UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Aplikace FishDiary pro iOS

Karel Borkovec

Diplomová práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Karel Borkovec**
Osobní číslo:	**I09356**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace FishDiary pro iOS**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem diplomové práce je vytvoření aplikace pro chytrý telefon, která bude sloužit k evidenci aktivit spojených s rybařením (evidence docházky, úlovků, počasí, polohy, atd.).
V úvodní části DP bude proveden rozbor operačního sytému iOS a objektového jazyka ObjectiveC a dále bude provedena rešerše dostupných aplikací na trhu, které tuto problematiku řeší.
V aplikační části práce bude proveden návrh architektury aplikace a její vlastní implementace, včetně zveřejnění aplikace v AppStoru.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

KOCHAN, Stephen G. Objective-C 2.0: Výukový kurz programování pro Mac OS X a iPhone. Brno: Computer press, a.s., 2010. ISBN 9788025126547.
APPLE INC. IOS Developer Library. 2007. Dostupné z: http://developer.apple.com /library/ios/
SADUN, Erica. The iOS 5 Developer's Cookbook: Core Concepts and Essential Recipes for iOS Programmers [online]. Third Edition. 2009 [cit. 2012-10-10]. ISBN 9780321832078. Dostupné z: http://www.amazon.com

Vedoucí diplomové práce: **Ing. Lukáš Čegan, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 17. 5. 2013 Karel Borkovec

Poděkování

Na tomto místě bych rád poděkoval Ing. Lukáši Čeganovi, Ph.D., za odborné vedení, rady a připomínky při tvorbě této práce.

Anotace

V teoretické části práce je proveden rozbor aplikací podobného typu, je popsána struktura programovacího jazyka Objective-C a znázorněna struktura operačního systému pro mobilní zařízení iOS. V praktické části je vysvětlena implementace aplikace FishDiary, rozebrány kroky nutné k publikaci aplikace v AppStore a popsány nástroje vhodné k dalšímu vývoji a analýze aplikace.

Klíčová slova

Apple, Objective-C, iOS, iPhone, ryba, rybář

Title

FishDiary application for iOS

Annotation

Theoretical part of the thesis analyzes similar applications, describes structure of the programming language Objective-C and shows the structure of operating system iOS for mobile applications. Practical part explains the implementation of FishDiary application, discusses the steps needed to publish application in the AppStore and describes the tools suitable for further development and analysis.

Keywords

Apple, Objective-C, iOS, iPhone, fish, angler

Obsah

Seznam zkratek **10**
Seznam obrázků **11**
Seznam tabulek **11**
Úvod **12**
1 Přehled aplikací pro rybáře **13**
1.1 Fishing Pro (go-fish) 13
1.2 Remember The Fish 14
1.3 Fishing Buddy 15
1.4 Fishbook 16
1.5 Vyhodnocení 17
2 Apple iOS **18**
2.1 Struktura (Vrstvy) 18
2.2 MVC v iOS aplikacích 19
2.3 Foundation framework 19
2.4 Delegace 20
2.5 Introspekce 21
2.6 UIKit 21
2.7 Core graphics 24
2.8 Quartz Core 24
2.9 Core Animation 24
2.10 Core Image 24
2.11 Core Data 24
2.12 Core Location 25
2.13 MapKit 25
2.14 Social Framework 25
2.15 Developer programy 25
2.16 Vývoj aplikací pro operační systém iOS 26
3 Jazyk Objective-C 2.0 ..... **27**
3.1 Datové typy 27
3.2 Zprávy 27
3.3 ARC 28

3.4 Protokoly ... 29
3.5 Kategorie ... 29
3.6 Bloky ... 30
4 Implementace aplikace FishDiary ... 31
4.1 Vývojové prostředí Xcode 4 ... 31
4.2 Architektura aplikace FishDiary ... 35
4.3 Ukládání dat ... 36
4.4 NSFetchedResultsController ... 41
4.5 Autorotace ... 43
4.6 Gesta ... 45
4.7 Práce s obrázky ... 47
4.8 Získání GPS souřadnic ... 47
4.9 Zobrazení vycházek a revírů na mapě ... 49
4.10 Lokalizace ... 52
4.11 Tipy pro optimalizaci ... 53
4.12 Dokončení ... 55
5 Marketing iOS aplikací ... 57
5.1 Webová stránka ... 57
5.2 Sociální sítě ... 58
5.3 Internetová fóra ... 58
5.4 Specializované PR magazíny ... 59
5.5 Promo kódy ... 59
5.6 Tiskový balíček ... 59
6 Zveřejnění aplikace v AppStore ... 61
6.1 Podepisování zdrojových kódů ... 61
6.2 iTunes Connect ... 62
6.3 Vyžadované grafické prvky ... 62
6.4 Nahrání aplikace do AppStore ... 62
6.5 Stavy aplikace ... 63
6.6 Stav – Ready for sale ... 64
7 Nástroje pro sledování aplikace ... 65
7.1 Testflight ... 65
7.2 Crashlytics ... 65

7.3 Google analytics 66
7.4 Nástroje pro sledování prodejů/stažení 67
8 Vyhodnocení výsledků aplikace po čtyřech měsících v AppStore 70
Závěr 71
Literatura a zdroje 72

Seznam zkratek

iOS	Operační systém pro mobilní zařízení společnosti Apple
Mac OS X	Operační systém pro osobní počítače společnosti Apple
UI	User interface – uživatelské rozhraní
ARC	Automatic Reference Counting – automatická správa paměti v iOS
MRC	Manual Reference Counting – manuální správa paměti v iOS
GPS	Global position systém – globální družicový polohový systém
ObjC	Objective-C – objektový programovací jazyk
API	Application programming interface – rozhraní pro programování
GCD	Grand central dispatch – rozhraní pro práci s vlákny v iOS
GSM	Groupe Spécial Mobile – globální systém pro mobilní komunikaci
HTTP	Hypertext Transfer Protocol – internetový protokol
WWDC	Worldwide Developers Conference – konference pro Apple vývojáře
XML	Extensible Markup Language – značkovací jazyk
Wi-Fi	Wireless Fidelity – bezdrátová komunikace
CMS	Content Management System – systém pro správu obsahu

Seznam obrázků

Obrázek 1 – Úvodní obrazovka aplikace Go-fish 14
Obrázek 2 – Úvodní obrazovka aplikace Remember The Fish 15
Obrázek 3 – Úvodní obrazovka aplikace Fishing Buddy 16
Obrázek 4 – Úvodní obrazovka aplikace Fishbook 17
Obrázek 5 – Základní vrstvy iOS, zdroj [2] 18
Obrázek 6 – MVC struktura včetně komunikace mezi vrstvami, zdroj [4] 19
Obrázek 7 – Struktura UINavigationControlleru, zdroj [2] 23
Obrázek 8 – UITabbarController, zdroj [2] 23
Obrázek 9 – Uživatelské rozhraní Xcode, zdroj [2] 31
Obrázek 10 – Storyboard, zdroj [2] 34
Obrázek 11 – Správa Gitu v Xcode 35
Obrázek 12 – UITabbarController aplikace FishDiary 36
Obrázek 13 – Databázové schéma aplikace FishDiary 39
Obrázek 14 – Podporované orientace v systému iOS, zdroj [2] 43
Obrázek 15 – Úvodní obrazovka aplikace FishDiary 55
Obrázek 16 – diagram vývoje iOS aplikací, zdroj [2] 61
Obrázek 17 – Rozhraní Crashlytics 66
Obrázek 18 – Google analytics pro mobilní aplikace 67
Obrázek 19 - iTunes Connect 68
Obrázek 20 – AppFigures – základní statistika 69
Obrázek 21 – AppFigures – grafy 69
Obrázek 22 – FishDiary – počet stažení 70
Obrázek 23 – FishDiary – počet stažení podle států 70

Seznam tabulek

Tabulka 1 – Porovnání dostupných rybářských aplikací 17
Tabulka 2 – Rozměry ikon a obrázků v pixelech 62
Tabulka 3 – Stavy aplikace v iTunes Connect 63

Úvod

Hlavním smyslem této práce je vyvinout aplikaci FishDiary pro mobilní zařízení s operačním systém iOS, která bude sloužit jako osobní záznamník rybáře. Současný trh nenabízí aplikaci, která sdružuje ulovené ryby během vycházky a umožňuje zobrazit celkový přehled ulovených ryb v jednotlivých revírech. Z tohoto důvodu bude vyvinuta aplikace FishDiary. Podmínky lovu se každoročně opakují a ryby v určitých situacích nebo podmínkách reagují stále stejně. FishDiary usnadní archivaci všech těchto údajů k pozdějšímu vyhledávání a analýze podmínek nebo konkrétních revírů. Zároveň aplikace FishDiary nabídne celkový roční přehled, který lze využít při kompletaci rybářského lístku v závěru sezóny.

Mobilní aplikace jsou v dnešní době velice populární. Na trhu s mobilními přístroji již chytré telefony převládají. Tím pádem se také rozrůstá trh s aplikacemi a vývojáři se snaží uživatelům usnadnit práci, aby byla co nejefektivnější.

Vývoj mobilních aplikací poskytuje práci s nejmodernější technologií a do budoucna má velký potenciál. Trh se stále vyvíjí a poptávka vzrůstá.

V úvodní části práce je proveden rozbor již existujících aplikací obdobného typu. Jsou analyzovány podporovaná zařízení, funkce a cena aplikace, tedy způsob monetizace, který byl autory zvolen.

V následující části je stručně představen jazyk Objective-C s důrazem na jeho specifické vlastnosti. Dále je popsána struktura operačního systému iOS, jeho vrstvy a nejpoužívanější frameworky.

V implementační části jsou podrobně popsány složitější části aplikace FishDiary, které byly při vývoji potřeba vyřešit, včetně ukázek zdrojových kódů. Marketing je nedílnou součástí vývoje každé aplikace, proto další kapitola popisuje způsoby marketingu mobilních aplikací.

V závěrečné části je popsán postup publikace aplikace v AppStore, cenová politika aplikace FishDiary a proveden rozbor nástrojů, které usnadní vývoj a analýzu dat po spuštění prodeje. Poslední kapitola vyhodnocuje dosažené výsledky po čtyřech měsících prodeje v AppStore. Jsou zde uvedena reálná data prodejů, včetně počtu prodejů v jednotlivých zemích. Závěrem je uveden souhrn možností, jak aplikaci vylepšit, aby byla více rentabilní a poskytovala větší uživatelskou hodnotu.

1 Přehled aplikací pro rybáře

V úvodní kapitole je provedena rešerše aplikací pro rybáře pro iOS. Aplikace jsou hodnoceny na základě podporovaných zařízení, funkcí, které nabízí uživatelům, a ceny.

1.1 Fishing Pro (go-fish)

Tato aplikace je pod záštitou anglického rybářského serveru, který má velkou uživatelskou základnu. Z tohoto důvodu se snaží jít směrem sociální sítě a maximálně využít již nashromážděná data.

