

Kontinuální zobrazování polohy mobilních zařízení v mapovém podkladu

Bc. Daniel Teplý

Diplomová práce
2015



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2014/2015

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Daniel Teplý**
Osobní číslo: **A12427**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Počítačové a komunikační systémy**
Forma studia: **prezenční**

Téma práce: **Kontinuální zobrazování polohy mobilních zařízení v mapovém podkladu**
Téma anglicky: **The Continuous Display of Mobile Device Positions in a Map Context**

Zásady pro vypracování:

1. Vypracujte rešerši možností iOS 8.
2. Popište nově vzniklý programovací jazyk Swift.
3. Analyzujte požadavky na mobilní aplikaci určenou pro snímání polohy z GPS zařízení a proveďte její návrh.
4. Dle návrhu vytvořte mobilní aplikaci pro iOS pomocí jazyku Swift.
5. Navrhněte a realizujte odesílání nasnímaných dat do jednoduché webové aplikace.
6. Věnujte pozornost zabezpečení aplikace.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:

1. APPLE INC. Uživatelské příručka pro iPhone s iOS 8.1. 2014. Dostupné z: <https://itunes.apple.com/cz/book/uzivatelska-prirucka-pro-iphone/id925163635?mt=11>
2. APPLE INC. The Swift Programming Language. 2014. Dostupné z: <https://itunes.apple.com/cz/book/swift-programming-language/id881256329?mt=11>
3. APPLE INC. Using Swift with Cocoa and Objective-C. 2014. Dostupné z: <https://itunes.apple.com/cz/book/using-swift-cocoa-objective/id888894773?mt=11>
4. AppCoda Community – Learn iOS Programming and Build iPhone App [online]. 2015 [cit. 2015-02-04]. Dostupné z: <http://www.appcoda.com>
5. Ray Wenderlich | Tutorials for iPhone / iOS Developers and Gamers [online]. 2015 [cit. 2015-02-04]. Dostupné z: <http://www.raywenderlich.com>
6. WWDC 2014 Session Videos: Learn about the latest in iOS and OS X with WWDC 2014 session videos. Apple Developer [online]. 2014 [cit. 2015-02-04]. Dostupné z: <https://developer.apple.com/videos/wwdc/2014/>

Vedoucí diplomové práce:

Ing. Petr Šilhavý, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

12. ledna 2015

Termín odevzdání diplomové práce:

15. května 2015

Ve Zlíně dne 6. února 2015

doc. Mgr. Milan Adámek, Ph.D.
děkan



Ing. Miroslav Matýšek, Ph.D.
ředitel ústavu

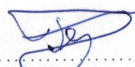
Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s tím, že vyrovnaní případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 17.5.2015


.....
podpis diplomanta

ABSTRAKT

V rámci této diplomové práce byla vytvořena mobilní aplikace pro platformu iOS, která poskytuje snímání polohy přihlášených uživatelů a kontrolu, zda se vyskytují v určité vymezené zóně. Tyto údaje jsou pak k dispozici administrátorovi, aby mohl sledovat, kde se pohybují uživatelé spadající pod jeho správu. Veškeré údaje o lokacích, zónách či uživateli se ukládají na server do MSSQL databáze. Komunikace s databází probíhá pomocí webového API a jeho metod. Pro tvorbu mobilní aplikace bylo využito nového programovacího jazyku Swift od společnosti Apple, kterému je věnována velká část teoretické části. Součástí práce je také popis poslední verze mobilního operačního systému iOS.

Klíčová slova: Apple, mobilní operační systém, iOS, programovací jazyk, Swift, mobilní aplikace, snímání polohy

ABSTRACT

In the context of this diploma thesis was created a mobile application for the iOS platform, which provides a position sensing of logged users and check, whether they occur in a defined zone. Then these data are available to the administrator to track, where users under his management are moving. All information about locations, zones or users are stored on the server in MSSQL database. Communication with database is done via web API and its methods. For creating mobile application was used a new programming language Swift by Apple, which is well described at theoretical part. The work also includes description of the latest version of the iOS mobile operating system.

Keywords: Apple, mobile operating system, iOS, programming language, Swift, mobile app, position sensing

Chtěl bych poděkovat panu Ing. Petru Šilhavému, Ph.D. za odborné vedení při tvorbě této diplomové práce.

Zároveň bych rád poděkoval své rodině a přítelkyni za veškerou podporu.

„Deciding what not to do is as important as deciding what to do.“

Steve Jobs

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD	10
I TEORETICKÁ ČÁST	11
1 MOBILNÍ OPERAČNÍ SYSTÉM IOS	12
1.1 HISTORIE.....	12
1.2 SYSTÉM IOS 8	13
1.2.1 Continuity.....	13
1.2.2 Widgety a notifikace s akcemi	13
1.2.3 Extensibility	13
1.2.4 Ostatní novinky	13
2 PROGRAMOVACÍ JAZYK SWIFT	15
2.1 ZÁKLADY	15
2.1.1 Konstanty a proměnné.....	16
2.1.2 Komentáře	16
2.1.3 Středníky	17
2.1.4 Celá čísla	17
2.1.5 Desetinná čísla.....	17
2.1.6 Určování datových typů a jejich ověřování.....	17
2.1.7 Logické hodnoty.....	17
2.1.8 Tuples (n-tice)	17
2.1.9 Optionals	18
2.2 ZÁKLADNÍ OPERÁTORY	21
2.2.1 Operátor přiřazení.....	21
2.2.2 Aritmetické operátory.....	21
2.2.3 Sloučené operátory přiřazení.....	22
2.2.4 Porovnávací operátory.....	22
2.2.5 Ternární podmínkový operátor.....	22
2.2.6 Nil coalescing operátor.....	23
2.2.7 Rozsahové operátory	23
2.2.8 Logické operátory	24
2.3 TEXTOVÉ ŘETĚZCE A ZNAKY	25
2.3.1 Řetězcové literály	25
2.3.2 Inicializace prázdného textového řetězce.....	25
2.3.3 Modifikovatelnost textových řetězců	25
2.3.4 Práce se znaky	26
2.3.5 Spojování textových řetězců a znaků	26
2.3.6 Interpolace textových řetězců.....	26
2.3.7 Porovnávání textových řetězců	27
2.3.8 Unicode	27
2.4 KOLEKCE – POLE, SADY A SLOVNÍKY	27
2.4.1 Modifikovatelnost kolekcí.....	28
2.4.2 Pole.....	28
2.4.3 Sady.....	31
2.4.4 Slovníky	32
2.5 ŘÍDÍCÍ STRUKTURY.....	36
2.5.1 „For“ cykly.....	36

2.5.2	„While“ cykly.....	37
2.5.3	Podmínkové příkazy.....	38
2.5.4	Příkazy pro přesun vykonávání kódu.....	41
2.6	FUNKCE.....	41
2.6.1	Definice a volání funkcí.....	41
2.6.2	Parametry funkcí a návratové hodnoty.....	42
2.6.3	Názvy parametrů funkcí.....	43
2.7	ENUMERATIONS (VÝČTY).....	44
2.7.1	Tvorba výčtů.....	45
2.7.2	Práce s výčtem ve „switch“ příkazu.....	45
2.7.3	Přiřazené hodnoty.....	46
2.7.4	„Raw“ hodnoty.....	47
2.8	TŘÍDY A STRUKTURY.....	47
2.8.1	Porovnání tříd a struktur.....	48
2.8.2	Hodnotové typy.....	49
2.8.3	Referenční typy.....	50
3	ANALÝZA POŽADAVKŮ NA APLIKACI.....	51
II	PRAKTICKÁ ČÁST.....	52
4	MOBILNÍ APLIKACE.....	53
4.1	PŘIHLÁŠENÍ.....	53
4.2	ROZHRANÍ ADMINISTRÁTORA.....	54
4.2.1	Sekce „Mapa“.....	54
4.2.2	Sekce „Skupina“.....	55
4.2.3	Sekce „Uživatel“.....	58
4.2.4	Sekce „O aplikaci“.....	59
4.3	ROZHRANÍ BĚŽNÉHO UŽIVATELE.....	60
4.3.1	Sekce „Mapa“.....	60
4.3.2	Sekce „Uživatel“.....	61
4.3.3	Sekce „O aplikaci“.....	62
5	MS SQL DATABÁZE.....	63
5.1	TABULKA „USER“.....	63
5.2	TABULKA „LOCATION“.....	64
5.3	TABULKA „AUTHSESSION“.....	65
5.4	TABULKA „GROUP“.....	65
5.5	TABULKA „ZONE“.....	66
6	POPIS API METOD.....	67

6.1	/LOGIN.....	67
6.2	/LOGOUT.....	69
6.3	/GETUSERSBYGROUP	69
6.4	/GETLOCATIONSFORUSER	71
6.5	/GETLOCATIONSBYDATEFORUSER	72
6.6	/ADDLOCATIONFORUSER	74
6.7	/GETZONEFORUSER.....	75
6.8	/SETZONEFORUSER	76
6.9	/CREATEUSER	77
6.10	/EDITUSER	78
6.11	/CHANGEPASSWORD	79
6.12	/DELETEUSER	80
6.13	/GETGROUPNAME	81
6.14	/EDITGROUPNAME	82
6.15	/SENDEMAIL	83
6.16	/ISVALIDSESSION	84
7	POPIS KÓDU MOBILNÍ APLIKACE	86
7.1	STORYBOARD, LAUNCHSCREEN A SLOŽKA IMAGES	86
7.2	TŘÍDA „APPDELEGATE“	86
7.3	VIEW CONTROLLER TŘÍDY	86
7.3.1	LoginViewController	86
7.3.2	MapViewController	87
7.3.3	GroupTableViewController	87
7.3.4	UserTableViewController	87
7.3.5	AboutViewController	87
7.4	OSTATNÍ TŘÍDY	87
7.5	EXTERNÍ KNIHOVNY	89
7.5.1	DatePickerDialog	89
7.5.2	Locksmith.....	89
7.5.3	SwiftHTTP	89
7.5.4	SVProgressHUD	89
8	ZABEZPEČENÍ APLIKACE.....	90
	ZÁVĚR	91
	SEZNAM POUŽITÉ LITERATURY.....	92
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	94
	SEZNAM OBRÁZKŮ	95
	SEZNAM PŘÍLOH	97

ÚVOD

Když se podíváme na současnou generaci, zjistíme, že chytré mobilní telefony s nějakým mobilním operačním systémem má už téměř každý. To znamená, že v dnešním světě se stávají mobilní aplikace, čím dál tím více důležitější, protože mohou různými způsoby zjednodušovat život. Můžeme pomocí nich vyhledávat nejbližší spoje, když chceme někam cestovat, nebo se nechat navigovat z jednoho místa na jiné či si nechat doporučit nějakou kvalitní restauraci nebo pivnici dle toho, kde se právě nacházíme. Mobilní aplikace nám nabízí nespočet možností využití, v obchodech s nimi se dají najít kancelářské aplikace, velké množství mobilních her, aplikace pro focení a úpravy fotek atd.

Nejvýraznějšími mobilními operačními systémy na trhu jsou iOS, Android a Windows Phone. Každý z nich je něčím jiný, má své charakterizující prvky a způsoby ovládání. A s tím i souvisí komunity uživatelů, kteří se vytváří okolo jednotlivých systémů a každá z nich si ráda prosazuje ten svůj. Co se týče mě, tak jsem si vyzkoušel všechny tři systémy a rozhodl jsem se pro iOS, protože co se týká přehlednosti, rychlosti či bezpečnosti si myslím, že nemá prozatím konkurenci.

Vzhledem k tomu, že jsem si tento systém velice oblíbil, tak jsem se rozhodnul, že v rámci diplomové práce bych se mu rád věnoval a zároveň demonstroval, jak se tvoří aplikace pro takovýto systém. A jelikož před necelým rokem Apple vydal svůj vlastní programovací jazyk Swift, tak jsem se rozhodl použít pro vývoj aplikace právě jej namísto staršího používaného jazyku Objective-C.

I. TEORETICKÁ ČÁST

1 MOBILNÍ OPERAČNÍ SYSTÉM IOS

Mobilní operační systém iOS (původně iPhone OS) vytvořila a vyvinula společnost Apple a je možno jej používat výhradně na hardwaru od této společnosti. V současné době běží na těchto zařízeních: iPhone, iPad a iPod Touch. Jeho upravená verze je implementovaná do nových „chytrých“ hodinek Apple Watch. [1]

1.1 Historie

Úplně první verze tohoto mobilního operačního systému byla vydána 29. června 2007 současně s iPhonem první generace. Až do června roku 2010 se tento systém jmenoval „iPhone OS“ a poté došlo k přejmenování na iOS. [2]

Od vydání v roce 2007 se iOS začal postupně vyvíjet. V druhé verzi se objevil App Store obchod s aplikacemi, kde mohou vývojáři třetích stran publikovat své aplikace. Třetí verze přinesla nové možnosti jako kopírování a vkládání textu, vyhledávání v telefonu pomocí Spotlight či push notifikace. A současně byl představen také iPad, tudíž iOS musel začít podporovat aplikace s větším rozlišením. Od čtvrté verze iOS je podporovaný multitasking, kdy uživatel po dvojitým tapnutí na „Home“ tlačítko má možnost zobrazit seznam otevřených aplikací. Dále byl představen v této verzi FaceTime pro videohovory, byla přidána možnost seskupovat aplikace do složek a iOS začal podporovat retina displeje s vysokým rozlišením. V páté verzi se objevila hlasová asistentka Siri, notificační centrum, uživatelé si mohli začít posílat mezi sebou zprávy pomocí iMessage a iOS se od této verze dá aktualizovat „over-the-air“, to znamená bez nutnosti připojení k PC s iTunes. A navíc byla v této verzi uvedena do chodu cloudová služba iCloud. Šesté vydání tohoto mobilního operačního systému přineslo zcela nové mapy nezávislé na Googlu a množství různých vylepšení jako je integrace Facebooku, sdílené fotky či zlepšení Siri. V roce 2013 došlo k největší grafické změně v systému iOS. Jeho sedmá verze přináší zcela přepracovaný design, který měl na starost především Jony Ive. Zároveň přibylo ovládací centrum pro jednoduché zapínání WiFi, Bluetooth, režimu „Nerušit“ atd., byl přepracován multitasking, změnila se aplikace „Fotky“ a „Fotoaparát“ a mnoho dalšího. A dosud dostupný AirDrop pro posílání souborů mezi systémy OS X je od této verze možný používat i v zařízeních s iOS. [3][4]

1.2 Systém iOS 8

Od podzimu roku 2014 je k dispozici iOS 8, který je aktuálně dostupný ve verzi iOS 8.3. Zatímco iOS 7 přinesl největší vizuální změny, iOS 8 se zaměřil více na vývojáře. Přinesl pro ně velké množství nových funkcí a nástrojů, pomocí kterých mohou aplikace získat více schopností než kdy předtím. Tato poslední verze systému se stala také nejvíce otevřenou verzí vůbec, co se týká modifikovatelnosti či vzájemné spolupráce aplikací. [4]

1.2.1 Continuity

„Continuity“ je velice výraznou novou vlastností, která zajišťuje interaktivitu iPhonů či iPadů s Mac počítačem s OS X Yosemite. Uživatel může např. pomocí tzv. „Handoff“ rozeslat mail na iPhonu, pak přejde k Macu a zde může pokračovat v psaní přesně tam kde skončil. Pomocí „AirDrop“ lze snadno přenášet soubory z telefonu do počítače. Nebo zajímavou novinkou je možnost přijímat telefonní hovory a klasické SMS zprávy i na Mac počítači. [4]

1.2.2 Widgety a notifikace s akcemi

V iOS 8 se otevřela možnost pro vývojáře vytvářet ke svým aplikacím interaktivní widgety do notifikačního centra. Zároveň došlo k vylepšení notifikací, že mohou nabízet různé akce. Např. když dojde notifikace s emailem, tak kromě jeho otevření jej lze nyní z notifikace rovnou i smazat. [4]

1.2.3 Extensibility

Extensibility neboli rozšiřitelnost je další velkou novinkou. Systém začal podporovat klávesnice třetích stran jako je Swift, Swype či Fleksy. Aplikace jsou schopny mezi sebou sdílet data a provádět různé interakce. A vývojáři dostávají přístup k TouchID systému rozeznávání otisků prstů, aby mohli tento senzor využívat pro zabezpečení svých aplikací. [4]

1.2.4 Ostatní novinky

Mezi další novinky patří přepracování aplikace „Zprávy“, která umožňuje sdílení lokace, posílání audio zpráv a mnoho dalšího. Vylepšení se také dočkal iCloud. Díky iCloud Drive ho lze využívat jako cloudové úložiště libovolných souborů. A také se zde objevila služba Photo Library, která je určena jako hlavní místo pro zálohu všech fotek ze zařízení. Na

závěr je důležité ještě zmínit novou systémovou aplikaci „Zdraví“, který slouží jako centrální místo pro zaznamenávání všech pohybových aktivit a zdravotních údajů uživatele. [4]

2 PROGRAMOVACÍ JAZYK SWIFT

Swift je novým programovacím jazykem pro vývoj iOS a OS X aplikací, který vynalezla společnost Apple. Poprvé byl představen veřejnosti 2. června 2014 na WWDC. Vychází z jazyku C a Objective-C. Swift je bezpečnější a přidává moderní prvky tak, aby bylo programování jednodušší. Vyvíjel se několik let a opírá se o oblíbený Cocoa a Cocoa Touch framework. Swift je podobný jazyku Objective-C a tudíž je pro programátory celkem snadné se jej naučit. Cílem bylo poskytnout nový jazyk, který bude přehlednější, svižnější a více uživatelsky přívětivý. Z velkého množství novinek se dá zmínit například možnost absence středníků za každý řádkem kódu, automatické odvození datových typů u proměnných atd. Swift také podporuje nový režim Playground, kde si programátor může za běhu aplikace experimentovat se svým kódem a vidět okamžitě výsledky bez nutnosti opětovné kompilace a spouštění aplikace. [5][6]

2.1 Základy

Swift poskytuje své vlastní verze datových typů oproti Objective-C. Mezi základní patří Int pro celá čísla, Double a Float pro desetinná čísla, Bool pro logické hodnoty (0, 1) a String pro textové řetězce.

