VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO SDÍLENÍ POLOHY VE SKUPINĚ

MOBILE APPLICATION FOR LOCATION SHARING IN A GROUP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. VOJTĚCH MATĚJÍČEK**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Matějíček Vojtěch, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Mobilní aplikace pro sdílení polohy ve skupině**
Mobile Application for Location Sharing in a Group
Kategorie: Uživatelská rozhraní

Zadání:

1. Vyhledejte a analyzujte existující nástroje pro sdílení polohy mezi uživateli.
2. Seznamte se s problematikou multiplatformního vývoje pro chytré telefony.
3. Navrhněte systém pro sdílení polohy ve skupině. Zaměřte se na kvalitní uživatelskou zkušenost, uchování soukromí, šetření se zdroji mobilního telefonu.
4. Implementujte navržený systém.
5. Testujte vyvinutý systém v provozu a s různými uživateli, iterativně vylepšujte jeho funkčnost a uživatelskou zkušenost.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Berkman, E., Hoober, S.: Designing Mobile Interfaces, O'Reilly Media, Inc. 2011
- Prajyot Mainkar, Salvatore Giordano: Google Flutter Mobile Development Quick Start Guide, Packt Publishing 2019
- Kyle Mew: Learning Material Design, Packs Publishing 2015

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4, značné rozpracování bodu 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 3. června 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Cílem této práce je vytvoření mobilní aplikace umožňující snadné sdílení polohy mobilního zařízení ve vytvořených skupinách. Práce obsahuje analýzu potřeb uživatelů, uživatelského rozhraní a možností přidávání členů skupin. Mobilní aplikace bude dostupná pro obě majoritní mobilní platformy, tedy iOS i Android. Z tohoto důvodu byla vyvíjena pomocí nástroje Flutter pro multiplatformní vývoj mobilních aplikací. Nedílnou součástí aplikace je webový server napsaný v jazyce PHP a frameworku Yii2.

Abstract

Target of this thesis is creation of mobile application enabling easy sharing of user's location in a group. Thesis includes analysis of needs of users, GUI and options of including new members of a group. Application is available for both major mobile platforms, iOS and Android. For that reason it was developed with tool Flutter providing multiplatform applications development. Necessary part of the application is web server written in language PHP and framework Yii2.

Klíčová slova

Mobilní aplikace, iOS, Android, Flutter, Dart, PHP, Yii2, API, MVC

Keywords

Mobile application, iOS, Android, Flutter, Dart, PHP, Yii2, API, MVC

Citace

MATĚJÍČEK, Vojtěch. *Mobilní aplikace pro sdílení polohy ve skupině*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Mobilní aplikace pro sdílení polohy ve skupině

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Matějíček
3. června 2020

Poděkování

Děkuji svému vedoucímu prof. Ing. Adamovi Heroutovi, Ph.D. za vedení práce a odbornou konzultaci.
Velké díky patří mým rodičům za podporu během celého studia.

Obsah

Kapitola 1

Úvod

Práce vznikla jako technická dokumentace k aplikaci **Flox**. Tato aplikace umožňuje registrovanému uživateli vytváření skupin přátel, v rámci nichž bude mezi jednotlivými členy možné sdílet a sledovat polohu mobilního zařízení. Aktuální poloha všech členů dané skupiny je v aplikaci indikována ikonami na mapě, jejichž umístění je v pravidelném intervalu aktualizováno. Podrobný popis funkcí aplikace je popsán v kapitole 2.

Aplikace je vyvíjena pomocí nástroje Flutter, který slouží pro multiplatformní vývoj mobilních aplikací. Díky tomu bylo možné sestavit mobilní aplikaci, kterou lze nainstalovat na mobilní zařízení s mobilními operačními systémy Android i iOS. Více o technologiích pro multiplatformní vývoj a samotném nástroji Flutter obsahuje kapitola 3. Vývoj samotné aplikace popisuje kapitola 5.

Pro uchovávání uživatelských dat byla implementována webová serverová aplikace napsaná v jazyce PHP a frameworku Yii2. S touto aplikací je spojená relační databáze, na které jsou uchovávána všechna potřebná data. Pro synchronizaci uložených dat s mobilní aplikací slouží aplikační rozhraní, se kterým aplikace komunikuje prostřednictvím HTTP požadavků. Podrobný popis serverové části aplikace je obsažen v kapitole 6.

Práce kromě podrobného popisu procesu vývoje aplikace pojednává o jejím uživatelském testování a změnách, které byly v návaznosti na toto testování na aplikaci provedeny. Proces testování blíže popisuje kapitola 7.

Testování i celkový vývoj byl prováděn iterativní metodou. Výstupem práce je přehledná a uživatelsky přívětivá mobilní aplikace, která splňuje všechny předem definované funkce.

Kapitola 2

Analýza a specifikace aplikace

Tato kapitola popisuje kroky, které byly podniknuty za účelem vytvoření specifikace funkcionality mobilní aplikace vyvíjené v této práci.

Součástí tohoto procesu bylo určení, která kategorie uživatelů mobilních zařízení bude považována za cílovou skupinu uživatelů aplikace. V dané cílové skupině pak proběhlo zjišťování požadavků pro funkce samotné aplikace. Posledním krokem byla analýza existujících aplikací. Na základě údajů získaných analýzou byly definovány specifikace a funkce výsledné aplikace.

2.1 Definice cílové skupiny uživatelů

Do cílové skupiny uživatelů patří všichni uživatelé chytrých mobilních zařízení, pro které je výhodné sdílení polohy napříč skupinou dalších uživatelů. Může se jednat zejména o:

- **rodiny s dětmi** – rodičům je díky aplikaci umožněno pohodlné sledování polohy jejich dětí a ostatních členů rodiny;
- **cestovatele ve skupinách** – pokud se skupina rozdělí, je díky aplikaci jednoduché opět najít ostatní členy. Tato skupina uživatelů zahrnuje i organizované zájezdy s průvodcem, který má díky aplikaci přehled o pohybu účastníků zájezdu;
- **návštěvníky hudebních festivalů** a jiných událostí konaných na širším prostranství – kvůli hluku může být problematické sejít se s ostatními členy skupiny pomocí telefonního hovoru. Díky této aplikaci je ale snadno může uživatel najít na mapě;
- **učitele na základních školách a nižších gymnáziích** – na školním výletě má učitel neustálý přehled o poloze a pohybu žáků při jejich volném rozchodu;
- a jiné.

Podmínkou pro používání aplikace je kromě schopnosti zařízení detekovat svou aktuální polohu také připojení k internetu, jehož prostřednictvím je poloha sdílena ostatním členům uživatelských skupin. O funkci sdílení polohy mezi uživateli blíže pojednává kapitola 4.3.

Protože je aplikace vyvíjena pro obě majoritní mobilní platformy, tedy iOS i Android, není cílová skupina uživatelů nijak omezena tím, který operační systém na svém mobilním zařízení používá. Výjimku tvoří pouze již zastaralá platforma Windows Mobile a ostatní mobilní zařízení, která se neřadí do kategorie chytrých telefonů.

2.2 Analýza požadavků uživatelů

Pro zjištění vlastností a funkcí aplikace, které by byly mezi uživateli žádané, sloužilo veřejné dotazování potenciálních budoucích uživatelů. Bylo k tomu využito rozeslání dotazníku Google Forms. Při rozesílání dotazníku byl kladen důraz na vyrovnaný poměr mezi uživateli mobilních zařízení se systémem Android a iOS.

Respondenti zde byli dotazováni na běžné způsoby používání svých mobilních zařízení včetně využití aplikací obsahujících napojení na polohové služby. Nejčastěji používané aplikace byly poté otestovány a jejich hlavní specifikace jsou popsány v kapitole 2.3.

Z analýzy výsledků dotazování vyplynulo, že požadavky uživatelů směřují více směrem k jednoduchosti aplikace než k vyššímu množství jejích funkcí. Aplikace, které nabízí uživatelům více funkcí, než je pro její přímočaré používání nutné, jsou pro některé uživatele nepřehledné a v mnohých případech jsou jejich funkce navíc považovány za zbytečné. Pokud je uživatel zahlcen velkým množstvím ovládacích prvků, stává se pro něj práce s aplikací nepříjemnou. Pokud jsou naopak nadbytečné funkce schovány v postranních nabídkách aplikace, jsou snadno přehlédnutelné a uživatel je mnohdy nikdy nepoužije.

2.3 Analýza existujících řešení

V této kapitole je popsáno několik existujících aplikací, které se svojí funkcionalitou a využitím podobají aplikaci vyvíjené v této práci. Jedná se jak o nativní aplikace z dílen výrobců mobilních operačních systémů, tak o aplikace od nezávislých vývojářů.

2.3.1 Nativní aplikace

Velké množství uživatelů projevilo spokojenost s používáním nativních nástrojů pro sdílení polohy. Těmito nástroji jsou aplikace ***Find My*** na zařízeních Apple iPhone a sdílením polohy prostřednictvím ***Google Maps*** na ostatních zařízeních. Obě tato řešení jsou funkční a použitelná pro okamžité sdílení polohy, avšak ani jedna z nich nedisponuje schopností vytváření uživatelských skupin. Uživatel je tedy odkázán na organizaci zobrazených dat „pouhým okem“. Z toho důvodu by větší množství dotázaných možnost vytvářet jednotlivé skupiny a přepínat zobrazení mezi nimi uvítalo. Problémem aplikace Find My je navíc nemožnost sdílení polohy s uživateli mobilních telefonů se systémem Android.

Dotazovaní uživatelé by dále uvítali funkci, která umožňuje stanovení „místa setkání“ pro celou skupinu v určitý čas. Tuto funkci nativní aplikace neobsahují.

2.3.2 Aplikace třetích stran

Kromě uživatelů výše zmíněných nativních aplikací se mezi dotazovanými vyskytovali i uživatelé aplikací třetích stran. Nejčastěji to byli uživatelé aplikace **GPS Locator**. Ta nabízí možnost vytváření „kruhů“ *(circles)* uživatelů pro vzájemné sdílení polohy. Tyto kruhy nemají předdefinovanou dobu vypršení platnosti. Uživatelé tedy sdílí svoji polohu po neurčitý čas, dokud se z kruhu neodpojí. Na tuto vlastnost může být nahlíženo jako na ubírání soukromí uživatele aplikace. Dalším problémem aplikace je její dostupnost pouze pro uživatele platformy Android.

U jiných aplikací třetích stran se vyskytovaly různé problémy či nedokonalosti aplikace. Příkladem takových nedokonalostí jsou softwarové chyby v aplikaci, vysoké nároky na baterii nebo její dostupnost pouze pro jednu či druhou platformu. Některé aplikace odrazovaly

uživatele taktéž vyšší pořizovací cenou nebo nutností zřízení předplatného pro jejich použití. Problémem některých aplikací bylo také příliš velké množství funkcí, které se projevovalo zvýšenou složitostí ovládání aplikace.

2.4 Výsledná definice funkcí aplikace

Vzhledem k informacím zjištěným během veřejného dotazování je tedy cílem práce vytvoření aplikace, která bude jednoduše ovladatelná pro uživatele a nabídne následující základní funkce:

- sdílení polohy zařízení,
- zobrazení polohy ostatních uživatelů na mapě,
- vytváření více uživatelských skupin a přepínání zobrazení mezi jednotlivými skupinami,
- plánování míst setkání pro členy jednotlivých skupin.

Tyto funkce je třeba splňovat s ohledem na dopady na stav baterie mobilního zařízení. Aplikace proto musí být schopna s vědomím uživatele běžet na pozadí a využívat funkce zařízení tak, aby byly co nejšetrnější k baterii. Odesílání polohy zařízení a přijímání aktualizací polohy ostatních členů skupin pak probíhá ve stanoveném intervalu a pouze v době, kdy je uživatel členem platné skupiny. Tím je kromě zajištění optimalizace energetické spotřeby zajištěno i zamezení nežádoucího sledování polohy uživatele. Podrobnosti o implementaci sdílení polohy jsou obsaženy v kapitolách 4.3 a 5.4.

Funkcemi aplikace pro posílení soukromí uživatele jsou dále časové omezení platnosti jednotlivých skupin (nelze vytvořit skupinu a po neomezenou dobu sledovat polohu jejích členů) či nemožnost vyhledávat ostatní uživatele v databázi aplikace. Uživatel vidí pouze ty ostatní uživatele, kteří se sami připojí k jeho skupině na základě odeslané pozvánky. Připojení uživatele ke skupině pro sdílení polohy je blíže specifikováno v kapitole 4.3.2.

Kapitola 3

Technologie použité při vývoji aplikace

V této kapitole je shrnut výčet technologií, principů a nástrojů použitých pro vývoj aplikace. Jednotlivé oddíly aplikace (server, mobilní aplikace) budou dále detailněji popsány v následujících kapitolách věnovaných implementaci těchto oddílů.

3.1 Poloha uživatele

Určování polohy uživatele je jednou z nejpoužívanějších funkcí mobilních zařízení s operačními systémy Android a iOS. Je to také jedna z hlavních funkcí mobilní aplikace vyvíjené v této práci.

V této kapitole jsou popsány technologie, které určování a ukládání polohy mobilního zařízení uživatele umožňují.

3.1.1 Technologie GPS

GPS *(Global positioning system)* [16] je systém využívající síť vesmírných družic, které obíhají okolo planety Země. Tyto družice umožňují určení přesné polohy elektronického přijímače na jejím povrchu. Systém družic GPS je provozován Ministerstvem obrany USA. Po roce 1983 se stal dostupným pro civilní uživatele.

Infrastrukturu systému GPS netvoří pouze systém družic na oběžné dráze okolo naší planety. Celý systém je rozdělen do těchto 3 segmentů:

- **Kosmický segment**: Skládá se z vesmírných družic obíhajících okolo Země. Pro svou plnou operační schopnost vyžaduje nejméně 24 fungujících družic. Momentálně se zde nachází 31 fungujících družic. Družice obíhají ve výšce cca 20 350 km nad zemským povrchem. Jsou vybaveny komunikační jednotkou, řídící jednotkou a atomovými hodinami.
- **Řídicí a kontrolní segment**: Systém pozemních stanic, které řídí, monitorují a spravují satelity v kosmickém segmentu. Tyto stanice se nachází na území USA a na základnách Letectva USA rozmístěných po celé Zemi.
- **Uživatelský segment**: Do tohoto segmentu patří uživatelé různých zařízení schopných přijímat a zpracovat signál GPS.

Poloha uživatele je určena na základě porovnání časových značek signálů odeslaných z jednotlivých družic s časem jejich přijetí na straně GPS přijímače. Podle rozdílů časových značek a rychlosti, kterou se šíří signál z družic, je přesně určena poloha přijímacího zařízení. Pro určení zeměpisných souřadnic GPS přijímače je nutný signál alespoň ze 3 družic, pro určení nadmořské výšky a dalších dat je třeba signál alespoň ze 4 družic.

Pro správné určení polohy je třeba zajistit minimum překážek v cestě rádiových vln mezi družicemi a GPS přijímačem. Je tedy nutná přímá viditelnost oblohy. Určování polohy ve vnitřních prostorách budov tedy není možné, případně bude velmi nepřesné.

Systém GPS není jediným systémem pro určování polohy na Zemi. Existují i další systémy, např. GLONASS, BeiDou a Galileo.

3.1.2 Světový geodetický systém

Pro jednoznačné určení polohy jakéhokoliv bodu na zemském povrchu byl v roce 1984 zaveden Ministerstvem obrany USA Světový geodetický systém označovaný ***WGS-84*** *(World Geodetic System)* [9].

K určování polohy bodu na Zemi je využíván systém souřadnic – zeměpisné šířky a zeměpisné délky. Obě tyto souřadnice představují úhly, jejich jednotkami jsou tudíž úhlové stupně (značka °).

Zeměpisná šířka představuje úhel, který svírá jím procházející osa Země s rovinou rovníku. Rovník dělí Zemi na severní a jižní polokouli a je kolmý k ose otáčení Země. Severní a jižní pól má zeměpisnou šířku 90°.

Zeměpisná délka představuje úhel, který svírá jím procházející osa Země s rovinou, nultého poledníku. Tento poledník prochází Královskou greenwichskou observatoří v Anglii a rozděluje Zemi na východní a západní polokouli. Poblíž poledníku se zeměpisnou délkou 180° se nachází *datová hranice Země*.

3.2 Ukládání uživatelských dat

Pro aplikaci, která umožňuje registraci a autentizaci uživatelů a práci s jejich uživatelskými daty, je třeba zajistit bezpečný způsob ukládání těchto dat. Data ukládána takovým způsobem nebudou smazána, pokud dojde k vypnutí či odinstalování aplikace, nebude-li jejich smazání explicitně vyžadováno.

Tato kapitola popisuje technologie využité pro implementaci a využití webové aplikace, která ukládání uživatelských a jiných dat umožňuje díky využití relační databáze a aplikačního rozhraní.

3.2.1 Webový server

Pro ukládání dat aplikace, které je nutné uchovat i po jejím vypnutí, či je sdílet mezi jednotlivými uživateli a jejich mobilními zařízeními, je třeba vytvořit pro aplikaci vhodnou databázi. Ta bude obsluhována serverovou aplikací, která poskytuje nástroje pro práci s modelovými entitami (abstrakce, validace přijatých uživatelských dat, ...), autentizaci uživatele a kontrola jeho rolí a práv a zpětné odesílání dat do aplikace po jejich zpracování.

Protože většina podobných aplikací napříč internetem je postavena v skriptovacím jazyce PHP, byl tento jazyk zvolen i pro tuto aplikaci. Použita byla nejnovější verze jazyka PHP v čase začátku vývoje aplikace (PHP 7.3.13). Jelikož není vhodné používat pro aplikaci většího rozsahu nestrukturovaný zdrojový kód, bylo třeba zvolit vhodný fra-

mework pro vývoj aplikace. Takových frameworků nad jazykem PHP existuje celá řada, mezi nejrozšířenější ve světě nebo v ČR patří Yii2, Symfony, Laravel nebo Nette.

3.2.2 Relační databáze

Součástí serverové aplikace je i relační databáze MariaDB. Tento druh databáze je založený na relační databázi MySQL, která poskytuje dostatečnou rychlost provádění databázových dotazů a výkon vhodný pro tuto mobilní aplikaci.

O vytvoření jednotlivých relací v relační databázi se stará zvolený PHP framework, který je schopen pomocí *databázových migrací* vytvořit kompletní strukturu databáze včetně relačních vazeb a cizích klíčů. Další funkcí migrací je naplnění databázi požadovanými základními či testovacími daty. Dále je v režii frameworku tvorba databázových dotazů v jazyce SQL. Těmito dotazy jsou při běhu aplikace data získávána z databáze pro další využití. Při zpracování výsledků funkcí serverové aplikace jsou pomocí těchto dotazů data do databáze zapisována.