Podporovaná zařízení

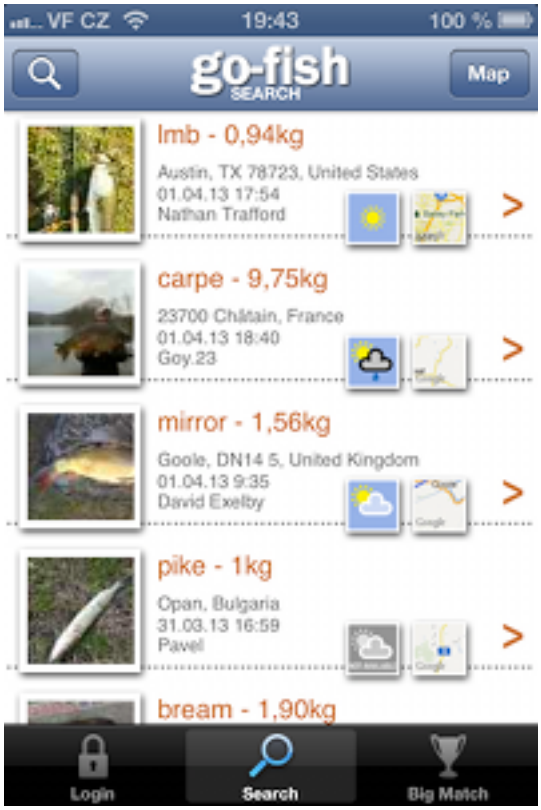
- iPhone, iPhone5,
- iPod touch,
- iPad,
- minimální verze iOS 4.3.

Funkce

Pro plné využití aplikace je nutná registrace. Po registraci a přihlášení je možné přidat svůj vlastní úlovek a k němu vyplnit další informace. K úlovku se automaticky načtou GPS souřadnice a úlovek se zobrazí ostatním uživatelům na mapě nebo v podobě tabulkového výpisu. V placené verzi je možné i přidávat rybářské obchody, které se taktéž zobrazí ostatním uživatelům.

Cena

Autor se rozhodl pro politiku dvou verzí aplikace. První „lite“, která je zdarma, zobrazuje reklamu a neobsahuje všechny funkce a druhá plná verze, která je však placená, bez reklam obsahuje všechny funkce a stojí $4.99.



Obrázek 1 – Úvodní obrazovka aplikace Go-fish

1.2 Remember The Fish

Aplikace slouží jako osobní zápisník jednotlivých úlovků.

Podporovaná zařízení

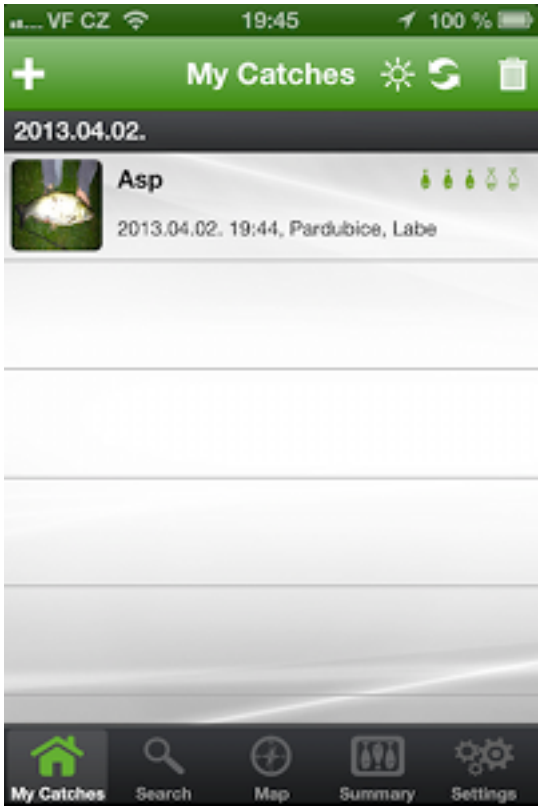
- iPhone, iPhone5,
- iPod touch,
- iPad,
- minimální verze iOS 5.

Funkce

Ke každému úlovku je možno přidat 2 fotografie. Úlovky lze zobrazit přehledně na mapě. Součástí je i jednoduchý agregovaný výpis, kdy lze zobrazit počet úlovků a jejich celkovou váhu za 1 rok. Aplikace umožňuje sdílet úlovky na Facebooku, ale nepodporuje integrovaný Facebook v iOS 6.

Cena

Aplikace je nabízena ve dvou verzích. „Lite“ verze, která je zdarma, má plnou funkcionalitu, ale je omezen počet úlovků na pět, kdežto v plné placené verzi je počet úlovků neomezený. Placená verze stojí $1.99.



Obrázek 2 – Úvodní obrazovka aplikace Remember The Fish

1.3 Fishing Buddy

Fishing Buddy je velmi jednoduchá aplikace, co se funkcí týče. I přesto je uživatelské rozhraní zbytečně složité a není příliš intuitivní.

Podporovaná zařízení

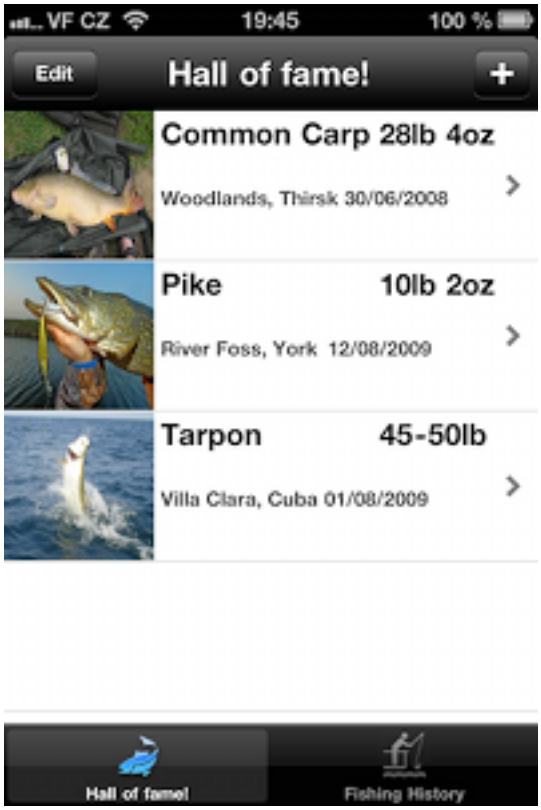
- iPhone,
- iPod touch,
- iPad,
- minimální verze i0S 3.0.

Funkce

Umožňuje pouze ukládat jednotlivé úlovky, včetně způsobu lovu a místa. Bohužel chybí GPS souřadnice a zobrazení na mapě.

Cena

Je nabízena pouze placená verze, která stojí $0.99.



Obrázek 3 – Úvodní obrazovka aplikace Fishing Buddy

1.4 Fishbook

Fishnook je jednoduchá aplikace, která plní svůj účel – osobní záznamník úlovků. Z pohledu rybáře chybí funkce jako GPS souřadnice pro zobrazení na mapě, agregovaný výstup dat a další.

Podporovaná zařízení

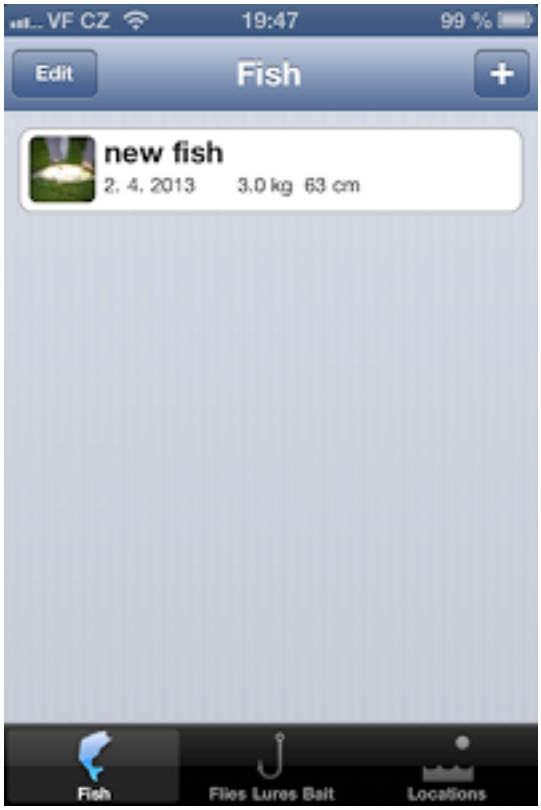
- iPhone,
- iPod touch,
- iPad,
- minimální verze iOS 4.0.

Funkce

Lze ukládat jednotlivé úlovky, navíc je možné evidovat nástrahy a revíry, které lze přiřadit k úlovkům. Je možné nastavit jednotky pro ukládání váhy a délky (kilogramy, libry, centimetry, palce).

Cena

Na trhu jsou dvě verze aplikace. „Lite“ verze má omezenou kapacitu pro ukládání úlovků, plná verze je neomezená a stojí $0.99.



Obrázek 4 – Úvodní obrazovka aplikace Fishbook

1.5 Vyhodnocení

Aplikace jsou vhodné k využití pro osobní účely, ale ani jedna z nich neumožňuje seskupovat pod jednotlivé revíry více vycházek a úlovků. Právě tuto funkcionalitu nabídne aplikace FishDiary. Celkový roční souhrn usnadní rybářům kompletaci rybářských lístků před odevzdáním na konci sezóny.

