagramů. Serverová část zajišťuje správu a persistenci dat a navenek poskytuje rozhraní, s kterým komunikuje mobilní aplikace.

3.1 Analýza a návrh

V rámci analýzy je důležité ujasnit si princip fungování a spojitosti mezi jednotlivými prvky systému. Analýza a návrh dovolují odhalit potenciální problémy a nalézt jejich řešení.

3.1.1 Výběr technologií

Volba technologií je jedním z prvních a velmi důležitých kroků při tvorbě informačního systému. Pro každý projekt je nutné vyhodnotit jeho potřeby, na jejichž základě se vývojový tým rozhodne, jaký programovací jazyk, knihovny a další služby využít. Prioritou může být výpočetní rychlost, rychlost vývoje, stabilita nebo naopak nové experimentální funkcionality či velikost podpůrné komunity.

Pro výčet nejpopulárnějších jazyků jsem zvolil PYPL index (PopularitY of Programming Language)[23]. Ten předpokládá fakt, že čím více je jazyk vyhledávaný, tím více je populární. Samotná data pocházejí z Google Trends[46] a seřazena jsou podle podílu z celkové sumy vyhledávacích dotazů. Data z dubna 2022 lze přehledně vidět v tabulce 3.1. Sloupec trend udává rozdíl oproti dubnu 2021.

V následujících sekcích uvedu, které technologie jsem zvolil a u hlavních i důvody, které mě k volbě vedly. Velké množství technologií bylo zvoleno na zá-

Pořadí	Programovací jazyk	Podíl	Trend
1.	Python	27.95 %	-2.2 %
2.	Java	18.09 %	+0.6 %
3.	JavaScript	9.14 %	+0.5 %
4.	C#	7.39 %	+0.5 %
5.	C/C++	7.06 %	+0.4 %
6.	PHP	5.48 %	-0.9 %
7.	R	4.41 %	+0.5 %
8.	TypeScript	2.27 %	+0.5 %
9.	Objective-C	2.23 %	-0.3 %
10.	Swift	2.06 %	+0.2 %

Tabulka 3.1: PYPL Index (PopularitY of Programming Language), duben 2022, trend oproti dubnu 2021[23]

kladě pracovních zkušeností s nimi a jejich použití na projekt, který nevyžaduje experimentální funkcionality, je vhodné, neboť jsou praxí osvědčené.

3.1.1.1 NodeJS (JavaScript / TypeScript)

„*JavaScript (JS) je lightweight, interpretovaný nebo just-in-time kompilovaný programovací jazyk s first-class funkcemi. I když je nejznámější jako skriptovací jazyk pro webové stránky, je hojně využíván také v prostředí bez prohlížeče, jako například Node.js, Apache CouchDB a Adobe Acrobat. JavaScript je prototypový, multiparadigmatický, jednovláknový, dynamický jazyk, který podporuje objektově orientované, imperativní a deklarativní styly*“[47] (přeloženo autorem)

TypeScript je následně nadstavbou JavaScriptu přinášející silné typování.

„*Node.js aplikace běží v jednom procesu bez vytváření nových vláken pro každý požadavek. Node.js poskytuje sadu asynchronních I/O operací ve své standardní knihovně, které zabraňují blokování JavaScript kódu a obecně, knihovny pro Node.js jsou psány s využitím neblokujících paradigmat, takže blokující chování je spíše výjimkou nežli normou.*

Provádí-li Node.js I/O operaci, jako například čtení ze sítě, přístup k databázi či souborovému systému, místo blokování vlákna a plýtvání CPU cykly čekáním Node.js pokračuje v operacích, až když dostane nazpět odpověď.“[48] (přeloženo autorem)

Node.js je tedy asynchronní, událostmi řízené běhové prostředí pro JavaScript postavené na *V8 JavaScript Engine*[49], pomocí kterého lze velmi lehce vytvářet škálovatelné systémy, u kterých odpadá nutnost starat se o správu vláken

a procesů. Důležitým pojmem je takzvaný *event-loop*[50], který stojí na pozadí asynchronního a neblokujícího chování.

Důvody výběru:

Vyvíjená aplikace nevyžaduje dlouhé a složité výpočty, naopak jednou z nejdůležitějších akcí je přístup do databáze. Asynchronní neblokující přístup je tedy velmi důležitý. Jednoduchost JavaScriptu a odpadající potíže se správou vláken/procesů či automatickou alokací a dealokací objektů jsou velmi vhodné argumenty pro vývoj, který není primárně zaměřený na výpočetní rychlost, a pro projekt, který si klade za cíl ověření konceptu a vytvoření MVP (minimum viable product, minimální životaschopný produkt). Rozšířenost a popularitu těchto technologií dokazuje velikost komunity, což se může hodit pro další vývoj projektu.

3.1.1.2 Yarn[51]

V rámci NodeJS se využívají dva hlavní správci balíčků a knihoven – npm[52] a Yarn. Při jejich používání nenalezneme příliš rozdílů, které se navíc časem stírají, avšak velmi často je upřednostňován Yarn díky lehce lepšímu výkonu[53]. Není-li však potřeba se zabývat každou vteřinou, není volba pro běžný projekt důležitá. Přechod z jednoho na druhý je navíc velmi triviální. V rámci této práci využívám správce Yarn.

3.1.1.3 NestJS[54]

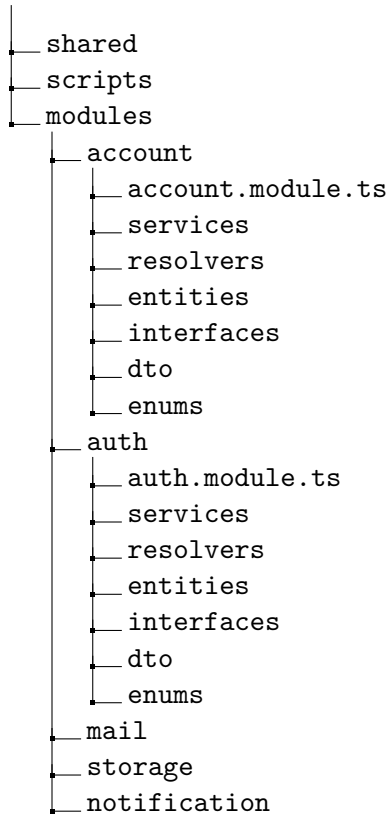
NestJS je pokročilý framework pro vývoj aplikací v TypeScriptu (potažmo JavaScriptu) postavený nad frameworkem express[55] (nabízí i možnost využít fastify[56] místo express). Express nabízí minimální a tedy flexibilní kostru pro vývoj mobilních či webových aplikací. NestJS přidává složitější funkcionalitu, kterou popíši v následujících odstavcích.

Modularizace

Základní prvkem pro přehlednost je možnost modularizace kódu. Modul v rámci frameworku NestJS obsahuje veškerý kód k obsloužení určité funkcionality. Zároveň však může mít závislost na některém z dalších modulů – například pro správu uživatelů, práv, souborů, notifikací či mailů. NestJS tedy primárně uskupuje soubory podle úlohy a až sekundárně podle typu (myšleno controller, definice entit, rozhraní).

Ukázka:

```
/src
├── config
└── migrations
```



Dependency injection

Dependency injection je technikou, ve které vývojář deleguje vytváření instancí závislostí na takzvaný DI Container. Nemusí se o tuto část tedy starat sám manuálně. NestJS toto nabízí pomocí dekorátoru @Injectable() a následného zaregistrování služby v modulu. V rámci jednoho modulu mají služby přístup ke všem dalším, avšak napříč moduly je nutné, aby jeden službu exportoval a druhý naopak importoval. NestJS má i řešení, vznikne-li kruhová závislost.

5 druhů middleware

NestJS nabízí 5 druhů middleware, kde každý má určitý způsob využití:

- **Middlewares** – funkce, které jsou volány před zpracováním požadavku konkrétním obsluhovačem v controlleru. Mají přístup k objektu žádosti i odpovědi.
- **Exception filters** – vrstva, která odchytává vyhozené výjimky v rámci zpracování požadavků.
- **Pipes** – jsou využívány k transformaci a validování příchozích dat.
- **Guards** – hlídají, zdali požadavek má být obsloužen, nejčastěji spojeno

s kontrolou práv.
- **Interceptors** – se využívají k přidání logiky navíc před zpracováním a po zpracování požadavku.

Pozornému člověku neujde, že jisté typy úloh by šly odbavit pomocí vícero metod, jejich dělení je tak z velké části i pro přehlednost.

Rozšíření

V rámci ekosystému NestJS je velmi jednoduché začlenit složitější služby a techniky – například zprostředkovatele zpráv a událostí (Redis, Kafka, RabbitMQ, MQTT), dotazovací jazyk GraphQL, validace přicházejících dat, ukládání do mezipaměti, serializace či verzování endpointů.

3.1.1.4 TypeORM[57]

Pro manipulaci s daty jsem vybral TypeORM, nástroj pro automatickou konverzi mezi objekty v rámci kódu aplikace a daty v relační podobě uložené v databázi. TypeORM nabízí rozvinutou funkcionalitu na práci s daty a relacemi mezi objekty, na vytváření migrací (změn v rámci schématu databáze) či na správu transakcí.

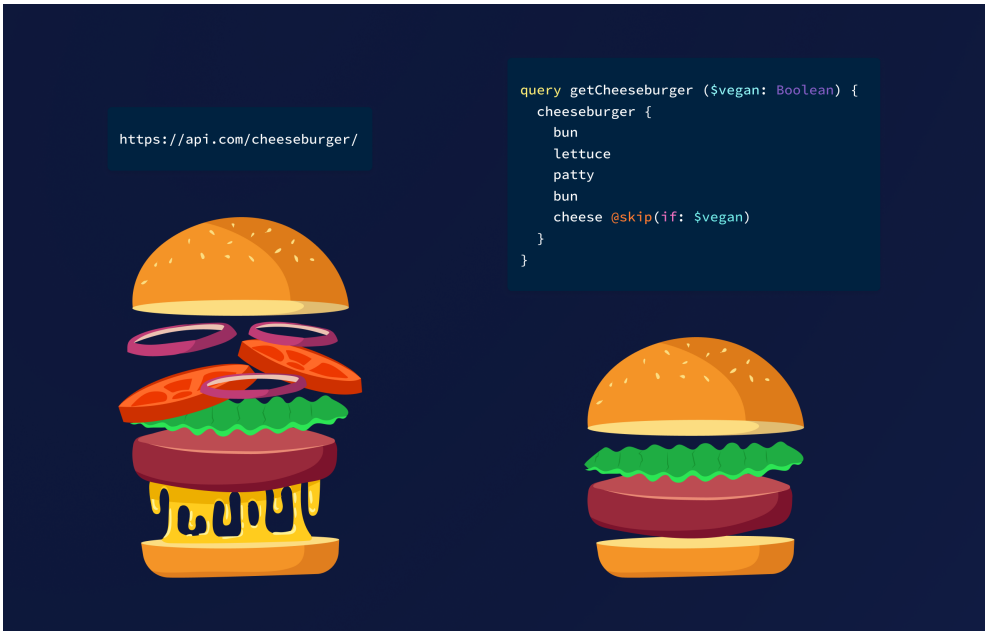
3.1.1.5 GraphQL a Apollo GraphQL[58]

GraphQL je jazykem pro dotazování a manipulaci s daty. Data jsou vnímána jako propojený graf, ze kterého si dotazem vybíráme pouze určitou část. GraphQL, ačkoliv není softwarový architektonický styl, se považuje jako alternativa k RESTful rozhraní. Jednoduchý rozdíl lze pozorovat na obrázku[1] 3.1.

Základem serverové části GraphQL komunikace je jediný koncový bod, na který jsou směřovány veškeré dotazy. Ten na základě obsahu dotazu volá příslušné operace *resolverů*, které konkrétní požadavek obslouží. Samotná implementace získávání a zpracování dat je čistě na programátorovi, a tím se může lišit projekt od projektu. Jelikož data jsou vnímaná jako graf, dotazující má možnost se neomezeně zanořovat, je-li k tomu server uzpůsoben. K tomu v rámci GraphQL jsou *field resolvers*, které slouží právě k získávání dat zanořených v existující entitě. Příklad uvádím na zdrojovém kódu 3.1.

Apollo GraphQL je sadou nástrojů pro usnadnění vývoje jak na straně klienta, tak i serveru. Přináší možnosti automatického generování GraphQL schémat na backendové straně a jednoduše je přenášet do webových i mobilních aplikací. Zároveň existuje balíček na integraci s NestJS vyvíjený přímo pod hlavičkou NestJS.

[1] Pro vložení obrázku do diplomové práce převážila jeho výstižná popisnost nad neformalitou.



Obrázek 3.1: GraphQL vs Rest[17]

```
1  @Resolver(of => Author)
2  export class AuthorsResolver {
3    constructor(
4      private authorsService: AuthorsService,
5      private postsService: PostsService,
6    ) {}

8    @Query(returns => Author)
9    async author(@Args('id', { type: () => Int }) id: number) {
10     return this.authorsService.findOneById(id);
11   }

13   @ResolveField()
14   async posts(@Parent() author: Author) {
15     const { id } = author;
16     return this.postsService.findAll({ authorId: id });
17   }
18 }
```

Zdrojový kód 3.1: Ukázka Apollo GraphQL v NestJS

3.1.1.6 PostgreSQL

Aplikace počítá s ukládáním dat v strukturované podobě, která je předem známá. Zároveň se předpokládá větší důraz na čtení dat před jinými opera-

cemi. Taktéž není potřeba brát ohled na souběžný zápis, ke kterému nebude docházet. Volba tedy byla namířena na relační databáze – konkrétně na nejznámější PostgreSQL a MySQL. Finální rozhodnutí bylo ovlivněno hostingem, kde bude celý systém testován a prvotně spuštěn. V rámci něho je nejvýhodnější možnost použití PostgreSQL databáze. Jelikož aplikace nevyžaduje funkcionalitu specifickou pro určitý databázový stroj, byla dána přednost tomuto systému. TypeORM má podporu obou strojů a případný přechod by znamenal vygenerování nových migrací a přenos dat.

3.1.1.7 Další knihovny a balíčky

Firebase[59]

Pro zasílání notifikací byla využita služba Firebase. Ta kromě notifikací nabízí služby na sledování statistik, přihlašování přes vícero externích služeb či vlastní databázové řešení. Avšak žádná z dalších možností není v systému ke dni odevzdání práce využita.

AWS S3[60]

Na ukládání fyzických souborů jako například profilová fotografie či přílohy v rámci funkce Trip je zvolena cloudová služba S3 od Amazon Web Services.

3.1.2 Návrh architektury

Architektura serverové části je pro větší přehlednost a udržitelnost rozdělena na tři vrstvy. Diagram lze vidět na obrázku 3.2.

3.1.2.1 Prezentační vrstva

Prezentační vrstva v rámci této práce obsahuje pouze jeden balíček tříd s názvem *Resolver*. Třídy v rámci tohoto balíčku odbavují požadavky GraphQL komunikace, kde jednotlivé funkce se mapují na konkrétní dotazy a mutace. V deklaraci jsou využity objekty z balíčku *Dto* (data transfer object) z business vrstvy, které obsahují specifikaci přijímaných dat a validačních pravidel. Tyto celé objekty jsou předávány ve volání specifických metod z business vrstvy a výsledek se následně vrací jako odpověď na požadavek.

3.1.2.2 Business vrstva

Business vrstva je, co se do počtu balíčků týče, nejobsáhlejší a implementuje logiku zpracování dat a kontroly přístupu.

Hlavní balíček má název *Service* a obsahuje třídy, jenž jsou zodpovědné právě za implementaci business logiky. Reagují na podněty z prezentační vrstvy,

zpracovávají data, volají metody na kontrolu práv, v případě problémů vyhazují výjimky a finálně volají funkce datové vrstvy na manipulaci s daty v databázi.

Voter skýtá třídy definující pravidla, kdo a za jakých podmínek může provádět určité akce a manipulovat s daty.