3.2.3 Aplikační rozhraní (API)

Sdílení dat mezi mobilní aplikací a webovým serverem probíhá prostřednictvím aplikačního rozhraní. To umožňuje předávání serializovaných a strojově čitelných dat jedním i druhým směrem. Nejčastěji dochází k předávání dat prostředicvím HTTP požadavků. Tím je zajištěna spolehlivá a účinná komunikace mezi oběma stranami. Existují dva hlavní typy (architektury) aplikačního rozhraní používaného pro komunikaci typu *klient-server (viz kapitola 4.2)*:

- **SOAP *(Simple Object Access Protocol)***: Tento protokol využívá pro serializaci formát XML. Struktura zprávy dodržuje standardizovanou šablonu. Tento způsob komunikace je vhodný pro systémy s přenosem velkého množství dat a jeho zpracování jednoduššími systémy může být kvůli jeho složitosti pomalejší [12].
- **REST *(Representational State Transfer)***: Používá se pro přenos dat nebo stavů aplikace, ačkoliv je každý požadavek bezstavový. Pro serializaci bývá často použitý jednodušší formát, např. RSS nebo JSON (JavaScript Object Notation). Struktura serializované zprávy je definována pouze v rámci aplikace. Pro svoji jednoduchost a efektivitu i v menších aplikacích je tento typ aplikačního rozhraní vhodný pro využití v této práci [15].

Pro bezpečné a bezproblémové používání aplikace závislé na komunikaci prostřednictvím aplikačního rozhraní je třeba toto rozhraní pečlivě dokumentovat. Dokumentaci API popisuje kapitola 6.3.

3.3 Mobilní aplikace

Mobilní aplikace slouží k vizualizaci dat a jejich zobrazení koncovému uživateli. Tato data jsou čerpána z dvou zdrojů, a to z databáze webového serveru, odkud jsou získávána prostřednictvím aplikačního rozhraní, a z vnitřní paměti mobilního zařízení, kam k jejich ukládání dochází za běhu aplikace.

Mobilní aplikace jsou v dnešní době vyvíjeny pro dvě majoritní mobilní platformy Android a iOS. Obě tyto platformy vyžadují implementaci aplikací pomocí jimi podporovaných nástrojů a programovacích jazyků. O těchto nástrojích pojednávají následující kapitoly.

3.3.1 Vývoj pro jednotlivé platformy

Pro vývoj aplikace určené pouze pro jednu cílovou mobilní platformu je nejvhodnějším řešením použití nástrojů, které jsou pro tento účel přímo určeny tvůrci těchto platforem. Tím je zajištěna možnost využití všech dostupných funkcí zařízení a jeho operačního systému.

Aplikace pro platformu Android byly od jejího představení vyvíjeny v jazyce Java. V roce 2011 představila společnost JetBrains jazyk Kotlin, který se poté rozšířil mezi vývojáře mobilních aplikací pro platformu Android a stal se jedním z oficiálních jazyků pro jejich vývoj. Editorem zdrojového kódu (IDE) pro vývoj aplikací pro Android je program **Android Studio**. Tento program umožňuje intuitivní práci se zdrojovým kódem a kompilaci projektu do spustitelné aplikace. Pro testování aplikací je možné využít emulátory zařízení s operačním systémem Android, které jsou také součástí programu Android Studio.

Vývoj aplikací pro mobilní platformu iOS s sebou přináší oproti vývoji pro platformu Android některá omezení. Zatímco při vývoji aplikací pro systém Android nejsme omezeni operačním systémem počítače, vývoj aplikace pro iOS je třeba provádět za použití programu **Xcode**. Ten lze nainstalovat pouze na počítače značky Apple, na nichž běží operační systém **MacOS**. Tento program poté umožňuje, podobně jako Android Studio, kompilaci zdrojového kódu a jeho spuštění na emulátorech mobilních zařízení. Další funkcí programu je napojení na Apple AppStore, které slouží jako internetový obchod s aplikacemi pro telefony iPhone a další chytrá zařízení produkovaná společností Apple. Aplikace pro platformu iOS je možné vyvíjet v jazyce Objective-C, který ale později nahradil jazyk Swift, který byl vytvořen společností Apple a je touto společností nadále vyvíjen a udržován.

3.3.2 Nástroje pro multiplatformní vývoj

Kromě programovacích jazyků a nástrojů pro vývoj aplikací dedikovaných pro jedinou mobilní platformu lze využít některá z existujících řešení vývoje aplikací pro obě platformy. Tato řešení umožňují kompilaci jediného zdrojového kódu na aplikace, které je možné spustit na zařízeních běžících na obou mobilních platformách, tedy Android i iOS.

Nejznámějšími z nástrojů, které multiplatformní vývoj aplikací umožňují, jsou **React Native** a **Flutter**. Nástroj React Native využívá programovací jazyk odvozený od jazyku JavaScript a knihovny React.js a překladu elementů vytvořených v tomto jazyce na nativní elementy jednotlivých platforem. Flutter využívá programovací jazyk Dart a narozdíl od React Native vytváří jednotlivé elementy svépomocí a vývojář tak ovládá vzhled každého prvku nezávisle na nativních prvcích jednotlivých platforem.

Využití multiplatformního vývoje pro tvorbu mobilních aplikací s sebou ale přináší omezení v podobě neumožnění plného využití všech možností a funkcí zařízení. Příkladem je problematická implementace běhu aplikace na pozadí. Tato omezení lze ale částečně vyřešit implementací potřebných funkcí pomocí nativního jazyka, tedy Java nebo Kotlin pro Android a Objective-C či Swift pro iOS. Tyto funkce lze pak začlenit do zdrojového kódu multiplatformního projektu.

Ačkoliv při vytváření zdrojového kódu aplikací v multiplatformních nástrojích nejsme omezeni operačním systémem počítače použitého pro vývoj aplikace, kompilace aplikací pro platformu iOS využívá program Xcode. Tím přetrvává omezení, díky kterému je aplikace pro iOS nutné vyvíjet (nebo alespoň kompilovat) na počítačích s operačním systémem MacOS.

3.4 Nástoje použité při vývoji aplikace

V této kapitole jsou blíže popsány programovací jazyky, frameworky a nástroje, které byly vybrány pro implementaci mobilní aplikace vyvíjené v této práci.

3.4.1 Flutter a Dart

Z výše zmíněných nástrojů, umožňujících multiplatformní vývoj mobilních aplikací, byl zvolen Flutter [6]. Flutter je sada nástrojů, která je vyvíjená společností Google a je určená k vytváření uživatelských rozhraní. Není to nástroj pouze pro vytváření mobilních aplikací, lze jej použít také pro tvorbu webových a desktopových aplikací [10]. Pro všechny tyto platformy je možná kompilace jediného zdrojového kódu, proto je Flutter vhodný mimo jiné pro multiplatformní vývoj mobilních aplikací. Mezi ostatními nástroji pro podobné využití je Flutter relativně nový, jeho první vydání v alfa verzi proběhlo v květnu roku 2017.

Pro přípravu projektu mobilní aplikace je Flutter třeba stáhnout z jeho webových stránek jako balík nástrojů obsahující zdrojové kódy, podporu programovacího jazyka Dart a konzolové aplikace, které umožňují samotný projekt inicializovat. V neposlední řadě instalační sada obsahuje konzolovou aplikaci `flutter-doctor`, která před zahájením i v průběhu vývoje kontroluje nastavení počítače a napojení nástrojů obsažených v balíku Flutter na sady SDK *(Software development kit)* jednotlivých platforem, pro které je aplikace vyvíjena.

Dart

Dart je programovací jazyk optimalizovaný pro tvorbu uživatelských rozhraní. Byl představen v Dánsku v říjnu roku 2011.

Stejně jako samotný nástroj Flutter je i jazyk Dart vyvíjen společností Google. Jedná se o třídní, objektově orientovaný jazyk. Syntax jazyka je odvozena od jazyků JavaScript, C# a Java. Pokud je využíván pro tvorbu uživatelského rozhraní webových aplikací, je možné ho kompilovat do jazyka JavaScript. Při využití pro vývoj mobilních aplikací v nástroji Flutter je překládán na nativní strojový kód mobilních zařízení platforem Android a iOS.

Jako IDE editor zdrojového kódu byl použit program Visual Studio Code od společnosti Microsoft. Tento volně dostupný program poskytuje možnost instalace pluginů umožňujících zvýraznění syntaxe jazyka Dart a spuštění debugu aplikace přímo z editoru. Pro verzování projektu byl opět použit nástroj Git na doméně www.gitlab.com.

3.4.2 Yii2 a DactylCMS

Z frameworků pro tvorbu webových aplikací ve skriptovacím jazyce PHP byl pro vývoj webového serveru použit framework Yii2. Tento framework poskytuje množství užitečných funkcí pro abstrakci databázových entit do modelů, validaci uživatelských dat a jejich datových typů. Dále poskytuje velmi dobré řešení směrování HTTP požadavků v aplikaci i širokou konfigurovatelnost všech svých funkcí podle potřeb vývojáře a projektu. V neposlení řadě je výhodou tohoto frameworku nástroj pro vytvoření aplikačního rozhraní REST, obsahující širokou škálu nástrojů pro jeho nastavení.

Pro vytvoření lokálního vývojového prostředí vhodného pro běh webové aplikace (server, databáze) byl použit nástroj *Docker*. Ten umožňuje vytvořit virtuální kontejner, jehož

prostředí odpovídá prostředí produkčního serveru a domény, na které bude webová aplikace následně zveřejněna pro umožnění HTTP komunikace.

Pro psaní zdrojového kódu byl použit IDE editor PHPStorm od firmy JetBrains. Součástí editoru je plugin poskytující možnost pozastavení programu *Xdebug* pro případné debugování chyb.

Yii2

Yii2 [17] je open source webový framework napsaný v skriptovacím jazyce PHP. Je založen na návrhovém vzoru **MVC** (viz kapitola 3.4.3) a vyznačuje se vysokým výkonem a škálovatelností, díky čemuž je vhodný pro tvorbu rozsáhlých webových aplikací všeho druhu. Navzdory této skutečnosti je však jeho zdrojový kód psán srozumitelně a co možná nejjednodušeji. Jeho výhodou je dále podrobná a přehledná dokumentace, která umožňuje bezproblémový náhled na jakékoliv výchozí funkce frameworku. Framework je využíván širokou komunitou vývojářů, což umožňuje snadné dohledání řešení možných problémů při vývoji aplikací.

DactylCMS

DactylCMS je modulární systém pro správu obsahu *(Content management system)*. Systém je kontinuálně vyvíjen a funguje jako nástavba nad frameworkem Yii2. Tato nástavba umožňuje definici uživatelských rolí a vytvoření obsahu webové aplikace pomocí dynamicky přidávaných a odebíraných modulů, které slouží jako rozšíření funkcionality webové aplikace. Nástavba dále poskytuje mnohá vylepšení základní struktury frameworku Yii2 pro práci s jeho prvky a komponentami.

Ve svém výchozím stavu je webová aplikace postavená na systému DactylCMS rozdělena na několik výchozích modulů. Těmito moduly jsou:

- **Společné jádro aplikace**: V tomto modulu se nachází základní definice modelů a konfigurací webové aplikace, které budou používány a rozšiřovány jednotlivými dalšími moduly. Nejedná se o koncový modul, jinak řečeno neobsahuje Views ani Controllery a uživatel s tímto modulem pracuje pouze prostřednictvím dědičnosti modelových entit dalších modulů.

- **Konzolová aplikace**: Modul spouštěný z ovládací konzole serveru, na kterém je webová aplikace vystavena. Pomocí tohoto modulu dochází k spouštění migrací či ovládání plánovaných a opakujících se úloh. Modul neposkytuje vizuální výstup (kromě textového výstupu na ovládací konzoli), a proto neobsahuje soubory typu View.

- **Administrace obsahu**: Pro ovládání obsahu stránek webové aplikace, ovládání uživatelů a jejich rolí a uživatelsky přívětivý a čitelný výpis instancí jednotlivých modelů slouží webová administrace. V této administraci se nachází také přehledné ovládací panely pro nastavení různých prvků webové aplikace a správa jednotlivých uživatelských rolí. Pro možnost přihlásit se do administrace je třeba, aby uživatel disponoval rolí s právy administrátora.

  Obsah webové aplikace je v administraci logicky rozdělen na kategorie podle jednotlivých modelů. V jednotlivých kategoriích je možné instance modelů prohlížet, vkládat, upravovat i mazat. Výchozím pohledem na kategorii je tabulkový výpis všech záznamů v databázové tabulce modelu, který je pro lepší přehlednost i z důvodů

výkonu aplikace rozdělen a stránkován. Při vkládání a úpravě jednotlivých záznamů je administrátor přesměrován na stránku s formulářem, který obsahuje jednoduché pluginy pro vkládání různých druhů obsahu (text, čísla, data, soubory apod.). Při odeslání tohoto formuláře dochází k validaci poskytnutých dat. V případě validační chyby je uživatel na tuto chybu upozorněn a vyzván k její opravě u příslušného pole pro vkládání obsahu, aniž by byl zbytek formuláře smazán.

- **Frontend webové aplikace**: V případě klasických webových aplikací je do v projektu obsažen modul Frontend. Ten obsahuje definici grafického obsahu, který si koncový uživatel prohlíží ve webovém prohlížeči. Jsou zde umístěny Modely, Controllery i soubory View, které tvoří webovou stránku za použití designu, který byl pro tuto stránku navržen, a obsahu, který byl vytvořen v administraci.

  Protože webová aplikace pro tuto diplomovou práci nevyžaduje implementaci webového frontendu, je tento modul použit pouze pro vytvoření tzv. landing page, tedy stránky odkazující návštěvníka na stažení mobilní aplikace do svého mobilního zařízení.

- **Aplikační rozhraní (API)**: Část webové aplikace, která je naopak pro funkci mobilní aplikace nezbytná, je Aplikační rohraní. Podobně jako v administraci webové aplikace dochází i zde k manipulaci s obsahem databáze, tedy zobrazení, vkládání, úpravě a mazání jednotlivých modelů.

  Na rozdíl od administrace, kde jsou data vyobrazena s důrazem na vizuální stránku a snadnou čitelnost pro uživatele (administrátora), aplikační rozhraní poskytuje data serializovaná do struktury JSON, která je vhodná pro reprezentaci jednotlivých modelů a hodnot jejich atributů. Tato serializace umožňuje snadné zpracování přijatých dat v mobilní aplikaci, jejíž nástroje dokáží data serializovaná ve formátu JSON zpracovat a převést na modelové entity ve své vnitřní struktuře. Veškerá práce s daty pak probíhá tam a webová aplikace je opět kontaktována až ve chvíli, kdy je změny třeba uložit do databáze.

  Pomocí aplikačního rozhraní dochází i k autentizaci uživatele. Z mobilního zařízení jsou odeslány přístupové údaje uživatele, které jsou na serveru ověřeny, a jsou navrácena kompletní uživatelská data spolu s vytvořeným přístupovým klíčem (access token), kterým se během veškeré další komunikace se serverem uživatel identifikuje.

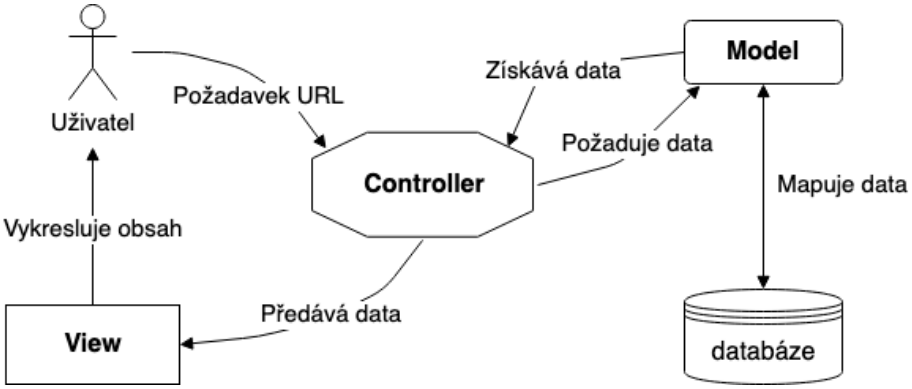
3.4.3 Návrhový vzor MVC

Tento návrhový vzor je pro použití při tvorbě webových aplikací v PHP velmi oblíben. Princip jeho fungování umožňuje oddělení programové logiky od vizuální části aplikace [19]. Logická část se nachází v *modelech*, vizuální stránka aplikace je definována v jednotlivých *pohledech* (view). Rozhraní mezi oběma těmito částmi aplikace obstarávají *ovladače* (controller).

Model

Třídy tohoto typu slouží jako základní stavební kameny webové aplikace postavené na návrhovém vzoru MVC. Tyto třídy poskytují abstrakci entit použitých v aplikaci.

Díky objektově-relačnímu mapování (ORM) je umožněna práce se záznamy v databázi aplikace jako s objekty, které jsou instancemi těchto tříd. Každý model obsahuje definici



Obrázek 3.1: Schéma návrhového vzoru MVC.

svých atributů, povinnosti vyplnění těchto atributů při vkládání do databáze a kontroly jejich datových typů. Tato pravidla jsou definována v metodě `rules()` každého modelu.

Modely dále poskytují možnost vytvoření relačních vazeb odpovídajících vazbám v databázovém modelu aplikace. Modely z těchto vazeb jsou přístupné pomocí metod `hasOne()` a `hasMany()` u relací 1:1 nebo 1:M, případně `hasMany()->via()` u relací M:N, které vyžadují existenci vazební tabulky v databázi, jejíž prostřednictvím je relace M:N transformována na posloupnost relací M:1 a 1:N.

Chování vazby M:N je definováno různě v jednotlivých frameworcích využívajících návrhový vzor MVC. V použitém frameworku Yii2 je vhodným postupem definice modelu, který svou strukturou odpovídá vazební tabulce. U některých jiných frameworků (např. Laravel) definice vazebního modelu není nutná, pokud jsou správně dodržovány konvence pojmenování relačně vázaných modelů a jejich atributů.

View

Tyto soubory obsahují definici vzhledu webové aplikace přístupné koncovému uživateli. Jedná se většinou o HTML kód s prvky jazyka PHP, které jsou použity zejména pro tvorbu dynamického obsahu. Syntax vloženého kódu v jazyce PHP se může lišit v různých frameworcích v závislosti na použité nástavbě nad čistým jazykem PHP (příkladem může být nástavba *blade.php* používaná ve frameworku Laravel [13]). V běžných případech je kód v jazyce PHP vložen do souborů view pro:

– výpis hodnoty proměnných nebo návratové hodnoty funkcí závislých na hodnotách získaných z uživatelských vstupů,

```
1 // ...
2 <?php echo $variable; ?> /* or */ <?= $variable ?>
3 // ...
```

Algoritmus 3.1: Výpis hodnoty ve views

– vytvoření podmínky pro vykreslení části stránky,

```
// ...
<?php if (condition): ?>
    /* content rendered on condition true */
<?php else: ?>
    /* content rendered on condition false */
<?php endif; ?>
// ...
```

Algoritmus 3.2: Podmínka ve views

– vykreslování v cyklu.

```
// ...
<?php foreach ($array as $key => $value): ?>
    /* iteratively rendered content */
<?php endforeach; ?>
// ...
```

Algoritmus 3.3: Foreach ve views

Hodnoty proměnných je nutné před vykreslením obsahu view stránky definovat v příslušné akci controlleru, který spravuje daný požadavek.

Protože aplikace Flox, vyvíjená v této práci, využívá pouze aplikační rozhraní webového serveru a nevyžaduje vizuální část na webu, nejsou v této práci soubory view použity.