Podobně jako v jazyku C, Swift využívá proměnných pro ukládání a odkazování se na jejich hodnoty dle jejich názvu. Kromě proměnných se také využívá v hojné míře konstant, které slouží pro ukládání neměnných hodnot.

Novým zajímavým datovým typem oproti Objective-C jsou tzv. "tuples". Jedná se o skupinu různých hodnot a využívá se jich například při definici funkcí, když chceme vrátit více než jednu hodnotu.

Ve Swiftu byly představeny také tzv. optionals, které řeší zda má proměnná uloženou nějakou hodnotu či ne. Je to podobné jako nil v Objective-C, ale optionals fungují pro jakýkoliv datový typ, ne pouze pro třídy. Optionals jsou bezpečnější a jasnější než nil pointery u Objective-C. Jsou také příkladem toho, že Swift je bezpečným jazykem, co se týče datových typů. Pokud se někde v kódu například očekává uložení textového řetězce a my se snažíme uložit celé číslo, tak se nám to nepodaří. [6]

2.1.1 Konstanty a proměnné

Konstanty a proměnné jsou definovány názvem a jejich přiřazenou hodnotou určitého datového typu. Jakmile je hodnota konstanty jednou nastavena, tak už se nemůže měnit. Oproti tomu proměnné mohou svou hodnotu měnit.

Konstanty i proměnné musí být definovány ještě předtím než se použijí. Konstanty se deklarují pomocí klíčového slova “let” a proměnné pomocí “var”.

```
let konstanta = 5
var promenna = 3
```

Pokud definujeme tímto způsobem konstantu / proměnnou, tak se automaticky odvodí datový typ. Ovšem v některých případech (např. pokud není ihned nastavena při deklaraci počáteční hodnota proměnné) je vhodné datový typ deklarovat pomocí dvojtečky a názvu datového typu za názvem proměnné.

```
var promenna : Int
```

Konstanty i proměnné mohou být pojmenovány téměř jakkoliv, i včetně různých emotikon. Ale pokud je konstanta či proměnná určitého datového typu už deklarována, tak ji nemůžeme deklarovat znovu se stejným jménem nebo změnit její datový typ. Zároveň také nelze udělat z proměnné konstantu a naopak.

Pro zobrazení aktuální hodnoty (logování) proměnné či konstanty do konzole slouží funkce “println()”, což je náhrada za “NSLog()” v Objective-C. [6]

```
println("Máme rádi Swift")
```

2.1.2 Komentáře

Komentáře ve Swiftu jsou víceméně stejné jako v Objective-C. Jednořádkové komentáře se dělají pomocí dvojitého lomítka.

```
// jednořádkový komentář
```

A víceřádkové komentáře začínají lomítkem s hvězdičkou a jsou ukončeny hvězdičkou s lomítkem.

```
/* víceřádkový komentář
na 2 řádky */
```

Výhodou komentářů ve Swiftu oproti Objective-C je možnost mít vnořené víceřádkové komentáře v již existujícím víceřádkovém komentáři. [6]

2.1.3 Středníky

Oproti většině ostatních programovacích jazyků, Swift nevyžaduje psaní středníků za konec každého příkazu v kódu. Středníky jsou potřeba pouze pokud příkazy řetězíme v jednom řádku. [6]

```
let promenna = "test"; println(promenna)
```

2.1.4 Celá čísla

Swift poskytuje celá čísla se znaménkem (signed) a bez znaménka (unsigned) v 8, 16, 32 a 64 bitové formě. Ve většině případů si programátor vystačí s definicí celého čísla pomocí klíčového slova "Int", popřípadě "UInt". [6]

2.1.5 Desetinná čísla

Desetinná čísla pokrývají daleko větší rozsah než celá čísla. Swift pro ně nabízí 2 datové typy "Double" a "Float". "Double" reprezentuje 64 bitovou variantu s přesností alespoň 15 desetinných míst. "Float" reprezentuje 32 bitovou variantu s přesností pouhých 6 desetinných míst. [6]

2.1.6 Určování datových typů a jejich ověřování

Jak již bylo zmíněno, Swift odvozuje automaticky datový typ, pokud ho sami neurčíme. A poté při kompilaci ověřuje, zda se někde v kódu nesnažíme uložit do proměnné nesprávný datový typ. Díky této kontrole je nám umožněno včas odchytit a opravit chyby a můžeme tím ušetřit spoustu času. Proto se Swift označuje jako bezpečný jazyk na datové typy. [6]

2.1.7 Logické hodnoty

Pro ukládání logických hodnot slouží ve Swiftu datový typ "Bool", který může nabývat pouze hodnoty "true" nebo "false". [6]

2.1.8 Tuples (n-tice)

Tuples (n-tice) sdružují více hodnot do jedné sloučené hodnoty. Hodnoty v tuple mohou být jakéhokoliv datového typu a přitom každá může být jiného.

```
let http404Error = (404, "Not Found")
```

V uvedeném příkladu je tuple obsahující 2 datové typy “Int” a “String”.

Pro získání jednotlivých hodnot z n-tice existuje více způsobů. Jedním z nich je metoda, kdy si vytvoříme pomocnou proměnnou. Do “statusCode” se uloží celé číslo 404 a do “statusMessage” textový řetězec “Not Found”.

```
let (statusCode, statusMessage) = http404Error
```

Další metodou přístupu je pomocí čísla indexu.

```
http404Error.0 -> vrátí 404
```

```
http404Error.1 -> vrátí “Not Found”
```

Jednotlivé prvky v n-tici lze při definování také pojmenovat obdobně jako u slovníků. A poté se k nim přistupuje dle jejich názvu.

```
let http200Status = (statusCode: 200, description: “OK”)
```

```
http200Status.statusCode -> vrátí 200
```

```
http200Status.description -> vrátí “OK”
```

Tuples nacházejí nejčastější využití v případech, když máme nějakou funkci, z níž chceme vrátit více než jednu hodnotu. [6]

2.1.9 Optionals

Optionals jsou novinkou oproti jazyku Objective-C. Používají se v situacích, když může nastat, že proměnná nemá definovanou žádnou hodnotu. To znamená v případech, když chceme povolit přidělení hodnoty “nil” do proměnné.

Optional proměnná či konstanta se definuje přidáním otazníku za její datový typ. Poté lze této proměnné přidělit hodnotu “nil”.

```
var message : String? = “Ahoj, jak se máš?”
```

```
message = nil
```

Pokud se při definici proměnné nenastaví výchozí hodnota, tak se nastaví její hodnota automaticky na “nil”.

```
var message : String?
```

Pro zjištění, zda optional proměnná obsahuje nějakou hodnotu, lze použít “if” podmínku s operátorem porovnáním rovnosti (“==”) či nerovnosti (“!=”).

```
if convertedNumber != nil
{
    println ("convertedNumber obsahuje číselnou hodnotu.")
}
```

Po ověření, že proměnná obsahuje nějakou hodnotu, k ní můžeme přistoupit pomocí přidání vykřičníku za její název. Ten nám jasně říká, že daná proměnná má přiřazenou hodnotu a že ji můžeme použít. Této metodě se říká tzv. “forced unwrapping” optional hodnoty.

```
if convertedNumber != nil
{
    println ("convertedNumber obsahuje číselnou hodnotu
\\(convertedNumber!).")
}
```

Pozn.: Znaky “\\(navezPromenne)” se používají pro zobrazení hodnoty proměnné při vypisování do konzole.

Kromě metody “forced unwrapping” existuje také více doporučená metoda “optional binding”. Ta se používá tak, že se nejdříve zjistí, zda optional proměnná obsahuje hodnotu. Pokud ano, přidělí se tato hodnota nové dočasné proměnné či konstantě. “Optional binding” se může používat v “if” podmínkách nebo v cyklu “while”.

```
if let actualNumber = possibleNumber.toInt()
{
    println("\\'(possibleNumber)\\' má číselnou hodnotu
\\(actualNumber).")
}
else
{
    println("\\'(possibleNumber)\\' nemůže být převedeno na
číslo.")
}
```

Na výše uvedeném příkladu lze vidět, jak funguje “optional binding”. Proměnná “possibleNumber” je datového typu “String” a volá se na ní metoda “toInt()”, která převádí textový řetězec na celé číslo s návratovým datovým typem optional “Int?” (tzn. že buď se vrátí číslo nebo “nil”). Pokud lze textový řetězec převést na číslo (tzn. “toInt()” vrátí hodnotu), uloží se do konstanty “actualNumber” číselná hodnota. A tuto konstantu lze využívat v první větvi “if” podmínky. A ačkoliv tato konstanta je definována s výchozí hodnotou (nejedná se o optional), tak již není třeba používat pro její zobrazení vykřičník.

Pokud je potřeba hodnotu ve větvi podmínky nějak modifikovat lze použít dočasnou proměnnou místo konstanty pomocí “if var nazevPromenne = optionalPromenna”.

Zvláštním druhem optionals jsou implicitně rozbalované optionals. Definují se přidáním vykřičníku za datový typ proměnné místo otazníku. Používají se v případech, kdy optional proměnné nastavujeme ihned nějakou výchozí hodnotu a počítáme s tím, že stále nějakou bude nadále mít. Tento druh optionals se využívá hlavně při inicializaci tříd, kdy se při vytváření instance třídy přiřadí každé proměnné ihned nějaká hodnota.

```
let possibleString: String? = "Optional textový řetězec."  
let forcedString: String = possibleString!  
// vyžaduje vykřičník pro rozbalení  
let assumedString: String! = "Implicitně rozbalovaný optional  
textový řetězec."  
let implicitString: String = assumedString  
// nevyžaduje vykřičník pro rozbalení
```

Jak lze vidět z příkladu, při definici implicitně rozbalované optional proměnné není potřeba při přístupu k ní psát vykřičník pro její vynucené rozbalení, protože její rozbalení probíhá automaticky při každém požadavku na její hodnotu. Ovšem v obou případech rozbalování musíme dávat pozor, aby proměnná obsahovala nějakou hodnotu, jinak při rozbalení dojde k chybě.

K implicitně rozbalovaným optionals lze přistupovat také jako k normálním, takže se dá ověřovat “if” podmínkou, jestli obsahují nějakou hodnotu či je rozbalovat pomocí “optional binding”. [6]

2.2 Základní operátory

Operátor je speciálním symbolem nebo frází, která se používá pro kontrolu, změnu či spojování hodnot. Swift podporuje většinu standardních operátorů z jazyka C a zlepšuje některé jejich možnosti, aby se eliminovaly běžné chyby při psaní kódu.

Operátory se dělí na unární, binární a ternární. Unární pracují pouze s jednou hodnotou a zapisují se buď před ni (např. „!a“) nebo za ni (např. „i++“). Binární operátory vyžadují dvě hodnoty a zapisují se mezi ně (např. „1 + 2“). Ternární operátor existuje ve Swiftu pouze jeden a to ternární podmínkový operátor „(a ? b : c)“. [6]

2.2.1 Operátor přiřazení

Operátor přiřazení „a = b“ inicializuje nebo mění hodnotu „a“ na hodnotu „b“.

```
let b = 6
var a = 4
a = b
// proměnné „a“ je přidělena hodnota 6
```

Tento druh operátoru ve Swiftu nevrací sám hodnotu, takže se nedá používat v „if“ podmínkách při porovnávání dvou hodnot. Je zde potřeba použít operátor „==“. [6]

2.2.2 Aritmetické operátory

Swift podporuje čtyři základní aritmetické operátory: sčítání „+“, odčítání „-“, násobení „*“ a dělení „/“. Narozdíl od aritmetických operátorů v Objective-C, Swift ve výchozím nastavení nepovolí těmto operátorům, aby přetekla jejich hodnota.

Pro výpočet zbytku po dělení se používá následujícího operátoru „a % b“. Oproti Objective-C se dá ve Swiftu získat také zbytek po dělení desetinných čísel.

```
9 % 2.5
// vrátí 1.5
```

Pro inkrementaci hodnoty (zvýšení o 1) proměnné či konstanty se používá operátor „++“, pro dekrementaci (snížení o 1) operátor „--“. Tyto operátory se mohou zapisovat buď před proměnnou („++i“) nebo za ni („i++“). Pokud je operátor napsaný před proměnnou, tak nejdříve upraví její hodnotu a poté ji vrátí. V případě zápisu operátoru za proměnnou se vrátí hodnota a pak dojde až k její změně.

```
var a = 2
let b = ++a
// „a“ i „b“ se rovná 3
let c = a++
// „c“ se rovná 3, „a“ se rovná 4
```

Pro změnu znaménka hodnoty se používá unární operátor „-“. [6]

```
let ctyri = 4
let minusctyri = -ctyri
```

2.2.3 Sloučené operátory přiřazení

Tyto operátory kombinují přiřazení „=“ s nějakou další operací. Nejčastěji se jedná o kombinaci se sčítáním „+=“.

```
var a = 2
a += 2
// hodnota „a“ je rovna 4
```

Narozdíl od inkrementačních a dekrementačních operátorů tyto nevrací žádnou hodnotu. [6]

2.2.4 Porovnávací operátory

Swift podporuje všechny standardní porovnávací operátory:

- je rovno „a == b“
- není rovno „a != b“
- je větší než „a > b“
- je menší než „a < b“
- je větší než nebo rovno „a >= b“
- je menší než nebo rovno „a <= b“

Každý operátor porovnání vrátí „Bool“ hodnotu, jestli bylo tvrzení splněno či ne. Tyto operátory se často využívají ve větvičích „if“ podmínkách. [6]

2.2.5 Ternární podmínkový operátor

Jedná se o speciální operátor skládající se ze tří částí ve formě „otázka ? odpověď1 : odpověď2“. Pokud „otázka“ nabývá hodnoty „true“ vykoná se „odpověď1“ a vrátí její

hodnota. V opačném případě se vykoná „odpověď“ a vrátí její hodnota. Tento operátor je zkráceným zápisem „if“ větvičí podmínky. [6]

```
let vyskaObsahu = 40
let obsahujeHlavicku = true
let vyskaRadku = vyskaObsahu + (obsahujeHlavicku ? 50 : 20)
// řádek obsahuje hlavičku, jeho výška bude 90
```

2.2.6 Nil coalescing operátor

Tento binární operátor zapisovaný ve tvaru „a ?? b“ rozbaluje optional proměnnou „a“, pokud obsahuje nějakou hodnotu nebo vrací výchozí hodnotu „b“, pokud „a“ nabývá hodnoty „nil“. Proměnná „a“ musí být vždy optional a proměnná „b“ musí odpovídat datovému typu „a“. [6]

```
let vychoziBarva = "green"
var vlastniBarva : String? // výchozí hodnota je nil
var aktualniBarva = vlastniBarva ?? vychoziBarva
// „vlastniBarva“ je nil, takže „aktualniBarva“ se nastaví na
// „green“
vlastniBarva = "red"
aktualniBarva = vlastniBarva ?? vychoziBarva
// „vlastniBarva“ již není nil, takže „aktualniBarva“ se
// nastaví na „red“
```

2.2.7 Rozsahové operátory

Swift nabízí dva rozsahové operátory, které slouží pro vyjádření rozsahu hodnot.

Uzavřený rozsahový operátor „a..b“ definuje rozsah od „a“ do „b“ s tím, že jsou obě tyto hodnoty do něj zahrnuty. A hodnota „a“ nesmí být větší než hodnota „b“.

```
for index in 1..3
{
    println("Index je: \(index)")
}
// Index je: 1
// Index je: 2
// Index je: 3
```

Polootevřený rozsahový operátor „a..<b“ definuje rozsah od „a“ do „b“ s tím, že „b“ do něj není zahrnuto. A stále platí, že hodnota „a“ nesmí být větší než hodnota „b“. Tento druh rozsahového operátoru je užitečný například při procházení polí. [6]

```
let jmena = ["Daniel", "Petr", "Adam"]
let pocetJmen = jmena.count
for i in 0..
```

2.2.8 Logické operátory

Logické operátory modifikují či kombinují logické hodnoty „true“ a „false“. Swift podporuje tři standardní logické operátory: NOT, AND, OR.

Logický NOT operátor („!a“) invertuje logickou hodnotu „true“ na „false“ a naopak.

Logický AND operátor („a && b“) je logickým výrazem, kde musí být obě hodnoty „a“ i „b“ „true“, aby byl výsledek výrazu „true“.