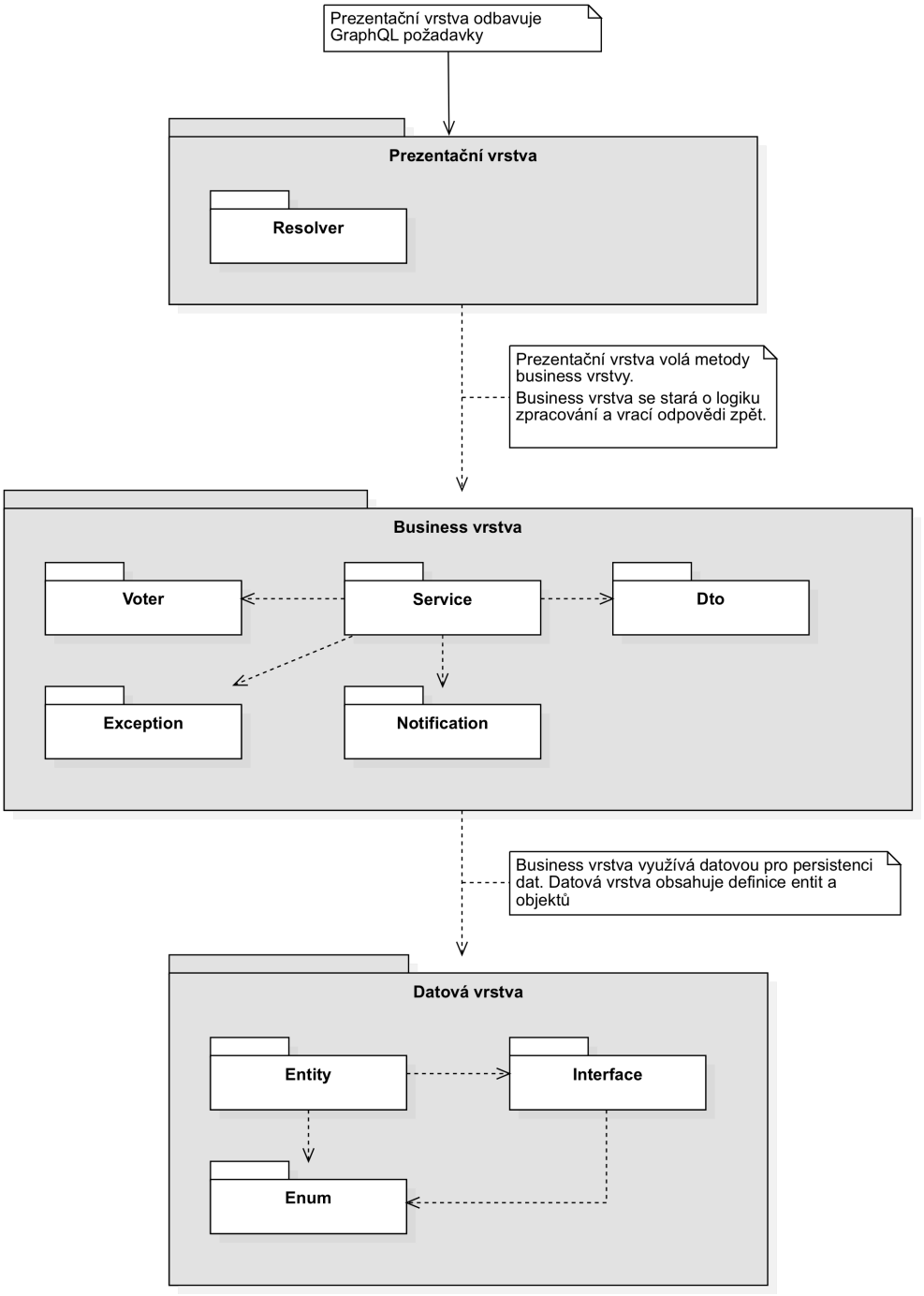
Dto obsahuje objekty používané k přenosu dat – dto je zkratkou pro data transfer object (objekty pro přenos dat). Ty se využívají již při externí komunikaci prezentační vrstvy a klientské aplikace. Kromě definice vlastností slouží i k definování jednoduchých validačních pravidel (délka řetězce, povinnost vyplnění) pomocí dekorátorů. O samotnou validaci se pak stará validační *pipa* popsaná v kapitole 3.1.1.3 týkající se frameworku NestJS.

Balíček *Notification* obsahuje pure funkce (čisté funkce) na generování těl notifikací. Takto pospolu je následně jednoduché dělat potenciální úpravy struktury upozornění.

Exception obsahuje obdobně jako *Notification* pouze pure funkce na generování výjimek, aby byly lehce přepoužitelné a v případě potřeby změny, aby veškeré definice byly na jednom místě.

3.1.2.3 Datová vrstva

Datová vrstva obsahuje definice entit v balíčku *Entity*, rozhraní entit v *Interface* a enumerace, například pro stavy či typy, v balíčku *Enum*. Vyčlenění rozhraní je z toho důvodu, že v samotných třídách entit jsou i dekorátory pro GraphQL komunikaci a definice pro uložení do databáze, a soubory tak narůstají na velikosti.



Obrázek 3.2: Návrh architektury

3.1.3 Diagram entitních tříd

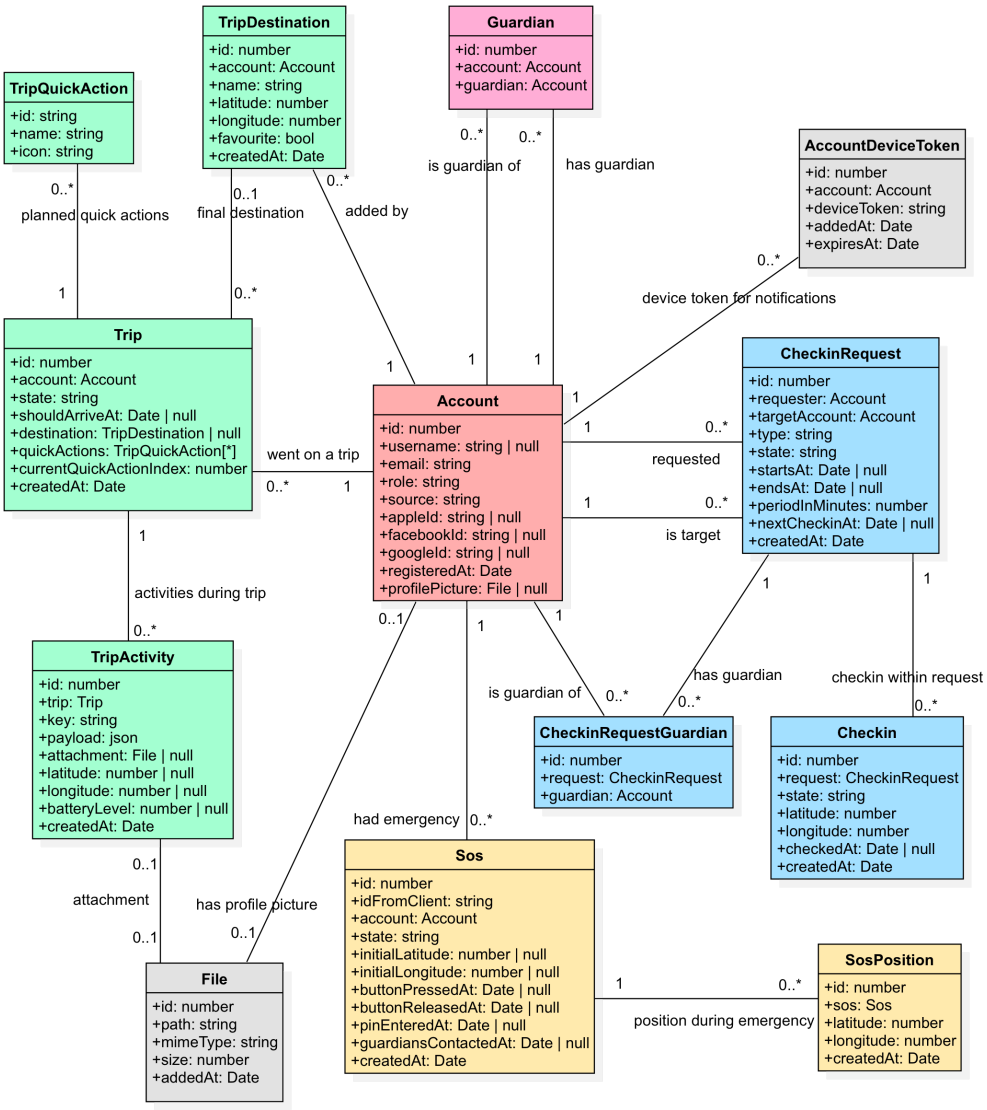
Na obrázku 3.3 lze vidět diagram entitních tříd (barevně odlišeny jednotlivé funkcionality). Základní entitou je *Account* reprezentující uživatelský účet, u kterého se evidují informace o emailu, uživatelském jméně, identifikátory na propojení se sociálními sítěmi či profilový obrázek.

Na vytvoření nouzových kontaktů slouží entita *Guardian*. *AccountDeviceToken* obsahuje údaj o identifikátoru fyzického zařízení pro posílání notifikací. *File* je entitou reprezentující soubor.

Pro funkcionalitu **SOS** tlačítka existují dvě entity s názvy *Sos* a *SosPosition*. První reprezentuje samotné stisknutí a stav, ve kterém se nachází. Slouží pro uchování informací o čase zmáčknutí, puštění, zadání PIN kódu či upozornění nouzových kontaktů. Druhá entita zastupuje informaci o uživatelově poloze v čase.

Checkin je reprezentován třemi entitami. *CheckinRequest* představuje samotné vytvoření požadavku na přihlášení a obsahuje informaci o typu (jednorázový, pravidelný) a pro pravidelný i údaje o začátku, konci a intervalu. *CheckinRequestGuardian* je vazbou mezi požadavkem a uživatelským účtem a funguje obdobně jako entita *Guardian*, reprezentuje tedy ochránce, kteří jsou informování ohledně stavu checkinu. Finální entitou je *Checkin*, jenž je záznamem ohledně jednoho konkrétního požadavku na přihlášení a skýtá informaci o stavu, zdali uživatel zadal, že je v pořádku a případně i lokaci, ve které požadavek potvrdil.

Pro funkcionalitu týkající se cest (**Trip**) jsou nadefinovány čtyři entity. *Trip* zastupující jednu cestu a obsahující informaci o předpokládaném čase příchodu, cílovou destinaci a předběžný plán cesty. Pro práci s místy je k dispozici entita *TripDestination* a pro jednotlivý krok plánu cesty slouží entita *TripQuickAction*. Dílčí aktivita finální cesty je reprezentována entitou *TripActivity*, která obsahuje informaci o typu, poloze, stavu baterie a následně dodatečné informace specifické pro jednotlivé typy aktivit.



Obrázek 3.3: Diagram entitních tříd

3.2 Realizace

V této kapitole rozebírám, jakým stylem je finální implementovaná serverová část rozdělena a jakým způsobem jsou realizovány důležité části systému. Není zde kompletní popis či dokumentace API rozhraní, neboť pomocí nástrojů Apollo GraphQL se striktně typované schéma samo propaguje napříč platformami.

3.2.1 Moduly

Serverová část se skládá z celkem 12 modulů, z nichž 2 nejsou aktuálně využívané (modul na odesílání mailů a modul na správu událostí – návrhový vzor publisher-subscriber). Zbylých deset vykonává následující funkce:

- **Account** – správa uživatelů,
- **Auth** – autentizace/autorizace,
- **Checkin** – funkcionalita ohledně checkinů,
- **Firebase** – napojení na Firebase knihovnu,
- **Guardian** – funkcionalita ohledně nouzových kontaktů,
- **Notification** – odesílání notifikací,
- **Sos** – funkcionalita ohledně kritických událostí,
- **Storage** – správa souborů, komunikace s AWS S3,
- **Trip** – funkcionalita ohledně cest,
- **Voter** – kontrola přístupu.

3.2.2 Konfigurace

Pro nakonfigurování určitých částí systému je vytvořena třída *ConfigService*, jež obsahuje veřejné metody, které vracejí objekty s příslušnými parametry. Je využito proměnných prostředí (environment variables), které jsou v některých případech nepovinné a mají tedy nadefinovanou výchozí hodnotu. Ukázka je dostupná na zdrojovém kódu 3.2.

```
1  class ConfigService {
2    constructor(private env: { [k: string]: string | undefined }) {}
3
4    private getValue(key: string, throwOnMissing = true): string {
5      const value = this.env[key];
6
7      if (!value && throwOnMissing) {
8        throw new Error(`config error - missing env.${key}`);
9      }
10
11     return value;
12   }
```

```
14    public getS3() {
15      return {
16        accessKeyId: this.getValue('S3_ACCESS_KEY_ID', true),
17        secretAccessKey: this.getValue('S3_SECRET_ACCESS_KEY', true),
18        region: this.getValue('S3_REGION', true),
19        bucket: this.getValue('S3_BUCKET', true),
20        version: this.getValue('S3_VERSION', true),
21        prefix: this.getValue('S3_PREFIX', false) || '',
22      };
23    }
24  }

26  const configService = new ConfigService(process.env).ensureValues(['MODE', '
        ↪ JWT_SECRET', 'DATABASE_URL']);

28  export { configService };
```

Zdrojový kód 3.2: Ukázka ConfigService

3.2.3 Komunikace

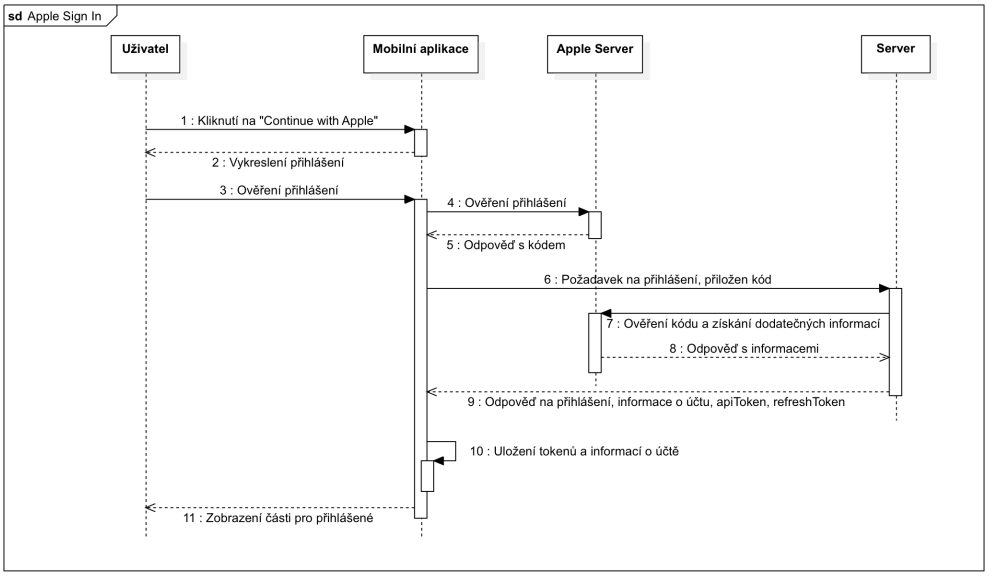
Jak již bylo zmíněno, komunikace probíhá pomocí dotazovacího jazyku GraphQL. V NestJS toho lze docílit nainstalováním balíčku *@nestjs/graphql* a následným zaregistrováním tohoto externího modulu pomocí naimportování do systému, nastavení určitých parametrů a vložením dekorátorů k příslušným metodám, které mají požadavky odbavovat.

3.2.3.1 Autentizace a přihlášení přes externí služby

Přihlášení přes všechny tři externí služby (Apple, Google, Facebook) bylo řešeno obdobným způsobem. Mobilní aplikace prezentuje uživateli přihlašovací okno, a pokud je výsledek úspěšný, pošle na server unikátní token/kód vygenerovaný externí službou, který je následně využit k získání údajů dalším dotazem, již však na straně backendu. V odpovědi tohoto požadavku jsou důležité informace jako email či identifikátor uživatele. Na serveru se vytvoří entita reprezentující uživatelův účet, uloží se do databáze a jako odpověď mobilní aplikaci se odešlou tokeny, jež se využívají v následující komunikaci pro autorizaci.

Pro dotazy na Apple a Google jsou využity přímo knihovny těchto poskytovatelů a v případě služby Facebook je získání údajů řešeno dotazem na konkrétní endpoint.

Další z možností, jak řešit autentizaci, by bylo využití jedné z autentizačních autorit, například Firebase Authentication[61]. Ty se starají o celý proces registrace, neboť nabízí komponenty do klientských částí i následné knihovny pro komunikaci v rámci backendu. Většinou nabízejí podporu pro velké množství



Obrázek 3.4: Průběh přihlášení přes Apple

přihlašovacích služeb, a tudíž přidání další možnosti je velmi jednoduché. Uživatelská data jsou v tomto případě však uložena na jejich serverech a vzniká zde závislost v kritické části aplikace, což je hlavním důvodem, proč toto řešení nebylo zvoleno.