Controller

Třídy tohoto typu poskytují rozhraní mezi soubory view, přístupné uživateli, a programovou logikou v modelech, která je záměrně před uživatelem skryta. Jejich hlavní částí je zpracování HTTP požadavku a jeho směrování do příslušné akce (metody této třídy), která tento požadavek obslouží. Před obsloužením požadavku může proběhnout kontrola, zda má uživatel právo přístupu k této akci. Příkladem důvodu zamítnutí přístupu je neautentizovaný uživatel přistupující do sekce určené přihlášeným uživatelům. Dalším kritériem ověření je přístup k akci controlleru použitím správné HTTP metody (GET, POST, ...). Toto chování lze nastavit v metodě `behaviors()`, která definuje pro každou akci podmínky přístupu uživatelů.

Funkcí controlleru je dále vyhledání modelu, který je pro daný uživatelský požadavek použit, a rozhodnutí o metodě tohoto modelu, která bude provedena. Příkladem jsou CRUD operace, tedy metody typu *create, read, update, delete*. Controller v takovém případě vyhledá požadovaný model pomocí třídy typu `ActiveQuery`, která vytvoří a provede SQL dotaz, poskytne mu uživatelská data, kterými se model sám naplní. Na konci procedury zavolá metodu modelu, jež změny promítne do relační databáze a tím je uloží. Při vyhledávání

kolekce více modelů je použita třída `DataProvider`, která poskytuje nástroje pro práci s kolekcemi modelových entit. Všechny operace prováděné nad modely jsou definovány přímo v daném modelu. Controller pouze invokuje metody, v nichž jsou definovány.

Kapitola 4

Návrh a architektura aplikace

Před zahájením implementace mobilní aplikace je třeba vytvořit její návrh. Od tohoto návrhu se bude odvíjet celková architektura aplikace, a to jak její mobilní část, tak webová serverová aplikace. Úkolem webové aplikace je uchování uživatelských dat a zpracování některých dalších úkolů. Součástí tohoto návrhu je definování vzhledu aplikace a jejího uživatelského rozhraní.

Tato kapitola popisuje architekturu aplikace a jejích součástí. Dále je popsán způsob řešení některých funkcí aplikace, které napomáhají k jednoduchosti a bezpečnosti jejího používání. V neposlední řadě popisuje kapitola proces vytváření grafického návrhu mobilní aplikace.

4.1 Modelové abstrakce aplikačních entit

Tato sekce obsahuje výčet jednotlivých modelových entit aplikace, jejich hlavních prvků a významu v kontextu celé aplikace. Většina těchto entit je použita shodně v mobilní i serverové části aplikace, kde jsou jejich data ukládána do relační databáze.

Uživatel *(User)*

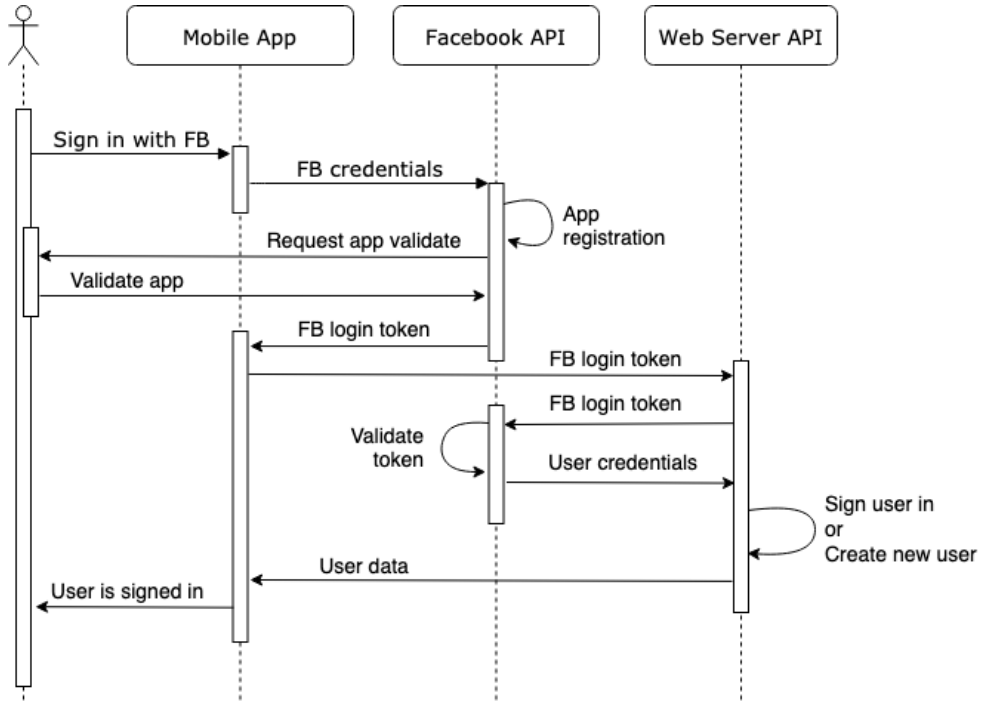
Hlavní entitou aplikace je Uživatel. Tato entita existuje v obou částech aplikace (server i mobilní aplikace) a poskytuje veškerou funkcionalitu spojenou s autentizací koncového uživatele aplikace. Většina z funkcí autentizace probíhá na serveru po odeslání uživatelských dat z mobilní aplikace HTTP metodou POST. Těmito funkcemi jsou:

- **Registrace**: Mezi požadovanými daty jsou e-mail a heslo, dále jméno a příjmení uživatele. Server nejprve ověří, zda se daný e-mail v databázi ještě nevyskytuje, a validuje poskytuná data pomocí definovaných validátorů. Pokud jsou data v pořádku, vytvoří záznam uživatele v databázi a zpět do mobilní aplikace odesílá odpověď na požadavek v podobě JSON struktury. Ta obsahuje uživatelská data a vygenerovaný přístupový klíč (access token), kterým uživatel autorizuje všechny své následující požadavky.
- **Přihlášení**: Uživatel zadává do mobilní aplikace přihlašovací údaje, které zadal při své registraci. Po odeslání údajů je nejprve ověřeno, zda v databázi existuje záznam uživatele s danou e-mailovou adresou. Poté je ověřeno, že heslo odpovídá heslu uloženému v databázi. To je z bezpečnostních důvodů šifrováno hashovací funkcí, proto je třeba před ověřením hesla předat kontrolovaný řetězec stejné hashovací funkci. Po

úspěšném přihlášení je mobilní aplikaci odeslána odpověď shodná s odpovědí při registraci, tedy data uživatele a přístupový klíč.

- **Obnova zapomenutého hesla**: Pokud je uživatel registrován, ale nezná své přihlašovací heslo, má možnost vygenerování registračního tokenu. Tento token je odeslán uživateli na e-mail a s jeho pomocí lze přistoupit k vytvoření nového hesla. Registrační token je platný pouze jednou, a to pouze po omezenou dobu. Po přístupu do webové aplikace je z databáze odstraněn.

- **Registrace pomocí sociálních sítí**: Uživatel neodesílá e-mail a heslo, ale pouze klíč poskytnutý použitou sociální sítí. Server pomocí tohoto klíče vyžádá od sociální sítě všechna potřebná uživatelská data. Odpověď serveru je shodná jako při běžné registraci. V administraci uživatelského profilu je poté možné registraci do aplikace Flox libovolně spravovat či zrušit. Průběh registrace pomocí sociální sítě Facebook je znázorněn diagramem na obrázku 4.1.



Obrázek 4.1: Přihlášení pomocí Facebook.

Uživatelské zařízení *(UserDevice)*

Do této entity jsou ukládány technické údaje o mobilním zařízení, které uživatel používá k přihlášení do mobilní aplikace. Tato data mohou být později využita pro vytváření statistik uživatelů (poměr uživatelů iOS a Android v různých verzích, průměrný počet zařízení na uživatele, ...).

Součástí těchto dat je dále klíč do aplikace Firebase od firmy Google. Díky propojení s touto aplikací lze do vybraných mobilních zařízení odesílat zprávy, které jsou na zařízeních zobrazeny jako notifikace. Více o využití Firebase pro push notifikace se nachází v kapitole 6.4.

Uživatelská lokace *(UserLocation)*

Tato entita je vytvořena pro uživatele při vytváření záznamu o jeho registraci a s modelem User je vázána relací 1:1. Slouží k ukládání dat polohy mobilního zařízení uživatele.

Při každém odeslání HTTP požadavku metodou POST jsou data v databázi aktualizována a skupinám, jejichž je uživatel členem, jsou odesílána pouze nejnovější data o aktuální poloze. Ukládanými daty o poloze jsou koordináty (zeměpisná šířka a délka) a čas zaznamenání polohy. Historie pohybu uživatele není pro mobilní aplikaci potřebná, a tudíž není ukládána.

Přístupový klíč *(UserAccessToken)*

Tato entita slouží pro ověření totožnosti uživatele při každém požadavku z mobilní aplikace na vzdálený server. Protože tento server je bezstavový, neukládá žádná data o aktuálně přihlášeném uživateli. Vytvořený přístupový klíč je při každé autentizaci přiřazen uživateli a aplikaci je předán pomocí náhodně vytvořené série alfanumerických a speciálních znaků. Ke každému uživateli může být přiřazeno více takových přístupových klíčů. Každý se váže k přihlášení z jiného zařízení. Mezi modely User (uživatel) a UserAccessToken (klíč) tedy platí relace 1:N.

Přístupový klíč dostává při vytvoření údaj o délce jeho platnosti. Po vypršení platnosti není klíč serverem přijat a na požadavek mobilní aplikace je odpovězeno chybou s HTTP kódem 401, který odpovídá neautorizovanému požadavku.

Mobilní aplikaci je při autentizaci uživatele odeslán pouze token přístupového klíče (náhodně vytvořená série znaků) a časové razítko doby vypršení platnosti klíče. Aplikace tento token poté přidá do hlavičky každého HTTP požadavku na server a veškeré ověření totožnosti uživatele je podle tohoto tokenu prováděno až na serveru.

Skupina *(Group)*

Tuto entitu vytváří uživatel po přihlášení v mobilní aplikaci. Skupina obsahuje svá konkrétní data a nastavení, které musí uživatel odeslat na server. Těmito daty jsou:

- **název skupiny**
- **doba platnosti skupiny** – každá skupina je platná pouze po omezenou dobu
- **přístupový kód skupiny** – tento kód je nutné zadat na straně uživatele, který se chce připojit ke skupině

Po přidání uživatele ke skupině jsou od vzdáleného serveru vyžádána data o poslední známé poloze jeho zařízení. Podle této polohy je uživatel zobrazen na mapových obrazovkách všech členů skupiny. Uživatel může skupinu opustit ještě před ukončením její platnosti. Ostatním uživatelům tím přestanou být viditelná data o jeho aktuální poloze.

Po vypršení doby platnosti skupiny tato skupina zcela nezaniká, ale její data jsou uchována pro případné prodloužení její platnosti. Po prodloužení platnosti jsou jednotliví členové o této skutečnosti notifikováni a mohou se rozhodnout, zda chtějí nadále být členy skupiny a sdílet svoji polohu, nebo si přejí být ze skupiny odstraněni.

Místo setkání *(MeetingPoint)*

Majitel (tvůrce) každé skupiny nebo její členové můžou přidat místo setkání. Toto místo je zvoleno na mapě pomocí jeho koordinátů (zeměpisná šířka a délka) a odesláno na server

spolu s plánovaným časem setkání na tomto místě. Čas setkání nelze nastavit vyšší, než je doba platnosti skupiny.

Členové skupiny jsou o vytvoření místa setkání notifikování. Uživatelé, kteří se ke skupině připojí po vytvoření místa srazu, jej vidí ve výpisu na obrazovce obsahující detaily o skupině. Dále jsou uživatelé notifikováni o blížícím se času setkání.

Místa setkání jsou pro případ nutnosti pozdějšího dohledání i po uplynutí doby setkání zobrazena na mapě, zanikají až se zanikající skupinou.

4.2 Komunikace s aplikačním rozhraním

Aplikace vyvíjená v této práci je rozdělena na webový server a mobilní aplikaci. Jak spojení *webový server* napovídá, bude využívána architektura *klient-server*. Klienty jsou zde mobilní aplikace nainstalované na množství mobilních zařízení. Komunikace klienta a serveru probíhá prostřednictvím internetu. Vzájemné dorozumívání mezi oběma stranami komunikace zajišťuje aplikační rozhraní.

Jak bylo řečeno v předchozích kapitolách, webová a částečně i mobilní aplikace je implementována pomocí návrhového vzoru *Model, View, Controller*. Díky tomu je umožněno využívat modelové abstrakce reálných entit, které provádí jisté operace ať na straně serveru, nebo na straně klienta.

Rozdělení těchto operací a jejich svěření jedné či druhé straně závisí od jejich podstaty a zdrojů nutných k jejich vykonání. Operace, jejichž součástí je předpokládaná práce s databází aplikace, jsou prováděny na serveru a jejich volání probíhá prostřednictvím aplikačního rozhraní. Operace, které lze naopak vykonat bez přístupu k databázi či je jejich vykonání možné pouze s dříve načtenými daty z jiné operace, jsou prováděny přímo na mobilním zařízení, tedy na straně klienta.

Příkladem kontrastu mezi takovými operacemi jsou různé typy validace formuláře pro registraci uživatele pomocí e-mailové adresy:

- **Validace formátu adresy**: E-mailová adresa zadaná uživatelem při registraci má předem specifikovaný formát, který musí být dodržen. Jeho dodržení lze ověřit porovnáním uživatelského vstupu s patřičným regulárním výrazem. Tuto operaci lze provést bez přístupu k datům v databázi, proto je prováděna přímo v mobilní aplikaci.
- **Ověření, zda adresa není používána**: E-mail uživatele musí být pro každého uživatele unikátní, duplicity by vedly k chybě při přihlášení obou uživatelů. Ověření, zda je daná e-mailová adresa používána jiným uživatelem, vyžaduje nahlédnutí do databáze existujících uživatelů, proto je prováděna na serveru.

Operace prováděné na serveru jsou vždy vyvolány požadavkem na aplikační rozhraní, který obsahuje příslušná vstupní data. Po provedení operace je serverovou aplikací vrácena návratová hodnota operace. Ta je prostřednictvím aplikačního rozhraní serializována a odeslána zpět do mobilní aplikace, kde proběhne její zpracování. Více o aplikačním rozhraní se nachází v kapitole 6.

4.3 Řešení bezpečného sdílení polohy

V této kapitole jsou popsány způsoby a principy chování aplikace, které umožňují bezpečné sdílení polohy uživatelova mobilního zařízení. Díky těmto principům je zachováno nejvyšší

možné soukromí uživatele a zabráněno nechtěnému sledování jeho aktuální polohy. Zároveň není omezená použitelnost aplikace pro skupinu více uživatelů.

4.3.1 Zachování soukromí uživatele

Jak bylo specifikováno v definici funkcí aplikace (kapitola 2.4), klade si tato aplikace za cíl zachování co možná nejvyšší úrovně soukromí uživatele. Aby bylo zamezeno zneužití této aplikace k nechtěnému nebo nežádoucímu sledování jeho lokace a pohybu, byly při návrhu pro implementaci zvoleny tyto funkce a praktiky:

- **Nelze vyhledat uživatele**: Aplikace nenabízí žádnou možnost, která umožňuje jednomu uživateli vyhledat jiného, který je v aplikaci registrován. Pozvání do skupiny lze provést pouze prostřednictvím osobní komunikace nebo s využitím libovolných dostupných komunikačních kanálů (SMS, sociální sítě, e-mail, ...).
- **Omezená doba platnosti skupiny**: Aplikace neumožňuje vytvořit skupinu s delší dobou platnosti, než je jeden měsíc. Po uplynutí této doby nelze sledovat lokaci členů skupiny. Vlastníkovi skupiny je sice umožněno prodloužit dobu platnosti skupiny, ale při takové úpravě jsou všichni její členové upozorněni notifikací.
- **Aktualizace pouze za běhu aplikace**: Poloha uživatele je odesílána na server pouze tehdy, když je aplikace na jeho mobilním zařízení spuštěná. Při běhu na popředí je poloha aktualizována vždy, při běhu na pozadí lze aktualizaci polohy vypnout v obrazovce nastavení. Pokud probíhá aktualizace polohy na pozadí, je o tom uživatel neustále uvědomován nesmazatelnou notifikací v oznamovacím centru systému Android nebo obarvením notifikační lišty v systému iOS.
- **Nutnost autentizace uživatele**: Příslušnost k dané skupině je ověřována webovým serverem a pouze přihlášený uživatel, který je členem nebo vlastníkem dané skupiny, může od serveru přijmout data ostatních členů této skupiny.

4.3.2 Užívání aplikace více uživateli

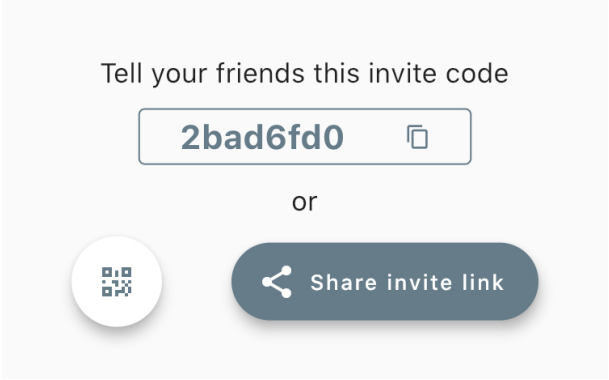
Z podstaty aplikace bylo nutné zajistit její použitelnost pro spolupráci více uživatelů – členů jedné a více skupin. Aby se uživatel mohl členem skupiny stát, musí ji buď vytvořit (tvůrce skupiny je automaticky jejím prvním členem), nebo do ní být pozván jejím tvůrcem či jiným členem, pokud to nastavení skupiny umožňuje.

Každá skupina obdrží při svém vytvoření **zvací kód**. Tímto kódem je pseudonáhodně vytvořený řetězec složený z 8 alfanumerických znaků, které musí uživatel zadat, pokud se chce ke skupině přidat.

Widget aplikace se zvacím kódem je vyobrazen na obrázku 4.2. Zadání zvacího kódu do formuláře na obrazovce pro připojení ke skupině (viz kapitola 4.4.2) je pouze jednou z možností, jak se stát jejím členem. Uživateli je umožněno zkopírovat klíč do schránky zařízení nebo využít ostatní metody pro pozvání uživatele do skupiny. Tyto metody jsou popsány v následujících podkapitolách.

Dynamické odkazy

Zadání zvacího kódu do obrazovky pro připojení ke skupině nebo jeho kopírování a vkládání prostřednictvím schránky zařízení by mohlo být některými uživateli vnímáno jako



Obrázek 4.2: Widget pro pozvání uživatele

problematické či neefektivní řešení. Z tohoto důvodu bylo třeba navrhnout řešení jiné, a to takové, které by uživateli celý proces připojení ke skupině zjednodušilo. Jedním z takových řešení je vytvoření dynamického odkazu pomocí aplikace Firebase [5]. Tyto odkazy může vygenerovat tvůrce skupiny nebo její členové a sdílet je se svými přáteli prostřednictvím jakéhokoliv dostupného komunikačního kanálu. Uživatel, který takový odkaz obdrží, se jeho prostřednictvím dostává do obrazovky pro připojení ke skupině, kde již má předvyplněný zvací kód skupiny, který byl z odkazu extrahován.