Tabulka 1 – Porovnání dostupných rybářských aplikací

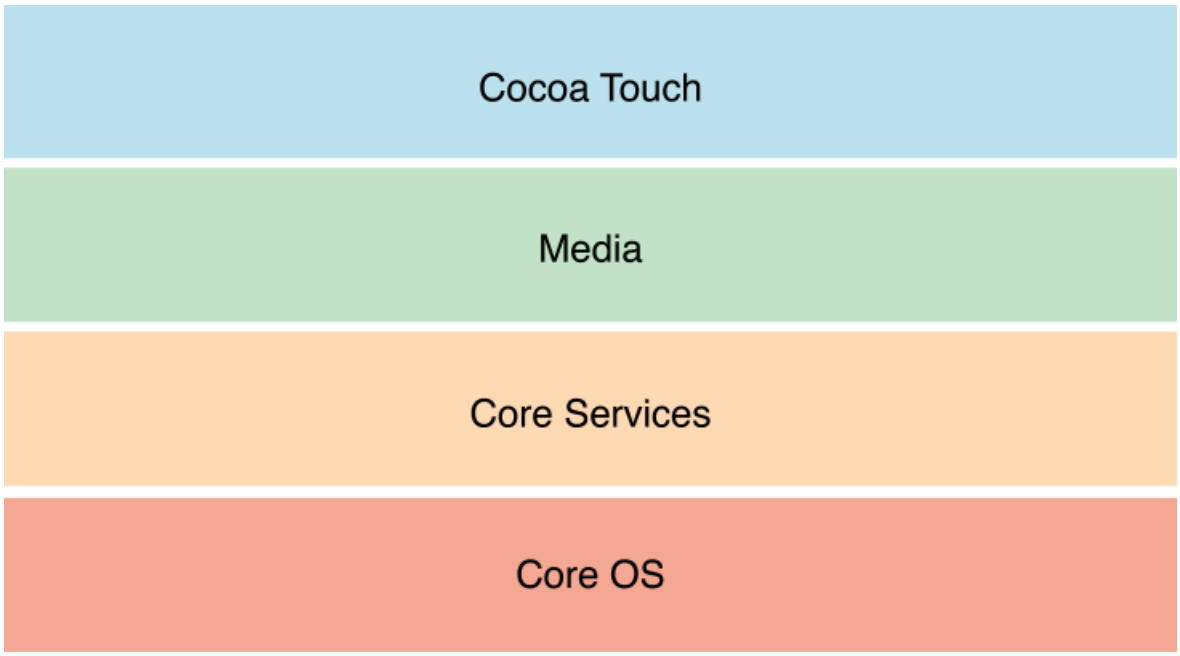
Název aplikace	Podporuje všechna zařízení	Minimální verze iOS	Cena
Fishing Pro	Ano	4.3	$4.99
Remember The Fish	Ano	5.0	$1.99
Fishing Buddy	Ne	3.0	$0.99
Fishbook	Ne	4.0	$0.99

Všechny analyzovaná aplikace podporují platformu iOS, která je podrobně představena v následující kapitole.

2 Apple iOS

iOS je mobilní operační systém společnosti Apple, který byl původně vytvořen pro mobilní telefon iPhone, později byl však využit i pro další zařízení jako je iPad, iPod touch nebo pro Apple TV. Jedná se o odlehčenou verzi Mac OS X, kterou Apple používá na svých osobních počítačích a noteboocích. Jedná se o operační systém na bázi UNIXu a jelikož je určen pro mobilní zařízení, neobsahuje všechny funkce OS X, ale navíc přidává např. podporu dotykových displejů.

2.1 Struktura (Vrstvy)



Obrázek 5 – Základní vrstvy iOS, zdroj [2]

Operační systém iOS je rozdělen do několika vrstev. Základní vrstvy používají C API, nejsou objektové. Vždy je doporučené používat nejvyšší vrstvy, pokud je to možné. Ve většině případů lze využít objektové nadstavby nad nižšími vrstvami.

Související služby jsou rozděleny do frameworků, které je nutno do projektu importovat, aby mohly být používány jejich funkce.

Cocoa touch

Definuje základní architekturu aplikace a zavádí podporu pro klíčové technologie jako multitasking, detekci dotykových gest a další systémové služby.

Media

Obsahuje vše, co se týká grafických prvků, audia a videa. Spadají sem frameworky jako Core graphics, Core Animation, Core Image, OpenGL ES a GLKit nebo Core Text.

Core Services

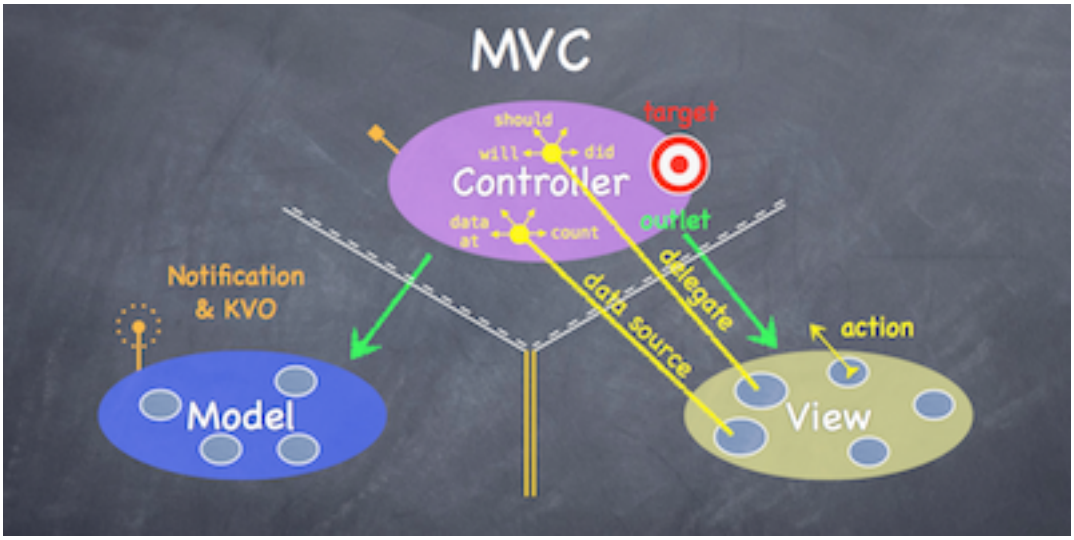
Jsou systémové služby, které programátor ani nemusí při vývoji používat, ale mnoho systémových funkcí je používá. Hlavní služby jsou iCloud storage, Automatic Reference Counting, bloky, Grand Central Dispatch (GCD). Z frameworků se zde nacházejí např. Accounts framework, Address Book framework, Core data, Core foundation, Core Location nebo Core Motion.

Core OS

Core OS je nejnižší vrstva iOS, nad kterou je postavena většina ostatních frameworků. Nejčastěji se používá, když je nutné komunikovat s externím hardwarem. Např. Core Bluetooth nebo Security framework.

2.2 MVC v iOS aplikacích

iOS aplikace používají návrhový vzor Model-View-Controller.



Obrázek 6 – MVC struktura včetně komunikace mezi vrstvami, zdroj [4]

Kontrolery jsou vždy potomky třídy UIViewController nebo tříd sloužících ke konkrétnímu způsobu zobrazování, jako je UITableViewController, UINavigationController nebo UITabbarController.

2.3 Foundation framework

Je souborem základních Objective-C tříd jako jsou třídy pro řetězce, čísla, datové struktury, data a ostatní. V následujících kapitolách je uveden seznam nejvíce používaných tříd.

NSObject

Je rodičem všech tříd v iOS, definuje metody pro vytváření jakýchkoliv objektů. Je to dvojice metod alloc/init. Metoda alloc je metodou třídní, kdežto metoda init je metodou instanční.

Datové struktury

Instance tříd obsahující slovo „mutable“ lze po inicializaci měnit (přidávat nebo odebírat hodnoty). Instance třídy bez slova „mutable“ se při inicializaci nastaví na konkrétní velikost a tu později změnit nelze.

- NSArray/NSMutableArray – klasické pole s indexy od 0,
- NSDictionary/NSMutableDictionary – slovník, stejný jako pole, jen indexy mohou být jakékoliv objekty (např. řetězce),
- NSSet/NSMutableSet – neřazená kolekce objektů.

Datové třídy

- NSString – uchovává řetězce, pole unicode znaků,
- NSNumber – uchovává jakékoliv číselné typy dostupné v jazyce C, včetně typů bool a char, implementuje metodu compare, tudíž lze hodnoty porovnávat,
- NSDate – třída pro práci s daty,
- NSData – práce s čistými byty dat.

Všechny ostatní třídy lze nalézt v dokumentaci[1].

2.4 Delegace

Cocoa touch velmi často využívá tzv. delegaci. K dosažení co nejvyšší míry abstrakce se využívá delegování určitých činností (např. zdroj dat pro UITableView) na jiné objekty. Jelikož delegát je ve třídě definován tak, že přijímá určitý protokol, je třída, vlastnící delegáta, povinná implementovat všechny metody a vlastnosti daného protokolu.

Např. třída UITableView má vlastnost delegate a dataSource. Vlastnost delegate přijímá protokol UITableViewDelegate a vlastnost dataSource přijímá protokol UITableViewDataSource. Všechny metody protokolu UITableViewDelegate jsou volitelné, takže je není nutné implementovat. Avšak UITableViewDataSource má dvě povinné metody a to tableView:cellForRowAtIndexPath: a tableView:numberOfRowsInSection:. Implementace těchto dvou metod bude vždy záviset

[1] https://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/ObjC_classic/_index.html

na konkrétním datovém modelu aplikace. Delegací se tudíž dosáhlo požadované vysoké míry abstrakce.

```
@interface FishingTripsViewController () <UITableViewDataSource,
UITableViewDelegate>

- (void)viewDidLoad
{
    self.tableView.delegate = self;
    self.tableView.dataSource = self;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    …
}

-(NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    …
}
```

2.5 Introspekce

Datový typ id se často používá jako návratová hodnota metod nebo jako typ parametrů metod. Jelikož definuje obecný typ objektu, je zapotřebí mít k dispozici nástroje, které umožňují určit typ objektu nebo zda odpovídá na konkrétní zprávu.

Kontrola na typ objektu se zavolá pomocí třídní metody class třídy NSObject:

```
[object isKindOfClass:[NSIndexPath class]];
```

U protokolů je možné definovat metody jako volitelné. Kvůli abstrakci není jisté, zda třída, která protokol přijímá, volitelnou metodu implementuje. Než se zavolá volitelná metoda protokolu, je nutné ověřit, zda na ni třída odpovídá.

```
[tableView respondsToSelector:@selector(headerViewForSection:)];
```

Podobně lze zjistit, zda třída přijímá konkrétní protokol.

```
[self conformsToProtocol:@protocol(UITableViewDelegate)];
```

Při introspekci je důležité se zamyslet, zda je vhodné vše testovat. Případně podmínky aplikovat pouze ve vydané aplikaci a při vývoji ne. Důvodem je snadnější hledání chyb ve zdrojovém kódu díky okamžitému pádu aplikace.

2.6 UIKit

Vše, co je spojeno s vizuálním uživatelským rozhraním, spadá do tohoto frameworku. UIKit poskytuje objekt samotné aplikace, obsluhu událostí, vykreslování, okna, pohledy a ovládací prvky navržené pro dotykové displeje.

Hlavní vlákno

Velmi důležitá je skutečnost, že veškeré změny v grafickém uživatelském rozhraní (všechna volání pomocí UIKit) musí být spouštěny na hlavním vlákně programu, které se označuje jako vlákno číslo 0. Jeden z důvodů je to, že celý UIKit framework není tzv. „thread safe“ a to úmyslně. Pokud by byl „thread safe“, byla by práce programátora příliš komplikovaná. Takto je nutné dávat pozor pouze na to, aby se UIKit metody vždy volaly na hlavním vlákně. Spuštění na jiném vlákně může vést k neočekávaným výsledkům a náhodným pádům aplikace. Každé takovéto volání blokuje uživatelské rozhraní. Uživatelům se nelíbí, když aplikaci něco blokuje a oni musí čekat. Dobrým zvykem je provádět na hlavním vlákně pouze kód, který je nenáročný a rychle proveditelný. Ostatní úkoly je nutné přesunout do pozadí. Pokud je z nějakého důvodu nutné spustit časově náročný proces na hlavním vlákně, je vhodné uživatele nějakým způsobem informovat, že musí počkat.