Ačkoliv je po technické stránce vše připravené, v době odevzdávání této práce není přihlášení přes Google a Facebook v provozu, neboť je potřeba nejdříve vyřešit verifikaci účtu a dodání dokumentů jako například zpracování osobních údajů.

Diagram průběhu přihlášení lze vidět na obrázku 3.4.

3.2.4 Autorizace

Po přihlášení a získání autorizačních tokenů následně mobilní aplikace první z nich (apiToken) odesílá v hlavičce každého požadavku. Server tuto hlavičku zpracuje a příslušné metodě dodá i konkrétní entitu uživatele, jež je následně využita pro kontrolu práv. Token má určitou dobu životnosti čítající několik jednotek hodin. V případě vypršení si musí aplikace vyžádat obnovení tokenů pomocí druhého (refreshToken). Druhý token má taktéž pevně nastavenou dobu platnosti, která je ovšem násobně delší (v desítkách dnů). Pokud není ani refreshToken platný, uživatel je vyzván, aby se znovu přihlásil.

3.2.5 Kontrola práv – Voters

O kontrolu práv, jak již bylo zmíněno, se stará modul Voter. Celý princip spočívá v globálně dostupné službě s názvem VoterService, která nabízí dvě veřejné metody. První funguje na zaregistrování voterů (hlasujících, obsluhovačů). Ty jsou implementovány v konkrétních modulech a starají se o rozhodování nad právy týkajících se daného modulu. V rámci modulu Voter je definováno rozhraní, které musejí jednotlivé votery implementovat. Hlavní VoterService má následně uloženo pole voterů, které při dotazu prochází. Samotný voter nejdříve implementuje jednoduchou funkci, která rozhodne, jestli může dotaz obsloužit a poté ho případně obslouží. Ukázku voteru lze pozorovat na zdrojovém kódu 3.3 a VoterService na zdrojovém kódu 3.4.

```
1  export class GuardianVoter extends IVoter {
2    async supports(attribute: string): Promise<boolean> {
3      return (<any>Object).values(GuardianAttributes).includes(attribute);
4    }

6    async voteOnAttribute(attribute: string, subject: any, account: IAccount):
         ↪ Promise<boolean> {
7      switch (attribute) {
8        case GuardianAttributes.REMOVE_GUARDIAN:
9          return this.removeGuardian(account, subject);
10       case GuardianAttributes.ACCEPT_GUARDIAN:
11         return this.acceptGuardian(account, subject);
12     }

14     return false;
15   }

17   async removeGuardian(account: IAccount, guardian: IGuardian): Promise<
         ↪ boolean> {
18     return account.id === guardian.account.id;
19   }

21   async acceptGuardian(account: IAccount, guardian: IGuardian): Promise<
         ↪ boolean> {
22     return account.id === guardian.guardian.id;
23   }
24 }
```

Zdrojový kód 3.3: Ukázka GuardianVoter

```
1  @Injectable()
2  export class VoterService {
3    private voters: IVoter[] = [];

5    async isGranted(attribute: string, subject: any, account: IAccount): Promise<
         ↪ boolean> {
```

```
6       for (const voter of this.voters) {
7         if (await voter.isGranted(attribute, subject, account)) {
8           return true;
9         }
10      }

12      return false;
13    }

15    register(voter: IVoter): void {
16      this.voters.push(voter);
17    }
18  }
```

Zdrojový kód 3.4: Ukázka VoterService

3.2.6 Nastavení externích služeb

V rámci celého systému se užívá řada externích služeb, které bylo potřeba nakonfigurovat a připravit pro použití. Jak již bylo zmíněno v kapitole 3.2.3.1 Autentizace a přihlášení přes externí služby, je rozdíl mezi přípravou služeb pro testování a pro produkční použití, neboť mnohdy je potřeba náročnějšího nastavení a verifikace, obvzláště jedná-li se o manipulaci s daty.

Příprava Firebase a Amazon web services S3 proběhla velmi hladce a zde jsem založil účty přes svojí fyzickou osobu. U přihlášení přes sociální sítě (tedy přes Google a Facebook) nastal problém při přechodu do ostrého provozu, kdy je potřeba účet verifikovat a dodat podmínky použití a z tohoto důvodu se nepodařilo do odevzdání diplomové práce služby zpřístupnit široké veřejnosti.

S přihlášením přes účet Apple nebyl žádný problém, neboť vývojem právě pro platformu od Apple a s předem definovaným cílem nasazení pro běžné uživatele nebylo tyto věci nutno řešit specificky pro přihlášení. Samotný proces nasazení je popsán později v kapitole týkající se mobilní aplikace, konkrétně v sekci 4.8 Technická příprava testování a nasazení.

3.2.7 Klíčová funkcionalita

V následujících několik sekcích popíši, jakým způsobem byla implementována logika klíčové funkcionality, jaké problémy během implementace nastaly a způsoby, kterými byly vyřešeny.

3.2.7.1 Nouzové kontakty

Přidávání nouzového kontaktu funguje přes vyplnění uživatelského jména, kdy druhá osoba musí následně požadavek na přidání odsouhlasit. Na správu nouzových kontaktů je připraveno 5 různých koncových bodů:

- *getMyGuardians* – vypsání mých nouzových kontaktů,
- *getGuardianRequests* – vypsání požadavků na přidání nouzových kontaktů,
- *addGuardian* – poslání požadavku na přidání,
- *acceptGuardian* – přijetí požadavku na přidání,
- *removeGuardian* – odstranění nouzového kontaktu.

Nejdříve bylo implementováno pouze jednosměrné přidání kontaktů – tedy přidal-li jsem si já nouzový kontakt, nebyl jsem automaticky já jeho ochráncem. Avšak v kódu i logice zpracování následně docházelo k velkému zmatku, a tak nakonec toto chování bylo změněno na obousměrné – při přijetí požadavku se tak vytvoří druhá entita s prohozeným pořadím.

3.2.7.2 SOS tlačítko

Princip, jenž je z části vidět na stavovém diagramu na obrázku 3.5, této funkcionality spočívá v tom, že uživatel zmáčkne tlačítko a během jeho držení aplikace odesílá pravidelně informaci o uživatelově poloze. Následně po puštění tlačítka uživatel buď do určitého časového horizontu zadá svůj PIN, nebo server odešle notifikaci nouzovým kontaktům. Pro obsloužení jsou vytvořeny čtyři koncové body:

- *sosButtonPressed* – uživatel stiskl tlačítko,
- *sosButtonReleased* – uživatel pustil tlačítko,
- *sosPinEntered* – uživatel zadal svůj PIN,
- *sosPositionUpdate* – aktualizace polohy během držení.

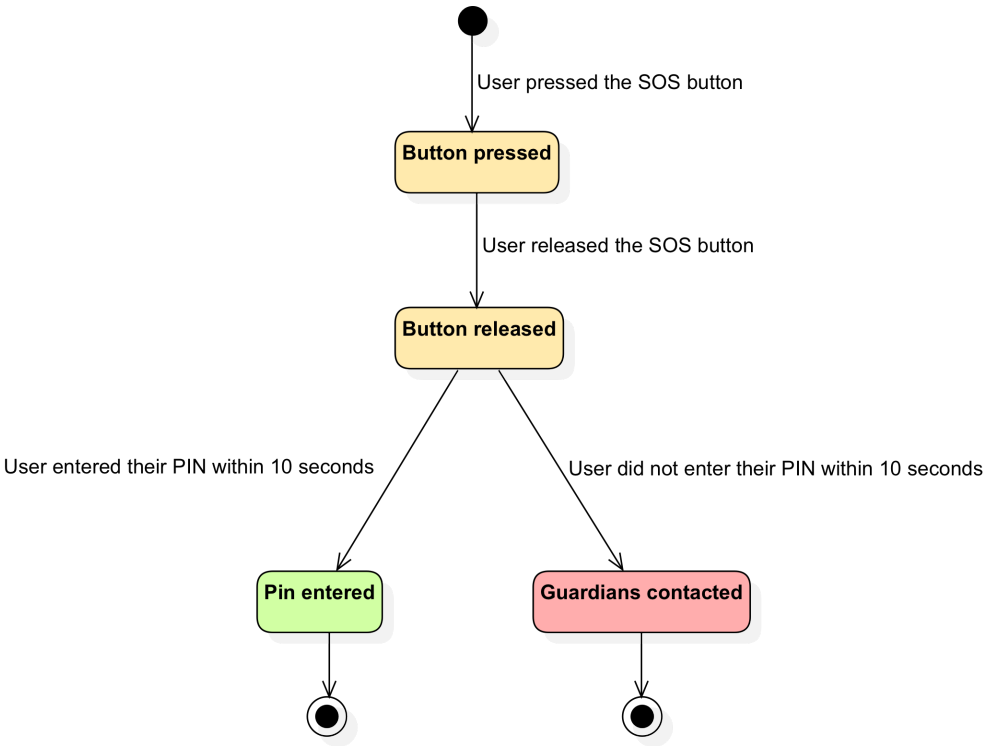
Pro zobrazení historie těchto mimořádných událostí jsou definovány tyto endpointy:

- *getMySosHistory* – výpis mých nouzových situací,
- *getSosesOfAccountsIAmGuardianOf* – výpis nouzových situací mých kontaktů,
- *getSosPositions* – výpis zaznamenaných poloh během držení tlačítka.

3.2.7.3 Checkin

Pro manipulaci s checkinem jsou vytvořeny celkem čtyři koncové body. Dva na vyžádání rozdělené podle typu, jeden na zrušení pravidelného a jeden na potvrzení.

- *requestCheckin* – vyžádání jednorázového přihlášení,
- *requestPeriodicalCheckin* – vyžádání pravidelného přihlášení,
- *cancelPeriodicalRequest* – zrušení probíhajícího pravidelného checkinu,
- *checkin* – potvrzení konkrétního checkinu.



Obrázek 3.5: Stavový diagram SOS

Samotný požadavek na checkin vytváří pouze záznam v databázi s příslušnými informacemi a o poslání požadavku konkrétnímu uživateli se stará až naplánovaná úloha, která se spouští každých 10 vteřin. Entita *CheckinRequest* má atribut udávající, kdy má být vyslán další požadavek (*nextCheckinAt*). Tato naplánovaná úloha tedy odbavuje záznamy, které mají tuto hodnotu v minulosti. V případě pravidelného checkinu rovnou uloží, kdy má dojít k dalšímu.

```
1  @Mutation(() => Boolean)
2  async requestPeriodicalCheckin(
3    @GqlAccount() account: IAccount,
4    @Args('periodicalCheckinRequest') checkinRequestPeriodicalDto:
         ↪ CheckinRequestPeriodicalDto,
5  ) {
6    await this.checkinService.requestPeriodicalCheckin({ account,
         ↪ checkinRequestPeriodicalDto });

8    return true;
9  }
```

Zdrojový kód 3.5: Ukázka zpracování požadavku na pravidelný checkin

```
@InputType()
export class CheckinRequestPeriodicalDto {
  @Field(() => ID)
  @IsNotEmpty()
  readonly targetAccountId: number;

  @Field()
  @IsNotEmpty()
  readonly startsAt: Date;

  @Field()
  @IsNotEmpty()
  readonly endsAt: Date;

  @Field()
  @IsNotEmpty()
  readonly periodInMinutes: number;

  @Field(() => [ID])
  @IsNotEmpty()
  readonly guardianAccountIds: number[];
}
```

Zdrojový kód 3.6: Ukázka DTO pravidelného checkinu

```
@Cron(CronExpression.EVERY_10_SECONDS)
async createCheckins(): Promise<void> {
  const requests = await this.checkinRequestRepository.find({
    relations: ['targetAccount', 'requester'],
    where: {
      nextCheckinAt: LessThan(new Date()),
    },
  });

  for (const request of requests) {
    await this.createCheckin({ request });
  }
}
```

Zdrojový kód 3.7: Ukázka plánované úlohy

Kromě manipulačních koncových bodů je vytvořeno 5 dalších na výpis a získávání informací:

- *getRequestsForMe* – výpis požadavků, kde jsem já byl požádán o přihlášení,
- *getRequestsIFollow* – výpis požadavků, ve kterých figuruji jako kontakt,
- *getRequest* – detail o konkrétním požadavku,

- *getCheckinsByRequest* – výpis jednotlivých dílčích požadavků o přihlášení (vztah 1:N u *CheckinRequest* a *Checkin*),
- *getUncheckedCheckinsForMe* – výpis požadavků, které jsem ještě nepotvrdil.

3.2.7.4 Trip

Finální popisovanou funkcionalitou, avšak pravděpodobně nejsložitější, je zaznamenávání průběhu cesty. Princip spočívá v zahájení cesty, kdy uživatel může zvolit čas příchodu a cílovou destinaci. Zároveň si může naplánovat průběh cesty pomocí rychlých akcí, kterými jsou například čekání, jízda tramvají, přestup či chůze pěšky. Po zahájení může volit z těchto přednastavených rychlých akcí, případně vybírat ze zbylých a také může přidávat zprávy a fotky. Pro vytváření tohoto průběhu cesty existuje 5 manipulačních koncových bodů:

- *createTrip* – vytvoření cesty,
- *endTrip* – ukončení cesty,
- *addMessageToTrip* – přidání zprávy k cestě,
- *addPhotoToTrip* – přidání fotky k cestě,
- *addQuickActionToTrip* – přidání rychlé akce k cestě.

V požadavcích na poslední tři koncové body jsou taktéž přítomny informace o uživatelově poloze a stavu baterie jeho zařízení. Pro zobrazení historie cest existuje následujících 5 endpointů:

- *getActiveTrip* – získání aktuálně aktivní cesty (je-li nějaká),
- *getMyTrips* – historie cest,
- *getTrip* – detail konkrétní cesty,
- *getMyGuardiansTrips* – historie cest mých kontaktů,
- *getActivitiesByTrip* – výpis aktivit konkrétní cesty.

V rámci této funkcionality existují ještě 2 finální koncové body:

- *getMyTripDestinations* – výpis historie cílových destinací,
- *toggleFavouriteTripDestination* – změna stavu oblíbení destinace.

Pro uživatele neexistuje komplexnější funkcionalita na správu destinací. Koncový bod na zahájení cesty přijímá buď identifikátor existující destinace, či objekt na vytvoření nové. Mazání není aktuálně dostupné, neboť výpis funguje jako historie.

Kapitola 4

Mobilní aplikace

V této kapitole se věnuji návrhu a realizaci mobilní aplikace. Nejdříve se zabývám rozvržením a konceptem, následně výběrem technologií a finálně samotnou implementací.

4.1 Low fidelity vs high fidelity prototyp[13, 14]

„Výraz 'prototyp' se často používá v různých kontextech. Z tohoto důvodu může docházet k nejasnostem ohledně jeho významu.

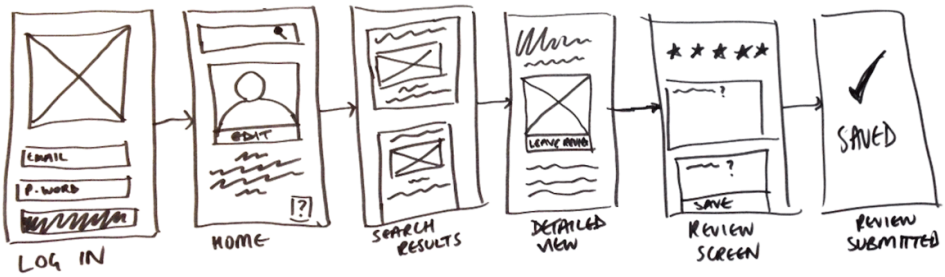
V jeho základní podobě je prototyp vyjádřením záměru designu. Prototypování umožňuje designérům prezentovat své návrhy a vidět je v reálné podobě. V kontextu digitálních produktů je prototyp simulací konečné interakce mezi uživatelem a rozhraním. V závislosti na tom, k čemu vývojový tým prototyp potřebuje, může simulovat celou aplikaci nebo pouze jednu konkrétní interakci.“[14] (přeloženo autorem)

Přesnost (fidelity) se vztahuje k míře detailů a propracovanosti prototypu. Návrhy se mohou lišit obsahem, interaktivitou či po vizuální stránce. V následujících podsekcích rozeberu dva hlavní způsoby a poté vyberu jakým způsobem budu navrhovat vyvíjenou aplikace.