Dynamické odkazy jsou zpracovatelné mobilními platformami a slouží jako odkazy do mobilní aplikace. V té lze na základě URL adresy odkazu nebo jeho parametrů definovat chování při použití takového odkazu. Pokud na mobilním zařízení aplikace není nainstalována, je možné uživatele odkázat na příslušnou stránku v Google Play Store (Android) nebo Apple AppStore (iOS), kde má možnost si aplikaci nainstalovat. Při otevření dynamického odkazu v jiném než podporovaném mobilním zařízení se odkaz chová jako běžná internetová adresa, která odkazuje na předem definovanou URL adresu, která je poté otevřena v internetovém prohlížeči.

Tvorba dynamického odkazu je možná přímo z webového rozhraní aplikace Firebase. Zde je nejprve nutné vytvořit doménu, na kterou se budou dynamické odkazy směrovat. Poté lze vytvořit samotný odkaz. Procesem jeho tvorby je uživatel provázen srozumitelným a jasným formulářem, ve kterém je nastaveno chování při otevření odkazu na jednotlivých platformách. Dalšími možnými nastaveními jsou marketingové kampaně, jejichž data jsou zobrazena při možném sdílení na sociálních sítích.

Druhým, častěji používaným způsobem vytvoření dynamického odkazu, je vytvoření požadavku na aplikační rozhraní Firebase přímo z mobilní aplikace. Tento způsob umožňuje vytvořit odkaz s aktuálně potřebnými daty – v tomto případě zvacím kódem pro připojení uživatele ke skupině. Dynamický odkaz vytvořený v aplikaci má, stejně jako při tvorbě ve webovém rozhraní, definované chování při jeho otevření na různých zařízeních. Na rozdíl od prvního způsobu ale není možné explicitně definovat jeho tvar. Doménové jméno URL adresy odkazu stále směruje uživatele na doménu, kterou bylo nutné vytvořit ve webovém rozhraní. Cesta odkazu za prvním znakem „/“ je ale vytvořena automaticky a má podobu náhodného textového řetězce. Tato podoba dynamického odkazu je zkrácenou verzí původně vygenerovaného dynamického odkazu, který všechna data obsahoval v parametrech URL adresy.

QR kód

QR kód *(Quick Response – rychlá odezva)* [1] slouží jako dvourozměrný kód obsahující různá textová data. Tato data jsou kódována do binární podoby a tato podoba je znázorněna pomocí strojově čitelného rastrového obrázku. Tento obrázek má standardizovanou strukturu, díky které je čitelný z libovolného úhlu a dokáže se vypořádat s jistou mírou nečitelnosti části kódu. Díky tomu je to ideálním prostředkem pro sdílení většího objemu dat v grafické či tištěné podobě na malém rozměru. QR kód je lehce rozeznatelný od jiných podobných strojově čitelných obrazců pomocí čtverců umístěných v obou horních rozích a levém dolním rohu (viz obrázek 4.3).

Obrázek 4.3: Ukázka QR kódu

Protože je díky dynamickým odkazům představeným v předchozí kapitole možné sdílet odkaz na přímé připojení uživatele ke skupině, je možné využít QR kódy k podobnému účelu. Do QR kódu je přímo kódován dynamický odkaz s pozvánkou k připojení ke skupině. Mobilní aplikací určenou pro skenování QR kódů pomocí fotoaparátu zařízení a načtení jejich obsahu je poté odkaz načten a pomocí mobilního internetového prohlížeče otevřen. Pokud je již na zařízení nainstalována aplikace Flox, je po načtení QR kódu přímo otevřena obrazovka pro připojení ke skupině s předvyplněným zvacím kódem. U zařízení s operačním systémem iOS disponuje funkcí pro skenování QR kódů přímo nativní aplikace Fotoaparát.

Jak bylo specifikováno v kapitole 2.1, jsou mezi cílovými skupinami uživatelů zájezdové skupiny s průvodcem a třídy na školním výletě. Právě pro tyto skupiny je určena možnost pozvání k připojení ke skupině prostřednictvím QR kódu. Ten je možné již před uskutečněním zájezdu vytisknout a poskytnout účastníkům prostřednictvím informativního letáku nebo vystavením v prostředku hromadné dopravy. Tím je umožněno rychlé hromadné sdílení dynamického odkazu mezi více potenciálních budoucích členů skupiny.

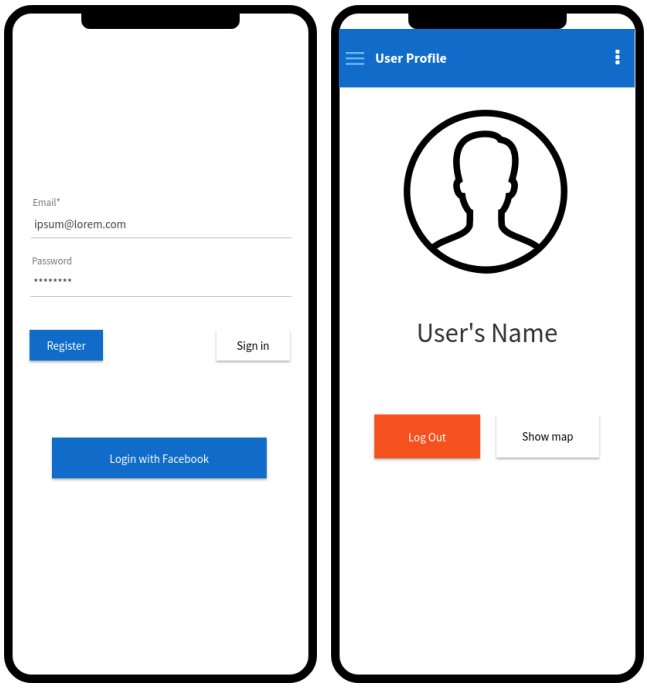
4.4 Grafický návrh aplikace

Před zahájením implementace aplikace je třeba vytvořit její grafický návrh. Ten bude určovat rozložení ovládacích prvků jednotlivých obrazovek aplikace a její celkový styl. Od tohoto grafického návrhu se potom bude odvíjet celková architektura mobilní aplikace, funkce jednotlivých obrazovek a navigace mezi jednotlivými obrazovkami.

Tato kapitola shrnuje proces návrhu grafického uživatelského rozhraní od úplného počátku práce na aplikaci. Jeho součástí je zhotovení základního návrhu obrazovek pro první verzi aplikace a jejich úpravy a rozvoj pro verzi druhou.

4.4.1 Základní návrh uživatelského rozhraní

Základní grafický návrh aplikace a jejího uživatelského rozhraní byl zhotoven před započetím práce na první verzi aplikace. Návrh zahrnoval pouze některé hlavní obrazovky (přihlášení uživatele, hlavní mapovou obrazovku, uživatelský profil). Příklady základních návrhů obrazovek pouze pro rozmístění prvků jsou ukázány na obrázcích 4.4 a 4.5.



Obrázek 4.4: Základní návrh přihlašovací obrazovky a uživatelského profilu

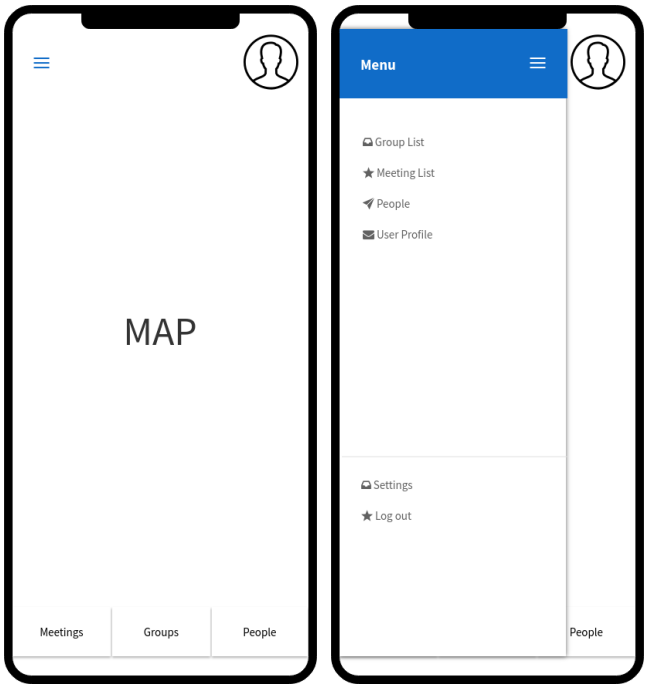
Ostatní obrazovky vznikaly jako deriváty těchto obrazovek nebo jako jednoduché obrazovky zobrazující pouze výčet prvků (seznam skupin, míst, ...).

Bylo tomu tak z toho důvodu, že účelem první verze aplikace bylo spíše otestování a ladění implementovaných funkcí a otestování rozmístění ovládacích prvků těchto obrazovek, než přiblížení se reálnému produktu.

V základním návrhu i ve finální aplikaci tvoří většinu stavebních jednotek designu aplikace prvky stylu *Material Design* [2]. Tento styl představuje open source souborový systém komponent, použitelných pro stavbu esteticky příjemných aplikací s přehledným ovládáním a přijatelnou uživatelskou zkušeností. Byl vyvinut společností Google v roce 2014 a je jedním z primárních zdrojů komponent definovaných v nástroji Flutter (který rovněž vlastní společnost Google).

Díky tomuto systému je možné komponenty řadit do přehledných responsivních šablon, které lze přizpůsobit velikosti obrazovky mobilního zařízení. Prvky jsou designovány tak, že mezi sebou díky barevnému odlišení a stínování vytváří prostorový efekt. Díky tomu má uživatel aplikace pocit, že např. tlačítka se „vznáší“ nad svým pozadím [11].

Tento styl byl použit pro většinu prvku na obou mobilních platformách. Výjimku tvoří některé prvky, jejichž podoba se odvíjí od mobilní platformy, na níž je aplikace spuštěna. Příkladem takových prvků jsou modální dialogy, které jsou v aplikaci na zařízení se systémem Android vyobrazeny pomocí Material Design. Na zařízeních se systémem iOS jsou vykresleny stylem Cupertino Dialog, tedy jako běžná dialogová okna systému iOS, na která



Obrázek 4.5: Základní návrh hlavní obrazovky a menu

jsou uživatelé zařízení iPhone zvyklí. Dalším prvkem, který na zařízeních se systémem iOS využívá styl Cupertino [3], jsou přepínače *(switch buttons)* nebo widgety *(prvky)* pro vložení data a času. Ukázka rozdílů mezi prvky na různých platformách je na obrázku **??**.

Jednotlivé základní prvky jsou spojovány do složitějších komponovaných bloků, které slouží jako stavební prvky jednotlivých obrazovek. Všechny obrazovky použité v aplikaci i ty, které byly použity v první verzi aplikace, ale v druhé verzi byly odstraněny, jsou popsány v následující kapitole 4.4.2.

4.4.2 Obrazovky mobilní aplikace

V této sekci jsou popsány obrazovky mobilní aplikace, jejich prvky a význam v kontextu celé aplikace. Dále je popsána navigace přechodů mezi těmito obrazovkami s pomocnými daty, které si jednotlivé obrazovky mezi sebou předávají.

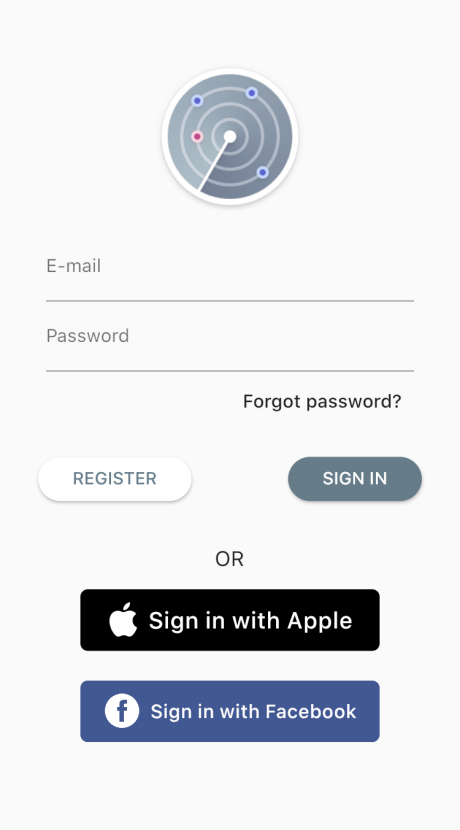
Autentizace uživatele

Tato obrazovka obsahuje jednoduchý prvek sestávající z dvou textových vstupů pro zadání e-mailové adresy a hesla, tlačítka pro přihlášení a tlačítka pro registraci. V případě, že se uživatel již dříve registroval a má platný uživatelský účet, vloží svá přihlašovací data do textových vstupů a odesláním formuláře se přihlásí. Pokud ještě uživatelský profil nemá, tlačítkem pro registraci je navigován do obrazovky registrace. Zde jsou mu postupně nabízeny textové vstupy pro e-mail, heslo, jméno a příjmení.

Obrazovka pro autentizaci uživatele je vykreslena pouze tehdy, není-li uživatel do aplikace registrován, nebo se dříve odhlásil. V opačném případě byla data o uživateli včetně přístupového klíče uložena do vnitřní paměti telefonu a sezení uživatele je automaticky po-

mocí těchto dat obnoveno. Spolu s přístupovým klíčem je taktéž uložena doba jeho platnosti. Po vypršení této doby je nutné se znovu přihlásit pomocí přístupových údajů.

Protože je kromě klasické autentizace e-mailem a heslem použita i registrace pomocí sociální sítě Facebook, je nutné ke schválení aplikace pro Apple AppStore přidat přihlášení pomocí AppleID. Na zařízeních iOS je proto oproti zařízením Android navíc tlačítko pro tuto možnost.



Obrázek 4.6: Obrazovka autentizace uživatele

Mapová obrazovka

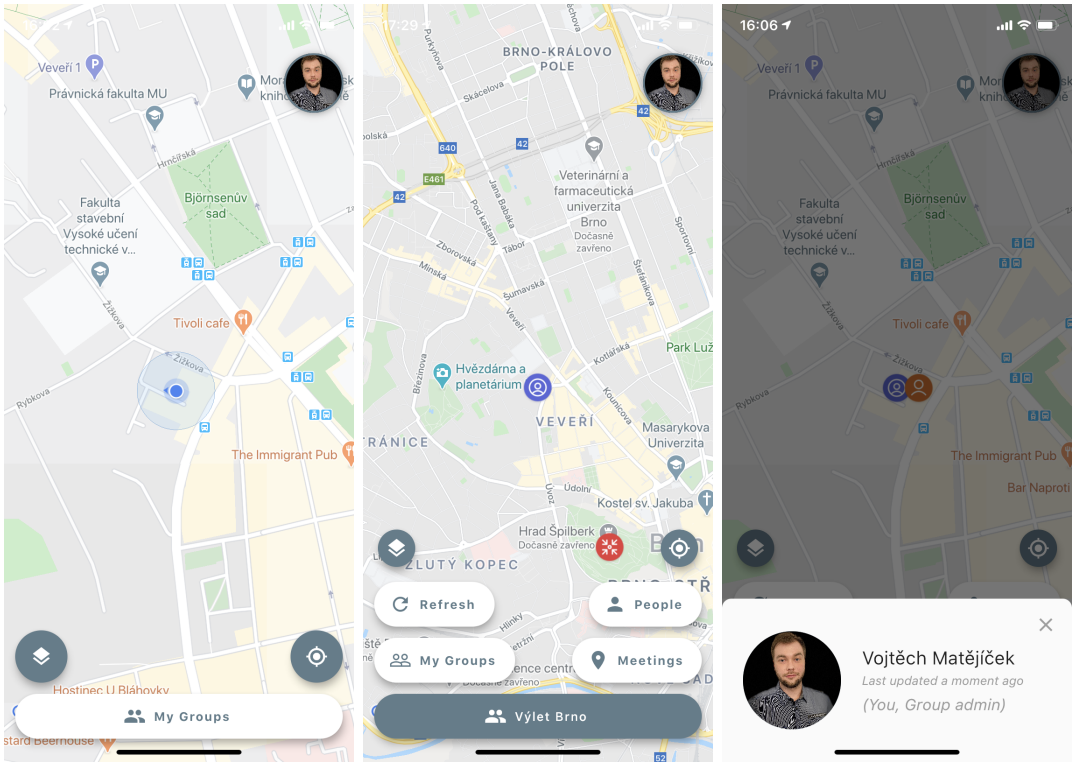
Tato obrazovka je hlavní obrazovkou celé aplikace. Uživatel je sem buď přesměrován po přihlášení do aplikace, nebo se aplikace spustí přímo na této obrazovce v případě, že uživatel byl automaticky přihlášen pomocí uloženého přístupového klíče.

Dominantou obrazovky je prvek obsahující mapový podklad Google Maps. Po spuštění aplikace, kdy ještě uživatel nezvolil, kterou skupinu chce aktuálně zobrazit na mapě, je zobrazena pouze aktuální poloha uživatele. Skupinu pro zobrazení si poté uživatel zvolí z nabídky, která se zobrazí při kliknutí na tlačítko **My Groups**. Toto tlačítko vyvolá vysouvací nabídku obsahující skupiny, jichž je uživatel členem či vlastníkem. Dále nabízí možnost vytvoření nové skupiny či připojení se k existující skupině. Uživatel si může pomocí plovoucího tlačítka zvolit, zda chce vidět normální mapový podklad, nebo letecké snímky zobrazené oblasti. Poslední plovoucí tlačítko slouží k vycentrování mapy na aktuální polohu uživatele.

Jakmile uživatel zvolí skupinu, je mu okamžitě zobrazena mapa s vykreslenými ikonami označujícími polohu, kde se nachází jednotliví členové, a ikonami označujícími místa setkání náležící aktuálně vybrané skupině. Stisknutím kterékoliv ikony na mapě je uživateli vypsán detail daného člena skupiny s údajem o poslední aktualizaci jeho polohy, nebo detail místa setkání. Obrazovka je v pravidelném intervalu obnovována pomocí načítání dat poslední polohy uživatelů. Toto obnovení je možné provést i mimo tento interval, a to pomocí plovoucího tlačítka **Refresh** v panelu navigačních tlačítek. Panel byl dále rozšířen o možnosti zobrazení seznamu míst setkání **Meetings** a seznamu členů skupiny **People**. Posledním tlačítkem je tlačítko odkazující na detail skupiny.

Vysouvací nabídka se seznamem členů je vyvolávána plovoucím tlačítkem People. Nabídka kromě seznamu obsahuje tlačítko pro pozvání nového uživatele k přidání se do skupiny. Po jeho stisknutí je uživateli zobrazen zvací kód skupiny, který může pozvanému uživateli sdělit, nebo mu může odeslat zvací odkaz, a to buď v textové zprávě, nebo jako vygenerovaný QR kód. Při vytváření skupiny si její tvůrce zvolí, zda umožňuje ostatním členům zvát nové uživatele, nebo si tuto možnost ponechá pouze pro sebe.

Posledním prvkem obrazovky je plovoucí tlačítko zobrazující seznam míst setkání skupiny. Tlačítko opět vyvolá vysouvací nabídku obsahující seznam míst a tlačítko pro přidání nového místa. Toto místo může přidat kterýkoliv uživatel, který je členem skupiny, pokud je mu to při jejím vytváření umožněno. V opačném případě může přidávat místa pouze vlastník skupiny. Upravovat a odstranit místo má pravomoc pouze vlastník skupiny a tvůrce tohoto místa. Obrazovka pro vytvoření či editaci místa setkání sestává z prvku s mapou, na kterém uživatel zvolí konkrétní bod, dále textový vstup pro název místa a datový vstup pro datum a čas setkání. Po vytvoření nebo editaci místa setkání jsou všichni členové skupiny upozorněni notifikací.