UIViewController

Slouží jako prostředník pro komunikaci mezi view objekty a modelem. Ve většině případů se používá potomek této třídy.

Samotný kontroler má vlastnost view, která je hlavním prvkem v zobrazení grafického uživatelského rozhraní.

UITableViewController

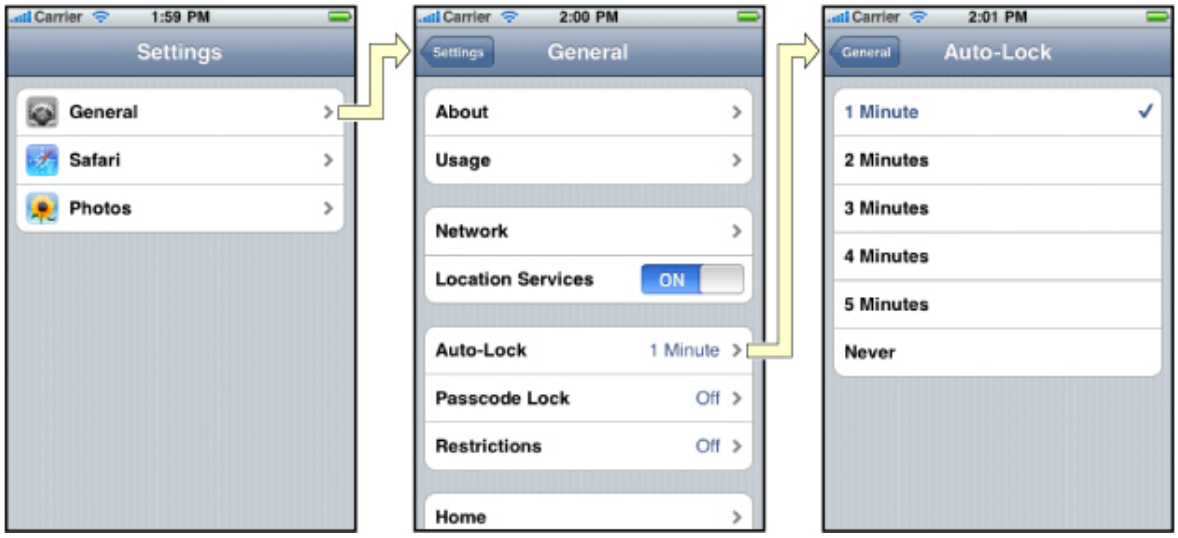
Jakýkoliv výpis, který má v iOS podobu tabulky, je vhodné implementovat pomocí potomka třídy UITableViewController. Úvodní obrazovka aplikace FishDiary, která zobrazuje jednotlivé vycházky, je toho příkladem.

Vlastnost view je v tomto kontroleru nastavena přímo na objekt UITableView, který vlastní objekty typu UITableViewCell. Jednotlivé buňky pak představují konkrétní vycházky.

UINavigationController

Jedná se o kontroler, který spravuje celou hierarchii kontrolerů. Z úvodní obrazovky FishDiary je možné po kliknutí na konkrétní vycházku přejít na detail vycházky a pomocí tlačítka „zpět“ je možné přejít zpět na celkový výpis. Toto chování je typické pro UINavigationController. Programátor se nemusí o nic starat, samotný kontroler sám ukládá a spravuje hierarchii a ví, kdy který kontroler zobrazit nebo vyjmout ze zásobníku.

Tento kontroler se používá téměř ve všech aplikacích.



Obrázek 7 – Struktura UINavigationControlleru, zdroj [2]

UITabbarController

Tabbar kontroler umožňuje jednoduché přepínání mezi kontrolery tak, že na spodní části obrazovky zobrazí panel, pomocí kterého je přepínání realizováno. Jedná se tak o velice rychlé, efektivní a přehledné přepínání mezi dílčími celky aplikace.

Obrázek 8 – UITabbarController, zdroj [2]

2.7 Core graphics

Core graphics je jeden z frameworků, který poskytuje pouze C API. Není nad ním žádná objektová nadstavba. Z toho vyplývá, že je potřeba velice pečlivě přistupovat ke správě paměti. Nová verze Xcode a kompilátoru již obsahuje mnoho upozornění, která programátora varují před možnými riziky úniku paměti.

2.8 Quartz Core

Tento Framework velice úzce souvisí s Core graphics. Poskytuje pokročilé vykreslování 2D grafiky. Podporuje hierarchie vrstev, vykreslování pomocí cest (velice často se používá vykreslování pomocí Bezierovy křivky) nebo vytváření či parsování PDF dokumentů.

2.9 Core Animation

Veškeré animace na obrazovce jsou v iOS pod záštitou tohoto frameworku. V nejnižší vrstvě se využívá grafického jádra, je tudíž použita nejvyšší míra efektivity. K vytvoření základních animací stačí pouze nastavit parametry a animaci spustit. O vše ostatní se framework postará sám. Díky tomu není tvorba animací složitá a výsledné aplikace jsou poté velmi příjemné pro koncové uživatele.

2.10 Core Image

Při zpracování obrázků a videa pomocí Core Image se rychlost téměř rovná reálnému času. Spolupracuje s Core Graphics a k vykreslování se využívá kapacita grafického jádra i procesoru, aby bylo dosaženo co nejvyšší efektivity.

Poskytuje mnoho vestavěných filtrů pro práci s obrázky, součástí je i velmi propracovaná detekce obličejů ve fotografiích.

2.11 Core Data

Při práci s relačními databázemi v iOS není nutná znalost SQL. Pomocí Core Data lze sestavovat plně objektové databázové modely. Nutno podotknout, že samotné Core Data nejsou relační databází, ale pouze objektovou abstrakcí nad datovým úložištěm, které nemusí mít s relační databází nic společného. Avšak velice často se Core Data používají ve spojení s SQLite databází.

Součástí je grafický editor, kde lze vytvářet databázové modely. Když je model hotový, je možné nechat Xcode, aby vygeneroval odpovídající třídy.

Migrace modelu

Problém s Core Data nastává, pokud se programátor rozhodne změnit nebo upravit model.

Jednoduché změny lze na model aplikovat automaticky pomocí nastavení parametrů při inicializaci Core Data. Složitější úpravy, jako je například změna nebo přidání vazeb mezi jednotlivými tabulkami, je nutné řešit pomocí systému zvaného „migrace modelu“.

To vyžaduje tvorbu vlastních migračních schémat, což nemusí být vždy úplně triviální záležitost.

2.12 Core Location

Umožňuje určit polohu zařízení a to i v případě pokud zařízení nemá vlastní GPS modul.

Stačí inicializovat třídu CLLocationManager a zapnout doručování změn polohy. Je možné vytvořit více instancí této třídy, ale doporučuje se udržovat pouze jednu konkrétní instanci v celé aplikaci. Toho lze dosáhnout jednoduše pomocí kategorií.

2.13 MapKit

Implementace map do vlastní aplikace se může zdát jako velice obtížná. MapKit API programátorovi velice usnadňuje práci. Stačí pouze vytvořit instanci MKMapView a tu přiřadit jako view kontroleru. Tímto krokem je kompletní mapa světa integrovaná do aplikace.

Do mapy je možné přidávat jakékoliv vlastní prvky, včetně překrytí mapy vlastními grafickými prvky.

2.14 Social Framework

Mimo jiné poskytuje interakci s dvěma nejvíce používanými sociálními sítěmi – Facebook a Twitter.

Bez tohoto frameworku by bylo nutné vytvářet ručně HTTP požadavky, což není vždy úplně komfortní. Práce se sítí v aplikacích pro mobilní zařízení je vždy náročný úkol. Není jisté, zda zařízení bude vůbec mít přístup k internetu nebo bude připojeno pomalým GSM signálem. Jako další je nutná znalost práce s API sociálních služeb.

Social Framework definuje třídu SLRequest, s jejíž pomocí je interakce se sociálními službami velmi jednoduchá.

2.15 Developer programy

K publikování aplikací v AppStore je nutné být registrovaný jako vývojář u společnosti Apple.

Po registraci se vývojářům otevře přístup k cenným studijním zdrojům jako je fórum, do kterého přispívají samotní Apple vývojáři, nebo možnost prohlížet videa z konferencí WWDC, která obsahují základní i podrobné teoretické znalosti iOS včetně ukázkových aplikací se zdrojovými kódy.

Během registrace se nabízí dvě možnosti.

Individuální vývojář

Tento typ vývojářského účtu je vhodný pro individuální vývojáře nebo pro malé týmy.

Společnost/Organizace

Pro společnosti nebo organizace se nabízí tento typ účtu. Umožňuje vytvářet vnitropodnikové aplikace, které nebudou publikovány v AppStore pro jiná zařízení, než jsou společností definovaná.

2.16 Vývoj aplikací pro operační systém iOS

Celý operační systém iOS je napsán v jazyce Objective-C, jež je objektovou nadstavbou nad jazykem C. Z tohoto důvodu musí být všechny aplikace pro iOS naprogramovány v tomto jazyce. Následující kapitola stručně popisuje jazyk Objective-C s důrazem na jeho specifické vlastnosti.

3 Jazyk Objective-C 2.0

Objective-C (ObjC) je objektově orientovaný jazyk, který je implementovaný jako nadstavba jazyka C. Jakýkoliv program napsaný v jazyce C lze přeložit pomocí Objective-C kompilátoru.

3.1 Datové typy

Lze používat všechny datové typy z jazyka C. Navíc jsou zavedené další datové typy specifické pro tento jazyk.

id

Slouží k uložení objektu jakéhokoliv typu. Typy id mohou být použity jako návratové hodnoty metod i jako hodnoty parametrů.

V jazyce Objective-C je datový typ id velice důležitou součástí, od kterého se odvíjí vlastnosti, jako je polymorfizmus a dynamická vazba.

SEL

Tzv. selektor umožnuje vytvořit ukazatel na metodu, která je součástí třídy, a předat ji např. jiné metodě jako parametr, která ji může spustit.

```
SEL selector = @selector(performSegueWithIdentifier:sender:);
```

Hodnota nil

Všechny neinicializované ukazatele na objekty v Objective-C mají základní hodnotu nil. To znamená, že ukazatel není nastaven na žádné konkrétní místo v paměti, resp. objekt není inicializován.

3.2 Zprávy

V názvosloví jazyka Objective-C se na místo volání metod používá: „poslat zprávu“. Vše vychází z jazyka Smalltalk, kde se Objective-C inspiroval.