4.1.1 Low fidelity prototyp

Low fidelity prototyp (lo-fi prototyp, prototyp s nízkou přesností) je hrubý nástin či rychlý jednoduchý funkční model. Může mít podobu náčrtu na papír stejně jako drátěného modelu (wireframe) vytvořeného v některém z dostupných nástrojů. Využívá obrazců nikoliv konkrétních prvků, neobsahuje konkrétní barvy, texty a data. Takovéto modely usnadňují komunikování myšlenek

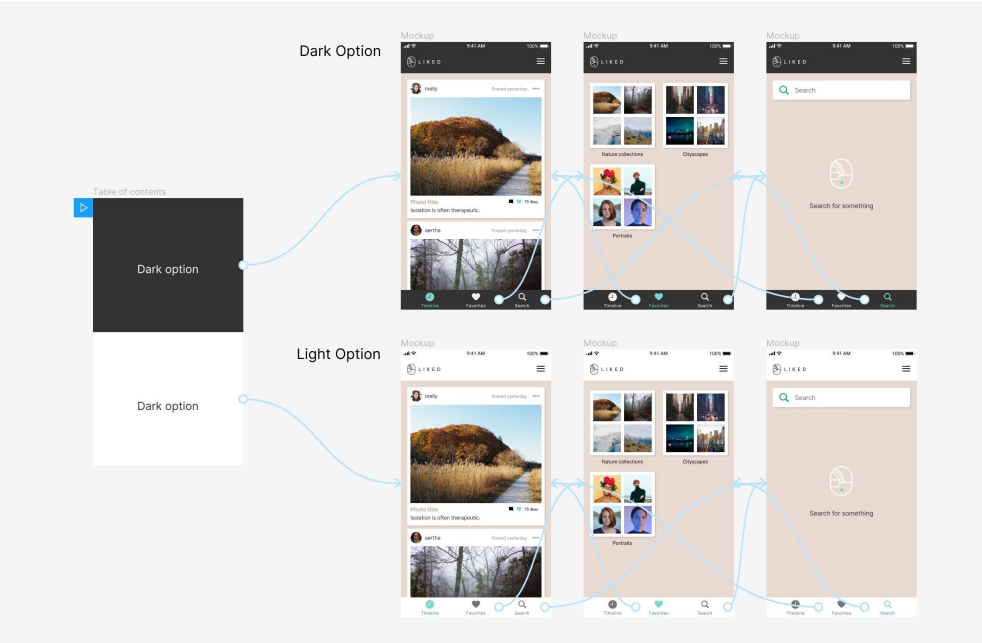
pomocí abstraktních elementů a slouží hlavně k nastínění rozložení. Příklad lo-fi wireframů je na obrázku 4.1.



Obrázek 4.1: Wireframe – ukázka[18]

4.1.2 High fidelity prototyp

High fidelity prototyp (hi-fi prototyp, prototyp s vysokou přesností) obsahuje více reálného obsahu oproti lo-fi modelu – například konkrétní typografie, uvedení rozlišení obrázků, použití skutečných obrázků nebo propracované a realistické interakce. Příklad hi-fi prototypu je na obrázku 4.2.



Obrázek 4.2: Figma Prototyping – ukázka[19]

4.1.3 Výběr pro návrh Safie

Pro tvorbu aplikace v rámci diplomové práce jsem se rozhodl vytvořit low fidelity wireframy, které budou sloužit pro nastínění rozhraní. Pár obrazovek následně graficky vylepším, aby byl patrný nádech, kterým se má aplikace vydávat. Absence klientské strany je důvodem nevytvoření pokročilého high fidelity klikatelného prototypu, jenž by byl primárně využit na získávání zpětné vazby. Taktéž uživatelské testování je plánováno až na funkční verzi aplikace.

4.2 Členění obsahu a wireframy

Před samotným zhotovením návrhu je potřeba si uvědomit jaké obrazovky a s jakou funkcionalitou bude aplikace skýtat a jakým stylem budou propojeny. V této sekci se podívám na propojení obrazovek – přehled lze vidět na obrázku 4.3. Vertikální obdélník značí stránku a šipka možnost přechodu (obousměrného pomocí tlačítek na vrácení).

4.3 Wireframy

Na vytvoření wireframů jsem využil grafického editoru Sketch[62] a jak již bylo zmíněno v kapitole 4.1.1, jedná se o hrubé nastínění rozložení prvků bez vizuálních detailů. Zhotovil jsem 4 obrazovky – konkrétně hlavní stránku, profil, žádost o checkin a probíhající cestu – viditelné na obrázku 4.4.

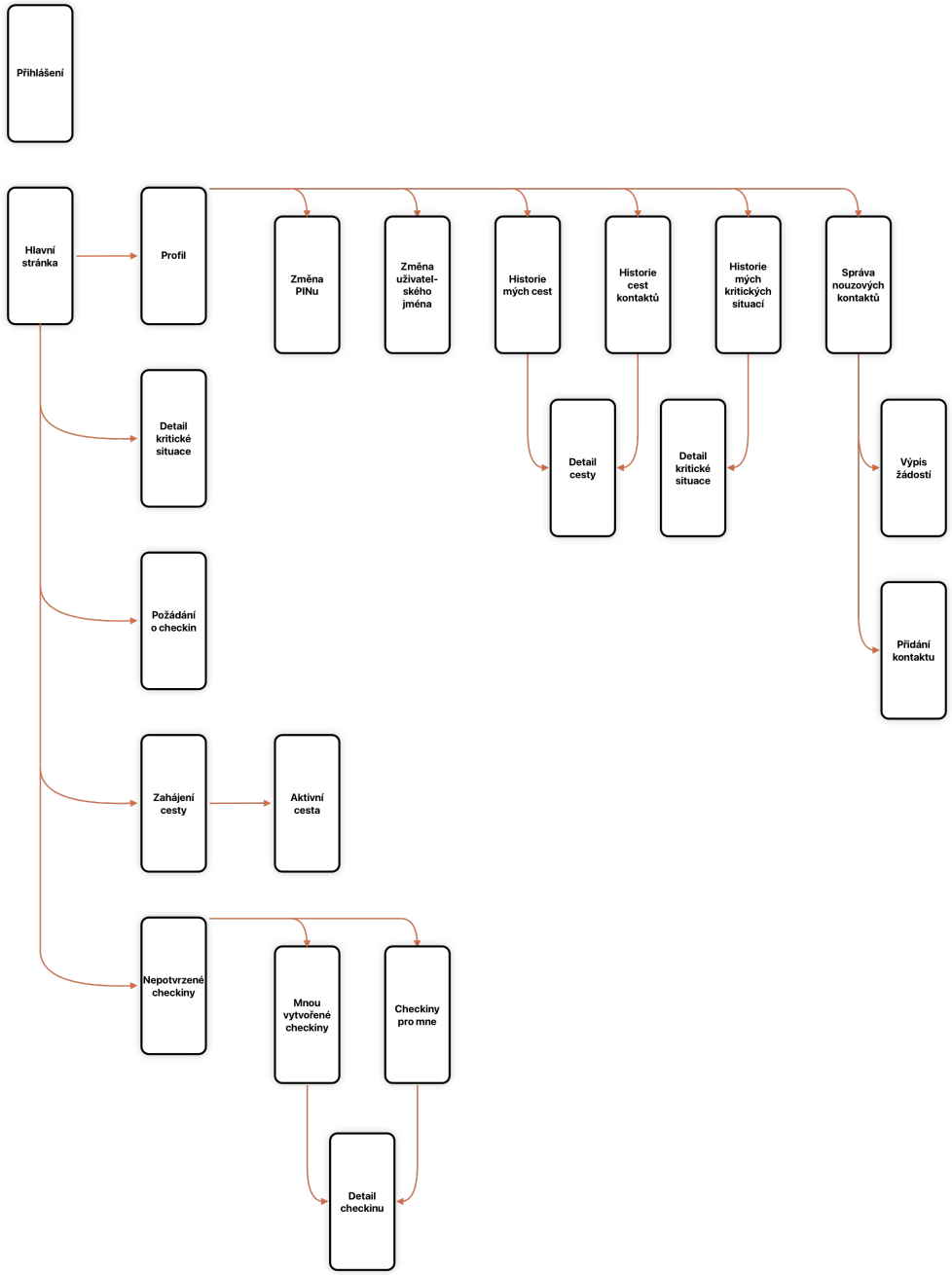
V horní části **hlavní stránky** se nachází profilová fotografie s uživatelským jménem sloužící jako proklik na profil. Níže lze vidět horizontální seznam posledních kritických situací uživatelových kontaktů pro rychlý přístup. Uprostřed obrazovky je velké SOS tlačítko a ve spodní části je menu s dalšími prokliky.

Profilová obrazovka sestává z velké fotografie a uživatelského jména. Následuje seznam kontaktů, přes který se lze prokliknout na jejich správu. Dále je zde seznam odkazů a nakonec tlačítko na odhlášení.

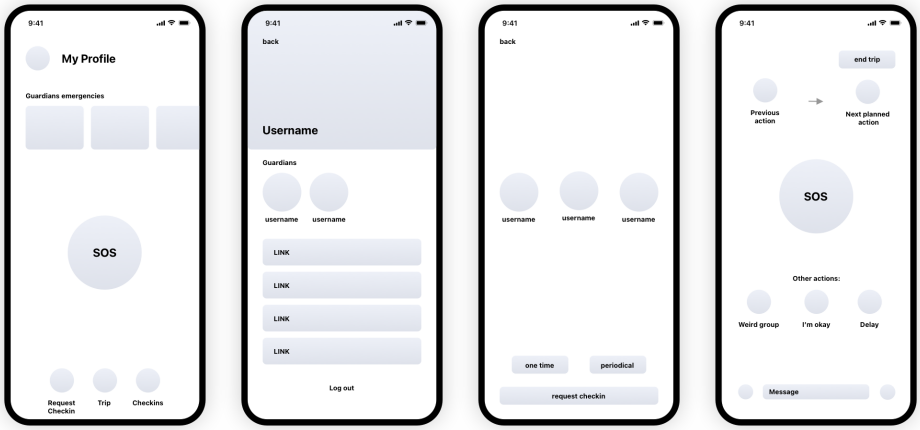
Žádost o checkin obsahuje seznam kontaktů v graficky jednoduché podobě, možnost volby jednorázového či pravidelného a následně tlačítko na odeslání.

Obrazovka aktivní cesty má v pravém horním rohu tlačítko na ukončení, poté poslední zvolenou akci s šipkou směřující na další naplánovanou akci. Uprostřed obrazovky je opět k dispozici velké SOS tlačítko, pod kterým je seznam ostatních rychlých akcí. Finálně má uživatel ve spodní části možnost přidat fotografii či napsat zprávu.

Pro další obrazovky nebyl návrh dělán, neboť šlo hlavně o nastínění rázu, a jelikož aplikaci vyvíjím sám nebylo potřeba předávat detailní ideu někomu dalšímu.



Obrázek 4.3: Organizace obsahu



Obrázek 4.4: Wireframy vyvíjené aplikace

4.4 Návrh designu

Návrh designu jsem opět provedl ve studiu Sketch[62] a zhotovil jsem koncept dvou obrazovek, loga a typů notifikací. Přehledně lze vidět na obrázku 4.5.

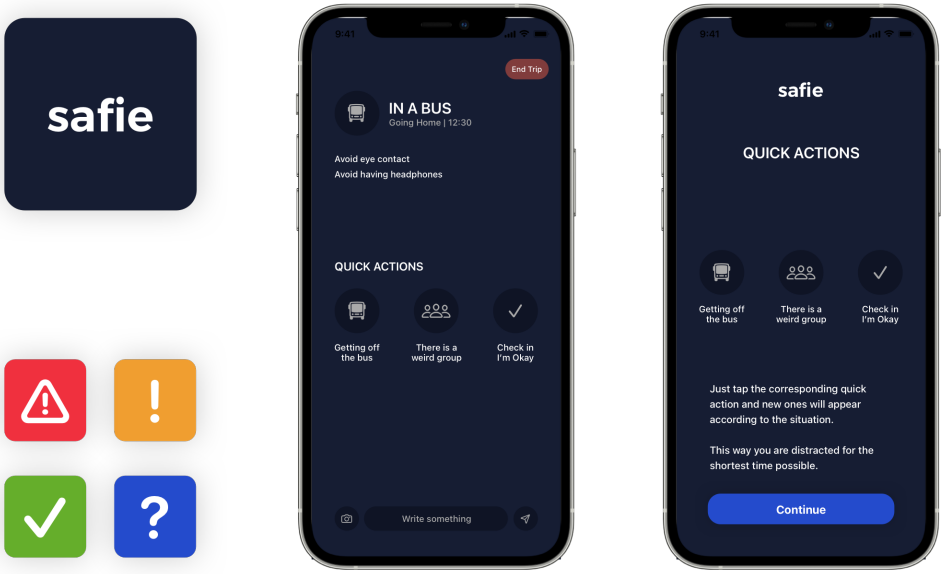
Logo je velmi jednoduché a skýtá pouze nápis „safie“ na tmavě modrém podkladu. První obrazovka znázorňuje aktivní cestu a lehce se liší od návrhu wireframu, neboť byla vytvořena dříve pro jednoduchou prezentaci idey. Konečný výsledek se tak od tohoto návrhu liší, což bude vidět během kapitoly 4.7 Realizace. Druhou obrazovkou je jeden z kroků takzvaného „onboardingu“ (pár úvodních obrazovek při prvním spuštění představujících aplikaci). Finálně lze pozorovat návrhy typu notifikací – tyto obrázky budou doplňovat notifikaci tak, aby uživatel na první pohled rozpoznal, jakou váhu má oznámení přiřadit.

4.5 Výběr technologií

V následujících sekcích se budu zabývat výběrem technologií pro vyvíjenou mobilní aplikaci.

4.5.1 Swift

„Swift by se dal stručně popsat jako programovací jazyk Apple světa. Uveden byl v roce 2014 jako náhrada dosavadního Objective-C. Po stránce návrhu se jedná o objektový, kompilovaný jazyk s typovou kontrolou. Poměrně blízko tak má k jazykům jako jsou Java, C# a jim podobným. Protože se jedná o relativně nový jazyk, tak si s sebou z minulosti nenese nejrůznější problémy a nabízí řadu moderních přístupů.“[63]



Obrázek 4.5: Náhled designu

4.5.2 SwiftUI

Pro psaní aplikací pro iOS je dnes možno využít dvou hlavních frameworků poskytovaných přímo od Apple – UIKit[64] a SwiftUI[65]. Oba mají své benefity stejně jako nedostatky a přináší trochu jinou filosofii nad procesem tvorby aplikace.

„Nejjednodušší způsob, jak si představit rozdíly mezi těmito dvěma frameworky, je, že Apple je vytvořil se záměrem pro snadné použití na základě typu programátora. Programátor orientovaný na vizuální charakter by preferoval UIKit, zatímco orientovaný na kód by upřednostnil SwiftUI. UIKit dává možnost vytvářet uživatelská rozhraní i bez silného programovacího pozadí. Důvodem je zavedení takzvaných storyboards – drag-and-drop řešení na vytváření uživatelských rozhraní. Tento přístup však přichází se svými nedostatky: vytváření uživatelského rozhraní pomocí kódu (bez storyboards) je v rámci UIKit výrazně obtížnější ve srovnání se SwiftUI. Posledním detailem stojícím za zmínku je, že UIKit je starší framework. To znamená, že časem můžeme vidět upadající podporu od Apple s tím, jak se SwiftUI bude vyvíjet dopředu.“[66] (přeloženo a zkráceno autorem)

Pro vývoj diplomové práce jsem tedy zvolil SwiftUI.

4.5.3 Apollo GraphQL Client[58]

Již v kapitole 3.1.1.5 ohledně výběru technologií pro serverovou část jsem zmiňoval, že Apollo GraphQL má nástroje pro práci s tímto dotazovací jazykem jak pro backendovou část, tak i pro klientskou stranu. Na straně klienta je potřeba přidat jednoduchý skript, který se stará o stažení schématu z určitého endpointu a následně vygenerování konkrétních tříd a metod pro použití v daném jazyce. Apollo GraphQL Client má podporu pro operační systémy Android a iOS a taktéž pro web v rámci jazyka JavaScript.