Obrázek 4.7: Hlavní mapová obrazovka.

Detail uživatele

Tato obrazovka obsahuje výpis detailů o přihlášeném uživateli. Slouží jako náhled na uživatelský profil a rozcestník pro možnosti jeho úprav. Zobrazenými daty jsou:

- **avatar** – náhled fotky uživatele,
- **jméno a příjmení**,
- **e-mailová adresa**.

Ikonou umístěnou u náhledu fotky uživatele lze vybrat novou fotku pro avatar uživatele z fotek v mobilním zařízením nebo pořídit novou fotku. Ikonou v pravém horním rohu aplikace se lze přemístit na obrazovku aktualizace údajů uživatele. Zde lze upravit jméno a příjmení uživatele. Další možností je změna hesla pro přihlášení, což je umožněno pouze po správném zadání hesla stávajícího. Po aktualizaci údajů je uživatel navigován zpět na obrazovku detailu uživatele.

Seznam skupin

Tato obrazovka poskytovala uživateli náhled na seznam všech skupin, jichž byl členem nebo vlastníkem. Dalším prvkem byl ovládací panel obsahující tlačítka pro přidání se k existující skupině či vytvoření nové skupiny.

Pro zjednodušení ovládání aplikace a zlepšení uživatelské zkušenosti byla později tato obrazovka z návrhu aplikace odstraněna a nahrazena vysouvací nabídkou, kterou je možné spustit z mapové obrazovky. Více o této nabídce je popsáno v oddílu popisujícím mapovou obrazovku.

Připojení se ke skupině

Na této obrazovce má uživatel možnost zadat zvací kód, který obdržel od jiného uživatele jako „pozvánku“ k připojení se ke skupině. Po zadání zvacího kódu je tento kód ověřen HTTP požadavkem na webovém serveru skrze aplikační rozhraní.

Je-li kód validní, tedy pokud skupina s daným kódem existuje a stále nedosáhla data a času ukončení své platnosti, je uživateli nabídnuta možnost se k ní připojit. V opačném případě je uživatel upozorněn na možnou chybu pomocí dialogové notifikace. Po připojení ke skupině je uživatel odkázán zpět na mapovou obrazovku, kde již vidí ostatní členy skupiny a její místa setkání.

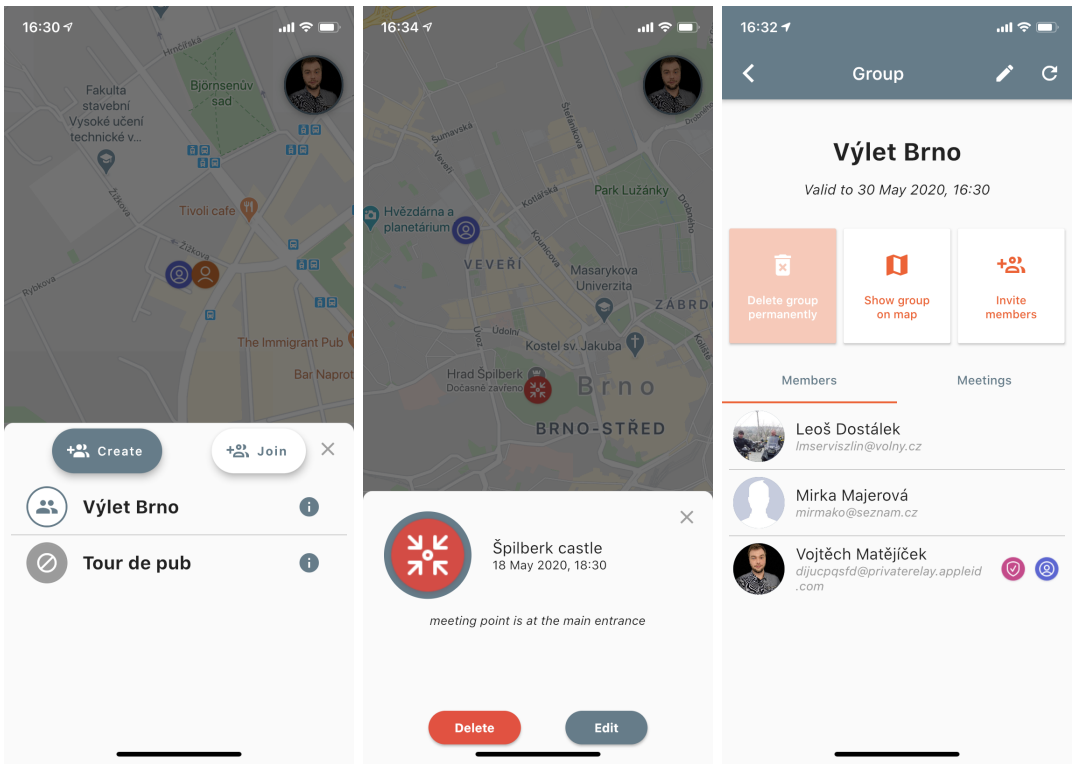
Detail skupiny

Obrazovka obsahuje veškerá data o skupině:

- **název skupiny**,
- **dobu platnosti skupiny**,
- **seznam členů** – seznam s možností prokliknutí se na aktuální pozici na mapě a detailu zvoleného uživatele,
- **seznam míst setkání** – položkami seznamu jsou názvy míst setkání a doba plánovaného setkání, proklik na dané místo zobrazí toto místo na obrazovce s mapou.

Pokud je uživatel vlastníkem (tvůrcem) skupiny, nachází se v horním pravém rohu tlačítko pro editaci skupiny. Toto tlačítko přesměruje uživatele na podobnou obrazovku jako tlačítko pro editaci uživatele, tentokrát jsou ale textové vstupy předvyplněny údaji o upravované skupině. Po odeslání úprav a aktualizaci skupiny je uživatel přesměrován zpět na obrazovku detailu skupiny s aktualizovanými daty. Členové skupiny jsou pomocí notifikace upozorněni na provedené změny v údajích o skupině.

V seznamu členů skupiny je vlastníkovi skupiny umožněno odstranit uživatele ze skupiny. Tato funkce je graficky ztvárněna jako odsouvací prvek *(dismissible item)* a k jejímu spuštění stačí pouze „přetáhnout“ prvek zprava doleva. Uživatel je poté vyzván k potvrzení této akce potvrzovacím dialogem.



Obrázek 4.8: Obrazovky skupiny (seznam, detail, detail místa setkání)

Uživatelské nastavení

Tato obrazovka poskytuje uživateli možnost přizpůsobit si chování aplikace podle svých preferencí. Protože aplikace nenabízí velké množství nastavitelných funkcí, obsahuje tato obrazovka pouze možnost vypnutí aktualizací polohy uživatele při běhu aplikace na pozadí.

Dalším prvkem, obsaženým v této obrazovce, je odkaz, který uživateli umožňuje přejít do systémového nastavení telefonu. Zde lze ovládat přidělení oprávnění k využití nativních funkcí zařízení (lokace, upozornění, . . .).

Hlavní menu

Toto menu sloužilo v první verzi aplikace jako rozcestník mezi všemi obrazovkami. Ve většině obrazovek bylo přístupné jako vysouvací menu z levé strany aplikace *(drawer)*. Z někte-

rých obrazovek (editace či vytváření entit a nastavení) toto menu přístupné nebylo a bylo třeba se pro jeho zpřístupnění vrátit na předchozí obrazovku.

Na základě poznatků z testování uživatelského rozhraní první verze aplikace nezávislými uživateli bylo hlavní menu odstraněno. Jeho funkce byla nahrazena navigačními tlačítky na hlavní (mapové) obrazovce aplikace. Tato tlačítka umožňují rychlejší a přehlednější ovládání celé aplikace.

Kapitola 5

Implementace mobilní aplikace

V této kapitole jsou popsány principy a postupy používané při tvorbě klientské mobilní aplikace, která je hlavní částí této práce. Bude popsán postup tvorby jednotlivých prvků, které jsou součástí aplikace, jejich naplnění uživatelskými daty a napojení aplikace na webový server, kde jsou tato uživatelská data uchována.

5.1 Struktura mobilní aplikace

Tato sekce obsahuje popis rozdělení a význam souborů v souborové struktuře projektu. Blíže se bude věnovat souborům specifickým pro projekt, tedy souborům vkládaným vývojářem mobilní aplikace.

5.1.1 Inicializace projektu

Projekt pro nástroj Flutter obsahuje rozsáhlé množství souborů, konfigurací a zdrojových kódů, které není třeba měnit. Proto jsou tyto soubory při inicializaci nového projektu automaticky kopírovány do své předurčené stromové struktury. Mezi těmito soubory jsou soubory nutné pro překlad zdrojových kódů v jazyce Dart do strojových kódů jednotlivých platforem. Dále se zde nachází optimalizátory kódu pro rychlý běh a optimální využívání zdrojů mobilního zařízení při používání výstupní aplikace.

Důležitou částí souborové struktury projektu jsou adresáře `android` a `ios`. V těchto adresářích se nachází konfigurační soubory pro nastavení, překlad a spuštění aplikace na obou platformách mobilního zařízení. Adresář `android` obsahuje konfigurace *build.gradle* a *AndroidManifest.xml*. Tyto konfigurační soubory využívá balík Android SDK při sestavování mobilní aplikace pro platformu Android. V adresáři `ios` se pak nachází konfigurační soubory typu *xcodeproj* a *Podfile* používané programem Xcode pro sestavení aplikace pro platformu iOS. Některá nastavení prováděná v těchto souborech jsou viditelná i samotnému koncovému uživateli. Jedná se například o název aplikace nebo výčet oprávnění k přístupu k nativním funkcím mobilního zařízení, které aplikace vyžaduje ke svému fungování.

Pro vývojáře je jedním z nejpodstatnějších adresářů adresář `lib`, který bude obsahovat veškeré soubory zdrojových kódů v jazyce Dart, specifických pro daný projekt. Tento adresář je při inicializaci projektu téměř prázdný a poskytuje tak vývojáři volnost při organizaci těchto souborů podle jejich kategorie. Jediným souborem, který je při inicializaci umístěn automaticky v adresáři, je soubor `main.dart`. Tento soubor obsahuje funkci `main()`, jejímž úkolem je spuštění samotné aplikace a definice základních atributů a nastavení aplikace.

5.1.2 Knihovny třetích stran

Navzdory krátké době, po kterou je Flutter poskytován široké veřejnosti, existuje početná komunita vývojářů a firem, která poskytuje předdefinované knihovny pro nejrůznější použití. Vývojáři je tak umožněna produkce přehledného a udržovatelného kódu díky možnosti tyto knihovny používat místo opakovaných implementací často používaných prvků. Katalog většiny těchto knihoven se nachází na webu www.pub.dev.

Mezi takové prvky patří například widgety pro zobrazování obsahu, nástroje pro usnadnění použití nativních funkcí mobilního zařízení (fotoaparát, GPS modul, ...) nebo třídy obsahující metody pro snadnější a optimalizovanou práci se samotným Flutterem. Některé z těchto knihoven byly později převzaty společností Google a zakomponovány do základního balíku (např. *Provider*).

Knihovny a balíky třetích stran jsou do aplikace importovány pomocí definice v konfiguračním souboru `pubspec.yaml`. Tento soubor obsahuje základní konfiguraci externích prvků celého projektu. Mezi tyto prvky patří mimo externí knihovny například také fonty nebo assety v podobě multimediálních souborů. V souboru je také specifikováno označení verze a sestavení aplikace.

Pro přidání nového balíku je třeba pouze připsat jeho identifikátor a požadovanou verzi do sekce `dependencies` (závislosti). Poté je konzolovým příkazem

```
flutter packages get
```

spuštěno stahování balíku a jeho instalace do projektu. Princip je podobný principu fungování správců závislostí jiných platforem, např. *composer* pro PHP nebo *npm (Node Package Manager)* pro javascriptový Node.js.

5.1.3 Organizace zdrojového kódu

Flutter sám nedefinuje předepsaný princip organizace struktury jednotlivých souborů zdrojových kódů, které jsou v jeho projektech použity. Přesto je vhodné jistý systém během vývoje aplikace dodržovat. Díky tomu je možný vývoj přehledného kódu a do budoucna dobře udržovatelného projektu. Další výhodou je možnost usnadnění průběhu týmové spolupráce na projektech.

Existuje více možností a technik, jak zdrojový kód organizovat do přehledné struktury. Tato kapitola popisuje jednotlivé typy a kategorie zdrojových souborů, které byly použity při implementaci mobilní aplikace vyvíjené v této práci.

Model

Podobně jako v návrhovém vzoru MVC (Model, View, Controller) se i v projektu vyvíjeném v této práci pomocí nástroje Flutter nachází modely. Jedná se o definice tříd jednotlivých objektů, které jsou ekvivalenty modelů v databázi vzdáleného serveru.

Modely obsahují definici atributů, gettery a settery relačně vázaných objektů a pomocné funkce pro vykonávání jednoduchých úkolů, které není třeba svěřovat vzdálenému serveru. Pro možnost využití jsou modely načítány pomocí providerů, které je pak předávají obrazovkám nebo jejich data odesílají na vzdálený server. Atributy tříd modelů mohou být, jako ve většině jiných programovacích jazyků, veřejné a soukromé. V jazyce Dart neexistuje žádný explicitní identifikátor přístupnosti atributu, existuje však zavedená konvence rozlišitelného pojmenování atributů. Podle této konvence obsahují názvy privátních atributů jako

počáteční znak podtržítko. Tuto konvenci Flutter přijal jako pravidlo, díky čemuž při pokusu o přístup k takto pojmenovanému privátnímu atributu z jiného souboru vrátí program výjimku.

Po změnách dat v modelech, které jsou použité pro vykreslení některého prvku nebo obrazovky, je třeba tuto obrazovku nebo prvek upozornit na nutnost překreslení své části s obnovenými daty. Pro tento účel slouží třída `ChangeNotifier`, kterou modely, jejichž data jsou viditelná na obrazovce mobilního zařízení, musí používat. Třída implementuje funkci `notifyListeners()`, jejíž volání upozorní všechny obrazovky závislé na obsahu daných modelů, že je třeba znovu vykreslit obrazovku nebo její část, kde se data z daného modelu nachází.

Provider (Poskytovatel)

Stejně jako modely jsou ekvivalenty modelů v serverové aplikaci, jsou providery ekvivalenty controllerů. Jejich hlavním úkolem je poskytnout data modelů jednotlivým obrazovkám, protože jsou v kontextu aplikace vždy dostupné během vykreslování těchto obrazovek.

Providery dále odpovídají za práci s daty modelů, jejich zpracování po načtení ze vzdáleného serveru a přípravu na odeslání po jejich úpravě či vytvoření v mobilní aplikaci.

Alternativou použití providerů je předávání potřebných modelů jako argumenty jednotlivých obrazovek při navigaci v aplikaci. Tento princip, ačkoli je funkční, není pro svou nepřehlednost příliš použitelný ve větších aplikacích. Docházelo by zde k několikanásobnému předávání modelu v rámci kontextového stromu obrazovek a widgetů, ačkoliv by k jejich využití pro vykreslení obsahu docházelo až na jeho konci.

Aby bylo možné použít providery, je třeba jejich třídy inicializovat v souboru `main.dart`. Po této inicializaci jsou providery drženy v paměti a připraveny poskytnout potřebná data libovolnému widgetu nebo obrazovce vykreslené v kontextu aplikace.

Widget (Prvek)

Widgety slouží k vykreslení části obrazovky aplikace. Každý widget obsahuje stromovou strukturu dalších prvků, které jsou vykresleny jako jeho potomci. Nemá-li widget potomka, stává se z něj prvek na popředí. Příkladem prvku na popředí je text, ikona nebo obrázek.

Prvky, které nejsou na popředí, obsahují atribut `child`, pokud jsou definovány pro obsah pouze jednoho potomka, nebo `children`, který těchto potomků obsahuje více. Součástí těchto prvků bývají zpravidla atributy pro nastavení vzhledu, velikosti, zarovnání a dalších nastavení vizuální podoby.

Existují prvky, které nemají vizuální výstup, ale slouží pouze pro úpravu vzhledu či chování svých synovských či rodičovských prvků. Příkladem takového prvku je `Padding`, který nastaví odsazení mezi hranicí rodičovského prvku a obsahem jeho potomka či `Stack` umožňující vrstvené skládání synovských prvků s absolutním umístěním na obrazovce.

Flutter obsahuje dva druhy widgetů, a to *stavové (stateful)* a *bezstavové (stateless)*. Pro většinu prvků, které se nenachází na popředí a bez změny obrazovky není důvod očekávat nutnost jeho překreslení v čase běhu aplikace, je vhodné využívat méně náročné bezstavové widgety. Tím je ušetřen výpočetní výkon nutný pro běh celé aplikace.

Prvky na popředí, jejichž obsah je častěji měněn v závislosti na provedení nějaké akce na dané obrazovce (stisk tlačítka, dotyk obrazovky, aj.), vyžadují implementaci jako stavový prvek. Tento prvek obsahuje privátní třídu `_WidgetState`, která uchovává aktuální data a různé hodnoty, u nichž se očekávají změny. Po provedení této změny lze vyvolat opětovné

vykreslení tohoto widgetu a změna je ihned patrná na obrazovce mobilního zařízení, kde se widget nachází.

Kvůli vyšším požadavkům na výpočetní výkon, který stavové prvky vyžadují, obzvlášť dochází-li ke změně dat a překreslení widgetu často, je vhodné omezit stavové prvky na minimální nutnou velikost a raději je jako drobné prvky obalit jako potomky jiných, bezstavových widgetů.

Screen (Obrazovka)

Speciálním případem widgetu je obrazovka. Ta se sice příliš funkcionálně neliší od běžného widgetu, avšak pro účely strukturování souborů v projektu je vhodné tyto obrazovky uchovávat ve vlastním adresáři. Obrazovky jsou na rozdíl od obyčejných widgetů registrovány v souboru `main.dart` pro umožnění navigace mezi nimi. Více o navigaci se nachází v sekci 5.2.

Tento widget zaujímá celou plochu obrazovky mobilního zařízení. V případě, že je v projektu používán jeden styl pro organizaci obrazovek, je tento styl použit pro všechny obrazovky nebo většinu z nich. Může se jednat např. o obalení obsahu obrazovky prvkem `Scaffold`, který poskytuje rozdělení této obrazovky na funkční prvky. Příkladem takového prvku je `AppBar` nacházející se v horní části obrazovky, který obsahuje tlačítka, jimž lze přiřadit různá funkcionalita. Dalším takovým prvkem je např. `Drawer`, tedy výsuvné menu.

Hlavním funkčním prvkem obrazovky je `body`, tedy tělo obrazovky s hlavním obsahem. Tělo má jako potomka další widget obsahující svůj vlastní strom prvků.

Ačkoliv bylo řečeno, že větší a obsáhlejší prvky je vhodné udržovat bezstavové, většina obrazovek bude implementována jako stavový prvek. Toto je z důvodu překreslování překrývajících se obrazovek a animace přesunu a navigace mezi těmito obrazovkami.

Exception (Výjimka)

Výjimky reprezentují chybu vzniklou při běhu programu aplikace. Existují různé typy výjimek, které jsou „vyhozeny“ (příkaz `throw`) na základě povahy chyby, která se právě stala.