Logický OR operátor („a || b“) je logickým výrazem, kde stačí, aby jedna z hodnot „a“ či „b“ byla „true“ a poté je výsledek výrazu „true“.

Tyto operátory mají časté využití v „if“ větvičkách podmínkách, kde se mohou různě kombinovat. [6]

```
if (zadanyKodOdDveri && naskenovanaSitnice) || maKlicOdDveri
{
    println ("Vítejte, přístup povolen!")
}
else
{
    println("PŘÍSTUP ZAMÍTNUT")
}
```

2.3 Textové řetězce a znaky

Textový řetězec je seřazenou kolekcí znaků a ve Swiftu je reprezentován datovým typem „String“. Pro jednotlivé znaky je definován datový typ „Character“. Tyto dva datové typy poskytují rychlý, Unicode kompatibilní způsob, jak pracovat s texty v kódu. [6]

2.3.1 Řetězcové literály

Řetězcový literál je pevná sekvence textových znaků ohraničená uvozovkami (“”), která se používá pro definici hodnoty proměnné či konstanty při datovém typu „String“. [6]

```
let textovyRetezec = "Obsah textového řetězce"
```

2.3.2 Inicializace prázdného textového řetězce

Pro vytvoření prázdného textového řetězce můžeme použít dva níže uvedené způsoby.

```
var prazdnyRetezec = ""  
// definice pomocí prázdného řetězcového literálu  
var dalsiPrazdnyRetezec = String()  
// definice pomocí inicializace instance třídy
```

Pro ověření, zda je textový řetězec prázdný se používá boolean vlastnosti „isEmpty“. [6]

```
if prazdnyRetezec.isEmpty  
{  
    println("Textový řetězec je prázdný.")  
}
```

2.3.3 Modifikovatelnost textových řetězců

Pokud definujeme textový řetězec jako proměnnou „var“, tak jej lze upravovat. V opačném případě, když je řetězec definován jako konstanta pomocí „let“, tak je nemodifikovatelný. Zde je rozdíl oproti Objective-C, kdy se modifikovatelnost určovala pomocí tříd „NSString“ a „NSMutableString“. [6]

```
var promenlivyRetezec = "Pes"  
promenlivyRetezec += " a kočka"  
// „promenlivyRetezec“ má nyní hodnotu „Pes a kočka“
```

2.3.4 Práce se znaky

Datový typ „String“ představuje kolekci „Character“ znaků ve specifikovaném pořadí. K jednotlivým znakům textového řetězce lze přistoupit pomocí „for-in“ cyklu. [6]

```
for znak in "Pes"
{
    println(znak)
}
// P
// e
// s
```

Pro vytvoření proměnné či konstanty datového typu „Character“ je třeba použít tohoto způsobu zápisu:

```
let znakPi : Character = "π"
```

2.3.5 Spojování textových řetězců a znaků

Textové řetězce se dají spojovat pomocí operátoru „+“.

```
let textovyRetezec1 = "ahoj"
let textovyRetezec2 = " jak se máš?"
var pozdrav = textovyRetezec1 + textovyRetezec2
// „pozdrav“ má hodnotu „ahoj jak se máš?“
```

K existujícímu textovému řetězci lze připojit další pomocí operátoru „+=“. [6]

```
var textovyRetezec3 = "čau"
textovyRetezec3 += textovyRetezec2
// „textovyRetezec3“ má hodnotu „čau jak se máš?“
```

2.3.6 Interpolace textových řetězců

Je to způsob, jak poskládat nový textový řetězec z různých konstant, proměnných, literálů a jiných výrazů pomocí zápisu formou řetězcového literálu. Každá položka, která se vkládá do literálu musí být ohraničena závorkami „()“, před které se napíše zpětné lomítko „\“. [6]

```
let cislo = 3
let zprava = "\ (cislo) krát 2.5 je \ (Double(cislo) * 2.5) "
// „zprava“ má hodnotu „3 krát 2.5 je 7.5“
```

2.3.7 Porovnávání textových řetězců

Pro porovnání textových řetězců se využívá binárních operátorů „je rovno“ (==) a „není rovno“ (!=).

```
let text1 = "Odpovídající si texty"
let text2 = "Odpovídající si texty"
if text1 == text2
{
    println("Texty si odpovídají!")
}
```

Pro zjištění, zda textový řetězec začíná nějakým textem se používá metody „hasPrefix()“ a pro zjištění, zda nějakým končí metody „hasSuffix()“. [6]

```
let jmeno = "Josef Novák"
if jmeno.hasPrefix("Josef")
{
    println("Daná osoba má jméno Josef")
}
// zobrazí se „Daná osoba má jméno Josef“
if jmeno.hasSuffix("Novotný")
{
    println("Daná osoba má příjmení Novotný")
}
// podmínka není splněna -> nevypíše se nic do konzole
```

2.3.8 Unicode

Unicode je mezinárodním standardem pro kódování, prezentaci a zpracování textu v různých systémech. Umožňuje vyjádření téměř jakéhokoliv znaku z jakéhokoliv jazyka ve standardizované podobě. Datový typ „String“ i „Character“ jsou s Unicode standardem plně kompatibilní. [6]

2.4 Kolekce – pole, sady a slovníky

Swift poskytuje tři základní typy kolekce: pole, sady a slovníky. Pole jsou uspořádanou kolekcí hodnot, sady neuspořádanou kolekcí unikátních hodnot a slovníky jsou neuspořádanou kolekcí typu klíč-hodnota.

U všech těchto typů kolekcí je vždy jasně daný datový typ hodnot či klíčů, které mohou obsahovat. To znamená, že uživatel nemůže omylem vložit do kolekce hodnotu nesprávného datového typu. A zároveň z toho vyplývá, že si může být uživatel jistý datovým typem při získávání dat z kolekce. [6]

2.4.1 Modifikovatelnost kolekcí

Pokud je vytvořená kolekce přiřazena proměnné („var“), stává se modifikovatelnou kolekcí. To znamená, že do ní lze přidávat položky, odebírat je z ní či je nějak upravovat. Naopak pokud je kolekce přiřazena konstantě („let“), stane se nemodifikovatelnou a nelze měnit její velikost ani obsah. [6]

2.4.2 Pole

Pole obsahují hodnoty stejného typu v uspořádaném seznamu. Stejná hodnota se může v poli vyskytnout několikrát na různých místech. Datový typ pole se zapisuje buď formou „Array<datovyTyp>“ nebo zkrácenou formou „[datovyTyp]“, která se obecně doporučuje používat. Položkou „datovyTyp“ je určen typ hodnot, které lze do pole ukládat.

Vytvořit prázdné pole lze pomocí následujícího příkazu:

```
var nejakeCisla = [Int]()
```

Pokud již bylo pole vytvořeno, tak jej lze vyprázdnit pomocí prázdných hranatých závorek „[]“.

```
nejakeCisla.append(45)  
nejakeCisla = []
```

Pole lze také vytvořit již s určitým počtem prvků, které mají nastavenou danou výchozí hodnotu. Provádí se to pomocí parametrů inicializátoru „count“ a „repeatedValue“.

```
var triDesetinnaCisla = [Double](count : 3, repeatedValue :  
0.5)  
// vznikne pole [0.5, 0.5, 0.5]
```

Pole stejného datového typu se dají spojovat pomocí operátoru „+“.

Pole se dá také vytvořit pomocí literálu ve formě „[hodnota1, hodnota2, hodnota3]“, kde zadáváme přímo počáteční hodnoty, které bude pole obsahovat.

```
var seznamNakup : [String] = [“vajíčka“, “mléko“]
```

Ačkoliv je zřetelné, že výše uvedené pole obsahuje pouze textové řetězce, tak Swift automaticky odvodí, že má obsahovat hodnoty datového typu „String“. Takže předchozí definici pole lze napsat takto:

```
var seznamNakupu = ["vajička", "mléko"]
```

Pro zjištění počtu položek v poli se používá vlastnost pole „count“.

```
println("Nákupní seznam obsahuje \ (seznamNakupu.count)  
položky.")
```

```
// zobrazí "Nákupní seznam obsahuje 2 položky."
```

Pro zjištění, zda je pole prázdné („count“ = 0) se dá využít „isEmpty“ vlastnosti.

```
if seznamNakupu.isEmpty  
{  
    println("Nákupní seznam je prázdný.")  
}
```

Přidání nové položky na konec pole se provádí pomocí metody „append(…)“.

```
seznamNakupu.append("mouka")
```

Jako alternativa pro přidání položky do pole lze použít operátor „+=“. Pomocí něj lze přidávat do pole i více položek najednou.

```
seznamNakupu += ["pečivo"]  
seznamNakupu += ["čokoláda", "sýr", "máslo"]
```

Pro získání hodnoty z pole je potřeba za název pole napsat hranaté závorky a do nich číslo požadovaného indexu.

```
var prvniHodnota = seznamNakupu[0]  
// „prvniHodnota“ se rovná „vajička“
```

Pro změnu hodnoty na daném indexu se postupuje následovně:

```
seznamNakupu[0] = "tři vajička"
```

Této syntaxe se dá využít také pro změnu hodnot v určitém rozsahu indexů. A to i s tím, že počet nových hodnot je jiný než je velikost rozsahu.

```
seznamNakupu[4...6] = ["banány", "jablka"]  
// nahradí tři položky („čokoláda“ a „sýr“ a „máslo“) novými  
dvěma položkami („banány“ a „jablka“)
```

```
// pole bude obsahovat po modifikaci o položku méně
```

Pro vložení hodnoty do pole na určitý index existuje metoda „insert(_:atIndex:)“.

```
seznamNakupu.insert("sirup", atIndex: 0)
```

Tento příkaz vloží na začátek pole položku „sirup“.

Pro odstranění položky z určitého indexu pole je definována metoda „removeAtIndex(_:)“, která zároveň i vrací smazanou položku pro případ, že by ji uživatel chtěl využít.

```
let sirup = seznamNakupu.removeAtIndex(0)
```

Po smazání položky „sirup“ z pole se první položkou pole stává opět položka „tři vejčička“.

Pokud vyžadujeme smazání poslední položky z pole, lze použít metodu „removeLast()“.

Procházet všechny hodnoty pole lze pomocí „for-in“ cyklu.

```
let seznam = ["pračka", "žehlička", "televize"]
for polozka in seznam
{
    println(polozka)
}
// pračka
// žehlička
// televize
```

Pokud potřebujeme při procházení pole zjistit kromě hodnot i indexy, tak se používá funkce „enumerate“, která vrací datový typ „tuple“ složený z indexu a hodnoty. [6]

```
for (index, hodnota) in enumerate(seznam)
{
    println("Položka \((index+1): \((hodnota)")
}
// Položka 1: pračka
// Položka 2: žehlička
// Položka 3: televize
```

2.4.3 Sady

Sady obsahují rozdílné hodnoty stejného typu bez žádného definovaného pořadí. Sady se mohou používat jako alternativa k polím v případě, že nezáleží na pořadí položek a nebo, když je potřeba, aby se položka vyskytovala pouze jednou.

Sada se definuje pomocí příkazu „Set<DatovyTyp>“, kde „DatovyTyp“ je datovým typem, který může sada obsahovat. Vytvoření prázdné sady, vložení položky do ní a její vyprázdnění se provádí následovně:

```
var pismenka = Set<Character>() // vytvoření sady
pismenka.insert("a")           // vložení znaku "a" do sady
pismenka = []                  // vyprázdnění sady
```

Sadu lze také naplnit hodnotami již při jejím vytváření. Definici datového typu „<String>“ lze v tomto případě vynechat, protože dle položek lze datový typ odvodit.

```
var zanry : Set<String> = ["Rock", "Pop", "Hip Hop"]
```

Počet položek sady se určuje pomocí vlastnosti „count“.

```
println("Sada obsahuje \(${zanry.count}) žánry.")
// zobrazí se „Sada obsahuje 3 žánry“
```

Pro ověření, zda sada obsahuje nějaké položky, slouží vlastnost „isEmpty“, která vrací „true“ v případě, že sada neobsahuje žádné položky.

```
if zanry.isEmpty
{
    println("Sada neobsahuje žádné položky")
}
```

Přidání položky do sady se provádí pomocí metody „insert(…)“.

```
zanry.insert("Jazz")
```

Odstranit položku ze sady lze pomocí metody „remove(…)“, která odstraní položku v případě, že byla její součástí a návratovou hodnotou funkce se stane smazaná položka. V opačném případě tato metoda vrací „nil“. Pro smazání všech položek ze sady je určena metoda „removeAll()“.

```
if let smazanaPolozka = zanry.remove("Rock")
{
    println("Ze sady byl smazán žánr \($smazanaPolozka)")
}
```

Pro zjištění, zda sada obsahuje nějakou konkrétní položku, se používá metoda „contains(_):“.

```
if zanry.contains("Hip Hop")
{
    println("Sada obsahuje zvolený žánr")
}
```

Procházení jednotlivých položek sady se provádí pomocí „for-in“ cyklu.

```
for zanr in zanry
{
    println("\(zanr)")
}
// Pop
// Hip Hop
// Jazz
```

Ačkoliv datový typ sady nemá určené pořadí položek, tak pro jeho procházení v určitém pořadí se používá funkce „sorted“. [6]

```
for zanr in sorted(zanry)
{
    println("\(zanr)")
}
// Hip Hop
// Jazz
// Pop
```

2.4.4 Slovníky

Slovník je tvořen spojeními mezi klíči stejného datového typu a hodnotami stejného datového typu. Každá hodnota je propojená s jedinečným klíčem, který slouží jako identifikátor pro tuto hodnotu v slovníku. Narozdíl od položek v polích, nemají položky ve

slovníku určeno specifické pořadí. Slovníky se používají pro vyhledávání hodnot na základě jejich identifikátoru.

Datový typ slovníku se zapisuje pomocí „Dictionary<Klic, Hodnota>“ či zkráceně „[Klic: Hodnota]“, kde „Klic“ je datovým typem klíčů slovníku a „Hodnota“ je datovým typem hodnot, které může slovník pro dané klíče obsahovat.

Vytvoření slovníku, přidání jedné položky (klíč-hodnota páru) a jeho vyprázdnění se provádí následovně:

```
//vytvoření slovníku
var nazvyCisel = [Int: String]()
// přiřazení hodnoty "šestnáct" ke klíči "16"
nazvyCisel[16] = "šestnáct"
// vyprázdnění slovníku
nazvyCisel = [:]
```

Pro vytvoření slovníku s počátečními položkami se používá slovníkový literál ve tvaru „[klic1: hodnota1, klic2: hodnota2, klic3: hodnota3]“.

```
var letiste : [String: String] = ["YYZ" : "Toronto Pearson",
"DUB" : "Dublin"]
```

Ačkoliv se ve výše uvedeném příkladu dá z položek slovníku odvodit datový typ klíče a hodnoty, nemusí se při definici proměnné uvádět její datový typ.

```
var letiste = ["YYZ" : "Toronto Pearson", "DUB" : "Dublin"]
```

Pro zjištění počtu položek ve slovníku se používá vlastnosti „count“.

```
println("Slovník obsahuje \(${letiste.count}) položky.")
// zobrazí se „Slovník obsahuje 2 položky.“
```

Vlastnost „isEmpty“ rozhoduje, zda slovník obsahuje nějaké položky.

```
if letiste.isEmpty
{
    println("Slovník je prázdný")
}
```

Pro přidání nové položky do slovníku lze použít metody nastavení pomocí indexu.

```
letiste["LHR"] = "London"
```

Pomocí této metody lze také měnit hodnotu přiřazenou ke klíči.

```
letiste["LHR"] = "London Heathrow"
```

Alternativou k indexování je použití metody „updateValue(_:forKey:)“, která nastavuje či aktualizuje hodnotu pro daný klíč. Tato metoda vrací předchozí hodnotu pro daný klíč. Tudíž lze zjistit, zda jde o aktualizaci či nastavení nové hodnoty.

```
if let staraHodnota = letiste.updateValue("Dublin Airport",
forKey: "DUB")
{
    println("Předchozí hodnota byla \(staraHodnota).")
}
// zobrazí se „Předchozí hodnota byla Dublin.“
// „staraHodnota“ nebyla nil, tudíž se jednalo o aktualizaci
hodnoty
```

Pomocí indexového přístupu lze také získat hodnotu pro požadovaný klíč slovníku. Návrátová hodnota je optional, ačkoliv v případě, že pro daný klíč neexistuje žádná hodnota, vrací se „nil“.

```
if let jmenoLetiste = letiste["DUB"]
{
    println("Jméno letiště je \(jmenoLetiste).")
}
else
{
    println("Letiště není součástí tohoto slovníku.")
}
// zobrazí se „Jméno letiště je Dublin Airport.“
```

Pro smazání položky ze slovníku lze použít také metody indexování s tím, že nastavíme jako hodnotu klíče „nil“.

```
letiste["YYZ"] = nil
// ze slovníku bylo odstraněno letiště Toronto Pearson
```

Alternativou k mazání klíč-hodnota páru ze slovníku je metoda „removeValueForKey(_:)“. Tato metoda v případě existence dané položky ji odstraní a vrátí její hodnotu a nebo vrátí „nil“, pokud daný klíč neměl přiřazenou žádnou hodnotu.

```
if let smazano = letiste.removeValueForKey("DUB")
{
    println("Ze slovníku bylo odstraněno letiště
\ (smazano).")
}
// zobrazí se „Ze slovníku bylo odstraněno letiště Dublin
Airport.“
```

Pro procházení klíč-hodnota položek slovníku se využívá „for-in“ cyklus. Každá položka slovníku se vrací ve formě „tuple“ jako „(klic, hodnota)“.

```
for (kod, nazev) in letiste
{
    println("\(kod): \(nazev)")
}
// YYZ: Toronto Pearson
// LHR: London Heathrow
```

Ze slovníku lze pomocí „for-in“ cyklu také získat zvlášť jeho klíče a hodnoty.

```
for kod in letiste.keys
{
    println("Kód letiště: \(kod)")
}
// Kód letiště: YYZ
// Kód letiště: LHR

for nazev in letiste.values
{
    println("Název letiště: \(nazev)")
}
// Název letiště: Toronto Pearson
// Název letiště: London Heathrow
```

Z klíčů či hodnot slovníku lze také jednoduše poskládat pole.

```
let kody = [String](letiste.keys)
// kody jsou ["YYZ", "LHR"]
let nazvy = [String](letiste.values)
// nazvy jsou ["Toronto Pearson", "London Heathrow"]
```

Datový typ slovník nemá své položky uspořádané, tudíž pokud je potřeba procházet jeho položky ve specifikovaném pořadí, je potřeba aplikovat metodu „sorted“ na jeho klíče či hodnoty. [6]

2.5 Řídící struktury

Swift poskytuje všechny běžné řídicí struktury. To zahrnuje „for“ a „while“ cykly pro opakované provádění stejné úlohy, „if“ a „switch“ příkazy pro vykonávání různých větví kódu dle určitých podmínek a příkazy jako „break“ či „continue“ pro přesun vykonávání na jiné místo v kódu. [6]

2.5.1 „For“ cykly

Swift nabízí dva druhy cyklů, které vykonávají sadu příkazů daný počet krát. Cyklus „for-in“ vykonává sadu příkazů pro každou položku v posloupnosti. Cyklus „for“ vykonává sadu příkazů dokud je splněna určitá podmínka, nejčastěji se jedná o inkrementaci proměnné na konci každého cyklu.