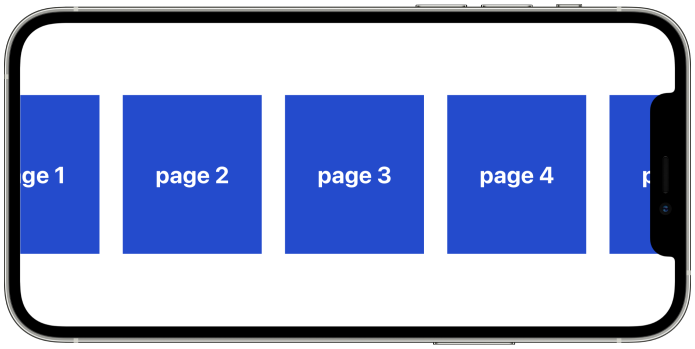
4.5.4 Lottie[67]

Lottie je knihovnou vyvíjenou americkou společností Airbnb pro vkládání vysoce kvalitních animací do mobilních aplikací. Vývojový tým má možnost exportovat animace pomocí rozšíření z programu After Effects od Adobe[68] či existují banky s animacemi vytvářenými komunitou. Jednou z nich je i LottieFiles[69], jejichž služby jsem v rámci realizace využil. Pro koncového uživatele vhodným použitím dochází ke zvýšení uživatelského zážitku.

4.5.5 Další knihovny a balíčky

SwiftUI ani UIKit nemají nativní podporu zobrazování obrázků z externích zdrojů. Pro toto použití se mi nejvíce osvědčila knihovna **Kingfisher**[70].

Knihovna **SwiftUIPager**[71] slouží k jednoduchému vytváření stránkovacích galerií (ukázka pro pochopení na obrázku 4.6) s pokročilými možnostmi konfigurace. Upravit jde například velikost stránek, rychlost animace či styl animace.



Obrázek 4.6: Ukázka stránkovací galerie

KeychainSwift[72] je soubor pomocných funkcí na práci s Keychain Services API – zabudovaná služba od Apple umožňující bezpečně ukládat malé kusy dat. Ve vyvíjené aplikace se hodí na ukládání tokenů ke komunikaci.

Firebase iOS SDK[73] slouží na práci se službami, které Firebase poskytuje. V rámci aplikace využito na notifikace.

Facebook iOS SDK[74] je využit pro přihlášení přes Facebook. Sada obsahuje další nástroje pro sdílení či analytiku, ty však využity nejsou.

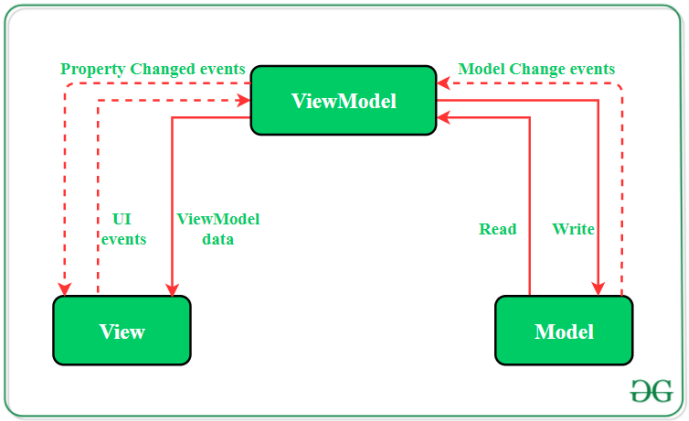
Google Sign In iOS[75] slouží k přihlášení přes Google.

4.6 Návrh architektury

„SwiftUI je deklarativní, stavem řízený framework. Nemůžeme se odkazovat na žádný pohled v hierarchii, ani nemůžeme přímo upravovat pohled jako reakci na událost. Místo toho měníme stav vázaný na pohled. Delegates, target-actions, responder chain, KVO — celá zoologická zahrada s technikami zpětného volání byla nahrazena closures a bindings.“[76] (přeloženo autorem, některé výrazy nechány pro jejich absenci v češtině či krkolomný překlad)

4.6.1 MVVM – Model-View-ViewModel

SwiftUI tedy přichází s implicitní architekturou MVVM (Model-View-ViewModel). Tento návrhový vzor se skládá ze tří komponent, každá s jinou zodpovědností. Pohled (View) se stará o vykreslení uživatelského rozhraní a zobrazení dat, informuje ViewModel o interakcích uživatele a pozoruje (observe) změny ve ViewModelu. Do pohledu nepatří žádná business logika. Model se stará o abstrakci datových zdrojů a ViewModel slouží k propojení pohledu a modelu a poskytuje pohledu přístup k datům. Vizuálně znázorněno na obrázku 4.7.



Obrázek 4.7: MVVM Schéma (Model-View-ViewModel)[20]

4.6.2 Singleton

V rámci celé aplikace se velmi často používá návrhový vzor jedináček (singleton). Ten omezuje inicializaci určité třídy pouze na jednu instanci používanou napříč systémem. Je uplatněn převážně pro modely, které jsou následně přístupné z vícero míst pro sdílení dat. Například věci týkající se uživatelovy historie jsou přístupné různými cestami a není potřeba je načítat zvlášť pro každé použití.

4.7 Realizace

V rámci této kapitoly se zaměřím na implementaci mobilní aplikace, jakým způsobem je strukturován celý projekt, jak byly řešeny klíčové funkcionality a jak proběhlo nasazení.

4.7.1 Souborová struktura

- `Shared` ..... soubory sdílené aplikací a rozšířeními
- `Safie Notifications` ..... rozšíření pro správu prezentování notifikací
- `Safie` ..... samotná mobilní aplikace
  - `Config` ..... konfigurace a parametry
  - `Localization` ..... soubory s překlady
  - `Gql` ..... dotazy, mutace a vygenerované třídy pomocí Apollo Client
  - `Models` ..... MVVM - Modely
  - `ViewModels` ..... MVVM - ViewModely
  - `Screens` ..... MVVM - Views - Obrazovky
    - `Sign`
    - `LoggedIn`
  - `Common` ..... obecné soubory
    - `Assets` ..... písma, Lottie animace
    - `Components` ..... komponenty
      - `UI`
      - `App`
    - `Extensions` ..... rozšíření existujících tříd
    - `Forms` ..... třídy pro validaci formulářů
    - `Structs` ..... struktury entit
    - `Styles` ..... styly
    - `Utils` ..... formátování dat a další jednoduché užitečné funkce

4.7.2 Nastavení Apollo GraphQL Client[58]

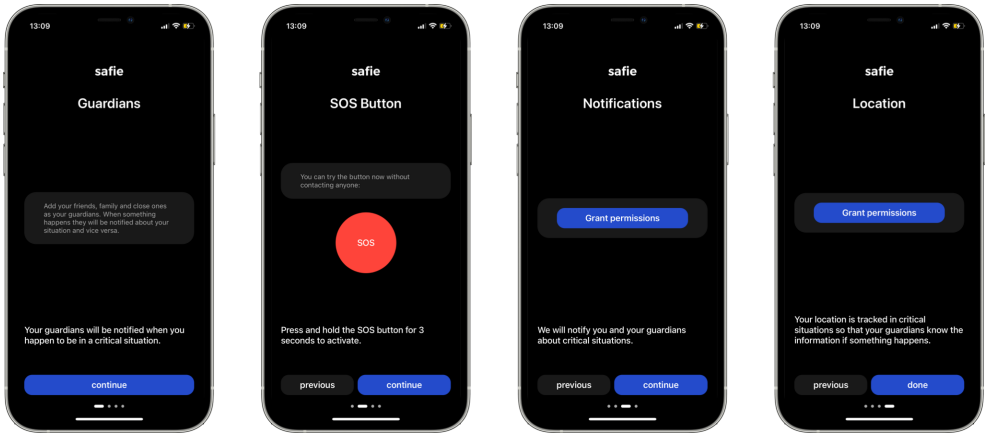
Pro nastavení GraphQL komunikace pomocí Apollo bylo potřeba přidat jednoduchý skript do jedné z prvních sestavovacích fází. Tento skript stáhne z určeného endpointu GraphQL schéma, projde složky s definicí dotazů a mu-

tací a finálně vygeneruje Swift třídy, které lze již jednoduše používat napříč aplikací.

Nadále bylo potřeba zajistit, aby do odeslaných požadavků byla vložena hlavička s api tokenem a v případě jeho vypršení aby došlo k obnovení pomocí refresh tokenu. Pokud je nevalidní i ten, je uživatel odhlášen. Apollo Client pro toto nabízí možnost vložení svého kódu do řetězce událostí, které nadcházejí odeslanému požadavku.

4.7.3 Onboarding

Onboarding slouží k rychlému seznámení uživatele s aplikací. V rámci něj je možné vysvětlit základní funkce a případně si vyžádat určitá práva například na posílání notifikací. Onboarding v rámci vyvíjené aplikace lze pozorovat na obrázku 4.8.



Obrázek 4.8: Onboarding – ukázka

4.7.4 Notifikace

Pro zprovoznění notifikací bylo nejdříve nutné vyřídit záležitosti na straně Firebase a Apple – založení účtů a generování klíčů. Poté bylo potřeba nastavit delegáta v rámci aplikace, který se stará o správu APN (Apple Push Notification) a Firebase tokenů sloužících pro adresování zařízení. Delegát je v kódu třída, která implementuje určité funkce na obsluhu jednotlivých záležitostí. Prvně je tedy potřeba získat APN token zařízení, který se předá Firebase a ten nám na oplátku přiřadí svůj token, který následně slouží k adresování notifikací ze serveru.

Poté bylo potřeba implementovat obsloužení příchodu notifikací, má-li uživatel aplikaci zapnutou. Pak také funkci, která se stará o obsloužení prokliku notifikace. V tomto případě je potřeba zanalyzovat obsah upozornění a zjistit,

zdali notifikace žádá zobrazení určité stránky – například notifikace ohledně žádosti o přidání kontaktů by měla otevřít stránku s takovými požadavky. Toho je dosaženo modelem, který je přístupný z metody pro obsluhu upozornění a zároveň z pohledu, který se stará o vykreslení obsahu pro přihlášené. Pohled je přihlášen ke změnám v modelu, na které reaguje zobrazením dané obrazovky.

Další částí je zprovoznění rozšíření *Notification Service Extension*[77], které dovoluje upravovat obsah před samotným zobrazením notifikace. V rámci této části kódu je implementován překlad obsahu notifikací (ačkoliv aktuálně je aplikace pouze v jazyce anglickém) a přidání obrázku typu notifikace (viz obrázek 4.5 v kapitole 4.4 Návrh designu).

4.7.5 Klíčová funkcionalita

4.7.5.1 SOS tlačítko

Funkcionalita SOS tlačítka je vyvolána jeho stiskem a následným držením po dobu minimálně tří sekund. Po uplynutí této doby je prezentována nová vrstva na stávající obsah pomocí *fullScreenCover*. Pro udržení informace o držení muselo být na gesto dlouhého držení navázáno gesto pohybu, neboť systém při ukázání nové vrstvy první gesto přerušil. Řešení je ukázáno na zdrojovém kódu 4.1.

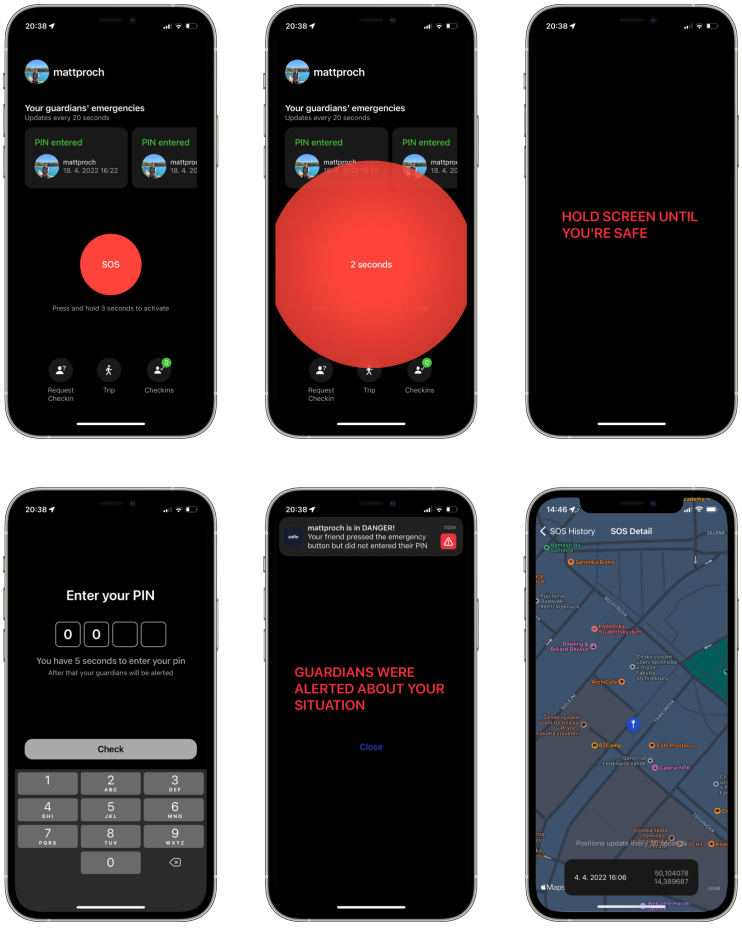
```
1  LongPressGesture(minimumDuration: 0.1)
2      .sequenced(before: DragGesture(minimumDistance: 0, coordinateSpace: .local))
3      .updating($pressingState) { value, state, transaction in
4          switch value {
5              case .second(true, nil):
6                  state = true
7              default:
8                  break
9          }
10     }
```

Zdrojový kód 4.1: SOS tlačítko – řešení pro udržení informace o gestu držení

Po uplynutí tří vteřin se systém přesune do stavu držení, během kterého pravidelně každých 5 sekund vysílá informaci o uživatelově poloze na server. V případě, že dotyčný obrazovku pustí, zobrazí se mu žádost o zadání jeho PINu, na kterou má 10 vteřin. Zadá-li svůj PIN vyšle se tato informace na server, který následně vyhodnotí, že vše je v pořádku a není potřeba upozornit kontakty. Nezadá-li ho však, na server se žádný požadavek neodešle a ten s absencí této informace do určité doby vyšle notifikaci uživatelově ochráncům.

Druhá osoba má možnost zobrazit si detail nouzové situace s konkrétními zaznamenanými body na mapě. Zde je implementováno i automatické obnovení

bodů po 30 sekundách tak, aby uživatel mohl mít tento detail stále otevřený. Informace o poloze se ukazují ve spodní části obrazovky v horizontálně posuvné galerii. Je zde údaj o času zaznamenání a souřadnice. Aktuálně zvolená poloha je i barevně vyznačena na mapě. Ukázky této funkcionality jsou vidět na obrázku 4.9.

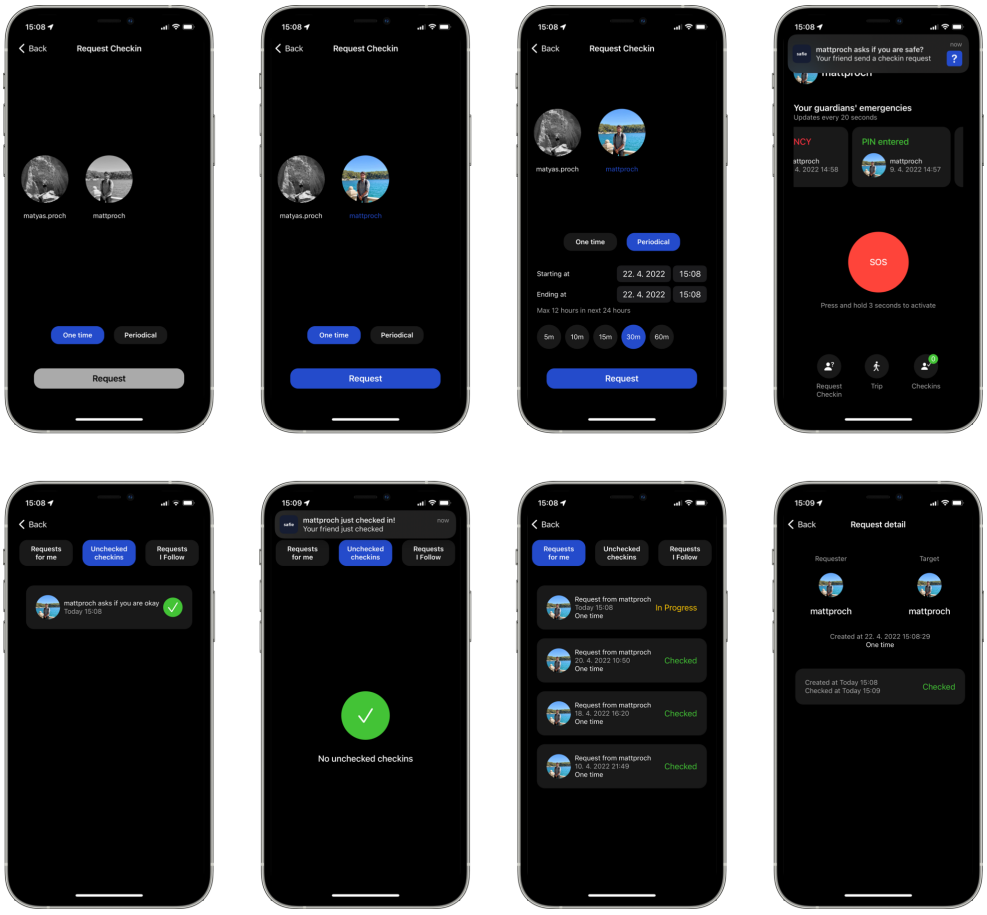


Obrázek 4.9: SOS tlačítko – ukázka

4.7.5.2 Checkin

Celý proces žádosti o checkin začíná výběrem jednoho z kontaktů (viz obrázek 4.10) a následným zvolením typu – jednorázový či pravidelný. V případě pravidelného se uživateli ukážou rozšířené možnosti nastavení – začátek a konec upozorňování a následně interval, u kterého má na výběr z pěti možností (5 až 60 minut). Po vyplnění stačí zmáčknout tlačítko na odeslání.