Většina často používaných výjimek je předdefinována již v samotném jádru jazyka Dart či nástroje Flutter. Pro vytvoření výjimek specifických danému projektu je ale možné definovat třídu, která rozšiřuje generickou třídu `Exception` a jejíž instance bude použita.

U částí zdrojového kódu, při jehož provádění vzniká riziko vyhození výjimky, je třeba tuto výjimku zpracovat, aby nedošlo k pádu celé aplikace. Zpracování výjimky spočívá v umístění rizikového úseku kódu do tzv. *try-catch* bloku, jehož úkolem je „odchytit“ vyhozenou výjimku a na základě jejího typu pokračovat v předdefinovaném chování.

```
try {
    final response = await SApi.get(
        'route',
        query: queryData,
    );
} on ApiException catch (e) {
    // do this on ApiException
} catch (e) {
    // do this on any other Exception
}
```

Algoritmus 5.1: Try-catch blok

Helper (Pomocná třída)

Posledním typem zdrojových souborů jsou helpery, soubory obsahující pomocné třídy nebo pouze pomocné funkce. Ty jsou využity ve více místech v projektu a jejich opětovnou definicí by docházelo k nechtěné redundanci kódu.

Příkladem pomocné třídy je třída pro vytváření HTTP požadavků obsahující statické metody pro vytvoření požadavků ve tvaru odpovídajícímu definicím webového serveru. Tyto statické metody jsou použity na více místech projektu, jejich funkci však vyžaduje více providerů, což neumožňuje implementaci potřebných metod například do některého z modelů.

5.2 Navigace v aplikaci

V této sekci je vysvětlen princip přecházení mezi jednotlivými obrazovkami aplikace a specifikována data, která jsou k tomuto úkolu potřebná.

5.2.1 Navigátor

Průběh celé navigace aplikací probíhá jako skládání, výměna či rušení vykreslení obrazovek. O tyto úkoly se stará třída `Navigator`. Nejčastějším jevem je skládání následujících obrazovek do datové struktury *zásobník*. Díky tomu, že se nově vytvořené obrazovky v průběhu vykreslování aplikace řadí do fronty LIFO (Last in, first out), je pro uživatele velmi jednoduché navigovat se zpět stejnou cestou, kterou se dostal do současného stavu.

Pokud z hlediska kontextu mobilní aplikace není vhodné při návratu navštívit některou z předchozích obrazovek, je možné během procházení LIFO fronty některé obrazovky odstranit. Toto lze provádět až po návštěvu domovské obrazovky a do vyprázdnění zásobníku.

5.2.2 Směrování

Podobně jako webová aplikace používá i aplikace napsaná pomocí nástroje Flutter směrování mezi obrazovkami. Toto směrování se řídí podle směrovacích pravidel, která definuje vývojář aplikace.

Směrovací pravidla pro každou obrazovku aplikace jsou definována v souboru `main.dart`. Každá obrazovka obsahuje svůj identifikátor, který označuje akci, jež je provedena při na-

vigaci pomocí tohoto identifikátoru. Ten může být libovolný, pokud je dodržena unikátnost mezi identifikátory všech obrazovek. Operací prováděnou při navigaci je většinou vykreslení nové obrazovky. Kvůli možnosti použití identifikátoru na více místech v aplikaci (například jeho předání jako argument navigátoru) je vhodné umístit tento identifikátor do statického atributu třídy, která definuje danou obrazovku.

Pokud je to nutné či vhodné, lze namísto volání obrazovky pomocí jejího identifikátoru vytvořit instanci třídy `ModalRoute`. Výsledkem jejího následného zavolání je obrazovka. Tuto metodu je vhodné použít, pokud v průběhu navigace dochází k zpracování dat předaných jako argumenty navigace.

5.2.3 Argumenty navigace

Jak již bylo zmíněno, při navigaci mezi obrazovkami lze mezi těmito obrazovkami předat libovolné argumenty. Pro rozsáhlé aplikace, kde může existovat celá řada zřetězených obrazovek, může nastat problém vícenásobného předávání stejného argumentu, kterému je doporučeno se z důvodu zbytečné konzumace výpočetního výkonu vyhýbat.

Ačkoliv lze jako argument předat objekt jakéhokoliv datového typu, při nutnosti použít toto předání je vhodné předávat pouze hodnoty jednoduchých datových typů. Příkladem je předání identifikátoru, který je na další obrazovce použit providerem k následnému vyhledání instance modelu s tímto identifikátorem.

5.3 Komunikace s webovým serverem

Tato sekce popisuje návaznost mezi mobilní aplikací a aplikačním rozhraním vzdáleného serveru. Komunikace probíhá pomocí HTTP protokolu.

5.3.1 Helper pro HTTP požadavky

Pro vytvoření HTTP požadavku byla do projektu importována veřejná knihovna, která tyto požadavky vytváří. Díky přístupu k dokumentaci aplikačního rozhraní webového serveru lze vytvořit strukturu parametrů, které se často vyskytují v jednotlivých požadavcích. Mezi tyto parametry patří:

- **Přístupový klíč**: Všechny autorizované požadavky musí obsahovat náhodně vygenerovaný přístupový klíč v hlavičce požadavku. Tento klíč slouží k identifikaci uživatele a bez jeho poskytnutí dochází k neúspěchu požadavku a jako odpověď je vrácena HTTP chyba 401 – *Unauthorized*.
- **Tělo požadavku**: Pokud se jedná o požadavek metodou POST nebo PATCH, jsou na server odesílána data o vytvoření či úpravě modelu. Data těla požadavku jsou před odesláním serializována do formátu JSON.
- **Expand**: Tento parametr je přidán k požadavku jako GET parametr URL. Jeho hodnota označuje relačně vázané objekty, jejichž data potřebujeme získat od serveru spolu s dotazovaným modelem. Příkladem je seznam uživatelů skupiny, který aplikace potřebuje při vykreslení detailu skupiny, avšak pro obrazovku výpisu seznamu skupin není tato informace potřebná.
- **Identifikátor modelu**: Jedná-li se o vyhledání konkrétního modelu, jeho úpravu či smazání, je serveru zasílán identifikátor, který umožní jeho vyhledání v relační

databázi. Identifikátor také slouží k ověření práv uživatele pro čtení a manipulaci s modelem.

Pro zamezení redundance kódu přidáváním těchto parametrů každému potřebnému požadavku byla vytvořena pomocná třída využívající veřejně dostupnou knihovnu pro vytvoření HTTP požadavku. Ta zaobaluje metody této knihovny a modifikuje je pro lepší použití v této aplikaci.

5.3.2 Asynchronní funkce

Komunikace se vzdáleným serverem probíhá v režii jednotlivých providerů. Účelem této komunikace je získat od vzdáleného serveru data z databáze, naplnit těmito daty příslušný model a po zpracování data vypsat na požadovaný výstup uživateli nebo je vrátit na server.

Protože ale komunikace po internetu trvá řádově mnohem déle, než operace prováděné kompletně v rámci mobilního zařízení, je třeba provádět tuto komunikaci v tzv. asynchronní funkci. Asynchronní funkce je taková, která umožňuje spouštění asynchronních operací. Funkce se vyznačuje klíčovým slovem `async` v její hlavičce.

Jako návratovou hodnotu asynchronní operace nelze zvolit libovolný datový typ, protože při jejím přiřazení může docházet k prodlevě a program by tak pokračoval s nesprávně přiřazenou hodnotou, což by mohlo mít za následek až pád aplikace.

Asynchronní operace proto vrací objekty typu `Future<T>` [7] (budoucí hodnoty daného typu T). Tyto objekty se vyznačují tím, že prodleva při jejich přiřazení nezpůsobí pád aplikace, ale je zřejmé, že v daném bodě musí program počkat na „naplnění“ této budoucnosti a tedy získání požadované hodnoty daného typu. Pro vyčkání na získání této hodnoty slouží klíčové slovo `await`, které pozdrží běh programu, dokud hodnota nebude správně přiřazena. Po získání přiřazené hodnoty je činnost funkce dokončena (pokud nedošlo k chybě například ve spojení) a výsledek HTTP komunikace je zpracován.

5.3.3 Výsledky komunikace

Data z aplikačního rozhraní serveru přichází do aplikace kódovaná ve struktuře JSON. Tato struktura ale není nic jiného, než obyčejný textový řetězec. Proto je nutné před zpracováním dat zprávu ze serveru dekódovat na požadovanou strukturu, se kterou umí jazyk Dart dále efektivně pracovat. Pro převod ze struktury JSON se používá struktura `Map<String, dynamic>`, tedy asociativní mapa obsahující prvky různých datových typů indexovaných pomocí textových řetězců. S touto mapou lze dále libovolně manipulovat, řadit, iterovat či přidávat a ubírat prvky.

Spolu s daty je vždy doručen i stavový kód shrnující výsledek HTTP komunikace [14]. Pokud komunikace proběhla bez chyb ať už na straně serveru, nebo z důvodu nesprávného uživatelského vstupu či nedostatečného oprávnění, byl doručen stavový kód kategorie *2xx* (tedy třímístné číslo s počátečním číslem 2). Mezi nejběžnější patří:

- **200 (Success)** – obecné označení úspěšné operace,
- **201 (Created)** – vytvoření nového záznamu,
- **204 (No content)** – bez obsahu, používá se po smazání dat,
- a další.

Po obdržení některého z těchto stavových kódů lze data zpracovat. Pokud se jedná o data jednotlivých modelů, je důležité, aby tyto modely byly přizpůsobeny tvaru odpovědi serveru. K získání znalosti o tomto tvaru slouží dokumentace aplikačního rozhraní obsahující seznamy endpointů pro požadavky, HTTP metody vhodné pro požadavky (nelze vytvářet modely pomocí metody GET, apod.) a příklad odpovědi serveru (viz kapitola 6.3).

Neúspěšná operace

Operace prováděné na straně serveru mohou z různých důvodu skončit neúspěšně. V takovém případě se stavové kódy dělí na kódy identifikující chybný požadavek od uživatele *(4xx)* a chybu na straně serveru *(5xx)*. Mezi časté chyby ze strany uživatele patří:

- **400 (Bad request)** – obecné označení chybného požadavku,
- **401 (Unauthorized)** – uživatel není přihlášen,
- **403 (Forbidden)** – uživatel je autorizován, ale k zpracování požadavku nemá povolen přístup,
- **404 (Not found)** – požadavek na neexistující stránku nebo objekt,
- **405 (Method not allowed)** – uživatel provedl požadavek nesprávnou HTTP metodou (GET namísto POST, apod.),
- a další.

Mezi časté chyby na straně serveru patří:

- **500 (Internal server error)** – vnitřní chyba serveru,
- **503 (Service unavailable)** – služba na serveru je dočasně mimo provoz,
- **504 (Gateway timeout)** – došlo k vypršení časového limitu pro zpracování požadavku,
- a další.

Pokud dojde k některé z těchto chyb, dochází k vyhození výjimky typu `ApiException` obsahující zprávu s odpovědí serveru a stavový kód chyby. Na základě této výjimky vypíše aplikace oznámení uživateli s příslušnou chybovou hláškou.

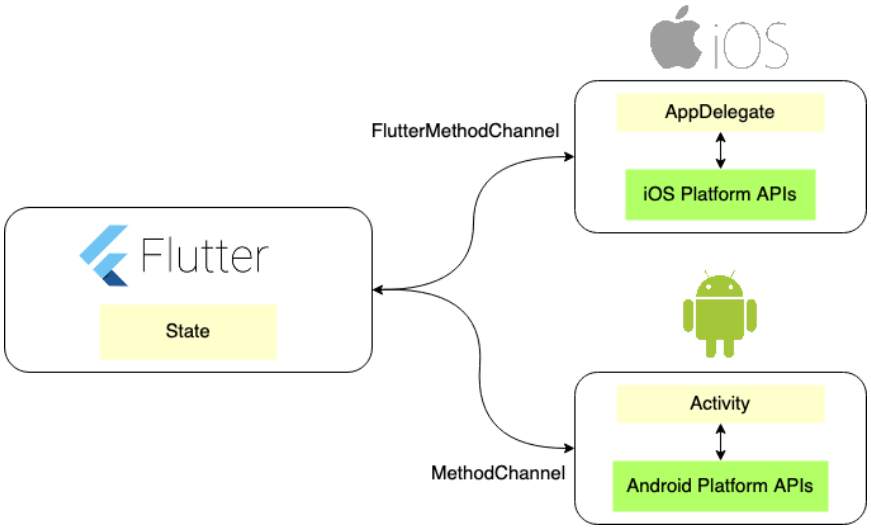
V případě chyby z důvodu nesprávného uživatelského vstupu (například při vytváření či úpravě modelu) lze podle této chyby opravit data a požadavek opakovat. Příkladem takové chyby je pokus o přihlášení s chybným heslem.

5.4 Běh aplikace na pozadí

Jak již bylo řečeno v kapitole 3.3, při vývoji mobilních aplikací pomocí nástrojů pro multiplatformní vývoj lze narazit na jistá omezení přístupu ke všem nativním funkcím mobilního zařízení. Jedno z těchto omezení nástroje Flutter je problematická implementace běhu aplikace na pozadí.

Pokud má uživatel aplikaci spuštěnou na popředí, fungují všechny prvky a funkce implementované pomocí Flutteru bez problému. Pokud je ale aplikace minimalizována pro umožnění spuštění jiné aplikace nebo je zařízení uspáno, je operačním systémem běh aplikace implementované pomocí Flutteru potlačen. To je v případě aplikace Flox kritický problém, protože při sledování polohy uživatele nelze spoléhat, že bude aplikace na jeho zařízení neustále spuštěna na popředí.

Na rozdíl od multiplatformního nástroje Flutter, nástroje pro vývoj aplikací v nativních programovacích jazycích běh na pozadí umožňují prostřednictvím tzv. *background services – služeb na pozadí*. Spuštění takové služby umožňuje běh na popředí jiným aplikacím, zatímco běh naší aplikace není přerušen. Přerušena je pouze ta část aplikace, která je implementována pomocí nástroje Flutter.



Obrázek 5.1: Volání metod v nativním kódu

Z tohoto důvodu je třeba služby na pozadí implementovat v nativních jazycích, tedy Kotlin nebo Java pro Android a Swift či Objective-C pro iOS. Flutter umožňuje vytváření jisté formy komunikačního kanálu mezi částí aplikace implementované v jazyce Dart a částí implementované v nativním jazyce [8]. Díky tomu je možné invokovat spuštění jakékoliv metody v nativní části zdrojového kódu. V tomto případě dochází ke spuštění služby na pozadí přímo ze základního kódu aplikace při zachycení události přesunu aplikace na pozadí.

Aplikace je takto minimalizována pouze zdánlivě, protože oba operační systémy vyžadují zobrazení trvalé notifikace v oznamovacím centru uživatelského rozhraní mobilního zařízení. Tato notifikace běží neustále na popředí, čímž je zajištěno, že aplikace není operačním systémem vyhodnocena jako nečinná a její běh není automaticky přerušen. Po úplném ukončení aplikace je její běh zastaven a notifikace není dále zobrazena.

Kapitola 6

Webový server a aplikační rozhraní

Webový server poskytuje aplikaci nástroje pro sdílení uživatelských dat a jejich ukládání do paměti relační databáze. Umožňuje také autentizaci uživatele pro vykonávání operací, které nejsou anonymním uživatelům povoleny. Komunikace s mobilní částí aplikace je zajištěna pomocí aplikačního rozhraní REST, jehož implementace je součástí aplikace webového serveru. Tato kapitola popisuje detaily implementace tohoto rozhraní i celé serverové aplikace.

6.1 Modely aplikačního rozhraní

Modely aplikačního rozhraní rozšiřují definice modelů společného jádra aplikace a dědí všechny jejich atributy a metody. Díky tomu s nimi lze pracovat jako s běžnými modely, které lze načíst z databáze nebo je do ní uložit. Mezi hlavní metody, které model dědí z jádra aplikace, patří metoda `rules()`. Ta obsahuje definice validátorů jednotlivých atributů modelu. Validátory kontrolují správnost zadaných údajů před vložením záznamu o modelu do databáze.

Hlavním rozšířením, ke kterému dochází v modelu aplikačního rozhraní, je definice metod `fields()` a `extraFields()`. Obě tyto metody vrací pole definic finální podoby struktury, která reprezentuje daný model a která bude odeslána pomocí aplikačního rozhraní do mobilní aplikace. Metoda `fields()` obsahuje atributy, které jsou zasílány při každém odeslání dat modelu. V metodě `extraFields()` jsou definovány atributy, které implicitně nejsou zasílány, ale lze o jejich zaslání explicitně požádat. Většinou se jedná o data relačně vázaných objektů k danému modelu, která nechceme posílat při běžné komunikaci, například výpisu seznamu objektů, ale pouze při získání detailních dat o jednom modelu. Příkladem definice modelu pro aplikační rozhraní je algoritmus 6.1.