„For-in“ prochází celou posloupnost, kterou mohou být např. rozsahy čísel, položky v poli či znaky v textovém řetězci.

```
for index in 1...5
{
    println("\(index) krát 5 je \(index * 5)")
}
// 1 krát 5 je 5
// 2 krát 5 je 10
// 3 krát 5 je 15
// 4 krát 5 je 20
// 5 krát 5 je 25
```

V případě, že není při vykonávání cyklu potřeba znát hodnoty z posloupnosti, může se místo názvu proměnné uvést podtržítka.

```
var odpoved = 1
for _ in 1...5
{
    odpoved += 2
}
// „odpoved“ je 11
```

Klasický cyklus „for“ se definuje pomocí podmínky a inkrementace.

```
for var index = 0; index < 3; ++index
{
    println("index je \(index)")
}
// index je 0
// index je 1
// index je 2
```

Jednotlivé části definice „for“ cyklu musí být odděleny středníkem jako v jazyku C, ale ve Swiftu není třeba psát závorky kolem definice cyklu. [6]

2.5.2 „While“ cykly

Tyto cykly provádějí sadu příkazů dokola, dokud se zadaná podmínka nestane „false“. Tento typ cyklů se nejčastěji využívá v případech, kde se nezná celkový počet iterací před začátkem vykonávání cyklu. Swift obsahuje dva druhy „while“ cyklů. „While“ kontroluje podmínku při každém startu cyklu a „do-while“ až na jeho konci.

Cyklus „while“ začíná kontrolou své podmínky. A ta pokud nabývá hodnoty „true“, sada příkazů se provádí tak dlouho, dokud se podmínka nestane „false“. Obecně se dá tento cyklus zapsat takto: „while podmínka { sadaPrikazu }“.

```
var i = 0
while i < 5
{
    i++
    println(i)
}
```

```
// 1
// 2
// 3
// 4
```

Cyklus „do-while“ vykoná vždy minimálně jednou sadu příkazů a až poté se ověřuje jeho podmínka s tím, že dokud podmínka nabývá hodnoty „true“, sada příkazů se opakuje. Obecný zápis tohoto cyklu je následovný: „do { sadaPrikazu } while podmínka“. [6]

```
var i = 0
do
{
    i++
    println(i)
}
while i < 5
// 1
// 2
// 3
// 4
// 5
```

2.5.3 Podmínkové příkazy

Ve Swiftu existují dvě možnosti, jak větvit kód – podmínkový příkaz „if“ a „switch“. Příkaz „if“ se nejčastěji využívá pro vyhodnocení jednoduchých podmínek s pár možnými výstupy. Oproti tomu „switch“ se používá pro řešení složitějšího větvení kódu.

Příkaz „if“ ve své nejjednodušší formě má pouze jednu podmínku, na jejímž základě se vykoná či nevykoná sada příkazů.

```
var teplota = 4
if teplota <= 5
{
    println("Teplota je velice nízká.")
}
// zobrazí se "Teplota je velice nízká"
```

„If“ příkaz lze dále větvit pomocí „else“ či „else if“ klauzulí na další sady příkazů. Větev „else“ není povinnou uvádět a vykoná se pokaždé, když není splněna ani jedna předchozí větvičí podmínka.

```
var teplota = 35
if teplota <= 5
{
    println("Teplota je velice nízká.")
}
else if teplota >= 30
{
    println("Teplota je velmi vysoká.")
}
else
{
    println("Teplota je průměrná.")
}
// zobrazí se „Teplota je velmi vysoká.“
```

Příkaz „switch“ pracuje se vstupní hodnotou, kterou porovnává vůči jednotlivým možným vzorům hodnot. Jakmile narazí u nějakého vzoru na shodu se vstupní hodnotou, vykoná se odpovídající část kódu. „Switch“ je alternativou k „if“ s výhodou v tom, že se lépe používá pro rozpoznání více možných stavů hodnoty.

Každý „switch“ se skládá z několika větví, přičemž každá začíná klíčovým slovem „case“ následovaným vzorem hodnoty či hodnot a dvojtečkou. Na konci musí tento podmínkový příkaz obsahovat větev „default:“, která se vykoná pokud nedojde ke shodě vstupní hodnoty s žádným vzorem.

```
let pismenko : Character = "a"
switch pismenko {
case "a", "e", "i", "o", "u":
    println("Jedná se o samohlásku.")
case "h", "k", "r", "d", "t", "n":
    println("Jedná se o souhlásku")
default:
    println("Nejedná se o samohlásku ani souhlásku")
```

```
}  
// zobrazí se „Jedná se o samohlásku.“
```

Narozdíl od Objective-C se jednotlivé větve „switch“ příkazu nemusí ukončovat příkazem „break“, ačkoliv Swift automaticky ukončí celý „switch“ příkaz po vykonání odpovídající větve. Povinností je, aby každá větev obsahovala aspoň jeden příkaz. Vykoná se vždy sada příkazů té větve, která jako první splnila podmínku.

Vzorové hodnoty ve „switch“ příkazu mohou být také rozsahy.

```
let cislo = 5  
switch cislo {  
case 0...10:  
    println("Číslo je z rozsahu 0 až 10.")  
default:  
    println("Číslo není z rozsahu 0 až 10.")  
}
```

Kromě toho „switch“ příkaz zvládá také porovnávat vůči datovému typu „tuple“. Každá položka tohoto datového typu může být porovnáвана vůči konkrétní hodnotě či rozsahu. Při napsání podtržítka „_“ odpovídá vzor položky každé hodnotě. [6]

```
let bod = (3, 0)  
switch bod {  
case (0, 0):  
    println("Bod má souřadnice (0, 0).")  
case (_, 0):  
    println("Bod má y souřadnici 0.")  
case (0, _):  
    println("Bod má x souřadnici 0.")  
default:  
    println("Bod nemá ani jednu souřadnici nulovou.")  
}  
// zobrazí se "Bod má y souřadnici 0."
```

2.5.4 Příkazy pro přesun vykonávání kódu

Tyto příkazy pomocí přesměrování kódu z jedné části do jiné mění posloupnost, v jaké je vykonáván kód aplikace. Swift má za tímto účelem celkem čtyři příkazy: „continue“, „break“, „fallthrough“ a „return“.

Příkaz „continue“ říká cyklu, aby přerušil, co právě provádí a aby začal vykonávat další iteraci cyklu.

Příkaz „break“ ukončuje okamžitě vykonávání cyklu či „switch“ příkazu. Používá se v případech, kdy je potřeba cyklus předčasně ukončit. Po jeho provedení se přesune vykonávání kódu za konec cyklu či „switch“ příkazu. Nejčastěji se „break“ využívá ve „switch“ příkazu u „default:“ větve.

„Fallthrough“ se používá u „switch“ příkazu a zapisuje se na konec „case“ větve, pokud uživatel chce, aby „switch“ testoval vstupní hodnotu i na dalších větvích. Docílí se tím chování, které je obvyklé pro „switch“ v jazycích založených na C.

Příkaz „return“ slouží pro okamžité ukončení právě prováděné funkce. [6]

2.6 Funkce

Funkce je soběstačnou částí kódu, která vykonává konkrétní úlohu. Každé funkci je třeba přidělit název pro její identifikaci a ten se poté používá i pro její „zavolání“, aby se vykonala.

Každá funkce ve Swiftu je dána určitým typem, který se skládá z typu parametrů a návratového typu. Tento typ se dá využít jako kterýkoliv jiný datový typ ve Swiftu. Např. pokud má funkce vstupní parametry textový řetězec s celým číslem a jako návratovou hodnotu textový řetězec, typ této funkce bude následovný: „(String, Int) -> String“. [6]

2.6.1 Definice a volání funkcí

Každá definice funkce začíná klíčovým slovem „func“, které je následované názvem funkce, který popisuje, co by měla vykonávat za úlohu. Za názvem se píše kulaté závorky „()“, ve kterých se může definovat jeden či více vstupních parametrů různého datového typu. Pokud má funkce i vracet nějakou hodnotu, zapíše se za vstupní parametry šipka „->“ a datový typ návratové hodnoty.

Když je potřeba funkci použít, zavolá se svým názvem a do kulatých závorek se uvedou její vstupní parametry (argumenty). Ty musí být zadávány ve stejném pořadí jak jsou definovány ve funkci. [6]

```
func pozdravOsobu(jmeno : String) -> String
{
    return "Ahoj" + jmeno + "!"
}
println(pozdravOsobu("Dane"))
// zobrazí se „Ahoj Dane!“
```

2.6.2 Parametry funkcí a návratové hodnoty

Funkce mohou mít více vstupních parametrů, které se zapisují do kulatých závorek a oddělují se čárkami.

```
func vypocitejVzdalenost(start : Int, konec: Int) -> Int
{
    return konec-start
}
println(vypocitejVzdalenost(1,5))
// zobrazí se „4“
```

Zároveň také funkce nemusí obsahovat žádný vstupní parametr, takže kulaté závorky zůstanou prázdné.

```
func pozdravSvet() -> String
{
    return "Ahoj světe!"
}
println(pozdravSvet())
// zobrazí se „Ahoj světe!“
```

Při definici funkce se nemusí uvádět ani návratový typ. V tomto případě se za kulaté závorky nepíše žádné další znaky a funkce se označuje typem „Void“ a její návratovou hodnotou je prázdný „tuple“ zapsaný jako „()“.

```
func poprejDobrouChut(jmeno : String)
{
    println("Dobrou chuť \ (jmeno)!")
}
```

```
}  
poprejDobrouChut („Dane“)  
// zobrazí se „Dobrou chuť Dane!“
```

Pokud je třeba, aby funkce vracela více návratových hodnot, může se využít datového typu „tuple“. Na příkladu uvedeném níže je předvedeno, jak se dá efektivně tento nový datový typ využít např. při funkci hledající nejmenší a největší celé číslo v poli.

```
func najdiMinAMax(pole: [Int]) -> (min: Int, max: Int)  
{  
    var min = pole[0]  
    var max = pole[0]  
    for cislo in pole[1..<pole.count]  
    {  
        if cislo < min  
        {  
            min = cislo  
        }  
        else if cislo > max  
        {  
            max = cislo  
        }  
    }  
    return (min, max)  
}  
let vysledek = najdiMinAMax([8, -6, 3, 112, 4, 80])  
println(„Nejmenší číslo je \(${vysledek.min}) a největší  
\(${vysledek.max}).“)  
// zobrazí se „Nejmenší číslo je -6 a největší 112.“
```

Návratový typ „tuple“ u funkcí může být definován také jako optional s tím, že může jako celek nabývat hodnoty „nil“. To se hodí v případech, když není jisté, jestli funkce bude vracet nějakou hodnotu. Takový „tuple“ se definuje např. takto: „(Int, Int)?“. [6]

2.6.3 Názvy parametrů funkcí

Všechny předchozí ukázky funkcí měly jen tzv. lokální názvy parametrů, které se používají pouze uvnitř funkce. Někdy je užitečné si pojmenovat parametry pro větší přehlednost i

externím názvem, který se zobrazuje při volání funkce. Tento název se zapisuje před lokální název funkce a odděluje se mezerou.

```
func spoj(retezec r1: String, sRetezcem r2: String, pomoci
spojka: String) -> String
{
    return r1 + spojka + r2
}
spoj(retezec: "Ahoj", sRetezcem: "světe", pomoci: ", ")
// funkce vrátí „Ahoj, světe“
```

Pokud uživateli vyhovuje použít stejný externí název parametru jako lokální, tak nemusí uvádět dva stejné názvy. Místo toho napíše pouze jeden a před něj napíše symbol mřížky „#“.

Při definici funkce lze definovat výchozí hodnotu pro jakýkoliv vstupní parametr. Pokud je výchozí hodnota definována, nemusí se při volání funkce parametr uvádět. Parametry s výchozí hodnotou se doporučuje uvádět až na konec seznamu parametrů a tyto parametry mají automaticky nastavený externí název stejný jako je lokální v případě, že uživatel jim nepřidělí vlastní externí název. [6]

```
func spoj(retezec r1: String, sRetezcem r2: String,
pomoci spojka: String = " ") -> String
{
    return r1 + spojka + r2
}
spoj(retezec: "ahoj", sRetezcem: "světe", pomoci: "-")
// funkce vrátí „ahoj-světe“
spoj(retezec: "ahoj", sRetezcem: "světe")
// funkce vrátí „ahoj světe“
```

2.7 Enumerations (výčty)

Výčet je běžným datovým typem pro skupinu souvisejících hodnot. V jazyku C musí mít každý jeho člen přiřazenou nějakou číselnou hodnotu. Výčty ve Swiftu jsou pokročilejší a předchozí uvedené pravidlo zde neplatí. Pokud je však hodnota (označuje se jako „raw“ hodnota) definována, může být textovým řetězcem, znakem, celým či desetinným číslem. [6]

2.7.1 Tvorba výčtů

Výčet se definuje pomocí klíčového slova „enum“ a složených závorek. Před název každého z členů se uvádí klíčové slovo „case“.

```
enum SvetovaStrana
{
    case Sever
    case Jih
    case Zapad
    case Vychod
}
```

Členové výčty se dají zjednodušeně definovat i na jednom řádku.

```
enum SvetovaStrana
{
    case Sever, Jih, Zapad, Vychod
}
```

Každá definice výčtu vytváří nový datový typ. Tudiž podobně jako ostatní názvy datových typů, by měl i název výčtu být v jednotném čísle a začínat velkým písmenem.

```
var smer = SvetovaStrana.Zapad
```

Proměnná „smer“ je nyní výčtového datového typu „SvetovaStrana“ a pokud jí chceme přidělit jinou hodnotu, lze použít zkrácené tečkové syntaxe. [6]

```
smer = .Vychod
```

2.7.2 Práce s výčtem ve „switch“ příkazu

Pomocí „switch“ příkazu lze jednoduše rozpoznávat výčtové hodnoty.

```
var smer = SvetovaStrana.Jih
switch smer {
case .Sever:
    println(„Severní směr“)
case .Jih:
    println(„Jižní směr“)
case .Zapad:
    println(„Západní směr“)
```

```
case .Vychod:
    println(„Východní směr“)
}
// zobrazí se „Jižní směr“
```

Ve výše uvedeném příkladě není třeba uvádět „default“ větev, ačkoliv jsou ve „switch“ příkazu zastoupeni všichni možní členi z výčtu. Pokud by tak nebylo, je třeba uvést „default“ větev. [6]

2.7.3 Přiřazené hodnoty

Výčty se mohou ve Swiftu definovat s přiřazenými hodnotami různého datového typu, přičemž pro každý člen může být definován jiný datový typ, když je to potřeba.

```
enum CarovyKod
{
    case UPCA(Int, Int, Int, Int)
    case QRCode(String)
}
```

Výše uvedený kód vytvoří výčtový datový typ s názvem „CarovyKod“, který obsahuje dva členy. První člen „UPCA“ má přiřazenou hodnotu datového typu „(Int, Int, Int, Int)“ a druhý člen „QRCode“ má přiřazenou hodnotu datového typu „(String)“.