Syntaxe pro poslání zprávy objektu foto s parametrem object vypadá následovně:

```
[foto containsObject:object];
```

Zprávy poslané na nil

Velmi důležitou vlastností jazyka Objective-C je, že pokud se pošle zpráva objektu, který je nil, nic se neděje (v programovacích jazycích jako je C# nebo Java by to způsobilo pád aplikace). Pokud má metoda návratový typ, vrátí se pouze nil, ale ve skutečnosti se žádná metoda nevolá. Této vlastnosti se dá velice dobře využívat a vede to k více přehlednému kódu, jelikož značně ubývá podmínek.

Tento kód nezpůsobí pád aplikace, nýbrž v pořádku proběhne:

```
NSArray *array = nil;
if ([array count] > 0) {
    ...
}
```

3.3 ARC

Porozumění správě paměti v jazyce Objective-C je velice důležité. Zejména v oblasti programování pro mobilní zařízení, která mají fyzickou paměť omezenou (starší zařízení pouze 256MB).

Objective-C neobsahuje žádný nástroj pro správu paměti jako je např. Garbage collector, který používá Java nebo C#. Garbage collector funguje tak, že v určitých cyklech spouští obsluhu, která dealokuje všechny objekty, které již nejsou potřeba. S tím je samozřejmě spojena nějaká režie, která může ubírat výkon.

Před verzí iOS 5 bylo nutné používat tzv. Manual Reference Counting (MRC). Dealokace objektů byla plně v režii programátora, a tudíž velice často docházelo k únikům paměti (memory leaks).

S vydáním iOS 5 Apple přidal Automatic Reference Counting (ARC), což mnohonásobně ušetřilo programátorům práci a zpřehlednilo kód.

Objective-C automaticky uchovává počet ukazatelů na objekt. Pokud se tento počet sníží na nulu, objekt je ihned dealokován.

Strong ukazatel na vlastnosti tříd

Když je vlastníkem objektu třída samotná a je nutné udržovat referenci po celou dobu existence instance třídy, využívá se strong reference. Tzn. objekt se strong referencí bude dealokován pouze v případě, kdy se nastaví na hodnotu nil.

Definice vlastnosti se strong ukazatelem („nonatomic“ znamená, že vlastnost nebude tzv. „thread safe“):

```
@property (strong, nonatomic) NSString *label;
```

Weak ukazatel na vlastnosti tříd

Weak referencí se dává kompilátoru najevo, že třída samotná nechce mít kontrolu nad životním cyklem objektu. Objekt bude dealokován v případě, že na něj žádná jiná třída nebude mít strong referenci. Tento přístup se využívá např. u outletů, kdy je zaručeno, že na ně vždy bude mít strong referenci view objekt.

Definice vlastnosti (outletu) s weak ukazatelem:

```
@property (weak, nonatomic) IBOutlet UISearchBar *searchBar;
```

3.4 Protokoly

Objective-C neobsahuje klasické rozhraní známé z jazyků jako Java nebo C#, ale zavádí tzv. protokoly.

Protokol je seznam metod a vlastností, které jsou sdíleny mezi jednotlivými třídami. Neobsahuje implementaci těchto metod, ale pouze jejich definici. Pokud třída přijímá konkrétní protokol, je jejím úkolem implementovat všechny povinné metody a vlastnosti.

Syntaxe pro vytvoření protokolu s jednou povinnou a jednou volitelnou metodou:

```
@protocol TripDetailViewControllerDelegate <NSObject>

@required
- (void)tripDetailViewController:(TripDetailViewController *)sender
saveTrip:(Trip *)trip;

@optional
- (void)tripDetailViewController:(TripDetailViewController *)sender
deleteTrip:(Trip *)trip;

@end
```

Za povšimnutí stojí, že samotný protokol může přijímat jeden nebo více jiných protokolů. Třída, která tento protokol bude přijímat, má za úkol implementovat povinné metody všech protokolů.

3.5 Kategorie

Existují situace, kdy je potřeba přidat k existující třídě další metody, ale dědičnost v tomto případě nedává smysl, proto jazyk Objective-C definuje vlastnost zvanou „kategorie“. Nově přidaná metoda bude mít plný přístup k instančním proměnným třídy.

Tato vlastnost je velice „mocná“, avšak dá se velmi snadno zneužívat. Vždy je potřeba se zamyslet, zda je kategorie to nejvhodnější řešení.

Omezení, které s sebou kategorie nesou, je, že není možné přidávat vlastní instanční proměnné.

Vytvoření kategorie na třídě Photo má následující syntaxi:

```
#import "Photo.h"

@interface Photo (Create)

+ (Photo *)createNewPhotoByDictionary:(NSDictionary *)photoDictionary
inManagedObjectContext:(NSManagedObjectContext *)context;

@end


@implementation Photo (Create)

+ (Photo *)createNewPhotoByDictionary:(NSDictionary *)photoDictionary
inManagedObjectContext:(NSManagedObjectContext *)context {
    …
}

@end
```

3.6 Bloky

V jazyce C# existuje vlastnost zvaná closures (někdy překládáno jako uzávěry), bloky v Objective-C jsou velice podobné. Jedná se o kus spustitelného kódu, který lze zadat např. jako parametr metody nebo ho definovat jako proměnnou. I když je blok spouštěn v jiném rozsahu, jsou v něm plně přístupné všechny lokální proměnné a dokonce je lze v bloku i měnit. Bloky mohou mít návratové hodnoty i parametry.

Velmi mnoho API v iOS bloky používá, ukázka znázorňuje jednoduchou animaci za pomoci bloků:

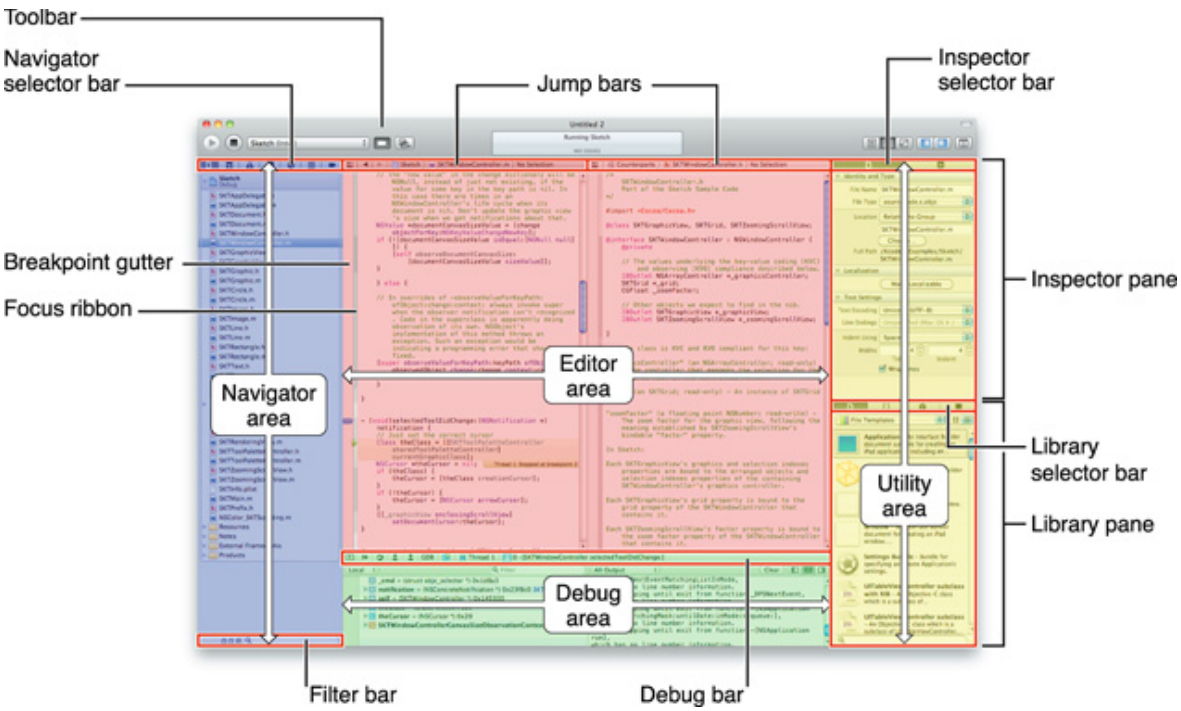
```
UIView *view = [[UIView alloc] init];
[UIView animateWithDuration:2.0
                 animations:^{
                     view.alpha = 0;
                 }
                 completion:^(BOOL finished) {
                     view.hidden = YES;
         }];
```

4 Implementace aplikace FishDiary

Pro vývoj aplikací v systému Mac OS X nebo iOS slouží integrované vývojové prostředí Xcode, které je popsáno v následující podkapitole.

4.1 Vývojové prostředí Xcode 4

Xcode je integrované vývojové prostředí určené pro vývoj iOS a Mac OS X aplikací. Součástí je jak editor zdrojových kódů, tak i editor grafického uživatelského rozhraní. Zároveň integruje podporu pro týmový vývoj a to zejména pomocí GITu nebo subversion. Při psaní Xcode kontinuálně nabízí automatické doplňování kódu a upozorňuje na chyby, ať už syntaktické nebo logické.



Obrázek 9 – Uživatelské rozhraní Xcode, zdroj [2]

Xcode nabízí mnoho funkcí, nejužitečnější popisuje následující seznam.

- Rozhraní v jednom okně – většina vývoje probíhá v jednom okně (je možné mít otevřené okna z více projektů a v každém okně mít otevřeno více záložek).
- Návrh grafického uživatelského rozhraní – pomocí tzv. „Interface builderu“ je možné nastavit většinu detailů grafické uživatelského rozhraní, jejich rozložení, propojení s business logikou a daty, které spravuje. „Interface builder“ je zároveň propojený s editorem zdrojového kódu kvůli rychlým přesunům mezi editory.
- „Asistovaná editace“ – když je potřeba pracovat na různých aspektech stejné komponenty, jako je např. grafické rozložení prvků a implementace funckionality, je možné zobrazit najednou více editorů a v nich různé soubory.

- Automatická identifikace chyb a oprava – Xcode automaticky kontroluje zdrojový kód a při výskytu chyby okamžitě vývojáře upozorní a v některých případech je schopný nabídnout řešení problému.

iPhone/iPad/Univerzální aplikace

Při zakládání nového projektu je nutné se rozhodnout, zda vytvořit aplikaci pouze pro iPhone, pouze pro iPad, nebo univerzální.

Univerzální aplikace je jednou aplikaci, která je optimalizovaná pro iPhone i iPad. Důsledkem je však nějaká práce navíc. Obrazovka u iPadu poskytuje širší možnosti díky její velikosti a při návrhu kvalitního grafického uživatelského rozhraní je to třeba brát v potaz. Z tohoto důvodů je možné definovat více samostatných rozhraní, jedno pro iPhone, druhé pro iPad. Implementace kontrolerů může zůstat pro obě rozhraní stejná nebo lze použít pro každé zařízení kontrolery jiné.

Je potřeba zvážit, zda implementovat pro každé zařízení jiný kontroler nebo použít stejný. Často je jednodušší použít kontrolery jiné, než použít jeden pro obě platformy. Pokud by se měl v obou opakovat stejný kód, je vhodné využít dědičnosti podle principů objektového programování.