Uživateli, pro kterého byl checkin určen, přijde upozornění a v rámci aplikace se mu ve spodním menu na prokliku na checkiny zobrazí v červeném poli počet nevyřízených požadavků. Buď manuálně nebo klikem na notifikaci se dostane na seznam čekajících žádostí, kde může jednotlivé žádosti potvrdit. Má zde možnost zobrazit si i historii a detaily jednotlivých požadavků.



Obrázek 4.10: Checkin – ukázka

4.7.5.3 Trip

V rámci této sekce popíši, jakým způsobem funguje funkcionalita cesty. Výsledné obrazovky z aplikace jdou vidět na obrázku 4.11 na konci této části.

Zahájení cesty

Zahájení cesty se skládá z několik částí – výběr času příchodu, cílové destinace a naplánování cesty pomocí rychlých akcí. Všechny tři možnosti jsou pro uživatele nepovinné.

Výběr cílové destinace uživatele přenese na novou obrazovku, kde má k dispozici historii svých destinací, které si může pomocí dlouhého podržení dát do oblíbených. Taktéž může přidat novou destinaci, na což má dva způsoby, oba využívající knihovny MapKit od Apple[78].

První možností přidání je výběr bodu na mapě, který si lze následně pojmenovat. Druhým způsobem je vyhledání využívající třídy *MKLocalSearchCompleter* z knihovny MapKit. V tomto případě ViewModel starající se o přidání destinací funguje jako delegát této třídy implementující metodu viditelnou na zdrojovém kódu 4.2. Po zadání vyhledávacího dotazu uživatelem se zavolá funkce na získání míst a v případě úspěchu se pomocí zobrazené metody naplní výsledky pro zobrazení. Z těch si následně může uživatel vybrat.

```
1  func completerDidUpdateResults(_ completer: MKLocalSearchCompleter) {
2      var searchResults: [SearchLocation] = [];
3      var count = 0

5      for result in completer.results {
6          if count >= 6 { continue }

8          count += 1

10         searchResults.append(
11             SearchLocation(
12                 id: UUID().uuidString,
13                 title: result.title,
14                 subtitle: result.subtitle
15             )
16         )
17     }

19     self.searchResults = searchResults
20 }
```

Zdrojový kód 4.2: Implementace získání výsledků vyhledávání

Jelikož se každá cesta může lišit, má uživatel možnost předem si cestu naplánovat pomocí rychlých akcí. Na začátku aktivní cesty se mu následně zobrazí první akce, po jejím odkliknutí druhá a tak dále. Příklad takové cesty může být: *čekání -> jízda tramvají -> přestup -> čekání -> jízda autobusem -> chůze pěšky*. Jedním ze způsobů vybrání je stisknutí akce, druhým je přetažení akce do části tvořené akce pomocí takzvaného drag-and-drop přístupu.

V rámci SwiftUI je implementace drag-and-drop velmi jednoduchá. Element, který lze táhnout implementuje metodu *.onDrag* vracející identifikátor. Část, kam lze prvek přetáhnout, funkci *.onDrop*, která se stará o vytažení identifikátoru a následně o upravení dat do chtěné podoby. Ukázka je na zdrojovém kódu 4.3.

```
1  .onDrop(
2      of: [.url],
3      isTargeted: .constant(false)
4  ) { providers in
5      if let first = providers.first {
6          let _ = first.loadObject(ofClass: URL.self) { value, error in
7              guard let url = value else { return }

9              let quickActions = self.quickActions.filter { $0.id == "\(url)" }

11             withAnimation{
12                 quickActions.forEach { self.selectedQuickActions.append($0) }
13             }
14         }
15     }

17     return false
18 }


21 .onDrag{
22     return .init(contentsOf: URL(string: quickAction.id))!
23 }
```

Zdrojový kód 4.3: Ukázka drag-and-drop

Aktivní cesta

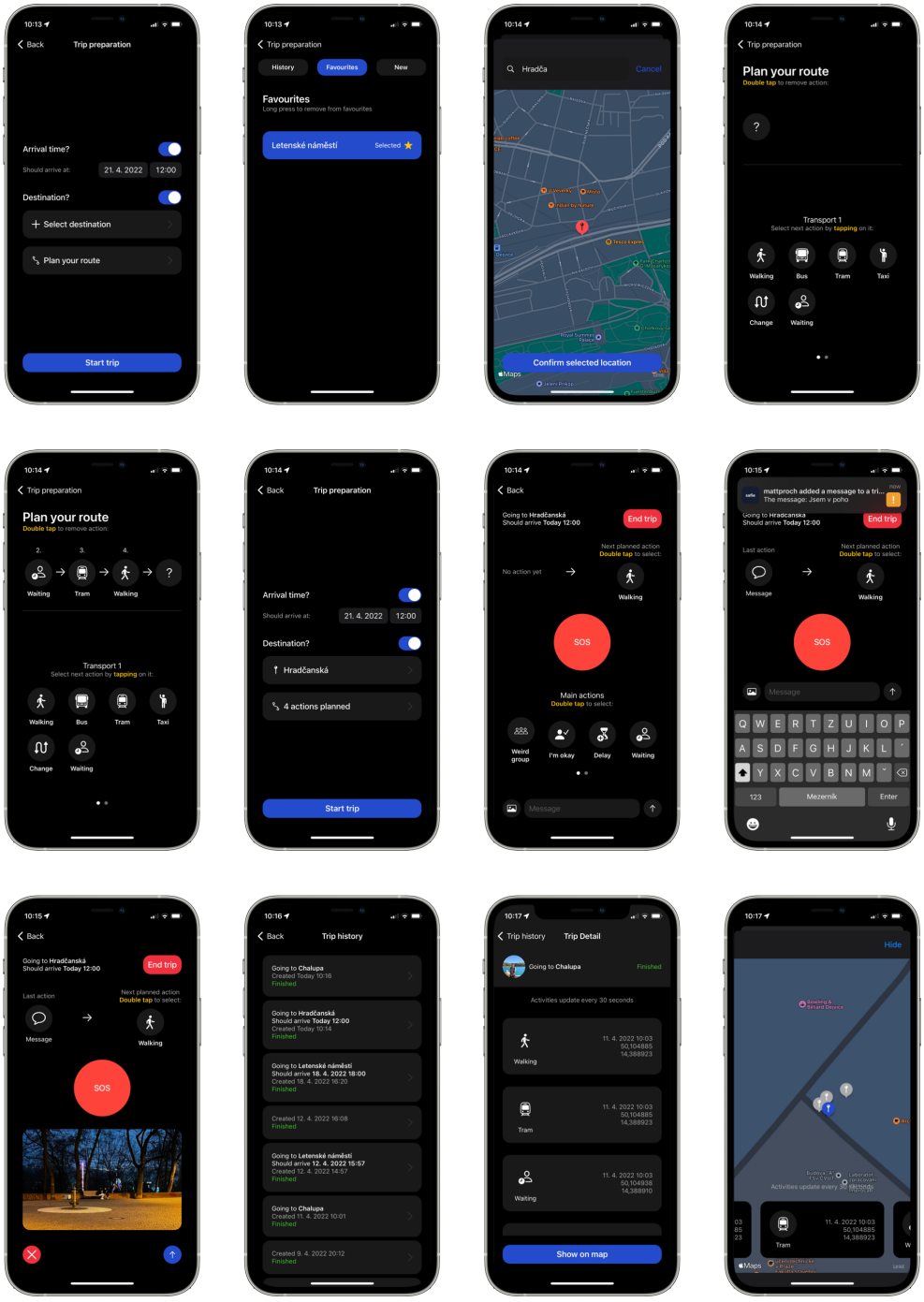
Rozložení na obrazovce aktivní cesty kopíruje návrh v rámci wireframů. V horní části jsou vidět základní informace o cílové destinaci a času příchodu a taktéž tlačítko na ukončení. Před ukončením cesty musí uživatel zadat svůj PIN. Pod těmito údaji a tlačítkem je poslední provedená akce a nadcházející plánovaná akce. Tu lze rychle zvolit dvojitým poklepáním. Následuje SOS tlačítko a další rychlé akce. Ve spodní části je možnost nahrát fotografii a přidat zprávu. Při-

dání fotografie je v rámci odevzdávání diplomové práce možné pouze z galerie a člověku se před samotným odesláním zobrazí náhled.

Uživatel má možnost vrátit se zpět a volně procházet aplikací. Zpátky na aktivní cestu se dostane pomocí stejného tlačítka jako na zahájení, které má změněnou barvu na indikaci, že aktuálně již cestu zahájil. Vypne-li člověk aplikaci, tak ta při opětovném spuštění vyšle na server dotaz, zdali existuje neukončená cesta, aby v ní mohl uživatel pokračovat.

Historie cest

Skrze svůj profil se může uživatel dostat na historii cest, ze které má možnost zobrazit si jednotlivé detaily. V rámci detailu vidí veškeré aktivity s časovou a polohovou informací. Také si může aktivity zobrazit na mapě, obdobně jako uložené polohy v rámci kritické situace SOS. V detailu cesty se aktivity samy každých 30 sekund aktualizují tak, aby mohl člověk sledovat svůj kontakt a neopouštět stránku.



Obrázek 4.11: Trip – ukázka

4.7.6 Critical alerts

Apple nabízí možnost, jak obejít tichý režim a režim nerušit, a vynutit si tak hlasité notifikace – pomocí takzvaných *critical alerts*. K tomuto je však nutno požádat o povolení, jehož vyřízení trvá v rámci týdnů až jednotek měsíců. Bohužel do odevzdání diplomové práce nebylo pro aplikaci Safie rozhodnuto. Co se týče technické implementace, ta je následně velmi jednoduchá a jedná se pouze o malé rozšíření stávajícího systému notifikací.

4.8 Technická příprava testování a nasazení

V rámci ekosystému Apple je oficiální cestou, jak vydávat a testovat aplikace, použití služeb App Store Connect[79] a TestFlight[80]. Vývojář archivuje aktuální stav aplikace a tento archiv odešle právě na App Store Connect. TestFlight je následně službou pro distribuci testerům. Ti si stáhnout stejnojmennou aplikaci do svého zařízení a v ní se jim následně dostupné verze objevují k nainstalování.

Pro samotné testování uvnitř organizace není potřeba dělat nic zvláštního, avšak pro testování vně je potřeba vyplnit popis aplikace a nechat ji zkontrolovat Applem. Tato kontrola většinou trvá v řádu jednotek hodin.

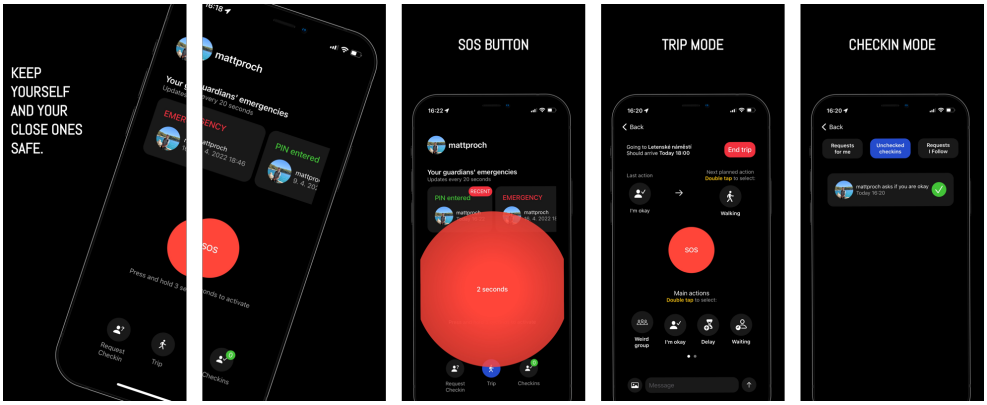
Nasazení a zveřejnění v App Store

Složitější situace je při nasazení přímo do obchodu pro širokou veřejnost. Je nutné splnit několik bodů:

- vyplnit popis aplikace,
- dodat náhledové fotografie (pro aplikace Safie je lze vidět na obrázku 4.12),
- vyplnit klíčová slova,
- zadat informace o dostupnosti (země) a věkovém omezení,
- vyplnit informace o využívání citlivých dat,
- dodat instrukce ohledně přihlášení pro kontrolu aplikace,
- stanovit cenu aplikace.

Následně vývojář odešle aplikaci ke kontrole Apple. Až po jejich schválení se dostane aplikace do oficiálního obchodu. Většinou však nastanou problémy a celý proces se může ze dnů protáhnout na týdny.

Vyvíjená aplikace byla během kontroly vrácena jednou kvůli nevyhovujícím náhledovým obrázkům pro iPad. Po následné úpravě již schválena byla a je dostupná pod jménem „Safie“ na adrese:
https://apps.apple.com/cz/app/safie/id1616323233.



Obrázek 4.12: App Store – Náhledové obrázky

Kapitola 5

Uživatelské testování

Testování lze provádět mnoha způsoby, kde každý se soustředí na určitý aspekt systému. Aplikaci lze testovat na úrovni kódu pomocí lokálních unit testů či integračních testů nebo již s koncovým zákazníkem v rámci akceptačního testování. Já jsem se v diplomové práci rozhodl pro uživatelské testování, které je účinným nástrojem pro zjištění problémů při používání aplikace a překážek při dosahování cílů.

Průběh testování se skládal ze tří částí:

1. pre-test,
2. uživatelský test,
3. post-test.

V první části (pre-test) jsem začal obecným popisem služby k jejímu přiblížení a otázkami, které zjišťovaly, jakým stylem účastníci aktuálně komunikují v nepříjemných situacích a zdali již používají obdobné aplikace. Následně jsem zadával úkoly, při jejichž plnění jsem dbal na nenastiňování průchodu tak, aby bylo pozorování autentické a neovlivněné. Po splnění jednotlivých zadání jsem sepsal pocity a názory uživatelů, jednak negativní, ale i pozitivní. Během testování byla nahrávána obrazovka zařízení a já jsem si dělal rovněž poznámky pro pozdější analýzu (na přiloženém médiu jsou dostupné ukázky nahrávek). Ve finální části (post-test) jsem zjišťoval celkový dojem z aplikace a reálnou využitelnost.

Na konci této kapitoly jsou všechny poznatky shrnuty a vyhodnoceny k návrhu potřebných úprav a zlepšení.

5.1 Testeři a Nielsenova křivka

Jakob Nielsen se zabýval zjištěním ideálního počtu testovacích subjektů. Ve svém výzkumu zjistil, že pro maximální poměr přínosů a nákladů je tento počet roven 5.[81] Samozřejmě jsou zde výjimky například pro kvantitativní studie, ale jde-li o informační systémy a aplikace, 5 se zdá býti optimem mezi počtem zjištěných problémů a náklady na samotné testování. S každým testerem navíc se nezvyšuje lineárně počet nalezených problémů.

Testery jsem vybíral ze svého okolí tak, aby spadali v rámci person do skupiny typických či občasných uživatelů.

5.2 Pre-test

Před testováním samotné mobilní aplikace jsem uživatele seznámil s principem a zaměřením aplikace a položil jim několik následujících otázek:

- Jaké využíváte způsoby komunikace při (nočním) cestování?
- Využíváte již nyní nějakou obdobnou aplikaci?
- Chybí vám na trhu něco v této oblasti?