Proměnná s čárovým kódem se vytvoří následovně:

```
var carovyKodZbozi = CarovyKod.UPCA(8, 85909, 51226, 3)
```

Této proměnné lze jednoduše přidělit i čárový kód jiného typu:

```
carovyKodZbozi = .QRCode(“ABCDEFGHJKLMNOP“)
```

Pro zjištění typu čárového kódu proměnné a vypsání jeho přiřazených hodnot se používá „switch“ příkazu. [6]

```
switch carovyKodZbozi {
case let .UPCA(cislo1, cislo2, cislo3, cislo4):
    println(“UPC-A:  \ \(cislo1),  \ \(cislo2),  \ \(cislo3),
\ \(cislo4)“)
case let .QRCode(kod):
    println(“QRCode: \ \(kod)“)
}
```

2.7.4 „Raw“ hodnoty

Alternativou k přiřazeným hodnotám jsou „raw“ hodnoty, které musí být stejného datového typu a přiřazují se k členům již při definici výčtu.

```
enum ASCIIZnaky : Character
{
    case Tab = "\t"
    case LineFeed = "\n"
    case CarriageReturn = "\r"
}
```

Tento výčet se skládá z formátovacích znaků a má přiděleny „raw“ hodnoty datového typu „Character“.

Přiřazené „raw“ hodnoty se nedají měnit a musí být unikátní. Pokud se používají „raw“ hodnoty datového typu „Int“, automaticky se inkrementují.

```
enum Planeta : Int
{
    case Merkur = 1, Venuse, Zeme, Mars, Jupiter, Saturn,
    Uran, Neptun
}
```

Pro přístup k „raw“ hodnotě členu výčtu se používá vlastnosti „rawValue“.

```
let poradizeme = Planeta.Zeme.rawValue
// zobrazí se 3
```

Když je výčet definován s „raw“ hodnotami, tak automaticky získává inicializátor, který dle „raw“ hodnoty vrátí členu výčtu nebo „nil“ v případě, když pro danou „raw“ hodnotu neexistuje žádný člen. [6]

```
let moznaPlaneta = Planeta(rawValue: 7)
// „moznaPlaneta“ je datového typu „Planeta?“ a rovná se
členu „.Uran“
```

2.8 Třídy a struktury

Třídy a struktury jsou základními stavebními kameny kódu aplikace. Definují se v nich jejich vlastnosti (konstanty, proměnné) a metody (funkce) pro zajištění požadované

funkcionality. Narozdíl od jiných programovacích jazyků, Swift nepotřebuje vytvářet oddělené rozhraní a implementační soubory pro vlastní třídy a struktury. Třída či struktura se definuje v samostatném souboru a je poté automaticky přístupná i z jiných definovaných tříd či struktur bez nutnosti provádění importu. [6]

2.8.1 Porovnání tříd a struktur

Třídy a struktury ve Swiftu mají společné tyto věci. Obě mohou:

- obsahovat vlastnosti pro ukládání hodnot
- obsahovat metody pro zajištění funkcionality
- obsahovat inicializátor pro nastavení výchozího stavu

Třídy mají navíc tyto schopnosti oproti strukturám:

- podporují dědičnost
- podporují deinicializátor
- počítadlo referencí povoluje více než jednu referenci na instanci třídy (struktury jsou vždy kopírovány, když se přemísťují různě v kódu)

Třídy i struktury se vytváří podobně. Třída se definuje klíčovým slovem „class“ a složenými závorkami, struktura se uvádí klíčovým slovem „struct“.

```
struct Rozliseni
{
    var sirka = 0
    var vyska = 0
}
class VideoRezim
{
    var rozliseni = Rozliseni()
    var prokladany = false
    var snimkovaFrekvence = 0.0
    var name : String?
}
```

Výše uvedený příklad definuje strukturu „Rozliseni“, která má dvě vlastnosti, které mají výchozí hodnotu „0“ a jejich datový typ je odvozen jako celé číslo „Int“. Zároveň se zde definuje třída „VideoRezim“ s čtyřmi vlastnostmi. První z nich je „rozliseni“, které je

instancí struktury „Rozliseni“, další je proměnná „prokladany“, „snimkovaFrekvence“ a na závěr vlastnost „name“, která je „optional“ typu a má výchozí hodnotu „nil“.

Vytváření instance třídy či struktury je velice podobné. Obojí používají inicializační syntaxe ve formě název třídy či struktury následovaný kulatými závorkami. Po vytvoření instance se vlastnosti inicializují na výchozí hodnoty.

```
let nejakeRozliseni = Rozliseni()  
let nejakyVideoRezim = VideoRezim()
```

Pro přístup k vlastnostem instance se používá tečková syntaxe. Za název instance třídy se napíše „tečka“ a za ní název požadované vlastnosti.

```
println("Šířka nějakého videa je \ (nejakeRozliseni.sirka) ")  
// zobrazí se „Šířka nějakého videa je 0“
```

Pomocí tečkové syntaxe lze i měnit hodnoty vlastností.

```
nejakyVideoRezim.rozliseni.sirka = 1280  
// šířka videa je nyní 1280
```

Všechny struktury mají navíc oproti třídám další automaticky vygenerovaný inicializátor, který nabízí přiřazení výchozích hodnot vlastnostem struktury dle jejich názvu. [6]

```
let vgaRozliseni = Rozliseni(sirka: 640, vyska: 480)
```

2.8.2 Hodnotové typy

Hodnotový typ je typem, jehož hodnota je kopírována, když je přiřazena proměnné či konstantě nebo když je předávána jako vstupní parametr funkce.

Většina základních datových typů jako jsou celá čísla, desetinná čísla, logické hodnoty, textové řetězce, slovníky či pole jsou hodnotové typy a jsou implementovány jako struktury.

Všechny struktury a výčty jsou hodnotovými typy. To znamená, že vlastnosti jejich instancí jsou vždy kopírovány. [6]

```
let hd = Rozliseni(sirka: 1920, vyska: 1080)  
var kino = hd  
kino.sirka = 2048  
println("kino má nyní šířku \ (kino.sirka) pixelů")  
// zobrazí se „kino má nyní šířku 2048 pixelů“
```

```
println("hd má stále šířku \ (hd.sirka) pixelů")  
// zobrazí se „hd má stále šířku 1920 pixelů“
```

2.8.3 Referenční typy

Oproti hodnotovým, referenční typy nejsou kopírovány, když se přiřazují proměnné či konstantě nebo když se předávají funkci jako vstupní parametr. Místo kopie se tvoří reference na stejnou instanci.

Typickým představitelem referenčního typu jsou třídy. Když se vytvoří proměnná či konstanta s instancí třídy a přiřadí se tato proměnná nějaké další proměnné, tak obě dvě mají referenci na stejnou instanci třídy.

Pro porovnání, zda proměnné či konstanty odkazují na stejnou instanci třídy se používá operátoru identity „==“, popřípadě „!=“. [6]

3 ANALÝZA POŽADAVKŮ NA APLIKACI

Jedním z hlavních požadavků při tvorbě mobilní aplikace pro snímání polohy je mobilní zařízení, které má integrovaný GPS modul. V mém případě bude vytvářena mobilní aplikace pro iPhone, který má už od své druhé generace GPS modul, který je schopen zaznamenávat různé údaje o poloze, implementován. V mobilní aplikaci bude potřeba získávat zeměpisnou šířku a délku a datum pořízení údajů o poloze. Pro veškerou práci s polohovými údaji je v iOS systému určen framework Core Location, pomocí jehož metod se bude aktivovat snímání polohy a provádět její uložení. Vzhledem k tomu, že dalším požadavkem je připojení k internetu a mobilní zařízení nemusí být v každém okamžiku připojeno, tak polohy se vždy budou nejdříve ukládat lokálně do databáze ve formě tzv. Core Data. A následovně, jak bude k dispozici připojení k internetu, tak budou odesílány do databáze na server. Komunikace mezi mobilní aplikací a databází bude probíhat pomocí vytvořeného webového API, které bude obsahovat jednotlivé metody od přidávání a získávání lokací až po správu uživatelů. Databáze bude navržena tak, aby se v ní daly ukládat veškeré nasnímané polohy a správně přiřazovat jednotlivým přihlášeným uživatelům. Navíc každému uživateli bude možno přiřazovat kruhovou zónu, v které se může pohybovat. Každý uživatel bude přidělen k nějaké skupině a tu bude mít na starost jeden či více administrátorů.

II. PRAKTICKÁ ČÁST

4 MOBILNÍ APLIKACE

V rámci praktické části této diplomové práce byla vytvořena mobilní aplikace s názvem LCTR (Locator) pro platformu iOS. Jedná se o univerzální aplikaci, kterou lze nainstalovat na všechny iPhony a iPady, na kterých běží iOS 8. Aplikace využívá zabudovaného GPS modulu v zařízení pro snímání poloh přihlášeného uživatele, které se následovně odesílají pomocí API metody na server do databáze. Zároveň může mít uživatel přiřazenou určitou povolenou zónu, v které se může pohybovat. Při vstoupení do ní či vykročení z ní se odešle email jeho přiřazenému administrátorovi.



Obrázek 1 - Logo mobilní aplikace

4.1 Přihlášení

Po spuštění aplikace se zobrazí obrazovka pro přihlášení, kde je uživatel vyzván k zadání svého uživatelského jména a hesla. Před provedením samotného přihlášení lze povolit volbu trvalého přihlášení, aby se přihlašovací údaje uživatele uložily do klíčenky a při příštím přihlašování se automaticky předvyplnily.

V aplikaci Locator existují celkem dva typy uživatelů. Buď se jedná o administrátora, který má pod sebou skupinu běžných uživatelů a může v aplikaci sledovat jejich poslední pohyb či jim nastavovat povolené zóny pohybu. A nebo se jedná o běžného uživatele, který patří k určité skupině spravované administrátorem a odesílá své polohy na server a zároveň, když překročí hranici své povolené zóny, tak se odešle email s touto informací administrátorovi. V aplikaci se rozpoznává po přihlášení typ uživatele a dle toho se upravuje uživatelské rozhraní aplikace.



Obrázek 2 - Přihlášení uživatele

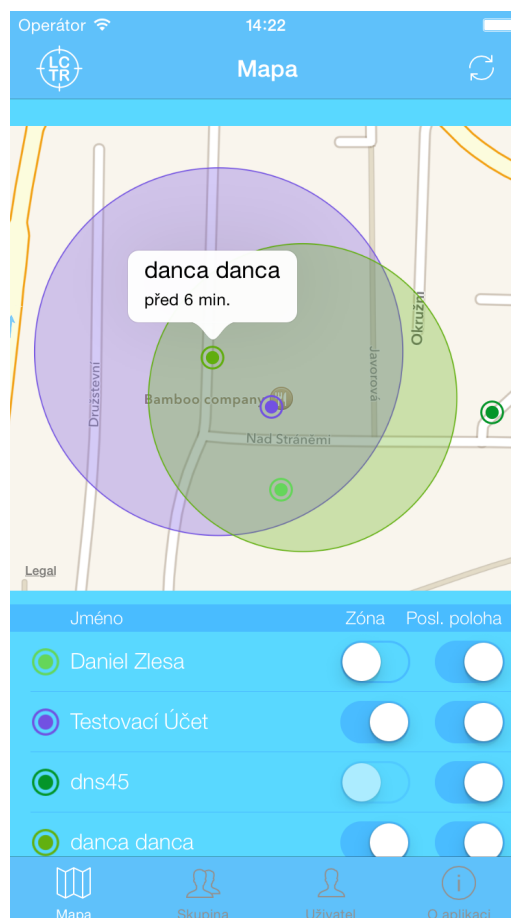
4.2 Rozhraní administrátora

Administrátor má po přihlášení k dispozici celkem čtyři záložky, mezi které patří „Mapa“, „Skupina“, „Uživatel“ a „O aplikaci“. Automaticky se zobrazuje jako první sekce „Mapa“.

4.2.1 Sekce „Mapa“

Tato sekce obsahuje mapu, v které se zobrazují body s poslední získanou polohou uživatelů skupiny a popřípadě kruhy znázorňující jejich nastavené zóny. Po tapnutí na bod s poslední polohou se zobrazí informace, čím je to poloha a kdy byla získána. Pod mapou se nachází tabulka, v níž je zobrazen seznam všech uživatelů spravovaných administrátorem. A dá se v ní pomocí přepínačů zvolit, zda se má pro požadovaného uživatele zobrazovat jeho poslední poloha či zóna. Pokud se přepínač nedá zapnout, znamená to, že pro tohoto uživatele není k dispozici poslední poloha či nemá nastavenou žádnou zónu. V horní části této sekce se nachází dvě tlačítka. Po levé straně je tlačítko s logem aplikace, které slouží

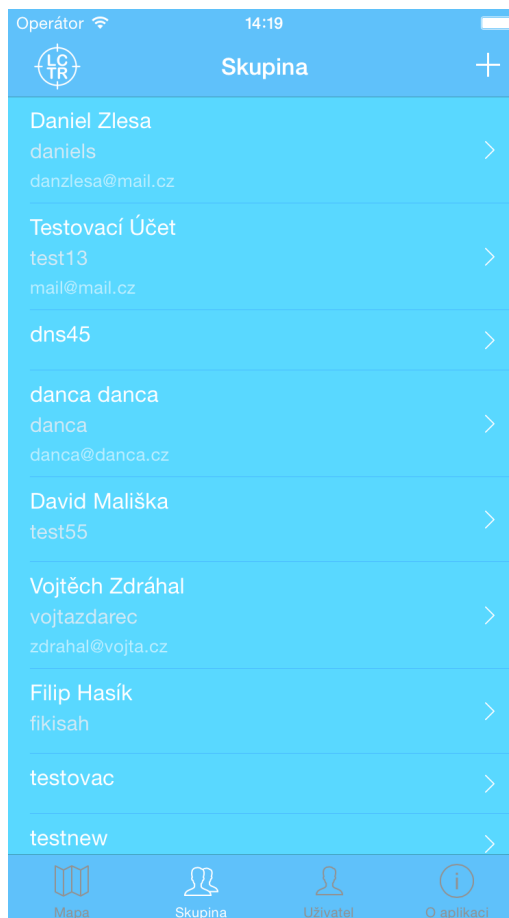
pro animované přiblížení mapy, aby byly viditelné všechny poslední lokace a zóny uživatelů. Na pravé straně je umístěné tlačítko pro znovu načtení dat ze serveru.



Obrázek 3 - Sekce "Mapa" administrátora

4.2.2 Sekce „Skupina“

Druhá sekce obsahuje tabulku, v níž jsou uvedeni všichni uživatelé spadající do skupiny přihlášeného administrátora. V každém řádku je uvedeno uživatelské jméno, jméno a příjmení a email uživatele. Vzhledem k tomu, že položky jméno, příjmení a email jsou volitelné, tak se může stát, že řádek bude obsahovat pouze uživatelské jméno. V horní části na pravé straně se nachází tlačítko „+“ pro přidání nového uživatele. Po tapnutí na řádek s uživatelem se zobrazí jeho detail.



Obrázek 4 - Sekce "Skupina" administrátora

Vytváření uživatele

Pomocí této obrazovky se dá vytvořit nový uživatel. Obsahuje formulář s celkem sedmi položkami, z nichž tři jsou nepovinné (jméno, příjmení a email). Uživatelské jméno musí být minimálně tři znaky dlouhé a nesmí se shodovat s žádným už předchozím existujícím. Heslo se automaticky generuje a nebo může administrátor zadat vlastní s tím, že musí být minimálně šest znaků dlouhé. Na konci se nastavuje oprávnění, které může být dvojího typu (běžný uživatel X administrátor). Pokud se vytváří běžný uživatel, položka s názvem skupiny se nedá změnit, ačkoliv nový uživatel bude náležet automaticky do skupiny, kterou administrátor spravuje. V případě vytváření administrátora lze změnit název skupiny a administrátor bude poté spravovat novou skupinu. Pokud byly všechny položky vyplněny správně, tak se po tapnutí na tlačítko „Přidat“ v pravém horním rohu vytvoří nový uživatel. Pro zrušení tvorby uživatele a návrat do sekce „Skupina“ je určeno tlačítko „Zrušit“ v levém horním rohu.

Operátor 14:19

Zrušit Přidat

Uživatelské jméno*

Zadejte uživatelské jméno

Jméno

Zadejte jméno

Příjmení

Zadejte příjmení

Email

Zadejte email

Heslo*

lwoekm

Oprávnění*

běžný uživatel

Skupina*

Master

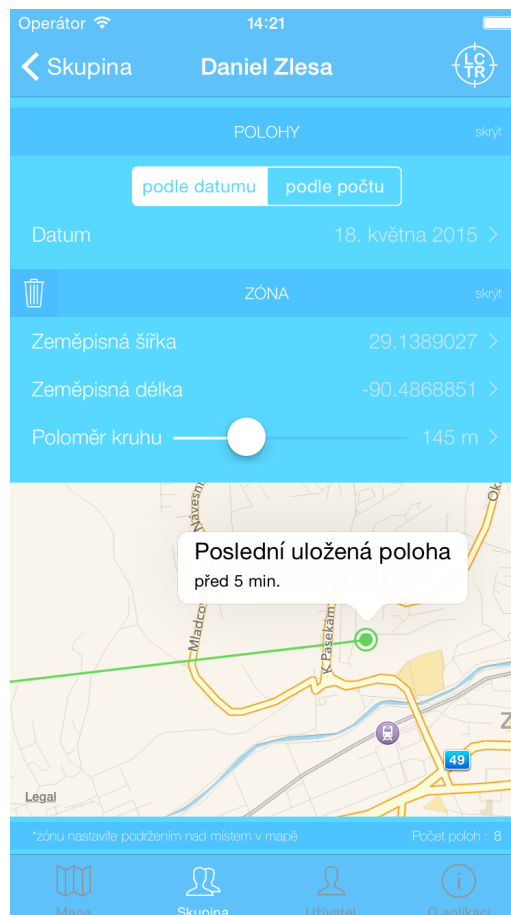
Položky označené * jsou povinné.