Pokud aplikace používá vlastní obrázky k definici grafického uživatelského rozhraní je nutné poskytnout 2 verze, menší pro iPhone a větší pro iPad. To samé platí pro ikonu a veškerou nutnou grafiku.

Ve správně navržené struktuře zdrojového kódu by nemělo být nutné explicitně zjišťovat, na kterém zařízení je kód spuštěn. Avšak někdy se objeví skutečnost, kdy to nutné je. Z tohoto důvodu je vývojářům k dispozici API, pomocí kterého je možné zjistit, zda je kód spuštěn na iPhonu, nebo na iPadu:

```
[[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPad
```

Podpora různých verzí iOS

Každá aplikace, která podporuje více verzí iOS a používá funkce z vyšších verzí, musí za běhu používat kontrolu, zda je možné určité funkce použít. Např. FishDiary podporuje iOS 5, ale samozřejmě funguje i na jakékoliv vyšší verzi. Při implementaci sdílení fotografií je pro uživatele, jejichž zařízení používá iOS 6, přívětivější využít integrovanou podporu Facebooku, která byla představena právě ve verzi iOS 6 (hlavním důvodem je, že se uživatelé nebudou muset k Facebooku přihlašovat v aplikaci). V iOS 5 je však nutné k této funkcionalitě použít jiný přístup. Ve zdrojovém kódu je tedy nutné rozhodnout, kterým směrem se vydat a to umožňuje právě kontrola za běhu aplikace. Existuje několik způsobů.

- Zjistit, zda je metoda třídy dostupná, je možné pomocí metody instancesRespondToSelector: nebo respondsToSelector:.

- Zjistit, zda je dostupné funkce z jazyka C, je možné jednoduchou kontrolou.

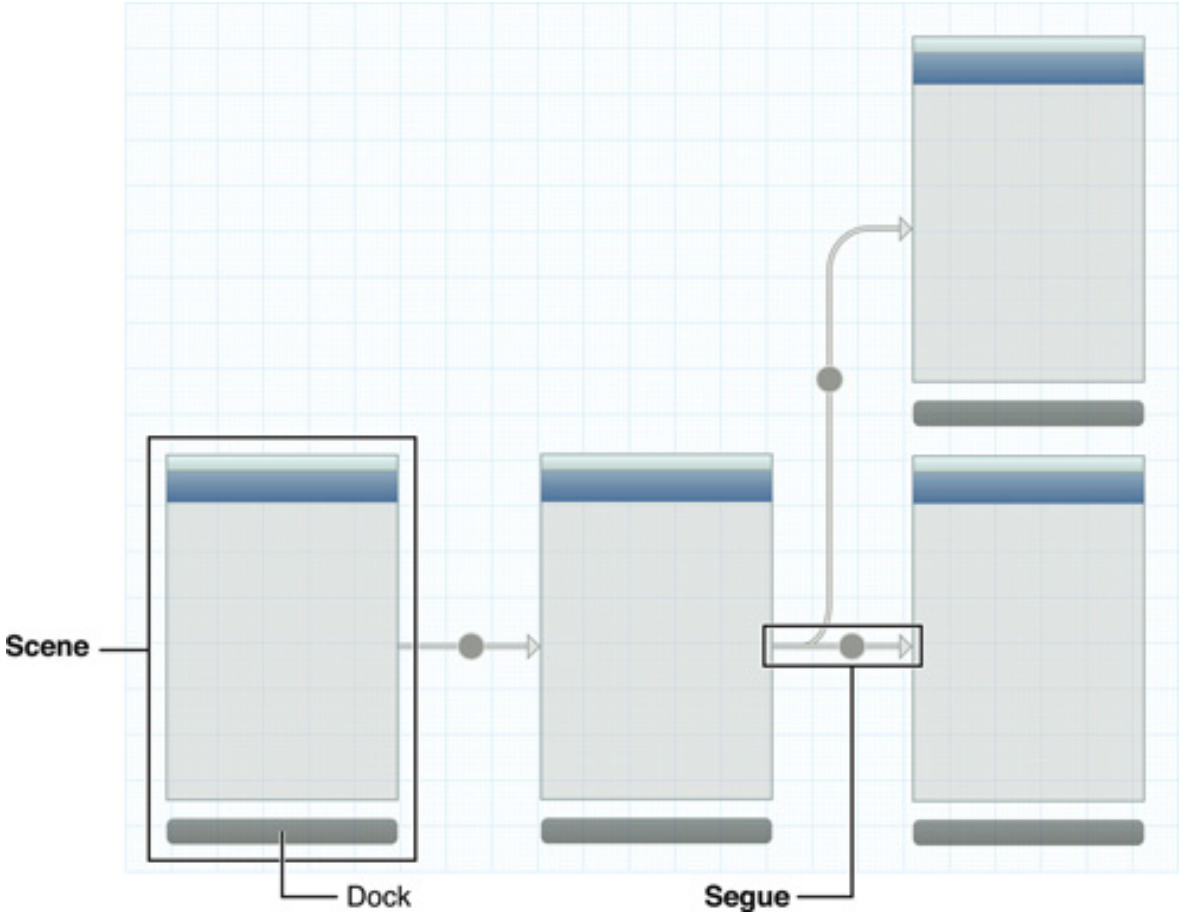
```
if (UIGraphicsBeginPDFPage != NULL)
{
    UIGraphicsBeginPDFPage();
}
```

- Ke zjištění, zda je možné použít konkrétní třídu, lze použít následují kód.

```
if ([UIPrintInteractionController class]) {
    // Je možné vytvořit instanci a třídu použít.
} else {
    // Třída není dostupná.
}
```

Storyboardy

S verzí iOS 5 se velmi výrazným způsobem změnila práce s grafickým uživatelským rozhraním. Storyboard představuje vizuální reprezentaci grafického uživatelského rozhraní iOS aplikací, zobrazuje obsah jednotlivých obrazovek a vazby mezi nimi. Každá obrazovka je propojena s konkrétním kontrolerem a jeho grafickými prvky, vazby jsou reprezentovány pomocí „segue“, které definují přechod mezi jednotlivými obrazovkami.



Obrázek 10 – Storyboard, zdroj [2]

Veškeré UI je tedy ve vývojovém prostředí sdruženo na jednom místě, kde lze také možné definovat hierarchie, např. pomocí tříd UINavigationController nebo UITabbarController.

Segue

Graficky určují vztahy mezi jednotlivými obrazovkami.

Každá „segue“ má svůj vlastní unikátní identifikátor, pomocí něhož je možné „segue“ identifikovat ve zdrojovém kódu. K použití se nabízí několik typů a každý typ má uplatnění v jiné situaci. Typ „push“ úzce souvisí s UINavigationControllerem, typ „modal“ je vhodné využít při akci, kterou je nutno dokončit, jako je např. vstup od uživatele, aby bylo možné pokračovat dále.

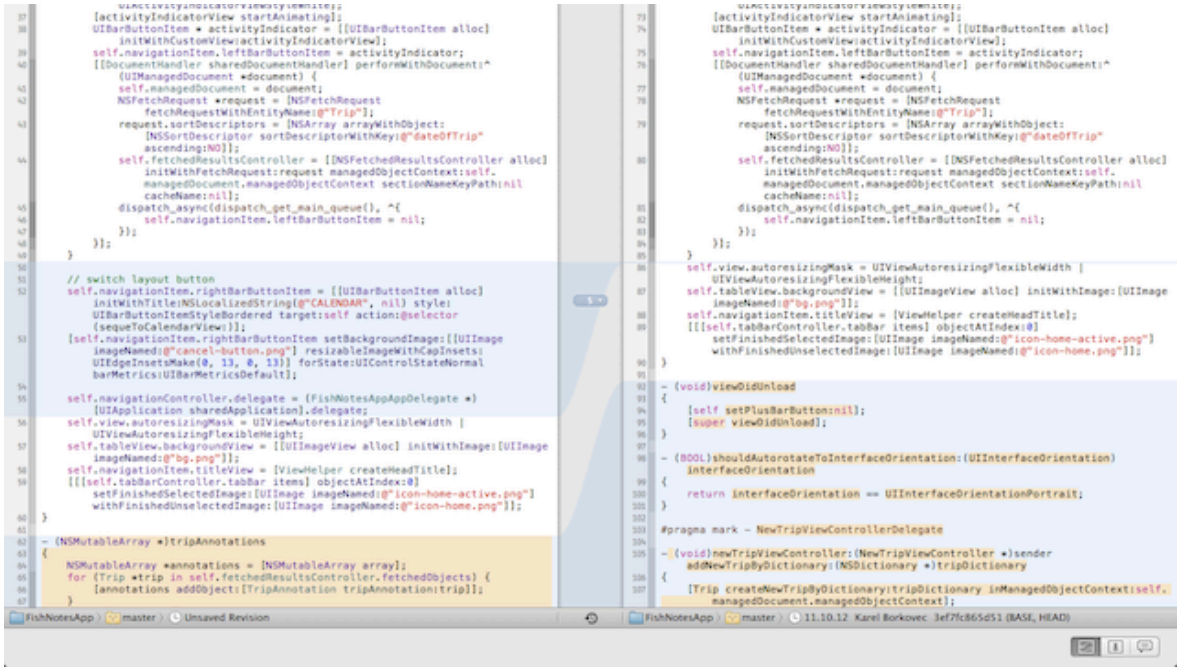
V některých případech je nutné se rozhodnout, kterou „segue“ použít, až ve zdrojovém kódu, resp. v kontroleru:

```
[self performSegueWithIdentifier:@"segue_identifier" sender:self];
```

Parametr sender je typu id, tudíž do něj lze vložit jakýkoliv objekt. Velmi často se používá pro předání prvku, který „segue“ vyvolal.

Verzování

Při vývoji aplikací v týmu je nutné zdrojový kód spravovat pomocí vhodných nástrojů. Správa zdrojového kódu však může být velice nápomocná i pro projekty, na kterých pracuje jen jeden vývojář. Xcode má přímou podporu pro dva nejpoužívanější nástroje a těmi jsou Subversion a Git. Subversion byl dříve velmi oblíben, ale v dnešní době se jeví jako lepší alternativa právě Git, který byl primárně vytvořen pro správu zdrojového kódu linuxového jádra.



Obrázek 11 – Správa Gitu v Xcode

4.2 Architektura aplikace FishDiary

FishDiary se skládá z několika dílčích celků, které je vhodné mít dostupné ze všech základních obrazovek. V aplikacích tohoto typu je vhodné využití UITabbarControlleru, který přesně poskytuje výše popsanou funkcionalitu.

Nejdůležitější funkcí je přidávání nových vycházek, proto je nutné tento krok v uživatelském rozhraní zvýraznit. Musí být přístupný ze všech základních obrazovek a proto se nabízí jeho umístění do spodní lišty UITabbarControlleru. Základní kontroler však neumožňuje zvýraznění konkrétního tlačítka, proto je nutné udělat malou změnu a výsledkem je takovýto vzhled.