Vyhodnocení:

Všech pět účastníků z testování nevyužívá žádnou obdobnou aplikaci a komunikují převážně přes zmíněný Messenger[33] či iMessage[34]. 2 z dotázaných s někým sdílí polohu. Všichni jsou na svůj způsob komunikace zvyklý, a tedy je nenapadá nic, co by jim v této oblasti chybělo.

5.3 Testovací scénáře

Testovací scénář by měl uživatele uvést do situace, avšak nenapovídat k jejímu řešení. Nebyl-li by takto scénář napsán či by moderátor nerespektoval nezasahování do průběhu, mohlo by dojít ke zjištění neodpovídajících a nekorektních výsledků.

5.3.1 Přihlášení

Očekávaný průchod:

1. otevření aplikace,
2. průchod onboardingem (seznámení se s SOS tlačítkem a potvrzení práv na notifikace a polohu),
3. přihlášení pomocí jedné z externích služeb,
4. nastavení unikátního PINu,

Popis	Scénář pro zjištění prvního dojmu z aplikace a potenciálních problémů, které uživatele donutí nepřihlásit se.
Zadání	Založte si v aplikaci účet.
Stav aplikace	Čerstvě nainstalovaná aplikace, nikdy nespuštěná.

Tabulka 5.1: Zadání testovacího scénáře pro přihlášení

5. nastavení uživatelského jména.

Vyhodnocení:

Se samotným přihlášením neměl žádný z účastníků problém. Zajímavý trend byl však v přeskakování onboardingu (úvodní obrazovky vysvětlující fungování), které si 4 z 5 uživatelů vůbec nepřečetlo.

5.3.2 Přidání prvních nouzových kontaktů

Popis	Scénář na odhalení překážek v přidávání nouzových kontaktů.
Zadání	Přidejte si libovolný kontakt (moderátor sdělí uživatelské jméno).
Stav aplikace	Uživatel se čerstvě přihlásil.

Tabulka 5.2: Zadání testovacího scénáře pro přidání prvních nouzových kontaktů

Očekávaný průchod:

1. kliknutí na svůj profil na hlavní stránce,
2. kliknutí na prázdný seznam kontaktů vyzývající přidání,
3. otevření formuláře pro přidání,
4. vyplnění uživatelského jména,
5. odeslání formuláře.

Vyhodnocení:

Při přidání prvního nouzového kontaktu mělo všech 5 testerů stejný problém, a tím byl proklik na svůj profil na hlavní stránce. Všichni byli ze začátku lehce zmatení a teprve po chvíli je možnost tohoto průchodu napadla. Avšak většina vyjádřila stejný názor, že při potenciálním dalším přidání by již žádný problém neměla. Jeden z účastníků také vyslovil přání, že by rád měl viditelnější potvrzení toho, že žádost opravdu odešla.

5.3.3 Přijetí žádosti o nouzový kontakt

Popis	Scénář na odhalení problémů s potvrzením žádostí o přidání kontaktu.
Zadání	Moderátor vám pošle žádost o přidání nouzového kontaktu. Potvrďte ji.
Stav aplikace	Uživatel přihlášen, jinak může být aplikace v libovolném stavu.

Tabulka 5.3: Zadání testovacího scénáře pro přijetí žádosti o nouzový kontakt

Očekávaný průchod:

1. obdržení notifikace se žádostí,
2. otevření svého profilu,
3. otevření správy kontaktů,
4. kliknutí na žádosti,
5. bod 2-4 lze alternativně přeskočit kliknutím na notifikaci,
6. potvrzení žádosti.

Vyhodnocení:

3 z 5 účastníků se prokliklo přes notifikaci, a nemělo se splněním scénáře tedy žádný problém. Zbylí dva tak neudělali, a tím nastal významný problém s průchodem a nalezením žádostí. Dosažení nebylo intuitivní, a je tak potřeba rozhodně upravit.

5.3.4 Požádání o jednorázový checkin

Popis	Scénář na získání zpětné vazby ohledně checkinů.
Zadání	Váš partner se již měl vrátit domu, ale není tomu tak. Požádejte ho o checkin.
Stav aplikace	Uživatel již musí mít přidán kontakt.

Tabulka 5.4: Zadání testovacího scénáře pro požádání o jednorázový checkin

Očekávaný průchod:

1. proklik z hlavní stránky na zadávání,
2. zvolení kontaktu,
3. odeslání.

Vyhodnocení:

Při požádání o jednorázový checkin neměl žádný z účastníku problém. 2 však vyhodnotili, že by bylo vhodné, kdyby při pouze jednom přidaném kontaktu byl automaticky označený.

5.3.5 Požádání o pravidelný checkin

Popis	Testovací scénář na zjištění, zdali se uživatel vyzná v nastavení pravidelného checkinu.
Zadání	Vaše dítě jde s kamarády ven. Vy po něm chcete, aby se pravidelně hlásilo, že je v pořádku.
Stav aplikace	Přidán alespoň jeden kontakt.

Tabulka 5.5: Zadání testovacího scénáře pro požádání o pravidelný checkin

Očekávaný průchod:

1. otevření obrazovky na zadávání,
2. vybrání kontaktu,
3. zvolení pravidelného checkinu,
4. vyplnění časových údajů,
5. vyplnění intervalu,
6. odeslání.

Vyhodnocení:

Nalezení a samotné odeslání pravidelného checkinu nikomu nezpůsobovalo nesnáze. Jeden z testerů zmínil, že komponenty pro vyplňování časů (které jsou však nativní) mu přišly lehce komplikované, a zdali by nešly nahradit něčím, kde se toho vyplňuje méně, popřípadě místo číselného data by bylo napsáno například „dnes“.

5.3.6 Potvrzení checkinu

Popis	Scénář na zjištění, zdali se uživatel vyzná v potvrzení checkinu.
Zadání	Partner vám odeslal žádost o checkin. Potvrďte ji.
Stav aplikace	Přidán alespoň jeden kontakt.

Tabulka 5.6: Zadání testovacího scénáře pro potvrzení checkinu

Očekávaný průchod:

1. obdržení notifikace se žádostí,
2. otevření stránky s checkiny,
3. alternativně proklik z notifikace,
4. potvrzení.

Vyhodnocení:

S potvrzením checkinu neměl žádný z testerů problém.

5.3.7 Kritická situace – bez ohrožení

Popis	Scénář na zjištění, zdali je mechanismus SOS tlačítka v aktuálním stavu použitelný a pochopitelný.
Zadání	Jdete večer sám/sama po ulici a vidíte divně vypadající skupinku. Jelikož se bojíte, použijete SOS tlačítko. Nakonec se však nestane nic ohrožujícího.
Stav aplikace	Přidán alespoň jeden kontakt.

Tabulka 5.7: Zadání testovacího scénáře pro kritickou situaci – bez ohrožení

Očekávaný průchod:

1. zmáčknutí a držení tlačítka,
2. puštění tlačítka,
3. zadání PINu do 10 vteřin.

Vyhodnocení:

Během testování SOS tlačítka si jeden z účastníků nevšiml odpočtu 10 vteřin při zadávání PINu pro potvrzení, že je v bezpečí. Je tedy potřeba toto číslo zvýraznit. Ostatní problém neměli.

5.3.8 Kritická situace – s ohrožením

Popis	Testovací scénář na získání zpětné vazby ohledně SOS tlačítka
Zadání	Jdete večer sám/sama po ulici a jde za vámi divně vypadající osoba. Jelikož se bojíte, použijete SOS tlačítko. Nakonec se rozhodnete, že chcete alarmovat své kontakty.
Stav aplikace	Přidán alespoň jeden kontakt.

Tabulka 5.8: Zadání testovacího scénáře pro kritickou situaci – s ohrožením

Očekávaný průchod:

1. zmáčknutí a držení tlačítka,
2. puštění tlačítka.

Vyhodnocení:

Při testování kritické situace, kdy měli testeři nechat systém alarmovat kontakty se nalezlo několik problémů. Jeden z účastníku si nebyl jistý, jestli při konečném zavření aplikace stále odesílá informace o poloze. Dva se vyslovili, že by rádi měli přítomné tlačítko, které přeskočí čekání a ihned informuje kontakty. Obecně zde byl problém v nedostatečném popisu, co v aktuálním momentě aplikace dělá.

5.3.9 Kritická situace z pohledu kontaktu

Popis	Scénář na zjištění, zdali by uživatel věděl, jak aplikaci použít, pokud se do kritické situace dostane jeho kontakt.
Zadání	Přišlo vám upozornění, že váš kontakt je v nebezpečí.
Stav aplikace	Přidán alespoň jeden kontakt.

Tabulka 5.9: Zadání testovacího scénáře pro kritickou situaci z pohledu kontaktu

Očekávaný průchod:

1. obdržení notifikace,
2. otevření aplikace,
3. otevření detailu kritické situace z hlavní stránky,
4. zjištění, kde se uživatel nachází,
5. kontaktování mimo aplikaci.

Vyhodnocení:

Nalezení detailu kritické situace nedělalo nikomu z testerů problém. Většině však v detailu chyběly nějaké prvky – moje poloha a jak daleko jsem od kontaktu, tlačítko pro rychlé zavolání, vizualizace poloh kontaktu formou cesty. Šlo tedy spíše o vylepšení.

5.3.10 Cesta – z práce domů

Popis	Testování funkcionality Trip. Na použití nejkomplexnější funkcionalita.
Zadání	Jdete pozdě večer z práce domů a chcete hrubý obrys cesty sdílet se svým partnerem.
Stav aplikace	Přidán alespoň jeden kontakt.

Tabulka 5.10: Zadání testovacího scénáře pro cestu – z práce domů

Očekávaný průchod:

1. prokliknutí se na zahájení cesty,
2. vyplnění času příchodu,
3. vyplnění destinace,
4. naplánování trasy,
5. spuštění trasy,
6. přidávání aktivit,
7. ukončení cesty.

Vyhodnocení:

Vyhodnocení tohoto scénáře jsem spojil s následujícím (cesta – procházka se psem), neboť se problémy týkají stejné funkcionality a velmi se protínají.

5.3.11 Cesta – procházka se psem

Popis	Obdobné testování jako při cestě domů, cílem je zjistit, jak si uživatel poradí s lehce jiným zadáním, které není striktní na čas příchodu či plánované akce.
Zadání	Chcete jít vyvenčit psa a sdílet cestu s partnerem.
Stav aplikace	Přidán alespoň jeden kontakt.

Tabulka 5.11: Zadání testovacího scénáře pro cestu – procházka se psem

Očekávaný průchod:

1. otevření zahájení cesty,
2. nepovinné vyplnění času příchodu, destinace a naplánování trasy,
3. spuštění trasy,
4. přidávání aktivit,
5. ukončení cesty.

Vyhodnocení:

S touto funkcionalitou nastaly rozhodně největší problémy. Část z účastníků přeskočila krok plánování cesty pomocí rychlých akcí, ostatní měli lehké nesnáze s pochopením, jak samotné plánování funguje. Čas příchodu a cílová destinace obtíže nikomu nedělaly.

Při aktivní cestě žádný z testerů správně nevyužil zamýšlené funkcionality plánovaných akcí. Jednomu z testerů chyběla možnost úpravy detailů (například čas příchodu). Obecně byl pohled na tuto část aplikace spíše skeptický a trvalo delší dobu, nežli se s ní spřátelili. Rozhodně zde chybělo větší vysvětlení jednotlivých prvků.

5.4 Post-test

- Ohodnoťte funkcionality aplikace na stupnici 1-5 (1 – nejlepší, 5 – nejhorší).
- Ohodnoťte používání aplikace na stupnici 1-5 (1 – nejlepší, 5 – nejhorší).
- Využil byste aplikaci ve svém životě?

Vyhodnocení:

Tabulka 5.12 ukazuje ohodnocení prvních dvou otázek. Testeři se prakticky shodli, že samotný nápad se jim líbí, avšak provedení není podle jejich ideálních představ. Co se týče třetí otázky ohledně osobního využití, opět zde byla shoda, kdy se všem velmi zamlouvalo SOS tlačítko, které by zužitkovali, ale naopak cestu by v aktuálním stylu nevyužil téměř nikdo. Účastníky testování jsem dále vyzval, aby po dobu jednoho týdne aplikaci dále testovali, což přineslo pouze potvrzení již zmíněných bodů – pozitivní reakce na SOS tlačítko, ovšem minimální využití funkce sdílení cesty. K tomuto se následně více vyjádřím v rámci závěru v sekci Budoucnost.

Otázka	Tester 1	Tester 2	Tester 3	Tester 4	Tester 5
Funkcionality	1-	1-	1	2	1
Používání	2-	3	3	3	2-

Tabulka 5.12: Ohodnocení funkcionalit a používání (1 – nejlepší, 5 – nejhorší)

5.5 Návrhy na zlepšení

Testování odhalilo mnohé nedostatky a slabiny aplikace, které v této sekci zformuluji a navrhnu konkrétní body pro vylepšení uživatelského zážitku (v době odevzdávání diplomové práce nebyly změny implementovány). Většina podnětů vychází z uživatelského testování, a navazuje tak na věci zmíněné v předchozích sekcích, některé však pocházejí z mé hlavy v rámci používání aplikace po delší dobu.

- Proklik na správu kontaktů dát do spodního menu na hlavní stránce a využívat barevného odznaku s číslem jako upozornění na nevyřízené žádosti.
- Vylepšení detailu kritické situace – jednotlivé polohy jako cesta, přidání tlačítka na zavolání (to znamená i vyplnění telefonu v rámci profilu), ukazovat moji polohu a teoreticky i možnost navigace.
- Vylepšení zpětné vazby při přidání kontaktu – hláška o potvrzení a obrazovka s odeslanými žádostmi.
- Lepší popis, co systém na pozadí dělá a jak se má uživatel chovat v průběhu kritické situace – SOS tlačítka.
- Zvýraznění ubíhajícího času při zadávání PINu v rámci SOS tlačítka.
- Celkové předělání funkcionality cesta (nebo přímo její odebrání pro zjednodušení uživatelova zážitku).
- Možnost mít stálé oblíbené nouzové kontakty a následně jednorázové na určitou dobu.
- Možnost zrušení pravidelného checkinu.
- Možnost přidání fotografie přímo pomocí fotoaparátu.

Závěr

Cílem této práce bylo prozkoumání možností využití mobilních zařízení na bezpečnost cestování, rešerše konkurence a následný vývoj vlastní aplikace na základě definovaných požadavků. V této části popíši, jak jsem postupoval pro naplnění tohoto cíle.

V první kapitole jsem se zaměřil na analytickou část, kdy jsem provedl průzkum chování a potřeb cílové skupiny a následnou rešerši existujících řešení. Z nabytých informací jsem definoval funkční a nefunkční požadavky, a stanovil tak hranice vyvíjené aplikace. Mezi hlavní funkcionality, které z těchto kapitol vzešly, jsou SOS tlačítko, požádání o checkin a zaznamenávání aktivit během cest.

Dále jsem prozkoumal metodiky vývoje softwaru a zvolil vhodnou metodu (iterativní), kterou jsem aplikoval na samotnou realizaci. Vývoj jsem rozdělil do dvou kapitol – první zabývající se pouze serverovou částí a druhou ohledně mobilní aplikace. V nich jsem nejdříve provedl analýzu skýtající výběr technologií, tvorbu podpůrné dokumentace formou diagramů a vizuálních návrhů. Poté přišly na řadu části ohledně samotné implementace.

Konečný produkt prošel uživatelským testováním, které odhalilo slabá místa, kterými jsou především na první pohled nezřejmé umístění některých prvků (například proklik přes můj profil na správu kontaktů) a následně funkcionalita na vytváření cest. Naopak na SOS tlačítko byla velmi pozitivní zpětná vazba. Otázkou do budoucna tak je, jestli není vhodné se vydat směrem k vylepšení SOS tlačítka na úkor ostatních hlavních funkcionalit. Samotná aplikace je nyní dostupná ke stažení na App Store[82].

Budoucnost

Finálně napíši pár slov ohledně budoucnosti celé aplikace a to jak z pohledu obsahu a funkcionalit, tak i po stránce monetizace.

Tento odstavec navazuje na skutečnosti zjištěné v rámci testování. Během něho účastníci mnohem kladněji reagovali na funkcionalitu SOS tlačítka a naopak lehce skepticky k tvorbě cesty, ačkoliv ta je po technické stránce nejkomplexnější. Dle mého je tak správné si položit otázku, zdali není vhodné se vydat cestou úplného odstranění druhé zmíněné funkcionality a naopak se snažit vylepšit a dovést k „dokonalosti“ SOS tlačítko, o které uživatelé vyjádřili zájem.