Obrázek 5 - Vytváření uživatele

Detail uživatele

Detail uživatele obsahuje ve spodní části mapu a v horní části dvě skrývatelné sekce s názvem „POLOHY“ a „ZÓNA“. Jejich skrývání se provádí tapnutím na jejich hlavičku s názvem. Uprostřed nahoře je uvedeno jméno uživatele. Pomocí sekce „POLOHY“ se dá filtrovat zobrazení poloh uživatele. Nachází se zde přepínač pro zvolení, zda zobrazovat polohy z určitého konkrétního dne či zda zobrazit určitý počet posledních lokací, s tím, že při zadání počtu „0“ se zobrazí všechny polohy uživatele. Po zvolení parametrů filtru se načtou požadované polohy z databáze umístěné na serveru a zobrazí se v mapě pomocí křivky, v níž je označen počáteční a konečný bod. V sekci „ZÓNA“ lze nastavit uživateli povolenou kruhovou zónu, v které se může pohybovat. Provádí se to pomocí nastavení zeměpisné šířky a délky jejího středu a poloměru jejího kruhu v metrech. Střed zóny se dá také nastavit pomocí podržení prstu nad požadovaným místem v mapě. Odstranění přidělené zóny se provádí pomocí tlačítka s ikonkou koše, které je umístěné vlevo v hlavičce sekce „ZÓNA“. Pod mapou se zobrazuje informační řádek, který ukazuje kolik

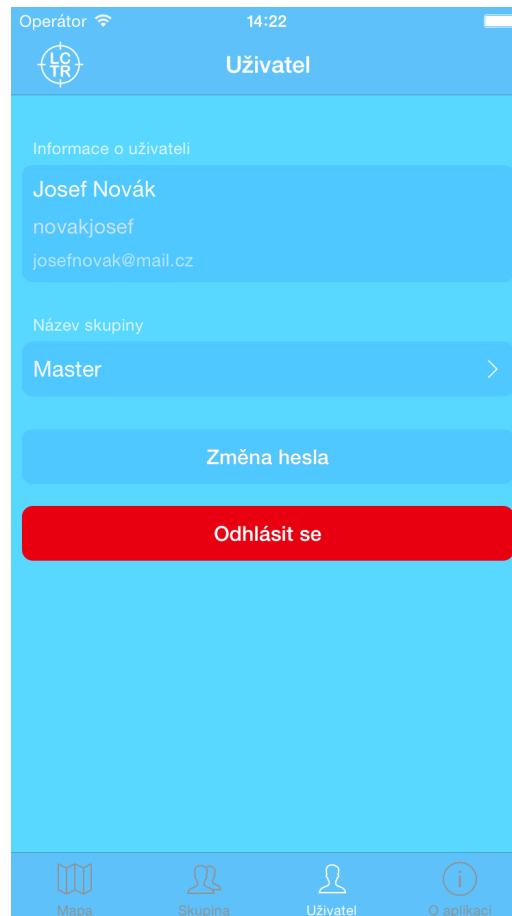
bylo načtených poloh pro zadané parametry filtru. Pomocí pravého horního tlačítka s logem aplikace lze provést animované přiblížení mapy na aktuálně načtené polohy a zónu uživatele.



Obrázek 6 - Detail uživatele

4.2.3 Sekce „Uživatel“

Třetí sekce sestává ze čtyř částí. V první části jsou zobrazeny informace o přihlášeném uživateli (jméno a příjmení, uživatelské jméno, email). V další části se dá změnit název skupiny uživatelů, které administrátor spravuje. Třetí část obsahuje tlačítko pro vyvolání dialogu pro změnu hesla přihlášeného uživatele. A v poslední části se nachází červené tlačítko sloužící pro odhlášení uživatele.



Obrázek 7 - Sekce "Uživatel" administrátora

4.2.4 Sekce „O aplikaci“

V poslední sekci se zobrazuje logo aplikace s jejím názvem, pod kterým je uveden základní popis aplikace a informace o jejím autorovi.



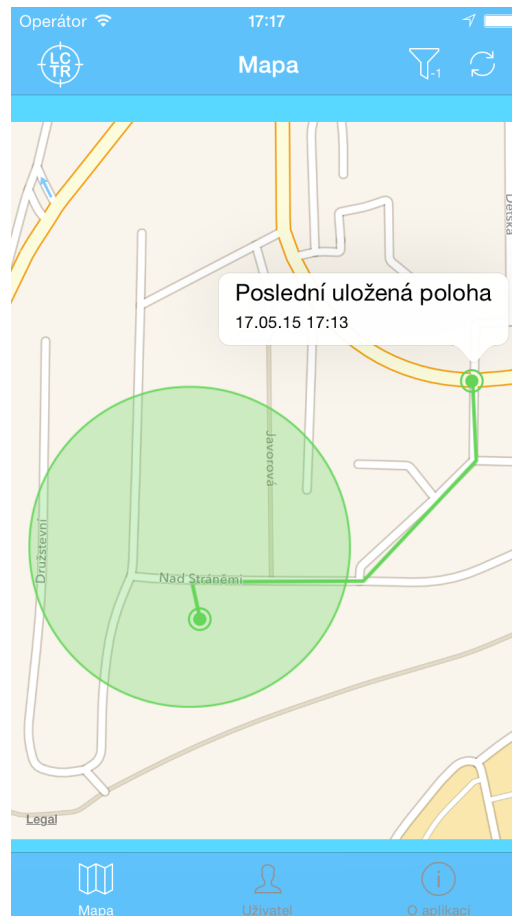
Obrázek 8 - Sekce "O aplikaci"

4.3 Rozhraní běžného uživatele

Běžný uživatel má po přihlášení k dispozici pouze tři záložky, mezi které patří „Mapa“, „Uživatel“ a „O aplikaci“. Automaticky se zobrazuje jako první sekce „Mapa“.

4.3.1 Sekce „Mapa“

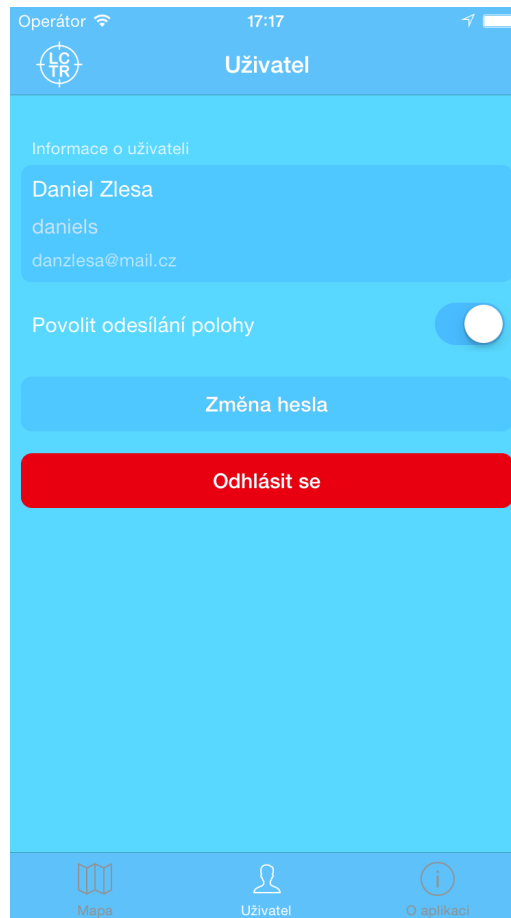
V této sekci se nachází mapa, ve které se po načtení obrazovky zobrazí křivka s nasnímanými polohami uživatele za aktuální den s vyznačenou první a poslední lokací. Zároveň se v mapě ukáže i povolená kruhová zóna, pokud ji přihlášenému uživateli administrátor nastavil. V horní části obrazovky se nachází tři tlačítka. Tlačítko v levé části s logem aplikace slouží pro animované přiblížení mapy na načtené polohy a zónu. Pomocí tlačítka s ikonou ve tvaru nálevky lze provést filtrování načtených poloh. A to tak, že si uživatel může zvolit, zda chce v mapě zobrazit polohy z aktuálního, včerejšího či předvčerejšího dne. Tlačítko umístěné v pravém rohu slouží pro znovu načtení dat z databáze.



Obrázek 9 - Sekce "Mapa" běžného uživatele

4.3.2 Sekce „Uživatel“

Druhá sekce se skládá ze čtyř částí. V první části jsou uvedeny informace o přihlášeném uživateli (jméno a příjmení, uživatelské jméno, email). V další části se nachází přepínač, pomocí kterého lze zapnout (popř. vypnout) odesílání poloh uživatele. Pro změnu hesla slouží tlačítko umístěné v další části. A jako poslední je červené tlačítko sloužící pro odhlášení uživatele.

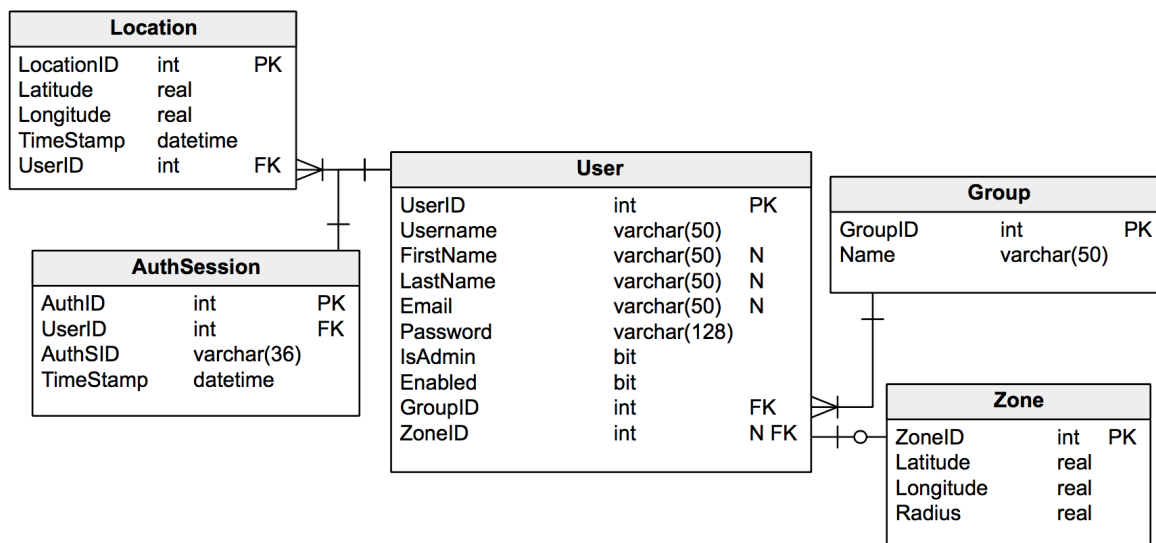


Obrázek 10 - Sekce "Uživatel" běžného uživatele

4.3.3 Sekce „O aplikaci“

Tato sekce je totožná se sekci „O aplikaci“ administrátora.

5 MS SQL DATABÁZE



Obrázek 11 - Schéma databáze

Pro ukládání dat a nastavení aplikace jsem použil databázi typu MS SQL 2012. Obsahuje celkem 5 tabulek: „User“, „Location“, „AuthSession“, „Group“ a „Zone“. Tabulka „User“ obsahuje jednotlivé běžné uživatele a administrátory, tabulka „Location“ shromažďuje nasnímané polohy uživatelů, tabulka „AuthSession“ slouží pro zabezpečení a ukládání aktuálně přihlášených uživatelů, tabulka „Group“ obsahuje jednotlivé skupiny uživatelů a tabulka „Zone“ slouží pro ukládání povolených kruhových zón uživatelů.

V databázi existují celkem 4 propojení tabulek. Prvním z nich je propojení tabulky „User“ s „Location“, kde každý uživatel může mít různý počet uložených nasnímaných lokací. Dalším je propojení tabulky „User“ s „AuthSession“, kde je po přihlášení jednomu uživateli vždy přiřazeno pouze jedno autorizované sezení. Třetí propojení je mezi tabulkou „Group“ a „User“, kde pro jednu skupinu uživatelů může existovat jeden či více uživatelů. Poslední propojené tabulky jsou „Zone“ a „User“, kde uživatel může mít definovanou svou vlastní unikátní povolenou kruhovou zónu.

5.1 Tabulka „User“

UserID

Identifikační číslo uživatele, které je primárním klíčem tabulky. Je unikátní a automaticky se inkrementuje.

Username

Uživatelské jméno uživatele o maximální délce 50 znaků.

FirstName

Křestní jméno uživatele o maximální délce 50 znaků. Nepovinný údaj.

LastName

Příjmení uživatele o maximální délce 50 znaků. Nepovinný údaj.

Email

Emailová adresa uživatele o maximální délce 50 znaků. Nepovinný údaj.

Password

Heslo uživatele ve formě SHA-512 hashe o délce 128 znaků.

IsAdmin

Může nabývat logické hodnoty 0 (běžný uživatel) nebo 1 (administrátor).

Enabled

Může nabývat logické hodnoty 0 (zakázaný uživatel) nebo 1 (povolený uživatel).

GroupID

Identifikační číslo skupiny, do které uživatel patří. Je cizím klíčem do tabulky „Group“.

ZoneID

Identifikační číslo povolené kruhové zóny, kterou má uživatel nastavenou. Je cizím klíčem do tabulky „Zone“. Nepovinný údaj.

5.2 Tabulka „Location“

LocationID

Identifikační číslo nasnímané lokace, které je primárním klíčem tabulky. Je unikátní a automaticky se inkrementuje.

Latitude

Zeměpisná šířka lokace ve formě desetinného čísla.

Longitude

Zeměpisná délka lokace ve formě desetinného čísla.

TimeStamp

Datum a čas pořízení lokace.

UserID

Identifikační číslo uživatele, který nasnímal tuto lokaci. Je cizím klíčem do tabulky „User“.

5.3 Tabulka „AuthSession“**AuthID**

Identifikační číslo pro aktivní sezení uživatele, které je primárním klíčem tabulky. Je unikátní a automaticky se inkrementuje.

UserID

Identifikační číslo uživatele, který je aktuálně přihlášený. Je cizím klíčem do tabulky „User“.

AuthSID

Náhodně vygenerovaný .NET GUID (globálně jedinečný identifikátor) o délce 36 znaků, který slouží pro zabezpečení komunikace.

TimeStamp

Datum a čas přihlášení uživatele.

5.4 Tabulka „Group“**GroupID**

Identifikační číslo skupiny uživatelů, které je primárním klíčem tabulky. Je unikátní a automaticky se inkrementuje.

Name

Název skupiny uživatelů o maximální délce 50 znaků.

5.5 Tabulka „Zone“**ZoneID**

Identifikační číslo povolené kruhové zóny, které je primárním klíčem tabulky. Je unikátní a automaticky se inkrementuje.

Latitude

Zeměpisná šířka středu zóny ve formě desetinného čísla.

Longitude

Zeměpisná délka středu zóny ve formě desetinného čísla.

Radius

Poloměr kruhu zóny v metrech ve formě desetinného čísla.

6 POPIS API METOD

Požadavky na server se posílají metodou GET s požadovanými parametry, odpovědi vrací API ve formě JSON dat. Celé API je postaveno na technologii ASP.NET a metody jsou psané v jazyku C#.

Po vykonání příkazu „/login“, který slouží pro přihlášení, získá uživatel unikátní číslo „authsid“, které musí používat při vykonávání všech ostatních příkazů pro jejich autorizaci.

Ve výstupních JSON datech se nachází vždy položky „code“ a „message“ a „data“. „Code“ obsahuje návratový kód, který může nabývat hodnot 100 (požadavek se úspěšně vykonal) nebo 101 (požadavek selhal). S tím souvisí položka „message“, která vrací při kódu 100 „Command was successfully finished“ a při kódu 101 „Command failed“.

6.1 /login

Popis

Provádí přihlášení uživatele. Nejdříve najde v databázi v tabulce „User“ záznam s odpovídajícím uživatelským jménem a porovná, zda si odpovídají hashe hesel. Před provedením přihlášení se navíc kontroluje, zda je uživatel povolený (položka Enabled). Pokud se přihlášení podaří, přidá se do tabulky „AuthSession“ záznam o aktuálním přihlášení uživatele (ID uživatele, vygenerované AuthSID a čas přihlášení). Zároveň se také zkontroluje, zda v tabulce „AuthSession“ neexistuje záznam se stejným ID uživatele. Pokud ano, provede se smazání tohoto předchozího přihlášení uživatele. Uživatel může být v jeden okamžik přihlášen pouze z jednoho zařízení.