Obrázek 12 – UITabbarController aplikace FishDiary

Všechny základní obrazovky mají možnost přecházet na další obrazovky, proto je nutné je vestavět do UINavigationControlleru, který podporuje právě tento typ přechodů.

První obrazovka po spuštění zobrazuje v tabulce všechny uložené vycházky, které jsou řazené podle data sestupně. Uživatel by měl mít možnost ve vycházkách vyhledávat podle určitých kritérií. Jelikož FishDiary slouží jako osobní zápisník a rybář bude pravděpodobně potřebovat historicky vyhledávat vycházky v konkrétních měsících v minulých letech, je uživatelsky přívětivé zobrazit vycházky v kalendáři. Z tohoto důvodu je na první obrazovce možné zobrazit vycházky jak tabulkově, tak v kalendáři.

Vyhledávat ve vycházkách je možné podle:

- názvu,
- popisu,
- ulovené ryby,
- revíru.

4.3 Ukládání dat

Téměř každá aplikace potřebuje ukládat vstupní data od uživatelů, se kterými poté pracuje. Při návrhu iOS aplikace je nutné analyzovat, který způsob ukládání dat zvolit. Systém nabízí několik způsobů.

NSUserDefaults

Jak samotný název třídy napovídá, tak tento způsob perzistence dat je určený pro ukládání uživatelských nastavení a menších objektů, tzn. není vhodný pro ukládání velkého množství dat. Snadno lze ukládat data, jako jsou řetězce, čísla, časová data, pole nebo slovníky. Je možné ukládat i vlastní objekty, ale většinou bude nutné je převést na objekt typu NSData.

FishDiary nabízí uživateli ukládat váhu a délku úlovku. Je nutné si uvědomit, že v různých zemích světa se používají různé jednotky, ať už metrické, imperiální nebo jiné. Tyto jednotky si uživatel může zvolit v nastavení, které je vhodné uložit právě do NSUserDefaults.

Property list

Ukládá data na disk ve stylu klíč – hodnota a používá standardizovaný formát XML. V iOS jsou velmi často používány. Xcode umí pracovat s XML soubory typu plist (zkratka používaná pro property list) a při otevření takového souboru je k dispozici grafický editor.

Není však možné ukládat všechny objekty, protože každý objekt nemusí být možné archivovat na typ NSData. To je možné pouze u objektů, které přijímají NSCoding protokol, který má 2 povinné metody.

```
-(void)encodeWithCoder:(NSCoder *)aCoder
-(id)initWithCoder:(NSCoder *)aDecoder
```

Když se program pokusí uložit objekt, který přijímá tento protokol je mu poslána zpráva encodeWithCoder: a jako parametr je předán objekt typu NSCoder. Úkolem této metody je uložit instanční proměnné pomocí objektu typu NSCoder předaného v parametru. Naopak zpráva initWithCoder: je objektu poslána při načtení z disku a jejím úkolem je nastavit instanční proměnné z dat předaných v parametru.

Document-based aplikace

Pro typ aplikací jako jsou různé poznámkové bloky nebo fotoalba se hodí tento typ ukládání dat. Data je možné ukládat na disk nebo do iCloudu. iCloud je vhodné zvolit v případě, že je v plánu vyvíjet aplikaci pro více zařízení – iPhone, iPad, ale třeba i OS X, což zajistí potřebnou synchronizaci dat mezi jednotlivými zařízeními.

Základem „document-based“ aplikací je třída UIDocument. Aplikace musí vytvořit potomka této třídy, který definuje, jak načítat data do vnitřních datových struktur této třídy a naopak jak data ukládat na disk. Např. v aplikaci typu poznámkový blok by každá poznámka představovala potomka třídy UIDocument a starala by se o uložení veškerých dat (text poznámky, datum vytvoření…). Jednotlivé objekty jsou reprezentovány jednotlivými soubory na disku, které zaručují jejich perzistenci.

Vytvoření nového dokumentu může trvat nějaký čas, proto vše probíhá na pozadí a po vytvoření se vykoná definovaný blok.

```
self.document = [[MyDocument alloc] initWithFileURL:url];
[self.document saveToURL:self.document.fileURL
        forSaveOperation:UIDocumentSaveForCreating
       completionHandler:^(BOOL success) {
                 if (success) {
                     ...
                 }
}];
```

Tato metoda se ihned navrátí a její obsluha je provedena v pozadí na jiném vlákně. Když je vytváření dokumentu dokončeno, zavolá se blok definovaný v parametru completionHandler. Důležité je, že blok bude spuštěn na stejném vlákně jako byla spuštěna metoda pro vytvoření (dobré je spouštět ji na hlavním vlákně), je to tudíž vhodné místo pro aktualizaci uživatelského rozhraní.

Při využití třídy UIDocument se programátor nemusí starat o ukládání změn, protože to probíhá automaticky v určitých časových intervalech. Časové intervaly nejsou přesně definovány, avšak je zaručeno, že dokument bude uložen. Uložení se provede vždy, když uživatel aplikaci ukončí.

Proces ukládání může být velice komplikovaný v případě použití iCloudu. Je nutné brát v potaz, že uživatel v tu chvíli nemusí být připojen k internetu nebo je jeho připojení velmi pomalé (GSM). UIDocument vše provede na pozadí, takže uživatel může pokračovat v práci a uložení proběhne ve chvíli, kdy to je možné.

Vlastní server

Další možností ukládání dat je vlastní server, který poskytuje API pro přístup k manipulaci s daty. Tento přístup je nejnákladnější ze všech. Jeho použití je nutné pro aplikace typu sociálních sítí nebo pro aplikace, jejichž vývoj je plánován pro různé platformy.

Dříve bylo nutné si celý server zařídit vlastními silami. To však pro mnoho vývojářů není možné kvůli omezeným financím, a proto vzniklo několik projektů, které umožňují ukládat data a sdílet je tak mezi aplikacemi na různých platformách. Velkou výhodou je, že poskytují knihovny ke komunikaci s jejich datovými úložišti. Implementace do aplikace je tedy velice snadná.

Příkladem takovéto služby může být Parse[2], který skýtá velké možnosti pro program zdarma. Je možné se zaregistrovat do jednoho ze tří programů. První program je zdarma a poskytuje 1 milion požadavků a 1 milion „push“ notifikací měsíčně. K dispozici je přívětivé API pro iOS, Android, Windows Phone, Windows 8 a OS X, včetně přehledné a kompletní dokumentace.

Core Data

Pro aplikace, které potřebují lokálně ukládat větší množství dat, je Core Data jedno z nejlepších řešení. Příkladem je právě aplikace FishDiary, která tento přístup využívá. Zároveň s třídou UIDocument, která byla implementována v iOS 5, vznikla i třída UIManagedDocument, která je jejím přímým potomkem a má za úkol správu Core Data objektů. Práce s Core Data se použitím této třídy velice zjednodušuje.

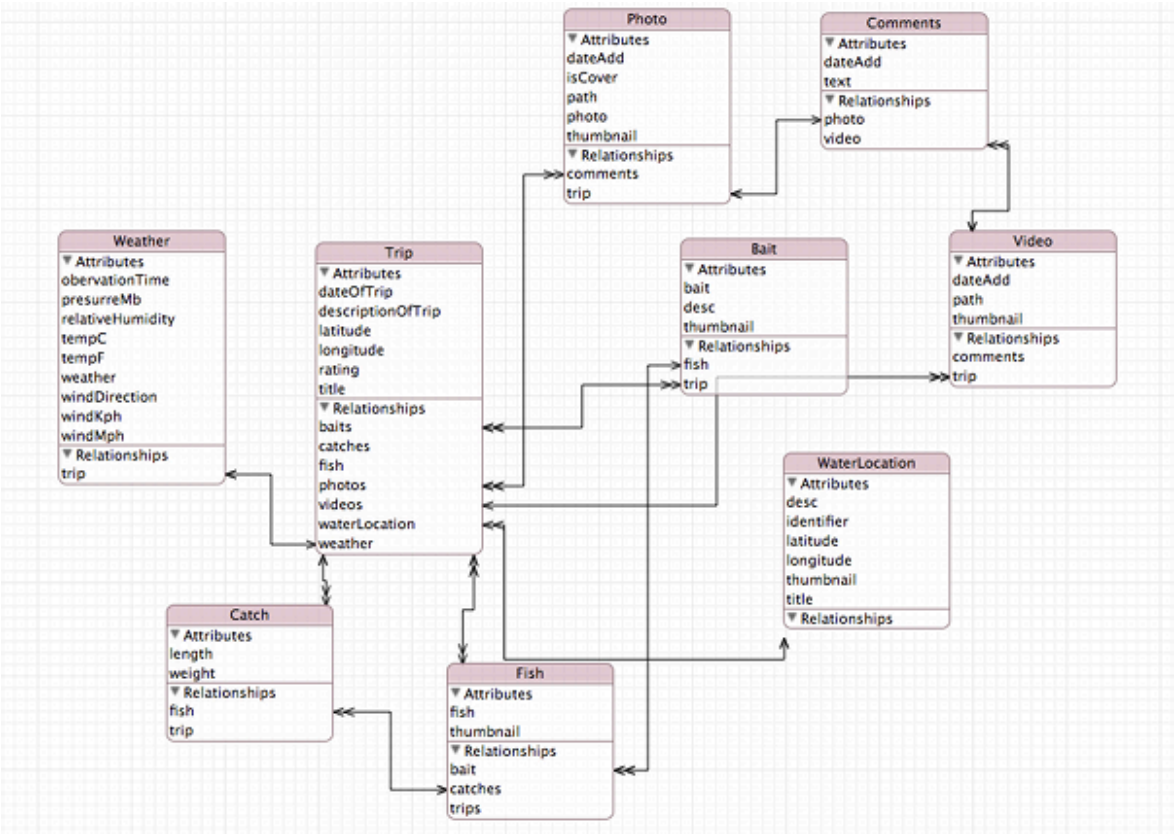
Core Data poskytuje:

- správu vztahů mezi entitami,
- co nejmenší využití paměti tím, že objekty jsou načítány až ve chvíli, kdy jsou potřeba,
- automatickou validaci objektů,
- migraci schématu,

[2] https://www.parse.com

- integraci s kontrolery pomocí NSFetchedResultsController,
- agregaci, filtrování dat a mnoho dalších.

Základními entitami databázového modelu v FishDiary jsou vycházky, počasí, úlovky, ryby, nástrahy, revíry, fotky a videa. Výsledkem analýzy těchto požadavků je následující schéma.