Výsledkem metod `fields()` a `extraFields()` je pole hodnot. Tyto hodnoty reprezentují:

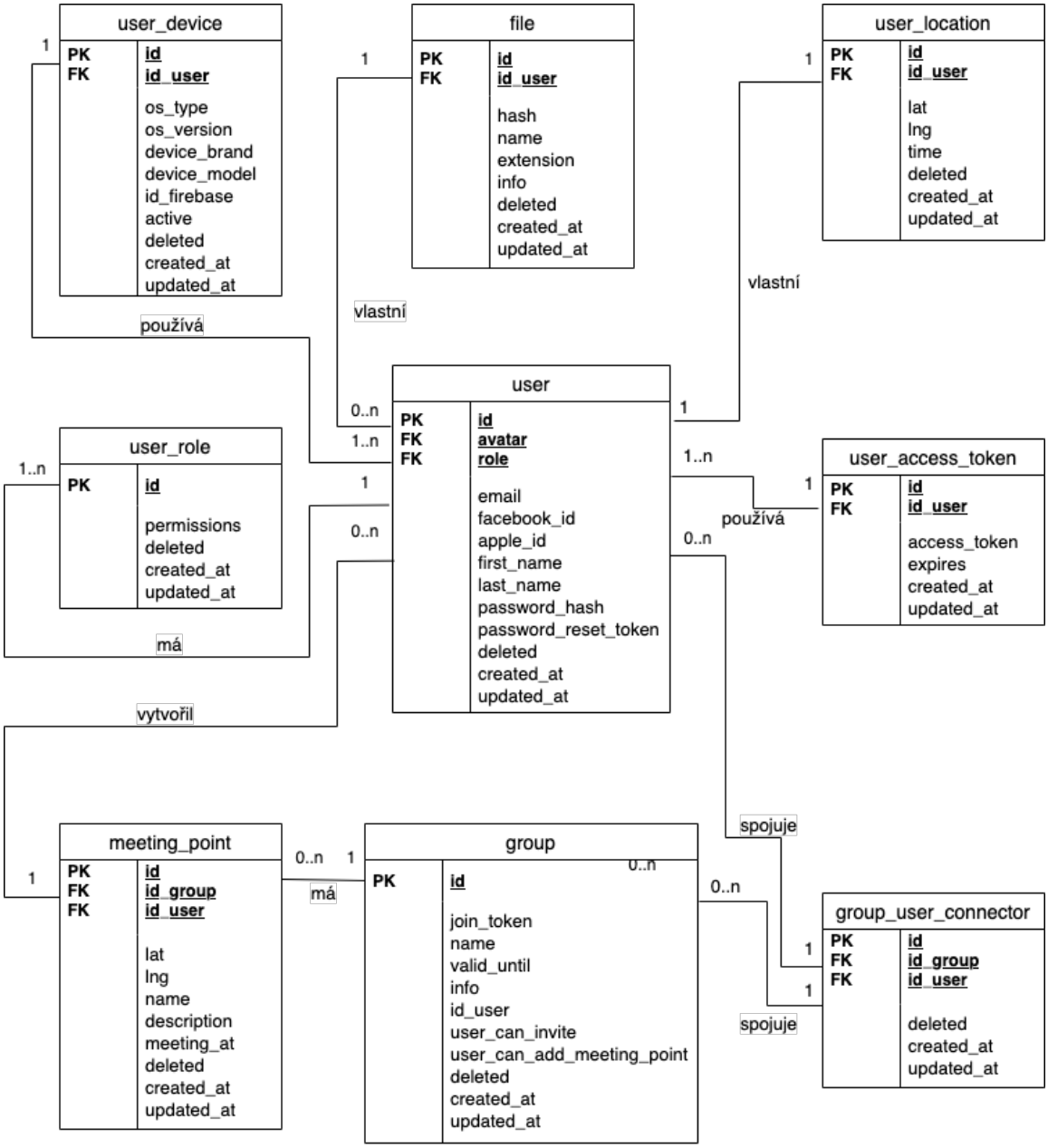
- **atributy** – názvy atributů modelu, které chceme odeslat,
- **gettery** – metody určené pro získání hodnoty privátního atributu či relačně vázaného objektu,
- **funkce** – v některých případech je nutné připravit odesílanou hodnotu v jednoduché anonymní funkci s návratovou hodnotou v podobě upraveného atributu. Tuto funkci lze v případě potřeby nahradit upraveným getterem.

```php
namespace api\versions\v1\models;
class Example extends \common\models\Example
{
    public function fields() {
        return [
            'id', // attribute by name
            'name' => function () {
                return "{$this->first_name} {$this->last_name}";
            }, // attribute by function
        ];
    }

    public function extraFields() {
        return [
            'relatedModels', // attribute by getter
        ];
    }

    public function getRelatedModels() {
        return $this->hasMany(RelatedModel::class, [
            'id' => 'id_related_model'
        ]);
    }
}
```

Algoritmus 6.1: Definice struktury API modelu

Modely jádra webové aplikace, které jsou modely aplikačního rozhraní rozšiřovány, na sebe mapují data z relační databáze webového serveru. Tím umožňují tato data serializovat a prostřednictvím aplikačního rozhraní odeslat mobilní aplikaci. Schéma datového modelu relační databáze je vyobrazeno na obrázku 6.1.

Protože je webová aplikace využívána pro manipulaci s daty modelů mobilní aplikace, kopírují modely webové aplikace svojí strukturou definice modelů mobilní aplikace. Tyto modely jsou popsány v kapitole 4.1. Tím je usnadněno veškeré předávání dat a jejich mapování během komunikace mezi oběma částmi aplikace.



Obrázek 6.1: Schéma relační databáze

6.2 Směrování požadavků API

Framework Yii2 a DactylCMS obsahují implicitní systém směrování požadavků pomocí syntaktické analýzy URL adresy požadavku a její rozdělení na logické části. Příkladem je rozdělení URL adresy požadavků

`GET` <*domain*>`/api/group/index`

a

`GET` <*domain*>`/api/group/view?id=1`.

V prvním případě je požadavek na doménu směrován na modul `API` (aplikační rozhraní), controller `GroupController` a akci `actionIndex()`.

V druhém případě je k URL požadavku připojen i GET parametr `id`. Tento požadavek je směrován na modul `API`, controller `GroupController` a akci `actionView($id)`, kde je argumentu `$id` metody `actionView()` GET parametr předán.

Tento systém bohužel funguje pouze pro nejčastější akce prováděné pomocí aplikačního rozhraní. Těmito akcemi jsou *index, view, create, update* a *delete*. Pro ostatní akce, které jsou specifické každému jednotlivému projektu, je třeba nadefinovat URL pravidlo, které požadavek přesměruje do správné metody správného controlleru. Pravidla, která jsou specifická pouze danému projektu, se nazývají `extraPatterns`.

Tato pravidla jsou definována při instanciaci modulu aplikačního rozhraní. Přidávaná pravidla jsou vytvářena jako instance třídy `RestUrlRule`, jíž jsou předávány údaje o cílovém controlleru, cílové akci (metodě) a tvaru a HTTP metodě požadavku (používanými metodami jsou GET, POST, PATCH a DELETE).

Příkladem definice pravidla pro směrování požadavku je zdrojový kód obsažený v ukázce 6.2. Přidání tohoto pravidla definuje směrování požadavků *status, login* a *register* na metody `actionStatus()`, `actionLogin()` a `actionRegister()` controlleru `UserController`.

```
// ...
$rule = [
   'class' => RestUrlRule::class,
   'controller' => 'user',
   'extraPatterns' => [
      'GET status' => 'status',
      'POST login' => 'login',
      'POST register' => 'register',
   ],
   'only' => [
      'index', 'status', 'login', 'register'
   ],
   'except' => [
      'view', 'create', 'update', 'delete'
   ]
];
// ...
```

Algoritmus 6.2: Definice směrovacích pravidel s omezením

Pokud je třeba ujistit se u vytvořeného pravidla, že uživateli aplikačního rozhraní nebude umožněno přistoupit k akcím, které vývojář aplikace nechce povolit, je možné tohoto docílit omezením pravidel. To lze provést pomocí atributu `only` nebo uzamčením směrovacích pravidel pomocí atributu `except`.

Nastavení pravidla v ukázce 6.2 umožní přistoupit pouze k požadavkům *index, status, login* a *register* a naopak požadavky *view, create, update* a *delete* proti přístupu uzamkne [18]. Při pokusu o zavolání URL adresy uzamčených požadavků je volajícímu vrácena odpověď s chybou 404 – stránka nebyla nalezena.

6.3 Dokumentace API

Při vývoji aplikace, která se spoléhá na manipulaci s daty uloženými v databázi pomocí aplikačního rozhraní, je třeba toto rozhraní pečlivě dokumentovat. Z dokumentace rozhraní musí být patrné všechny *endpointy* (konkrétní akce API controllerů) a ostatní údaje, jež musí vývojář aplikace, která pracuje s aplikačním rozhraním, znát.

Existuje větší množství údajů a atributů každého endpointu, které lze v dokumentaci aplikačního rozhraní uvést. Mezi ty nejčastější a ty, jejichž uvedení je pro kvalitní dokumentaci vyžadováno, patří:

- **URL adresa**: Protože aplikace komunikuje s webovým serverem pomocí HTTP požadavků, je třeba pro každý endpoint definovat URL adresu, pomocí které lze volat příslušnou akci.
- **HTTP metoda**: Metody HTTP požadavku slouží pro rozlišení operace, kterou daný požadavek provádí. Pokud dochází pouze k čtení dat uložených na serveru, lze použít metodu GET. Pro vkládání nových záznamů libovoného typu či provádění akcí využívající uživatelský vstup (např. login) je nutné použít metodu POST. Úprava existujících záznamů je prováděna metodou PATCH a odstranění záznamu z databáze je prováděno metodou DELETE. Při použití nesprávné metody vrátí aplikační rozhraní odpověď se stavovým kódem 405 – Method not allowed.
- **Verifikace uživatele**: Protože některé endpointy jsou pro aplikaci přípustné pouze po autorizaci uživatele, může být vyžadováno vložení uživatelských údajů. Aby nedocházelo k odesílání uživatelského jména a hesla při každém takovém požadavku, jsou spolu se všemi zabezpečenými endpointy odesílány uživatelské přístupové *tokeny*, které jsou generovány při přihlášení uživatele a jejichž prostřednictvím je ověřena identita uživatele. Přístupové tokeny jsou odesílány v hlavičce HTTP požadavku.
- **Parametry požadavku**: Pokud při provádění požadavku dochází k odesílání dat z aplikace na webový server, je nutné tato data odesílat v přesně definovaném formátu. Tento formát je proto prostřednictvím dokumentace sdělen vývojáři koncové aplikace.
- **Formát odpovědi**: Webový server na každé volání aplikačního rozhraní reaguje odesláním HTTP odpovědi. V některých případech obsahuje tato odpověď data, která byla vyžádána aplikací. Tato data jsou odeslána v přesně definované struktuře a je třeba tuto strukturu kvalitně dokumentovat pro zajištění jejího správného použití v koncové aplikaci.

6.3.1 Swagger

Pro dokumentaci aplikačního rozhraní pro aplikaci Flox byl použit nástroj Swagger. Tento nástroj umožňuje vytvoření komplexní dokumentace všech endpointů. Ty jsou přehledně rozděleny podle své kategorie a vizuálně rozlišeny podle použité HTTP metody.

Kromě běžné dokumentace jednotlivých endpointů umožňuje Swagger také volání těchto endpointů včetně vložení uživatelského vstupu. Volané endpointy jsou zpracovány příslušným API controllerem. Jejich vstup je zpracován a uživateli je odeslána odpověď podobně, jako by byla odeslána samotné aplikaci. Díky tomu lze i bez implementace samotné aplikace, závislé na aplikačním rozhraní, toto rozhraní otestovat a odstranit případné chyby. Umožněna je i autentizace uživatele na základě přístupového tokenu, který je obsažen v odpovědi na požadavek o přihlášení.

6.3.2 Spustitelné endpointy

Po nadefinování endpointu je spuštěno skenování souborů se zdrojovým kódem v předdefinovaných adresářích. Nalezené definice jsou poté načteny a vykresleny pluginem `swagger-php` do jedné přehledné webové stránky. Na této stránce je také možné spouštět jednotlivé endpointy za účelem jejich testování.

Aby bylo možné spustit testování funkčnosti endpointu v dokumentaci generované pomocí nástroje Swagger, je nutné definovat všechny parametry HTTP požadavku a ostatní údaje o endpointu.

Pro definování struktury reprezentující data modelové entity je nejprve definováno pojmenované schéma této entity, které bude dále použito v samotné definici endpointu aplikačního rozhraní. Definice schémat i HTTP požadavků jsou vkládány do zdrojového kódu jako anotace modelů či akcí (metod) controllerů.

```
/**
 * @OA\Schema(
 *     schema="myModelDataScheme",
 *     description="Example model scheme",
 *     @OA\Property(property="id", type="integer"),
 *     @OA\Property(property="name", type="string")
 * )
 */
class MyModel extends ActiveRecord {
    public function rules() {
        return [
            [['id'], 'integer'],
            [['name'], 'string'],
        ];
    }
}
```

Algoritmus 6.3: Definice modelového schématu

Definované schéma je poté použito v definici samotného endpointu. Součástí této definice je dále specifikace HTTP metody používané pro endpoint, URL adresa, možné parametry HTTP požadavku, použité zabezpečení a tvar odpovědi. Pokud se jedná o požadavek metodou POST, může být součástí definice struktury těla požadavku. I zde je možné použít definované schéma modelu.

Jak je patrné z definice endpointu metodou POST v algoritmu 6.4, lze pro endpoint definovat nejen podobu odpovědi pro úspěšně provedený požadavek, ale také pro odpověď na požadavek ukončený s chybovým stavem (zde chyba 404 – *Page not found*).

```
class ExampleController extends RestApiController
{
    /**
     * @OA\Post(
     *   path="/example/foo",
     *   summary="Example POST endpoint",
     *   tags={"Example"},
     *   security={{"access-token":{}}},
     *   @OA\Parameter(
     *       name="expand",
     *       in="query",
     *       required=false,
     *       style="form",
     *       @OA\Schema(type="string")
     *   ),
     *   @OA\RequestBody(
     *       @OA\MediaType(
     *           mediaType="multipart/form-data",
     *           @OA\Schema(ref="#/components/schemas/myModelDataScheme")
     *       )
     *   ),
     *   @OA\Response(
     *       response=200,
     *       ref="#/components/schemas/myModelDataScheme"
     *   ),
     *   @OA\Response(
     *       response=404,
     *       ref="#/components/schemas/notFoundResponseScheme"
     *   ),
     * )
     */
    public function actionFoo() {
        // ...
    }
}
```

Algoritmus 6.4: Definice endpointu pro akci controlleru

Dalším prvkem endpointu je možnost přidání nepovinného parametru `expand`. Protože bylo definováno umístění parametru v *query*, bude jeho hodnota serializována do formátu *url kódování* jako součást výsledné URL adresy:

$$<domain>\texttt{/api/example/foo?expand=}value.$$

6.4 Firebase

Aby bylo možné notifikovat uživatele aplikace o nastalých změnách dat skupin, jichž jsou členy, je nutné zajistit způsob zpětné komunikace směrem od serveru k uživateli. Komunikace pomocí REST API probíhá pouze na vyžádání ze strany mobilní aplikace, proto je tento způsob zjišťování změn za běhu aplikace nevhodný.

Tento problém řeší funkce Firebase Cloud Messaging [4]. Tato funkce je hojně využívána vývojáři mobilních aplikací, protože umožňuje odeslat zprávu do mobilního zařízení na základě jeho unikátního identifikátoru. Tento identifikátor je získáván během procesu přihlášení uživatele a je odesílán pomocí aplikačního rozhraní spolu s daty o používaném zařízení (viz kapitola 4.1 – *Uživatelské zařízení*).

Data o zařízení jsou spárována s uživatelským účtem, jehož data jsou uchovávána v relační databázi webového serveru. Při provedení události, o které by měl být uživatel notifikován, proběhne hledání aktivních zařízení daného uživatele (uživatel může vlastnit více zařízení). Identifikátory všech zařízení určených k odeslání notifikace jsou zaslány na aplikační rozhraní aplikace Firebase. Ta na tato zařízení rozešle zprávu obsahující titulek, tělo zprávy *(body)* a nepovinný atribut *data*. Pokud je hodnota atributu *data* definována, lze ji poté využít při zachycení zprávy mobilní aplikací například k upřesnění směrování obrazovek aplikace po kliknutí na obdrženou notifikaci.

Zprávy jsou odchytávány v mobilní aplikaci pomocí předdefinované funkce, která je součástí definice základních funkcí aplikace. Díky tomu lze zobrazovat notifikace i při vypnutém stavu aplikace.

Kapitola 7

Uživatelské testování a vydání aplikace

Posledním krokem ve vývoji mobilní aplikace je její vydání na obchody s aplikacemi jednotlivých platforem. Obchod s aplikacemi pro systém Android společnosti Google se nazývá Google Play Store. Společnost Apple provozuje pro svá zařízení se systémem iOS obchod AppStore.

V této kapitole je popsán proces zveřejnění aplikace na oba tyto obchody, ale i průběh testování funkčnosti aplikace, které bylo před samotným zveřejněním nutné provést. Součástí testování aplikace je i test ovladatelnosti a uživatelské zkušenosti grafického rozhraní aplikace.

7.1 Testování první verze aplikace

Prvním milníkem ve vývoji aplikace byl vznik její základní verze, který neobsahoval většinu hlavních funkcí, ale pouze základní funkce. Příklady chybějících funkcí jsou možnost registrace pomocí sociálních sítí, výběr místa setkání na mapě či možnost aktualizace polohy ostatních uživatelů mimo plánovaný interval. Důvodem pro vznik této verze byla potřeba „reálné“ aplikace, již bylo možné otestovat uživateli na reálném mobilním zařízení. Distribuce instalačního balíku pro instalaci aplikace do mobilního zařízení probíhala bez využití oficiálních testovacích kanálů (viz kapitoly 7.2.1 a 7.2.2), ale pouze předáváním pomocí sdílení souboru na cloudovém úložišti.

Testování aplikace odhalily řadu jejích nedostatků ať v rozmístění ovládacích prvků grafického uživatelského rozhraní, nebo v samotných funkcích aplikace. Tyto nedostatky jsou blíže popsány v následujících kapitolách. Dalším, velmi podstatným nedostatkem první verze aplikace, byla nemožnost běhu aplikace a aktualizace polohy na pozadí. Na základě zkoumání těchto nedostatků a chyb vznikla druhá verze aplikace, jejíž testování je dále popsáno v kapitole 7.2.

7.1.1 Ovládací prvky GUI

Grafické uživatelské rozhraní této verze aplikace vycházelo z původního návrhu představeného v kapitole 4.4. Rozmístění ovládacích prvků odpovídalo základním předdefinovaným funkcím pro vykreslení prvků obsažených v jádru nástroje Flutter. Hlavním prvkem celé aplikace byl mapový podklad s umístěnými markery označujícími polohu jednotlivých uživatelů.

Ačkoliv by tato grafická podoba aplikace mohla být pro využívání všech jejích funkcí použitelná, neposkytovala uživateli komfort při jejím používání a ovládání. Pro navigaci v aplikaci bylo použito příliš mnoho obrazovek, k nimž musel uživatel přistupovat složitě prostřednictvím menu v boční nabídce základní obrazovky. Většina těchto obrazovek byla tedy nahrazena modálními výsuvnými nabídkami, dostupnými přímo z hlavní obrazovky aplikace, a boční nabídka menu byla z aplikace odstraněna.

7.1.2 Vliv běhu aplikace na baterii zařízení

První verze aplikace poskytovala možnost testování jejích základních funkcí. Nejdůležitější touto funkcí je sdílení polohy uživatelova mobilního zařízení a její vyobrazení na mapovém podkladu. Protože využívání polohových služeb zařízení je náročné na baterii, bylo třeba nalézt vhodný způsob jejich používání.

Poloha uživatele je údaj, který je při jejím sdílení nutné pro správné vykreslování na mapě v pravidelném intervalu obnovovat. Dále je tento údaj třeba sdílet s aplikačním rozhraním webového serveru prostřednictvím internetu, což má na stav baterie také nezanedbatelný vliv. V první verzi probíhalo odesílání polohy na webový server v intervalu 5 sekund. Ve stejném intervalu byly poté přijímána data o ostatních uživatelích skupiny. Toto nastavení poskytovalo přehledné a plynulé obnovování informací o poloze ostatních členů skupiny, avšak po delším používání aplikace docházelo k rychlému vybíjení baterie a zvyšování teploty zařízení.

Při odesílání polohy zařízení v intervalu 5 sekund se spotřeba baterie pohybovala cca 24–32 % poklesu nabití baterie za 1 hodinu používání aplikace. Tento údaj se lišil v závislosti na použitých testovacích zařízeních s různými operačními systémy. Při použití stejného intervalu se teplota zařízení citelně zvýšila už během prvních 10–25 minut používání aplikace. Je zde však třeba brát ohled na nutnost neustále zapnutého displeje zařízení, protože první verzi aplikace schází podpora běhu na pozadí.

Zvýšení intervalu odesílání a příjmu polohových údajů na 2 minuty snížilo spotřebu baterie i rychlost zvyšování teploty zařízení. Při stále zapnutém displeji se během 1 hodiny používání aplikace (stále trvale zapnuté na popředí) snížilo nabití baterie o 11–17 %. Vysoký interval obnovy dat však s sebou přinesl podstatné snížení kvality sledování pohybu ostatních členů skupiny na mapě. Při rychlosti 5 km/h, která odpovídá průměrném tempu chůze, urazí uživatel za 2 minuty vzdálenost cca 167 m. Tato vzdálenost je však příliš velká pro účinné sledování rychlosti nebo nalezení přesné polohy daného uživatele.