Parametry požadavku

username – uživatelské jméno

password – SHA-512 hash uživatelského hesla

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/login?username=novakjosef&password=abce51a73c98c3d88c312a555b97d7c242c1958b343e4aec6e314be438381dbd5da973e083daa1b991ff86cc6ffe9c1af9fc186512b480abbef5fa1da0b5e0ab
```

Obrázek 12 - Požadavek metody "/login"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data":
    {
      "authsid": "63e48d2c-bcc4-4de7-b20d-cf067fdd855f",
      "userinfo":
      {
        "userid": 4,
        "username": "novakjosef",
        "firstname": "Josef",
        "lastname": "Novák",
        "email": "josefnovak@mail.cz",
        "isadmin": "true",
        "groupid": 1,
        "zoneid": 3
      }
    }
  }
}
```

Obrázek 13 - Odpověď metody "/login"

Popis odpovědi

authsid – vygenerovaný .NET GUID, unikátní ID sloužící pro zabezpečení komunikace, používá se pro ověření uživatele při volání dalších příkazů

userinfo – souhrn informací o přihlášeném uživateli

userinfo -> userid – ID uživatele

userinfo -> username – uživatelské jméno

userinfo -> firstname – křestní jméno uživatele

userinfo -> lastname – příjmení uživatele

userinfo -> email – email uživatele

userinfo -> isadmin – true (uživatel je admin) / false (jedná se o běžného uživatele)

userinfo -> groupid – ID skupiny, ke které uživatel náleží

userinfo -> zoneid – ID zóny, kterou má uživatel nastavenou

6.2 /logout

Popis

Provádí odhlášení uživatele. Na základě parametru „authsid“ najde odpovídající záznam v tabulce „AuthSession“ a odstraní jej.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/logout?authsid=63e48d2c-bcc4-4de7-b20d-cf067fdd855f
```

Obrázek 14 - Požadavek metody "/logout"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data": ""
  }
}
```

Obrázek 15 - Odpověď metody "/logout"

6.3 /getusersbygroup

Popis

Slouží pro získání skupiny uživatelů, která náleží přihlášenému adminovi. Před provedením příkazu se ověří v tabulce „User“, jestli je přihlášený uživatel admin (položka „IsAdmin“). Poté se na základě parametru „groupid“ vyberou uživatelé z tabulky „User“, kteří mají nastavený parametr „IsAdmin“ na false.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

groupid – ID skupiny

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/getusersbygroup?authsid=63e48d2c-bcc4-4de7-b20d-cf067fdd855f &groupid=1
```

Obrázek 16 - Požadavek metody "/getusersbygroup"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data":
    {
      "users":
      [
        {
          "userid": 6,
          "username": "petrych",
          "firstname": "Petr",
          "lastname": "Rychlý",
          "email": "rychlypetr@mail.cz",
          "enabled": true,
          "zoneid": 3
        },
        ...
      ]
    }
  }
}
```

Obrázek 17 - Odpověď metody "/getusersbygroup"

Popis odpovědi

users – načtená skupina uživatelů

users -> **userid** – ID uživatele

users -> **username** – uživatelské jméno

users -> **firstname** – křestní jméno uživatele

users -> **lastname** – příjmení uživatele

users -> **email** – email uživatele

users -> *enabled* – true (uživatel je povolený) / false (uživatel je zakázaný)

users -> *zoneid* – ID zóny, kterou má uživatel nastavenou

6.4 /getlocationsforuser

Popis

Slouží pro získání zvoleného počtu (parametr „*numberoflocations*“) posledních uložených lokací zvoleného uživatele (parametr „*userid*“). Data se získávají z tabulky „*Location*“ dle „*userid*“ seřazené podle času nasnímání lokace „*timestamp*“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

numberoflocations – počet posledních uložených lokací k zobrazení (0 = zobrazit všechny uložené lokace)

userid – ID uživatele, jehož uložené lokace chceme zjistit

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/getlocationsforuser?authsid=63e48d2c-bcc4-4de7-b20d-cf067fdd855f &numberoflocations=5&userid=6
```

Obrázek 18 - Požadavek metody "/getlocationsforuser"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data":
    {
      "locations":
      [
        {
          "latitude": 46.587,
          "longitude": 23.245,
          "timestamp": "2015-04-02T10:00:00"
        },
        {
          "latitude": 45.874,
          "longitude": 24.474,
          "timestamp": "2015-04-02T09:50:00"
        },
        ...
      ]
    }
  }
}
```

Obrázek 19 - Odpověď metody `"/getlocationsforuser"`

Popis odpovědi

locations – seznam načtených lokací uživatele

locations -> *latitude* – zeměpisná šířka lokace

locations -> *longitude* – zeměpisná délka lokace

locations -> *timestamp* – datum a čas pořízení lokace

6.5 /getlocationsbydateforuser

Popis

Slouží pro získání uložených lokací k určitému datu (parametr „date“) pro zvoleného uživatele (parametr „userid“). Data se získávají z tabulky „Location“ dle „userid“ a porovnáváním „date“ s „timestamp“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

date – datum, z kterého chceme načíst uložené lokace

userid – ID uživatele, jehož uložené lokace chceme zjistit

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/getlocationsbydateforuser?authsid=63e48d2c-bcc4-4de7-b20d-cf067fdd855f&date=2015-04-02&userid=6
```

Obrázek 20 - Požadavek metody "/getlocationsbydateforuser"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data":
    {
      "locations":
      [
        {
          "latitude": 46.587,
          "longitude": 23.245,
          "timestamp": "2015-04-02T10:00:00"
        },
        {
          "latitude": 45.874,
          "longitude": 24.474,
          "timestamp": "2015-04-02T09:50:00"
        },
        ...
      ]
    }
  }
}
```

Obrázek 21 - Odpověď metody "/getocationsbydateforuser"

Popis odpovědi

locations – seznam načtených lokací uživatele

locations -> *latitude* – zeměpisná šířka lokace

locations -> *longitude* – zeměpisná délka lokace

locations -> *timestamp* – datum a čas pořízení lokace

6.6 /addlocationforuser

Popis

Provádí přidání nasnímané lokace do databáze. Lokace se přidává do tabulky „Location“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

userid – ID uživatele

latitude – zeměpisná šířka

longitude – zeměpisná délka

timestamp – datum a čas pořízení lokace

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/addlocationforuser?authsid=63e48d2c-bcc4-4de7-b20d-  
cf067fdd855f&userid=6&latitude=12.456&longitude=15.123&timestamp=2015-04-02  
17:00:00
```

Obrázek 22 - Požadavek metody "/addlocationforuser"

Odpověď

```
{  
  "response":  
  {  
    "code": 100,  
    "message": "Command was successfully finished",  
    "data": ""  
  }  
}
```

Obrázek 23 - Odpověď metody "/addlocationforuser"

6.7 /getzoneforuser

Popis

Slouží pro získání povolené kruhové zóny uživatele dle parametru „zoneid“. Data se získávají z tabulky „Zone“ dle „ZoneID“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

zoneid – ID zóny

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/getzoneforuser?authsid=63e48d2c-bcc4-4de7-b20d-cf067fdd855f &zoneid=8
```

Obrázek 24 - Požadavek metody "/getzoneforuser"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data":
    {
      "latitude": 32.585,
      "longitude": 10.443,
      "radius": 45.0
    }
  }
}
```

Obrázek 25 - Odpověď metody "/getzoneforuser"

Popis odpovědi

latitude – zeměpisná šířka středu zóny

longitude – zeměpisná délka středu zóny

radius – poloměr kruhu zóny v metrech

6.8 /setzoneforuser

Popis

Provádí nastavení zóny pro vybraného uživatele. Před provedením příkazu se ověří v tabulce „User“, jestli je přihlášený uživatel admin (položka „IsAdmin“), protože nastavovat zóny smí pouze admin. Poté se zjistí dle parametru „userid“ v tabulce „User“, zda má uživatel nastavenou nějakou zónu. Pokud ano, tak se dle jeho „ZoneID“ upraví v tabulce „Zone“ parametry zóny. Pokud ne, tak se v tabulce „Zone“ vytvoří nová zóna a její „ZoneID“ se uloží do tabulky „User“ danému uživateli.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

userid – ID uživatele

latitude – zeměpisná šířka středu zóny

longitude – zeměpisná délka středu zóny

radius – poloměr kruhu zóny v metrech

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/setzoneforuser?authsid=63e48d2c-bcc4-4de7-b20d-cf067fdd855f &userid=6&latitude=58.475&longitude=25.845&radius=50.0
```

Obrázek 26 - Požadavek metody "/setzoneforuser"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data": ""
  }
}
```

Obrázek 27 - Odpověď metody "/setzoneforuser"

6.9 /createuser

Popis

Provádí vytvoření nového uživatele dle zadaných vstupních parametrů. Před provedením příkazu se ověří v tabulce „User“, jestli je přihlášený uživatel admin (položka „IsAdmin“), protože vytvářet nové uživatele smí pouze admin. Nový uživatel se ukládá do tabulky „User“. Pokud se vytváří admin, nastaví se mu buď id skupiny „GroupID“ stejné jako má přihlášený admin (bude spravovat stejnou skupinu uživatelů) a nebo se vytvoří v tabulce „Group“ nová skupina a vytvořenému adminovi se přidělí „GroupID“ nové skupiny. Při tvorbě běžného uživatele se nastavuje „GroupID“ stejné jako má přihlášený admin. Před samotným vytvořením uživatele se provede kontrola, zda již neexistuje zvolené uživatelské jméno. Položka „Enabled“ se při vytvoření uživatele nastavuje automaticky na „true“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

username – uživatelské jméno

firstname – křestní jméno (nepovinný parametr)

lastname – příjmení (nepovinný parametr)

email – email (nepovinný parametr)

password – SHA-512 hash hesla

isadmin – „true“ – admin, „false“ - běžný uživatel

groupname – název nové skupiny uživatelů (nepovinný parametr – pokud je tento parametr zadán a tvoří se admin, tak se vytvoří nová skupina pro uživatele)

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/createuser?authsid=63e48d2c-bcc4-4de7-b20d-  
cf067fdd855f&username=alfredo&firstname=Alfréd&lastname=Viskózní&email=alfredo  
v@mail.cz&password=abce51a73c98c3d88c312a555b97d7c242c1958b343e4aec6e314be  
438381dbd5da973e083daa1b991ff86cc6ffe9c1af9fc186512b480abb5fa1da0b5e0ab&isa  
dmin=true&groupname=Alfréd Big Brother Group
```

Obrázek 28 - Požadavek metody "/createuser"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data": ""
  }
}
```

Obrázek 29 - Odpověď metody "/createuser"

6.10 /edituser

Popis

Provádí uložení změn údajů existujícího uživatele dle zadaných vstupních parametrů. Před provedením příkazu se ověří v tabulce „User“, jestli je přihlášený uživatel admin (položka „IsAdmin“), protože editovat uživatele smí pouze admin. Pokud se při editaci změní uživatelské jméno „Username“, musí se provést kontrola, zda již není obsazené. Edituje se uživatel v tabulce „User“ dle parametru „userid“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

userid – ID uživatele, který se má editovat

username – upravené uživatelské jméno

firstname – upravené křestní jméno

lastname – upravené příjmení

email – upravený email

enabled – „true“ – uživatel povolen, „false“ - uživatel zakázán

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/edituser?authsid=63e48d2c-bcc4-4de7-b20d-
cf067fdd855f&userid=10&username=alfredo24&firstname=Alfréd&lastname=Viskózní&
email=alfredov2@mail.cz&enabled=true
```

Obrázek 30 - Požadavek metody "/edituser"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data": ""
  }
}
```

Obrázek 31 - Odpověď metody "/edituser"

6.11 /changepassword

Popis

Provádí změnu hesla u přihlášeného uživatele. Dle vstupního parametru „userid“ se najde v tabulce „User“ přihlášený uživatel. Porovná se SHA-512 hash „currentpassword“ s hashem „Password“ uloženým v tabulce. Pokud si odpovídají, uloží se do tabulky „User“ nové heslo pro uživatele ze vstupního parametru „newpassword“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

userid – ID přihlášeného uživatele

currentpassword – SHA-512 hash původního hesla

newpassword – SHA-512 hash nového hesla

Požadavek

```
http://lctr.aspone.cz/LCTR.aspx/changepassword?authsid=63e48d2c-bcc4-4de7-b20d-
cf067fdd855f&userid=10&currentpassword=3f08b178cf44b2ba1745bd72cf6c7db6f50973
85511e85ca9f210f12376ea0a43bc8323b0ec4a6b92c5ab3912bb7cc1b4b043ac811f664ba4
af7f3c51974935c&newpassword=abce51a73c98c3d88c312a555b97d7c242c1958b343e4a
ec6e314be438381dbd5da973e083daa1b991ff86cc6ffe9c1af9fc186512b480abbeb5fa1da0b
5e0ab
```

Obrázek 32 - Požadavek metody "/changepassword"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data": ""
  }
}
```

Obrázek 33 - Odpověď metody `"/changepassword"`

6.12 /deleteuser

Popis

Provádí mazání uživatele. Před provedením příkazu se ověří v tabulce „User“, jestli je přihlášený uživatel admin (položka „IsAdmin“), protože mazat uživatele smí pouze admin. V tabulce „User“ se dle vstupního parametru „userid“ najde požadovaný uživatel a ten se odstraní. Zároveň se smažou uživatelovy uložené lokace z tabulky „Location“ a popřípadě jeho zóna z tabulky „Zone“ a jeho přihlášení z tabulky „AuthSession“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

userid – ID uživatele k smazání

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/deleteuser?authsid=63e48d2c-bcc4-4de7-b20d-  
cf067fdd855f&userid=10
```

Obrázek 34 - Požadavek metody `"/deleteuser"`

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data": ""
  }
}
```

Obrázek 35 - Odpověď metody "/deleteuser"

6.13 /getgroupname

Popis

Slouží pro získání názvu skupiny uživatelů dle parametru „groupid“. Data se získávají z tabulky „Group“ dle „GroupID“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

groupid – ID skupiny

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/getgroupname?authsid=63e48d2c-bcc4-4de7-b20d-  
cf067fdd855f&groupid=3
```

Obrázek 36 - Požadavek metody "/getgroupname"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data":
    {
      "groupname": "Moji zaměstnanci"
    }
  }
}
```

Obrázek 37 - Odpověď metody `"/getgroupname"`

Popis odpovědi

groupname – název skupiny uživatelů

6.14 /editgroupname

Popis

Provádí úpravu názvu uživatelské skupiny. Před provedením příkazu se ověří v tabulce „User“, jestli je přihlášený uživatel admin (položka „IsAdmin“), protože měnit jméno uživatelské skupiny smí pouze admin. V tabulce „Group“ se dle vstupního parametru „groupid“ najde požadovaná skupina a změní se její název dle vstupního parametru „groupname“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

groupid – ID skupiny k editaci

groupname – nový název skupiny

Požadavek

```
http://lctr.aspone.cz/LCTR.aspx/editgroupname?authsid=63e48d2c-bcc4-4de7-b20d-cf067fdd855f&groupid=3&groupname=Moji věrní zaměstnanci
```

Obrázek 38 - Požadavek metody `"/editgroupname"`

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data": ""
  }
}
```

Obrázek 39 - Odpověď metody "/editgroupname"

6.15 /sendemail

Popis

Tato metoda se používá pro odesílání emailů administrátorovi, když některý z jeho uživatelů skupiny překročí hranici povolené zóny. Provádí odeslání emailu z emailové adresy „dp.locator.app@gmail.com“ na adresu z parametru „emailAddress“ s předmětem „subject“ a obsahem „content“.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

emailAddress – emailová adresa příjemce

subject – předmět emailu

content – obsah emailu

Požadavek

```
http://lctr.aspone.cz/LCTR.aspx/sendemail?authsid=63e48d2c-bcc4-4de7-b20d-
cf067fdd855f&emailAddress=josefnovak@mail.cz&subject=LCTR app -> překročení
zóny&content=Uživatel daniels překročil hranici své povolené zóny.
```

Obrázek 40 - Požadavek metody "/sendemail"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data": ""
  }
}
```

Obrázek 41 - Odpověď metody "/sendemail"

6.16 /isvalidsession

Popis

Slouží pro ověření, zda je uživatel přihlášený. Na základě parametru „authsid“ se ověří v tabulce „AuthSession“ existence záznamu o přihlášeném uživateli.

Parametry požadavku

authsid – unikátní ID pro autorizaci příkazů vygenerované příkazem „/login“

Požadavek

```
http://lctr.aspone.cz/LCTR.asmx/isvalidsession?authsid=63e48d2c-bcc4-4de7-b20d-
cf067fdd855f
```

Obrázek 42 - Požadavek metody "/isvalidsession"

Odpověď

```
{
  "response":
  {
    "code": 100,
    "message": "Command was successfully finished",
    "data":
    {
      "isvalid": true
    }
  }
}
```

Obrázek 43 - Odpověď metody "/isvalidsession"

Popis odpovědi

isvalid – true (uživatel je přihlášen) / false (uživatel není přihlášen)

7 POPIS KÓDU MOBILNÍ APLIKACE

Aplikace Locator (LCTR) byla vyvíjena v aplikaci Xcode verze 6.3.1 pomocí nového jazyka Swift od společnosti Apple, který byl aktuálně dostupný ve verzi 1.2. V aplikaci jsou obsaženy jednak třídy sloužící jako controllery pro jednotlivé obrazovky a mimo to také třídy, které zajišťují konkrétní funkce aplikace. Navíc bylo využito frameworku CoreLocation pro práci se snímáním polohy a frameworku MapKit pro práci s mapou. V projektu byly použity čtyři externí knihovny od cizích vývojářů.