Na to navazuje i další bod, který zde chci probrat. Samotný nápad jsem velmi často konzultoval s jedním ze svých kamarádů, který projevil možný zájem začlenit aplikaci i do své diplomové práce na téma crowdfundingových kampaní. Prováděl jsem s ním i testování, během kterého jsme se dostali k jednomu z jeho kamarádů, který aktuálně podniká v Mexiku v oblasti soukromé ochranky – lidé si za měsíční poplatek mohou platit službu, kdy jejich lidé jsou rozmístěni napříč danou lokalitou, a vlastně tak zastupují policejní složky. Dohromady jsme dali zajímavý nápad, napojit jeho strukturu a logistiku rozmístění ochranky na systém SOS tlačítka.

Další možností na monetizaci by mohla být spolupráce s veřejným sektorem či naopak soukromým a vydat aplikaci pod záštitou některé z firem podnikající v tomto odvětví.

Literatura

[1] Sayan Kumar Pal. *Software Engineering | Classical Waterfall Model*. GeeksforGeeks [online]. 2022. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/

[2] Sayan Kumar Pal. *Software Engineering | Iterative Waterfall Model*. GeeksforGeeks [online]. 2021. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/

[3] saumyasaxena2730. *Software Engineering | Incremental process model*. GeeksforGeeks [online]. 2022. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-incremental-process-model/

[4] 02DCE. *Software Engineering | Prototyping Model*. GeeksforGeeks [online]. 2022. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-prototyping-model/

[5] Sayan Kumar Pal. *Software Engineering | Spiral Model*. GeeksforGeeks [online]. 2022. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-spiral-model/

[6] 02DCE. *Software Engineering | Rapid application development model (RAD)*. GeeksforGeeks [online]. 2018. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-rapid-application-development-model-rad/

[7] ashvika99. *Difference between Prototype Model and RAD Model*. GeeksforGeeks [online]. 2018. [cit. 2022-03-29]. Dostupné z:

https://www.geeksforgeeks.org/difference-between-prototype-model-and-rad-model/

[8] SakshiBhakhra. *Scrum (software development)*. GeeksforGeeks [online]. 2019. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/scrum-software-development/

[9] Sayan Kumar Pal. *Software Engineering | Agile Software Development*. GeeksforGeeks [online]. 2021. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-agile-software-development/

[10] Mlejnek, J.: Agilní přístup. [online], [cit. 2022-03-29]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/506281/mod_resource/content/5/12.prednaska.pdf

[11] Sayan Kumar Pal. *Software Engineering | Extreme Programming (XP)*. GeeksforGeeks [online]. 2018. [cit. 2022-03-29]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-extreme-programming-xp/

[12] Thomas Hamilton. *What is Test Driven Development (TDD)? Tutorial with Example*. Guru99 [online]. 2022. [cit. 2022-03-31]. Dostupné z: https://www.guru99.com/test-driven-development.html

[13] Josef Pavlíček. *UI Design Steps* [online]. 2020. [cit. 2022-03-31]. Dostupné z: https://docs.google.com/presentation/d/1CldSHyC8D8cN2LGrqUVJ2U5eEKn5z9MpkFPmzwZX4Zc

[14] Nick Babich. *Prototyping 101: The Difference between Low-Fidelity and High-Fidelity Prototypes and When to Use Each*. Adobe [online]. 2017. [cit. 2022-03-31]. Dostupné z: https://blog.adobe.com/en/publish/2017/11/29/prototyping-difference-low-fidelity-high-fidelity-prototypes-use

[15] Breakout: Rapid Application Development - 2021 Complete Guide. [online], [cit. 2022-03-29]. Dostupné z: https://getbreakout.com/no-code/rapid-application-development/

[16] wrike: Guide to Scrum Sprints. [online], [cit. 2022-03-29]. Dostupné z: https://www.wrike.com/scrum-guide/scrum-sprints/

[17] Ronak Ganatra. *GraphQL Vs. REST APIs*. GraphCMS [online]. 2021. [cit. 2022-04-14]. Dostupné z: https://graphcms.com/blog/graphql-vs-rest-apis

[18] Babich, N.: Common Wireframing Issues to Avoid. [online], 2020, [cit. 2022-03-31]. Dostupné z: https://xd.adobe.com/ideas/process/wireframing/common-wireframing-issues-to-avoid/

[19] Lowry, T.: 5 ways to improve your prototyping workflow. [online], [cit. 2022-03-31]. Dostupné z: https://www.figma.com/best-practices/five-ways-to-improve-your-prototyping-workflow/

[20] Rishu Mishra. *Difference Between MVC and MVVM Architecture Pattern in Android*. GeeksForGeeks [online]. 2020. [cit. 2022-04-21]. Dostupné z: https://www.geeksforgeeks.org/difference-between-mvc-and-mvvm-architecture-pattern-in-android/

[21] *Kvantitativní a kvalitativní výzkum (srovnání)*. Survio [online]. 2020. [cit. 2022-03-09]. Dostupné z: https://www.survio.com/cs/blog/jak-vytvorit-dotaznik/kvantitativni-vyzkum-kvalitativni-vyzkum

[22] Matthew Smith. *Women feel less safe walking home alone at night than in 2018*. YouGov [online]. 2021. [cit. 2022-03-05]. Dostupné z: https://yougov.co.uk/topics/lifestyle/articles-reports/2021/11/01/women-feel-less-safe-walking-home-alone-night-2018

[23] Pierre Carbonnelle *PYPL PopularitY of Programming Language* [online]. [cit. 2022-04-14]. Dostupné z: https://pypl.github.io/PYPL.html

[24] Apple Inc. *Apple* [online]. [cit. 2022-02-1O]. Dostupné z: https://www.apple.com

[25] Veronika Víšková. *Kvantitativní a kvalitativní výzkum (srovnání)*. WikiKnihovna [online]. 2011. [cit. 2022-03-09]. Dostupné z: https://wiki.knihovna.cz/index.php?title=Kvantitativn%ED_a_kvalitativn%ED_v%FDzkum_(srovn%E1n%ED)

[26] Google LLC *Google* [online]. [cit. 2022-02-14]. Dostupné z: https://google.com

[27] YouGov PLC. *YouGov | About YouGov Company* [online]. [cit. 2022-03-09]. Dostupné z: https://yougov.co.uk/about/

[28] Twitter, Inc. *Twitter* [online]. [cit. 2022-03-06]. Dostupné z: https://twitter.com

[29] Curtice., K.: [online], 2018, [cit. 2022-03-06]. Dostupné z: https://twitter.com/KaitlinCurtice/status/1045353284788203521

[30] Apple Inc. *App Store* [online]. [cit. 2022-02-1O]. Dostupné z: https://apps.apple.com

[31] Google *Google Play* [online]. [cit. 2022-02-21]. Dostupné z: https://play.google.com

[32] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. Nielsen Norman Group [online]. 1994. [cit. 2022-03-06]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[33] Meta *Messenger* [online]. [cit. 2022-02-1O]. Dostupné z: https://www.messenger.com

[34] Apple Inc. *Messages for iPhone, iPad, Apple Watch, and Mac - Official Apple Support* [online]. [cit. 2022-02-1O]. Dostupné z: https://support.apple.com/messages

[35] Apple Inc. *iCloud - Find My - Apple* [online]. [cit. 2022-02-1O]. Dostupné z: https://www.apple.com/icloud/find-my/

[36] Follo App Ltd. *Follo – Walking Safety App | Simple. Automated. Private.* [online]. [cit. 2022-02-1O]. Dostupné z: https://folloapp.co

[37] SafeTrek, Inc. *Noonlight: Feel Protected 24/7 on the App Store* [online]. [cit. 2022-02-21]. Dostupné z: https://apps.apple.com/us/app/noonlight-feel-protected-24-7/id716262008

[38] Mlejnek, J.: Analýza a sběr požadavků. [online], [cit. 2022-03-16]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/506241/mod_resource/content/7/03.prednaska.pdf

[39] Matthew Smith. *Functional and Nonfunctional Requirements: Specification and Types*. AltexSoft [online]. 2021. [cit. 2022-03-16]. Dostupné z: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/

[40] Jonathan Dyson. *Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements*. LinkedIn [online]. 2019. [cit. 2022-03-16]. Dostupné z: https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson/

[41] HP Inc. *Hewlett Packagrd* [online]. [cit. 2022-03-09]. Dostupné z: https://www.hp.com

[42] Rami Anwar. *ANDROID VS IOS DEVELOPMENT: WHICH PLATFORM SHOULD I DEVELOP FOR FIRST?* Eastern Peak [online]. 2021. [cit. 2022-03-27]. Dostupné z: https://easternpeak.com/blog/android-vs-ios-development-which-platform-first/

[43] StatCounter: Android Version Market Share Worldwide. [online], [cit. 2022-03-27]. Dostupné z: https://gs.statcounter.com/os-version-market-share/android

[44] Apple Inc. *App Store - Support - Apple Developer* [online]. [cit. 2022-03-13]. Dostupné z: https://developer.apple.com/support/app-store/

[45] Usability.gov: Use Cases. [online], [cit. 2022-03-27]. Dostupné z: https://www.usability.gov/how-to-and-tools/methods/use-cases.html

[46] Google LLC *Google Trends* [online]. [cit. 2022-04-14]. Dostupné z: https://trends.google.com/trends

[47] MDN Contributors. *JavaScript*. mdn [online]. 2021. [cit. 2022-04-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[48] *Introduction to Node.js*. nodejs.dev [online]. [cit. 2022-04-13]. Dostupné z: https://nodejs.dev/learn/introduction-to-nodejs

[49] *The V8 JavaScript Engine*. nodejs.dev [online]. [cit. 2022-04-13]. Dostupné z: https://nodejs.dev/learn/the-v8-javascript-engine

[50] *The Node.js Event Loop*. nodejs.dev [online]. [cit. 2022-04-13]. Dostupné z: https://nodejs.dev/learn/the-nodejs-event-loop

[51] Yarn *Home | Yarn - Package Manager* [online]. [cit. 2022-04-14]. Dostupné z: https://yarnpkg.com

[52] npm, Inc. *npm* [online]. [cit. 2022-04-14]. Dostupné z: https://www.npmjs.com

[53] Alfrick Opidi. *NPM vs. Yarn: Which Package Manager Should You Choose?* WhiteSource [online]. 2020. [cit. 2022-04-14]. Dostupné z: https://www.whitesourcesoftware.com/free-developer-tools/blog/npm-vs-yarn-which-should-you-choose/

[54] Nest JS *NestJS - A progressive Node.js framework* [online]. [cit. 2022-04-14]. Dostupné z: https://nestjs.com

[55] StrongLoop, I.; other expressjs.com contributors: Express - Node.js web application framework. [online], [cit. 2022-04-14]. Dostupné z: https://expressjs.com

[56] Fastify, Fast and low overhead web framework, for Node.js. [online], [cit. 2022-04-14]. Dostupné z: https://www.fastify.io

[57] TypeORM *TypeORM* [online]. [cit. 2022-04-14]. Dostupné z: https://typeorm.io

[58] Apollo Graph Inc. *Apollo GraphQL* [online]. [cit. 2022-04-14]. Dostupné z: https://www.apollographql.com

[59] *Firebase Cloud Messaging* [online]. [cit. 2022-04-20]. Dostupné z: https://firebase.google.com/docs/cloud-messaging

[60] *Amazon S3* [online]. [cit. 2022-04-20]. Dostupné z: https://aws.amazon.com/s3/

[61] *Firebase Authentication | Simple, no-cost multi-platform sign-in* [online]. [cit. 2022-04-24]. Dostupné z: https://firebase.google.com/products/auth

[62] Sketch B.V. *Design, collaborate, prototype and handoff · Sketch* [online]. [cit. 2022-04-20]. Dostupné z: https://www.sketch.com

[63] Filip Němeček. *Lekce 1 - Úvod do jazyka Swift, platformy a Xcode*. ITnetwork [online]. 2018. [cit. 2022-04-24]. Dostupné z: https://www.itnetwork.cz/swift/zaklady/uvod-do-jazyka-swift-platformy-a-xcode

[64] Apple Inc. *UIKit | Apple Developer Documentation* [online]. [cit. 2022-04-22]. Dostupné z: https://developer.apple.com/documentation/uikit/

[65] Apple Inc. *SwiftUI | Apple Developer Documentation* [online]. [cit. 2022-04-22]. Dostupné z: https://developer.apple.com/documentation/swiftui/

[66] Bobby Gill. *SwiftUI vs UIKit in 2022, Which Framework Should Your App Use?* Blue Label [online]. 2022. [cit. 2022-04-20]. Dostupné z: https://www.bluelabellabs.com/blog/swiftui-vs-uikit/

[67] Airbnb, Inc. *Lottie* [online]. [cit. 2022-04-20]. Dostupné z: https://airbnb.design/lottie/

[68] Adobe *VFX and motion graphics software | Adobe After Effects* [online]. [cit. 2022-04-20]. Dostupné z: https://www.adobe.com/products/aftereffects.html

[69] Design Barn Inc. *Free Lottie Animation Files, Tools & Plugins - LottieFiles* [online]. [cit. 2022-04-20]. Dostupné z: https://lottiefiles.com

[70] *KingFisher* [online]. [cit. 2022-04-20]. Dostupné z: https://github.com/onevcat/Kingfisher

[71] *SwiftUIPager* [online]. [cit. 2022-04-20]. Dostupné z: https://github.com/fermoya/SwiftUIPager

[72] *KeychainSwift* [online]. [cit. 2022-04-20]. Dostupné z: https://github.com/evgenyneu/keychain-swift

[73] *Firebase iOS SDK* [online]. [cit. 2022-04-20]. Dostupné z: https://github.com/firebase/firebase-ios-sdk

[74] *Facebook iOS SDK* [online]. [cit. 2022-04-20]. Dostupné z: https://github.com/facebook/facebook-ios-sdk

[75] *Google Sign In iOS* [online]. [cit. 2022-04-20]. Dostupné z: https://github.com/google/GoogleSignIn-iOS

[76] Alexey Naumov. *Clean Architecture for SwiftUI* [online]. 2019. [cit. 2022-04-21]. Dostupné z: https://nalexn.github.io/clean-architecture-swiftui/

[77] Apple Inc. *User Notifications | Apple Developer Documentation* [online]. [cit. 2022-04-20]. Dostupné z: https://developer.apple.com/documentation/usernotifications

[78] Apple Inc. *MapKit | Apple Developer Documentation* [online]. [cit. 2022-04-22]. Dostupné z: https://developer.apple.com/documentation/mapkit/

[79] Apple Inc. *App Store Connect - Apple Developer* [online]. [cit. 2022-04-22]. Dostupné z: https://developer.apple.com/app-store-connect/

[80] Apple Inc. *TestFlight - Apple* [online]. [cit. 2022-04-22]. Dostupné z: https://testflight.apple.com

[81] Jakob Nielsen. *How Many Test Users in a Usability Study?* Nielsen Norman Group [online]. 2012. [cit. 2022-04-25]. Dostupné z: https://www.nngroup.com/articles/how-many-test-users/

[82] *Safie* [online]. [cit. 2022-04-29]. Dostupné z: https://apps.apple.com/cz/app/safie/id1616323233?platform=iphone

Seznam použitých zkratek

API Application Programming Interface
APN Apple Push Notification

FURPS Functionality, Usability, Reliability, Performance, Supportability

Hi-Fi High Fidelity

iOS iPhone Operating System

Lo-Fi Low Fidelity

MVVM Model-View-ViewModel

npm Node Package Manager

ORM Object–relational Mapping

PYPL PopularitY of Programming Language Index

RAD Rapid Application Development
REST Representational State Transfer

SDK Software Development Kit

Příloha **A**

Obsah přiložené SD karty

/
- `NodeJS Server` ..... zdrojové kódy serverové části
- `iOS App` ..... zdrojové kódy mobilní aplikace
- `Thesis` ..... zdrojová diplomová práce ve formátu LaTeX
  - `DP_Prochazka_Matyas_2022.tex` ..... hlavní soubor
- `DP_Prochazka_Matyas_2022.pdf` ... diplomová práce ve formátu PDF
- `assignment.pdf` ..... zadání diplomové práce
- `Design` ..... zdrojové soubory k designům a wireframům
- `User Testing` ..... ukázky nahrávek obrazovky během testování