Tímto experimentálním způsobem bylo následně zjištěno, že aktualizace polohy uživatele v intervalu 40 sekund šetří baterii zařízení. Spotřeba baterie za 1 hodinu při stále zapnutém displeji oproti vyššímu intervalu mírně vzrostla, avšak v nové verzi aplikace je již možné používání aplikace na pozadí, čímž je podstatná část baterie ušetřena. Při běhu aplikace na pozadí taktéž téměř nedochází k většímu zahřívání zařízení, než jaké odpovídá běžnému používání mobilního telefonu.

Aktualizace polohových dat ostatních členů skupiny probíhá v intervalu 30 sekund. Menší interval byl zvolen kvůli vyšší přesnosti sledování polohy jiných uživatelů, jejichž zařízení odesílají svou polohu na aplikační rozhraní serveru v různý čas. Komunikace s webovým serverem má ve srovnání s používáním polohových služeb zařízení menší vliv na nabití baterie. Díky tomu nezpůsobuje toto nastavení zrychlení vybíjení zařízení.

Nastavení intervalů odesílání a aktualizace polohových dat bylo použito u druhé verze aplikace. S možností běhu aplikace na pozadí i při zamknutém stavu se spotřeba snížila na rozsah 4–10 %, a to v závislosti na typu a stáří zařízení a kondici jeho baterie. Nejnižší

spotřebu vykazovalo zařízení iPhone 11, průměrnou spotřebu 6 % zařízení Samsung Galaxy A40 a iPhone 7 Plus. U staršího zařízení Xiaomi Redmi 4s byla již zaznamenána vyšší spotřeba, stejně tak vyšší pozorované zahřívání zařízení.

7.2 Testování druhé verze aplikace

Z důvodu odhalení většího počtu nedostatků a chyb v první verzi aplikace Flox byl zahájen vývoj druhé verze této aplikace. Tato verze vznikla jako nový projekt za použití nástroje Flutter a využívá některé funkce implementované v první verzi. Příkladem těchto funkcí je helper pro komunikaci pomocí HTTP či statická třída pro ovládání funkcí pro získávání aktuální polohy mobilního zařízení.

Ostatní funkce a obrazovky aplikace byly přepracovány. Jednalo se z pohledu aplikační logiky především o strukturu modelových tříd aplikace a napojení jejich dat na vykreslovací prvky aplikace pomocí *providerů*. Z pohledu grafického uživatelského rozhraní se změna týkala rozmístění ovládacích prvků na všech obrazovkách, přepracování způsobu navigace uživatele v aplikaci a úpravu responsivity na různých velikostech obrazovek zařízení.

Při testování druhé verze aplikace byl kladen důraz na správné fungování všech funkcí aplikace, komfortní a funkční používání ovládacích prvků aplikace a fungování nativních funkcí, jako jsou např. notifikace. Během testování jedné z průběžných testovacích verzí aplikace došlo k odhalení problémů při sdílení aktuální polohy uživatele při běhu aplikace na pozadí. Tato chyba byla následně odstraněna implementací sdílení polohy na pozadí pomocí zdrojového kódu v nativních jazycích pro vývoj mobilních aplikací (viz kapitola 5.4). Dalším předmětem testování aplikace bylo zkoumání rozdílů chování aplikace na zařízeních s různými operačními systémy, odhalování příčiny těchto rozdílů a jejich eliminace.

Testování druhé verze aplikace probíhalo iterativní metodou a mezi uživatele bylo publikováno celkem 26 sestavení obsahujících opravy jednotlivých chyb, které byly během tohoto testování odhaleny.

7.2.1 Testování v Android Alfa kanálu

Pro testování druhé verze aplikace již nebyl instalační balíček distribuován pouhým sdílením souboru na cloudových službách. Pro aplikaci určenou k instalaci na zařízení s operačním systémem Android byl vytvořen kanál pro Alfa testování v obchodě Google Play Store.

Založení tohoto kanálu vyžaduje založení vývojářského účtu. Tento účet je založen s osobními daty vývojáře (nebo vývojářské firmy), z nichž některé jsou v obchodě viditelné pro uživatele mobilního zařízení, který si jím vydanou aplikaci hodlá pořídit. Další nutností je úhrada jednorázového poplatku za zřízení účtu.

Proces vydání aplikace v Alfa kanálu je poté započat vytvořením položky aplikace v ovládacím panelu vývojářského účtu. V této položce je možné vytvořit různá vydání aplikace ať už testovací, nebo veřejná. Protože k testování aplikace vydané v Alfa kanálu má přístup pouze omezený počet testovacích uživatelů, je třeba přiřadit tyto uživatele do seznamu testerů. Těm je pak umožněno aplikaci zobrazit v aplikaci Google Play na jejich zařízení a aplikaci Flox nainstalovat.

Pokud dojde k nalezení chyby v aplikaci, je po odstranění chyby možné nahrát nové sestavení aplikace. Toto sestavení je označeno pořadovým číslem, které musí být v rámci jedné aplikace, nahrané do vývojářského účtu, unikátní. Po nahrání tohoto sestavení a jeho vydání v Alfa kanálu je uživatelům zapojeným do testování aplikace umožněno aplikaci aktualizovat v aplikaci Google Play.

7.2.2 Testování v iOS TestFlight

Obdobně jako testování aplikace pro Android v Alfa kanálu Google Play Store byla testována i aplikace pro iOS. Společnost Apple pro testování aplikací pro svůj obchod AppStore využívá nástroj TestFlight. Jedná se o „kopii“ samotného obchodu AppStore, odkud je možné instalovat nahrané aplikace do zařízení. Aplikace zde jsou však dostupné pouze vybraným uživatelům s daným AppleID.

Nahrání aplikace do prostředí TestFlight je mírně složitější než do Alfa kanálu pro Android. Nejprve je třeba založení vývojářského účtu Apple Developer. Stejně jako vývojářský účet od Google i tento účet vyžaduje vyplnění osobních údajů vývojáře, poplatek za vedení účtu není však jednorázový, ale platí pouze 1 rok. Pro nahrání aplikace do prostředí vývojářského účtu je třeba její sestavení pomocí programu Xcode. Jednotlivá sestavení je třeba taktéž označovat unikátním číslem. Ta nemusí být na rozdíl od Google Play unikátní v rámci celé aplikace, ale pouze v rámci jedné verze. K odeslání aplikace je třeba podepsat ji certifikátem vývojáře, který lze získat v prostředí vývojářského účtu. Pro uložení sestavené aplikace slouží prostředí App Store Connect. Do tohoto prostředí se vývojář přihlašuje údaji svého vývojářského účtu.

Do prostředí App Store Connect je aplikace odeslána přímo z aplikace Xcode na počítači vývojáře. Po jejím zpracování je vývojáři umožněno ji publikovat pro vybrané testovací uživatele v prostředí TestFlight. Těm je pak aplikace přístupná k instalaci do mobilního zařízení prostřednictvím mobilní aplikace TestFlight. Podobně jako v Alfa kanálu Google Play je i zde možné opravenou chybu distribuovat mezi testující uživatele prostřednictvím aktualizace aplikace.

Publikování jednoho sestavení aplikace v prostředí TestFlight je možné pouze po omezenou dobu 90 dní. Poté přestává být aplikace dostupná pro instalaci na mobilní zařízení a je nutné publikovat nové sestavení, nebo zveřejnit stávající v obchodě AppStore.

7.2.3 Testování aplikace v terénu

Aplikace byla po vydání na testovacích kanálech obou platforem průběžně testována několika uživateli. Samotné testování probíhalo při různých příležitostech, např. při pěší turistice, cyklistice nebo během cest autem a prostředky hromadné dopravy.

Kromě testů funkčnosti aplikace otestoval každý z účastníků také samotné ovládání aplikace. Uživatelé si vyzkoušeli vytváření skupin, použití zvacích kódů i dynamických odkazů ve formě textu i QR kódu a vytváření míst setkání, na kterých se poté setkali. Díky vyzkoušení všech ovládacích prvků bylo zajištěno, že aplikace a její grafické uživatelské prostředí je dostatečně intuitivní a nabízí dobrou uživatelskou zkušenost při jejím používání.

Testy reálného použití aplikace trvaly různou dobu. Tato doba se lišila v závislosti na situaci, při které uživatel aplikaci spustil. Pokud se jednalo o pouhou procházku v přírodě či ve městě, jednalo se o používání o délce nižších desítek minut. Při používání aplikace během delší turistiky nebo cykloturistiky se pak jednalo o nižší až střední jednotky hodin (v průměru 1,5–4 hodiny).

7.3 Veřejné vydání aplikace

Po úspěšném důkladném otestování funkcí a ovládání aplikace byla tato aplikace zveřejněna v obchodech Google Play Store a AppStore. Takto zveřejněná aplikace je poté dostupná pro instalaci všem uživatelům mobilních zařízení s operačními systémy Android a iOS.

Aby bylo možné aplikaci zveřejnit, je třeba vyplnit prohlášení o její nezávadnost a vhodnosti či nevhodnosti pro děti určitého věku. Jelikož aplikace Flox neobsahuje žádný nevhodný obsah, je označena jako vhodná pro děti od 4 let. Dále je třeba vyplnit informace o aplikaci, její popis a ukázky v podobě snímků obrazovky mobilních zařízení různých velikostí. Další položkou, kterou je nutné vložit, je ikona aplikace v předem určeném rozlišení.

Po vyplnění všech nutných údajů a obrázků je aplikace předložena ke kontrole správci vydání v obchodech s aplikacemi. Tato kontrola může trvat i několik dní v závislosti na složitosti aplikace a kvalitě poskytnutých informací o ní. Pro otestování aplikace před vydáním v obchodě AppStore bylo třeba vytvořit krátké video popisující její funkci.

Na rozdíl od vydání aplikace pro Google Play Store, které bylo bez výhrad schváleno v prvním sestavení určeném pro veřejné vydání, aplikace pro vydání v AppStore byla zamítnuta. Bylo to z důvodu absence možnosti registrace uživatele pomocí AppleID, kterou společnost Apple ve svých aplikacích vyžaduje, pokud je umožněna registrace pomocí jiných sociálních sítí. Po implementaci této funkce (která je dostupná pouze pro zařízení s operačním systémem iOS) byla již aplikace schválena a zveřejněna.

Aplikace je nyní dostupná v obou obchodech pod názvem Flox a je možné ji zdarma stáhnout a nainstalovat do chytrých telefonů s operačními systémy iOS a Android.

7.4 Pokračování projektu

Po vydání aplikace a jejím rozšíření mezi více uživateli chytrých mobilních zařízení je třeba sledovat ohlasy uživatelů aplikace. Na základě těchto ohlasů bude potom možné navázání na vývoj projektu implementací nových funkcí aplikace nebo vyladění nedokonalostí aplikace, které se mohou projevit při jejím dlouhodobějším používání.

V této kapitole jsou shrnuty některé body, které by se mohly ukázat jako vhodné při budoucím rozšiřování spektra funkcí aplikace.

7.4.1 Využití mapového podkladu pro navigaci

Aplikace využívá mapový podklad Google Maps pro zobrazení polohy uživatelů připojených ke stejné skupině na mapě. Další funkcí je vytváření a plánování míst srazu skupiny, na které jsou její členové upozorněni prostřednictvím notifikací.

S využitím dalších funkcí poskytovaných platformou Google Maps by mohla být vhodným rozšířením funkcí aplikace možnost spuštění hlasové navigace k místu srazu nebo k aktuální pozici jiného člena skupiny. Navigaci by bylo nutné implementovat tak, aby byla schopná upravovat navigační instrukce podle změn polohy cílových bodů (uživatele či místa setkání).

7.4.2 Rozšíření podporovaných zařízení

V současném stavu je aplikace přizpůsobena pouze pro používání na chytrých telefonech a zařízeních s délkou úhlopříčky displeje do cca 6,5". Implementací podpory zařízení s větší velikostí displeje by bylo umožněno aplikaci pohodlně využívat např. na tabletech disponujících hardwarem pro určení polohy a mobilním internetovým připojením. Tato změna by si vyžádala rozšíření responsivity aplikace a změnu rozvržení ovládacích prvků na jejích obrazovkách.

Další kategorií chytrých zařízení, na která by mohla aplikace být rozšířena, jsou chytré hodinky. Tyto hodinky většinou spolupracují s chytrým telefonem, se kterým jsou neustále

spárovány. Přenesení některých funkcí na zápěstí uživatele by mohlo ale poskytnout větší komfort. Mezi těmito funkcemi by mohlo být např. nastavitelné připomenutí času srazu na daném místě či navigace uživatele k místu srazu (viz kapitola 7.4.1). Pomocí hodinek s podporou samostatného připojení k mobilnímu internetu by pak mohla přibýt možnost sdílení aktuální polohy uživatele. Tuto funkci ale chytré hodinky doposud v České republice nenabízejí.

7.4.3 Nativní programování

Jak bylo zmíněno v několika předchozích kapitolách, multiplatformní programování mobilních aplikací s sebou přináší některá omezení přístupu k využívání nativních funkcí mobilních zařízení. Pokud by došlo k obsáhlejší aktualizaci aplikace z důvodu rozšiřování jejích funkcí, bylo by vhodné tuto aplikaci rozdělit na dva samostatné projekty v nativních jazycích mobilních platforem. Pro systém Android by tedy vznikl projekt např. v jazyce Kotlin a pro iOS v jazyce Swift.

Kapitola 8

Závěr

Cílem této práce bylo vytvoření mobilní aplikace pro umožnění sledování polohy uživatelů ve skupině. Aplikace byla vytvořena pro obě majoritní mobilní platformy, Android a iOS. K vývoji aplikace byl použit nástroj Flutter, určený pro multiplatformní vývoj aplikací. Specifikace funkcí aplikace byla zhotovena na základě analýzy současných existujících aplikací, jež jsou určeny k podobnému využití, spolu s analýzou požadavků uživatelů, které byly zjištěny veřejným dotazováním.

Za účelem zpracování a ukládání uživatelských dat byl jako součást aplikace implementován webový server. Ten je napojen na relační databázi MariaDB. Přenos dat mezi ním a mobilní aplikací probíhá prostřednictvím HTTP požadavků na aplikační rozhraní typu REST.

K vytvoření návrhu a rozložení ovládacích prvků aplikace a jejich implementaci a finální zpracování byl použit styl Material Design. Pro některé prvky v aplikaci běžící na zařízeních se systémem iOS byl použit styl Cupertino Design. Při vytváření obrazovek bylo dbáno na snadnou ovladatelnost a co nejlepší uživatelskou zkušenost s grafickým rozhraním aplikace.

V průběhu vývoje aplikace byla zjištěna mnohá omezení funkcí nástrojů pro multiplatformní vývoj mobilních aplikací na rozdíl od použití nativních jazyků obou platforem. Hlavním problémem byla nemožnost aktualizace polohy uživatele při běhu aplikace na pozadí. Tuto funkci bylo třeba implementovat nativními jazyky Kotlin a Swift zvlášť pro Android a iOS a spouštět pomocí volání metod komunikačním kanálem.

Z tohoto důvodu bych označil Flutter jako nástroj vhodný spíše pro implementaci aplikací, jejichž cílem je pouze zobrazovat obsah a zpracovávat data zadaná uživatelem. Příkladem takových aplikací jsou mobilní e-shopy, ve kterých není třeba zajišťovat běh aplikace na pozadí. Pro tyto funkce je vhodnější aplikace vyvíjet v nativních jazycích, které mají větší přístup ke všem nativním funkcím mobilního zařízení.

Aplikace pro obě platformy byla zveřejněna v internetových obchodech s mobilními aplikacemi Google Play Store a Apple AppStore. Zde jsou zdarma k dispozici široké veřejnosti.

Literatura

[1] DENSO WAVE Inc.. *What is a QR code?* [online]. [cit. 2020-05-21]. Dostupné z: https://www.qrcode.com/en/about/.

[2] Google. *Build beautiful products, faster.* [online]. [cit. 2020-05-23]. Dostupné z: https://www.material.io/.

[3] Google. *Cupertino (iOS-style) widgets* [online]. [cit. 2020-05-23]. Dostupné z: https://flutter.dev/docs/development/ui/widgets/cupertino.

[4] Google. *Firebase Cloud Messaging* [online]. [cit. 2020-05-17]. Dostupné z: https://firebase.google.com/docs/cloud-messaging.

[5] Google. *Firebase dynamic links* [online]. [cit. 2020-05-21]. Dostupné z: https://firebase.google.com/docs/dynamic-links.

[6] Google. *Flutter Documentation* [online]. [cit. 2020-01-14]. Dostupné z: https://flutter.dev/docs.

[7] Google. *Future<T> class* [online]. [cit. 2020-01-14]. Dostupné z: https://api.flutter.dev/flutter/dart-async/Future-class.html.

[8] Google. *Writing custom platform-specific code* [online]. [cit. 2020-05-21]. Dostupné z: https://flutter.dev/docs/development/platform-integration/platform-channels.

[9] Hrubý, M. *Geografické Informační Systémy (GIS) – Studijní opora* [online]. [cit. 2020-05-19]. Dostupné z: http://perchta.fit.vutbr.cz/vyuka-gis/uploads/1/GIS-final2.pdf.

[10] Mainkar, P. a Giordano, S. *Google Flutter Mobile Development Quick Start Guide: Get up and running with iOS and Android mobile app development.* Packt Publishing [cit. 2020-04-29]. ISBN 978-1789344967.

[11] Mew, K. *Learning Material Design.* Packt Publishing [cit. 2020-05-23]. ISBN 978-1785289811.

[12] Mitra, N. a Lafon, Y. *SOAP* [online]. [cit. 2020-05-18]. Dostupné z: https://www.w3.org/TR/soap12-part0/.

[13] Otwell, T. *Blade Templates* [online]. [cit. 2020-05-17]. Dostupné z: https://laravel.com/docs/7.x/blade.

[14] Penland, J. *A Complete Guide and List of HTTP Status Codes* [online]. [cit. 2020-01-14]. Dostupné z: https://kinsta.com/blog/http-status-codes/.

[15] Pichlík, R. *A REST* [online]. [cit. 2020-05-18]. Dostupné z: https://dagblog.cz/a-rest-c5156313d79e.

[16] Weill, L. R., Grewal, M. S. a Andrews, A. P. *Global Positioning Systems, Inertial Navigation, and Integration*. John Wiley & Sons, Inc. [cit. 2020-05-19]. ISBN 9780470041901.

[17] Yii Software. *The Definitive Guide to Yii 2.0* [online]. [cit. 2020-01-14]. Dostupné z: https://www.yiiframework.com/doc/guide/2.0/en.

[18] Yii Software. *Routing* [online]. [cit. 2020-01-14]. Dostupné z: https://www.yiiframework.com/doc/guide/2.0/en/rest-routing.

[19] Čápka, D. *MVC architektura* [online]. [cit. 2020-01-10]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor.

Příloha A

Obsah přiloženého paměťového média

```
/
├── dip.pdf ..... Technická dokumentace
├── media/
│   ├── poster/
│   │   ├── cs.pdf
│   │   └── en.pdf
│   └── video.mp4
├── project/
│   ├── mobile/ ..... Zdrojové soubory mobilní části aplikace
│   └── server/ ..... Zdrojové soubory serverové části aplikace
└── text/ ..... Zdrojové soubory technické dokumentace
```