7.1 Storyboard, LaunchScreen a složka Images

Storyboard je speciálním typem souboru, ve kterém se zobrazují vykreslené všechny obrazovky aplikace a je zde možno přizpůsobovat vzhled aplikace pomocí přidávání různých textových polí, popisků, tlačítek apod. LaunchScreen je podobným souborem s tím, že slouží pouze pro nastavení vzhledu úvodní obrazovky, která se zobrazuje než se načte aplikace. Složka Images je složkou určenou pro shromažďování všech obrázků, které se vyskytují v aplikaci.

7.2 Třída „AppDelegate“

Tato třída je hlavní třídou každé iOS aplikace. Vykonává se jako první po spuštění aplikace. Dají se zde odchytné různé změny v aplikaci, jako je minimalizace aplikace, její zavření atd. Z této třídy aplikace pokračuje do třídy „LoginViewController“.

7.3 View Controller třídy

Každá tato třída zajišťuje spravování své přidělené obrazovky.

7.3.1 LoginViewController

Jedná se o controller starající se o přihlašovací formulář. Zajišťuje správné načítání a ukládání uživatelského jména a hesla do klíčenky pomocí externí knihovny Locksmith. Validuje také vstupy formuláře před přihlášením a spouští přihlašovací proces. Během něj se postupně vykonávají tyto metody API : login (přihlášení uživatele a získání „authsid“), getgroupname (získání názvu skupiny), getusersbygroup (získání skupiny uživatelů spadajících pod správu administrátora, nevykonává se při přihlášení běžného uživatele), getzoneforuser (získání zóny běžného uživatele / získání zón uživatelů skupiny) a

getlocationsforuser (získání posledních lokací skupiny uživatelů, u běžného uživatele se volá getlocationsbydateforuser pro získání jeho poloh za aktuální den).

7.3.2 MapViewController

Tento controller se stará o správné zobrazování bodů s lokacemi uživatelů a zón v mapě v sekci „Mapa“. Pokud je přihlášený administrátor, tak zajišťuje i plnění tabulky správnými daty a funkčnost přepínačů. Zároveň jsou zde definovány metody pro odchytní tapnutí na jednotlivá tlačítka v horní části obrazovky. Pokud se zobrazuje tato sekce běžnému uživateli, spouští se zde i snímání polohy.

7.3.3 GroupTableViewCellController

Tato třída je zodpovědná za zobrazování sekce „Skupina“, když je přihlášený administrátor. Jedná se o controller spravující svou tabulku, zajišťuje správné vykreslení jednotlivých řádků a odchytní tapnutí na ně, kdy se poté zobrazí obrazovka s detailem, o kterou se stará už GroupUserDetailViewController. Součástí této třídy je i metoda starající se o modální zobrazení formuláře pro přidání uživatele, který je spravován třídou AddUserTableViewCellController.

7.3.4 UserTableViewCellController

Tento controller má na starost správné zobrazení a funkčnost sekce „Uživatel“. Obsahuje metody, které řídí správné zobrazení tabulky, která obsahuje vždy čtyři položky. Druhá položka se mění na základě typu přihlášeného uživatele. V případě administrátora se vyskytuje na druhém řádku buňka s možností změny názvu skupiny. V případě běžného uživatele je zde umístěn přepínač pro zapnutí či vypnutí odesílání polohy.

7.3.5 AboutViewController

Třída AboutViewController zajišťuje správné zobrazení sekce „O aplikaci“. Rozhoduje na základě typu zařízení, jaká bude velikost zobrazovaného loga aplikace.

7.4 Ostatní třídy

CoreDataLocation

Jedná se o singleton třídu (tzn. může být vytvořena pouze její jedna jediná instance), která zajišťuje ukládání, načítání a mazání lokací z core data. Lokace se ukládají do core data (v

podstatě se jedná o lokální sqlite databázi) vždy po jejich nasnímání a poté se až ukládají na server do databáze.

MapController

MapController je třídou, která zajišťuje veškerou práci s mapou. Poskytuje metody pro přidávání či odebírání bodů z mapy, pro vykreslování křivky do mapy nebo pro přidávání či odebírání kruhů znázorňujících povolené zóny uživatelů.

LocationManager

LocationManager je další singleton třídou a obsahuje metody sloužící k snímání polohy uživatele a sledování vstupů či odchodů uživatele z povolené zóny. Zároveň tato třída obsahuje metody k uložení a načtení, zda je povoleno snímání polohy.

ApiClient

ApiClient je obsáhlou singleton třídou, která obsahuje metody pro odesílání requestů pomocí GET metody na server. Všechny tyto metody odpovídají metodám API, tudíž tato třída se využívá pro komunikaci se serverem a databází. Také jsou zde definovány proměnné, které slouží k dočasnému uložení načtených dat z databáze.

User

Tato třída slouží k vytváření instancí jednotlivých uživatelů. Využití má při ukládání přihlášeného uživatele či skupiny uživatelů, která náleží pod správu nějakého administrátora. Obsahuje metodu, pomocí které lze formátovat zobrazení uživatelského jména.

Zone

Pomocí této třídy se tvoří instance jednotlivých povolených kruhových zón uživatelů. Každá zóna má definovány tři vlastnosti: zeměpisnou šířku, zeměpisnou délku a poloměr kruhu zóny v metrech.

Colors

Tato singleton třída zajišťuje generování pole náhodných barev, které se využívají poté pro rozpoznání jednotlivých uživatelů skupiny administrátora.

Strings

Strings jsou obsáhlou třídou, která obsahuje různé metody týkající se textů v aplikaci. Jsou zde definovány chybové hlášky při selhání dotazů na API, texty do alert boxů, popisky do HUD či formátovače datumů atd.

Extensions

V Extensions třídě jsou vytvořeny rozšíření pro tři základní datové typy. Pro datový typ String je zde definována metoda, která převádí textový řetězec na SHA-512 hash. Pro datový typ UIImage byla vytvořena metoda, která provádí přebarvení obrázku zvolenou barvou. A pro datový typ NSDate se zde nachází čtyři metody, které provádí různé operace s datemem.

7.5 Externí knihovny

7.5.1 DatePickerDialog

Tato knihovna slouží pro zobrazení dialogového okna s formulářem pro výběr datumu. V aplikaci je využita v detailu uživatele pro volbu datumu, z něhož chce administrátor zobrazit polohy. [11]

7.5.2 Locksmith

Jedná se o knihovnu, která je nadstavbou nad iOS klíčenkou. Slouží pro ukládání, načítání a mazání dat z klíčenky pomocí zjednodušených metod. V aplikaci se tato knihovna používá pro ukládání a načítání uživatelských údajů při přihlašování. [12]

7.5.3 SwiftHTTP

Knihovna SwiftHTTP slouží jako nadstavba okolo NSURLSession. Používá se v jazyku Swift pro zjednodušení tvorby HTTP requestů. V Locator aplikaci je hojně využita ve třídě ApiClient, kde se používá při volání API metod. [13]

7.5.4 SVProgressHUD

Poslední použitou knihovnou je SVProgressHUD, který slouží k zobrazení HUD s informací, že se načítají data nebo že bylo něco úspěšně či neúspěšně vykonáno. V aplikaci se zobrazuje vždy při načítání dat ze serveru při volání API metod. Zároveň také zobrazuje, zda byl dotaz na server úspěšný či neúspěšný. [14]

8 ZABEZPEČENÍ APLIKACE

V mé aplikaci bylo stěžejní zabezpečit komunikaci mezi mobilní aplikací a serverem, kde je umístěná databáze. Bezpečnost komunikace mezi mobilní aplikací a databází je zajištěna tím, že se k databázi nepřístupuje přímo, ale pomocí jasně definovaných metod v API. Zároveň pro úspěšné vykonání jakéhokoliv příkazu (kromě „/login“ příkazu) je potřeba mít vygenerován GUID textový řetězec o délce 36 znaků (v mém případě označený jako „authsid“), který získá uživatel pokaždé, když se úspěšně přihlásí a poté ho využívá pro autentizaci dalších příkazů. Hesla všech uživatelů se ukládají do databáze ve formě SHA-512 hashe, který má délku 128 znaků.

Celé webové API běží pro testovací účely na freehostingu na sdíleném serveru od firmy ASPone s.r.o.. Pro zvýšení bezpečnosti by se dalo nasadit API někam na placený webhosting, kde je podporován HTTPS protokol, který využívá šifrování pomocí SSL či TLS a zabránilo by se tím možným odposlechům komunikace.

ZÁVĚR

Hlavním cílem této diplomové práce bylo vytvořit mobilní aplikaci, která bude snímat aktuální polohy přihlášených uživatelů a odesílat je pomocí metod webového API do databáze na server. K těmto naměřeným datům uživatelů má poté z mobilní aplikace přístup administrátor, který má přístupné polohy těch uživatelů, kteří patří pod jeho správu (jsou členy jeho skupiny). Běžný uživatel má možnost v aplikaci sledovat svůj pohyb za poslední tři dny a zobrazuje se mu v mapě jeho povolená kruhová zóna (pokud má nějakou přidělenou). Při překročení hranic této zóny se automaticky odesílá jeho administrátorovi upozorňující email. Administrátor si může v aplikaci prohlížet libovolně vyfiltrované uložené polohy uživatelů z jeho skupiny, či jakémukoliv svému uživateli nastavit povolenou zónu. Mezi jeho kompetence patří také přidávání, editace a mazání uživatelů.

V rámci praktické části jsem detailně popsal, jak vypadá a funguje mobilní aplikace. Dále jsem se zabýval návrhem databáze, která slouží pro ukládání všech dat o polohách, zónách, uživatelích a skupinách. Poté jsem se zaměřil na webové API, které obsahuje celkem šestnáct metod. Ty jsou určeny pro práci s databází jako je přidávání, editace či mazání údajů z ní. Na závěr jsem rozebral strukturu aplikace, její jednotlivé třídy a celkové zabezpečení komunikace mezi ní a databází.

Co se týče teorie, nejdříve jsem se zaměřil na popis mobilního operačního systému iOS a jeho historie. Zároveň jsem popsal zásadní novinky jeho poslední verze. Ve druhé části teoretické části jsem se zabýval detailním popisem základů nového programovacího jazyku Swift. Na závěr teoretické části byla provedena analýza požadavků na vytvářenou mobilní aplikaci.

SEZNAM POUŽITÉ LITERATURY

- [1] APPLE INC. Uživatelské příručka pro iPhone s iOS 8.3. 2014. Dostupné z: <https://itunes.apple.com/cz/book/uzivatelska-prirucka-pro-iphone/id925163635?mt=11>
- [2] IOS. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2015 [cit. 2015-05-17]. Dostupné z: <http://en.wikipedia.org/wiki/IOS>
- [3] History of iOS. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2015 [cit. 2015-05-17]. Dostupné z: http://en.wikipedia.org/wiki/History_of_iOS
- [4] IOS: A visual history. *The Verge* [online]. 2013 [cit. 2015-05-17]. Dostupné z: <http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad>
- [5] Swift (programming language). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2015 [cit. 2015-05-17]. Dostupné z: [http://en.wikipedia.org/wiki/Swift_\(programming_language\)](http://en.wikipedia.org/wiki/Swift_(programming_language))
- [6] APPLE INC. The Swift Programming Language. 2014. Dostupné z: <https://itunes.apple.com/cz/book/swift-programming-language/id881256329?mt=11>
- [7] APPLE INC. Using Swift with Cocoa and Objective-C. 2014. Dostupné z: <https://itunes.apple.com/cz/book/using-swift-cocoa-objective/id888894773?mt=11>
- [8] AppCoda Community - Learn iOS Programming and Build iPhone App [online]. 2015 [cit. 2015-02-04]. Dostupné z: <http://www.appcoda.com>
- [9] Ray Wenderlich | Tutorials for iPhone / iOS Developers and Gamers [online]. 2015 [cit. 2015-02-04]. Dostupné z: <http://www.raywenderlich.com>
- [10] WWDC 2014 Session Videos: Learn about the latest in iOS and OS X with WWDC 2014 session videos. Apple Developer [online]. 2014 [cit. 2015-02-04]. Dostupné z: <https://developer.apple.com/videos/wwdc/2014/>
- [11] Squimer/DatePickerDialog-iOS-Swift. *GitHub, Inc.* [online]. 2015 [cit. 2015-05-20]. Dostupné z: <https://github.com/squimer/DatePickerDialog-iOS-Swift>
- [12] Matthewpalmer/Locksmith. *GitHub, Inc.* [online]. 2015 [cit. 2015-05-20]. Dostupné z: <https://github.com/matthewpalmer/Locksmith>

-
- [13] Daltoniam/SwiftHTTP. *GitHub, Inc.* [online]. 2015 [cit. 2015-05-20]. Dostupné z: <https://github.com/daltoniam/SwiftHTTP>
- [14] TransitApp/SVProgressHUD. *GitHub, Inc.* [online]. 2015 [cit. 2015-05-20]. Dostupné z: <https://github.com/TransitApp/SVProgressHUD>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface – rozhraní pro programování aplikací.
GPS	Global Positioning System – globální družicový polohový systém.
GUID	Globally Unique Identifier – globálně jedinečný identifikátor.
HTTP	HyperText Transfer Protocol – internetový protokol.
HTTPS	HyperText Transfer Protocol Secure – zabezpečený šifrovaný internetový protokol.
HUD	Head-Up Display – metoda zobrazení informací uživateli, většinou ve formě malého poloprůhledného okna, které se zobrazuje na popředí.
JSON	JavaScript Object Notation – způsob zápisu dat.
MSSQL	Microsoft Structured Query Language – relační databázový systém.
OS	Operating System - operační systém.
OTA	Over-The-Air – metoda distribuce updatů softwaru
PC	Personal Computer – osobní počítač.
SHA	Secure Hash Algorithm – rozšířená hashovací funkce.
SMS	Short Message Service – krátká textová zpráva.
SSL	Secure Sockets Layer – kryptografický protokol pro zabezpečení komunikace na Internetu.
TLS	Transport Layer Security – kryptografický protokol pro zabezpečení komunikace na Internetu, nástupce SSL.
URL	Uniform Resource Locator – řetězec znaků sloužící k přesné specifikaci umístění zdrojů informací na Internetu.
WiFi	Wireless Fidelity – bezdrátová komunikace v počítačových sítích.
WWDC	Worldwide Developer Conference – každoroční konference firmy Apple pro vývojáře.

SEZNAM OBRÁZKŮ

Obrázek 1 - Logo mobilní aplikace	53
Obrázek 2 - Přihlášení uživatele	54
Obrázek 3 - Sekce "Mapa" administrátora	55
Obrázek 4 - Sekce "Skupina" administrátora	56
Obrázek 5 - Vytváření uživatele	57
Obrázek 6 - Detail uživatele	58
Obrázek 7 - Sekce "Uživatel" administrátora	59
Obrázek 8 - Sekce "O aplikaci"	60
Obrázek 9 - Sekce "Mapa" běžného uživatele	61
Obrázek 10 - Sekce "Uživatel" běžného uživatele	62
Obrázek 11 - Schéma databáze	63
Obrázek 12 - Požadavek metody "/login"	67
Obrázek 13 - Odpověď metody "/login"	68
Obrázek 14 - Požadavek metody "/logout"	69
Obrázek 15 - Odpověď metody "/logout"	69
Obrázek 16 - Požadavek metody "/getusersbygroup"	70
Obrázek 17 - Odpověď metody "/getusersbygroup"	70
Obrázek 18 - Požadavek metody "/getlocationsforuser"	71
Obrázek 19 - Odpověď metody "/getlocationsforuser"	72
Obrázek 20 - Požadavek metody "/getlocationsbydateforuser"	73
Obrázek 21 - Odpověď metody "/getocationsbydateforuser"	73
Obrázek 22 - Požadavek metody "/addlocationforuser"	74
Obrázek 23 - Odpověď metody "/addlocationforuser"	74
Obrázek 24 - Požadavek metody "/getzoneforuser"	75
Obrázek 25 - Odpověď metody "/getzoneforuser"	75
Obrázek 26 - Požadavek metody "/setzoneforuser"	76
Obrázek 27 - Odpověď metody "/setzoneforuser"	76
Obrázek 28 - Požadavek metody "/createuser"	77
Obrázek 29 - Odpověď metody "/createuser"	78
Obrázek 30 - Požadavek metody "/edituser"	78
Obrázek 31 - Odpověď metody "/edituser"	79
Obrázek 32 - Požadavek metody "/changepassword"	79

Obrázek 33 - Odpověď metody "/changepassword"	80
Obrázek 34 - Požadavek metody "/deleteuser"	80
Obrázek 35 - Odpověď metody "/deleteuser"	81
Obrázek 36 - Požadavek metody "/getgroupname"	81
Obrázek 37 - Odpověď metody "/getgroupname"	82
Obrázek 38 - Požadavek metody "/editgroupname"	82
Obrázek 39 - Odpověď metody "/editgroupname"	83
Obrázek 40 - Požadavek metody "/sendemail"	83
Obrázek 41 - Odpověď metody "/sendemail"	84
Obrázek 42 - Požadavek metody "/isvalidsession"	84
Obrázek 43 - Odpověď metody "/isvalidsession"	84

SEZNAM PŘÍLOH

PI CD s diplomovou prací v elektronické podobě a zdrojovými kódy mobilní aplikace a webového API