Obrázek 13 – Databázové schéma aplikace FishDiary

Core Data jsou pouze objektovou nadstavbou nad konkrétním fyzickým úložištěm, jak již bylo zmíněno výše. Bez explicitního nastavení využívají SQLite databázi. Je však nutné definovat objekty, které slouží jako prostředníci mezi jednotlivými typy úložišť, a ty se nazývají „persistent stores“. Dalším důležitým prvkem v této hierarchii objektů je „managed context“. Zjednodušeně lze říci, že to je vrstva, která se stará o převod dat modelových objektů mezi „persistent stores“ a uživatelskými objekty, se kterými pracuje programátor.

K práci s Core Data je zapotřebí instance třídy UIManagedDocument a vytvoření, resp. otevření dokumentu.

```
NSURL *url = [[[NSFileManager defaultManager]
URLsForDirectory:NSLibraryDirectory inDomains:NSUserDomainMask]
lastObject];
url = [url URLByAppendingPathComponent:@"Trips database"];
UIManagedDocument *document = [[UIManagedDocument alloc]
```

```
initWithFileURL:url];
NSDictionary *options = [NSDictionary dictionaryWithObjectsAndKeys:
[NSNumber numberWithBool:YES],
NSMigratePersistentStoresAutomaticallyOption,
[NSNumber numberWithBool:YES], NSInferMappingModelAutomaticallyOption,
nil];
document.persistentStoreOptions = options;

if (![[NSFileManager defaultManager]
fileExistsAtPath:[self.document.fileURL path]]) {
     [self.document saveToURL:self.document.fileURL
        forSaveOperation:UIDocumentSaveForCreating
        completionHandler:^(BOOL success) {
            ...
    }];
} else if (self.document.documentState == UIDocumentStateClosed) {
    [self.document openWithCompletionHandler:^(BOOL success) {
        ...
    }];
} else if (self.document.documentState == UIDocumentStateNormal) {
    ...
}
```

Toto je celý kód, který je potřeba pro vytvoření instance pracující s Core Data. Třída UIManagedDocument automaticky vytvoří příslušné objekty „persistent store“ a „managed context“. Objekt NSManagedContext je přístupný přes instanční proměnnou managedContext a je potřeba při veškeré práci s úložištěm a objekty typu NSManagedObject.

Objekty typu NSManagedObject představují konkrétní záznamy z databázového schématu (v relační databázi se jedná o konkrétní řádek). Xcode umožňuje jednoduché vygenerování těchto tříd podle schématu. Stačí otevřít příslušné schéma, označit entity, pro které se mají třídy vygenerovat, a v menu „Editor“ stisknout „Create NSManagedObject subclass…“. Do tříd se vygenerují všechny vlastnosti, které jsou definované v modelu, včetně metod pro práci se vztahy. Pokud je potřeba přidávat vlastní metody, vlastnosti nebo instanční proměnné, je vhodné nevpisovat je přímo do vygenerovaných souborů, ale vytvořit si kategorie. Kdyby došlo ke změně schématu a následnému regenerování tříd, Xcode by přepsal všechny změny, s využitím kategorií se toto nestane.

V aplikaci se bude velmi často vytvářet nová vycházka, tedy objekt typu Trip. Z tohoto důvodu je vhodné vytvořit si kategorii typu Trip+Create a v ní implementovat třídní metodu createNewTripByDictionary:inManagedObjectContext:.

```
+ (Trip *)createNewTripByDictionary:(NSDictionary *)tripDictionary
inManagedObjectContext:(NSManagedObjectContext *)context
{
    Trip *trip = nil;
    trip = [NSEntityDescription insertNewObjectForEntityForName:@"Trip"
inManagedObjectContext:context];
    trip.latitude = [tripDictionary objectForKey:TRIP_GPS_LATITUDE];
    trip.longitude = [tripDictionary objectForKey:TRIP_GPS_LONGITUDE];
…
return trip;
}
```

Z ukázky jednoznačně vyplývá, jak Core Data zajišťují typovou kontrolu dat. Vlastnosti longitude a latitude třídy Trip jsou typu NSNumber, kompilátor tudíž nedovolí nastavit objekt jiného typu.

Zavolání této metody stačí k tomu, aby se vytvořil nový objekt typu Trip a zároveň se uložil do databáze. Jak již bylo zmíněno, uložení probíhá automaticky.

4.4 NSFetchedResultsController

K získávání dat z databáze Core Data slouží objekt typu NSFetchRequest, pomocí něhož lze vytvářet komplexní dotazy. Objekt musí obsahovat název entity, ve které bude vyhledávat, a dále může obsahovat:

- predikát specifikující, které vlastnosti se mají vybrat, a omezení, která se aplikují na výslednou množinu,
- pole, které určuje jak jsou výsledky seřazeny.

Predikáty lze řetězit a vytvářet tak komplexní struktury pro získávání požadovaných výsledků. Ukázka inicializace a nastavení objektu typu NSFetchedRequest, který získá všechny vycházky seřazené sestupně podle data vytvoření.

```
NSFetchRequest *request = [NSFetchRequest
fetchRequestWithEntityName:@"Trip"];
request.sortDescriptors = [NSArray arrayWithObject:[NSSortDescriptor
sortDescriptorWithKey:@"dateOfTrip" ascending:NO]];
```

Jednotlivé vycházky je třeba zobrazit na úvodní obrazovce aplikace. Řešení, které se nabízí jako první, je uložit si všechny výsledky do pole a toto pole použít jako datový zdroj pro UITableView. V tomto případě by bylo nutné ošetřit všechny případy manipulace s daty, jako jsou vkládání nových vycházek, mazání vycházek či editace. Kvůli ulehčení práce iOS nabízí třídu NSFetchedResultsController, která řeší všechny výše zmíněné problémy téměř automaticky.

Tato třída je speciálně vytvořena pro efektivní správu dat získaných z Core Data a zobrazovaných v buňkách UITableView. Stačí jí nastavit kontext a „fetch request“, aby věděla odkud a jak získat data, vše ostatní třída dopočítá sama.

Mimo jiné nabízí i funkce jako:

- monitorování změn objektů v poskytnutém kontextu a informování delegáta o těchto změnách,
- cachování výsledků, aby nedocházelo k zbytečné opakované komunikaci s úložištěm.

Na základě toho, zda má NSFetchedResultsController specifikovaného delegáta a nastavenou cache, nabízí tři operační módy:

- žádné sledování – delegát ani cache nejsou nastaveny – NSFetchedResultsController poskytuje pouze přístup k datům v takovém stavu, v jakém byly při prvním načtení,
- sledování paměti – delegát je nastaven – veškeré změny jsou reflektovány v UITableView,
- úplné sledování – delegát i cache jsou nastaveny – změny jsou reflektovány v UITableView a navíc je udržována stálá cache.

Dokumentace uvádí, že je vždy výhodné implementovat instanci třídy NSFetchedResultsController jako instanční proměnnou. K inicializaci je potřeba zadat čtyři parametry, první dva jsou povinné, poslední dva jsou volitelné:

- objekt typu NSFetchRequest,
- kontext,
- klíčovou cestu pro získání názvu sekce,
- název cache.

Po inicializaci je zapotřebí poslat zprávu performFetch:. Tímto krokem je nastavení kontroleru hotové. Pokud se načtený objekt z Core Data změní, je delegát kontroleru okamžitě upozorněn. Velmi užitečné je implementovat tuto metodu delegátu:

```
- (void)controller:(NSFetchedResultsController *)controller
   didChangeObject:(id)anObjec
       atIndexPath:(NSIndexPath *)indexPath
     forChangeType:(NSFetchedResultsChangeType)type
      newIndexPath:(NSIndexPath *)newIndexPath;
```

Při vložení nové vycházky se tato metoda postará o zobrazení vycházky v UITableView, stejně tak při smazání nebo editaci názvu zajistí synchronizaci uživatelského rozhraní s datovým zdrojem.

Delegát UITableViewDataSource má dvě povinné metody, které se implementují pomocí „fetch results“ kontroleru.

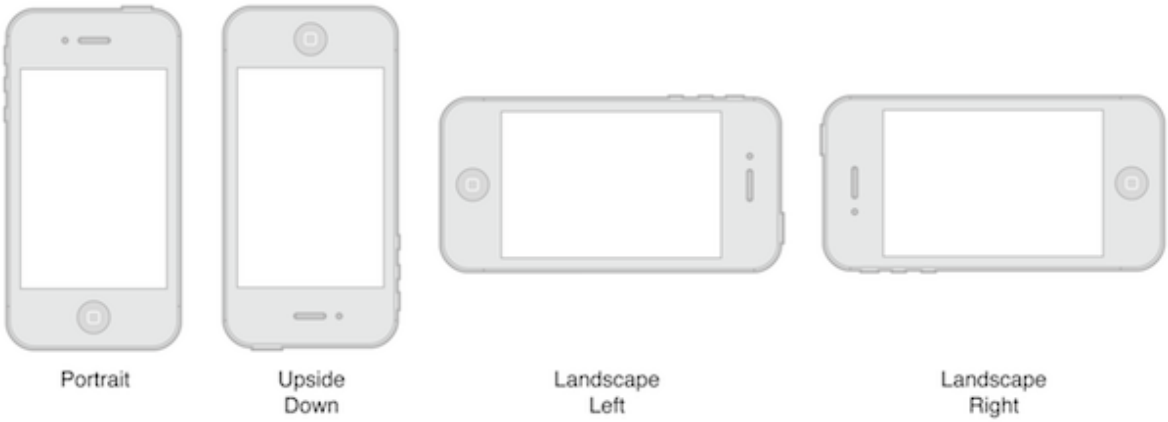
```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    return [[[self.fetchedResultsController sections]
objectAtIndex:section] numberOfObjects];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
```

```
    Trip *trip = [self.fetchedResultsController
objectAtIndexPath:indexPath];
    ...
    return cell;
}
```

4.5 Autorotace

Akcelerometr v iOS zařízeních umožňuje detekovat změny polohy. V základním nastavení podporují aplikace všechny čtyři možné orientace – portrétovou a krajinnou. Při změně orientace systém pošle notifikaci, aby bylo možné na změny reagovat a ve většině případů systém sám vykoná potřebné změny.



Obrázek 14 – Podporované orientace v systému iOS, zdroj [2]

Když se změní orientace, velikost okna hlavního kontroleru se automaticky upraví na nový rozměr a tento rozměr je dále propagován skrze hierarchii objektů. Na objektech je možné definovat, jak se mají chovat při změně orientace. V iOS 5 a nižších verzích se tento systém nazývá „struts and springs“, v iOS 6 a vyšších verzích je možno využít funkce zvané „Autolayout“, která je velmi propracovaná a umožňuje téměř jakékoliv nastavení.

Mohou nastat situace, kdy je nutno základní chování rotací upravit. Je možné ovlivnit následující funkce:

- definovat jaké orientace bude aplikace podporovat,
- nastavit jak bude probíhat animace na obrazovce během změny orientace.

Podporované orientace lze definovat v souboru Info.plist, který je konfiguračním souborem pro celou aplikaci. Pro podporu všech orientací je potřeba definovat slovník s klíčem „UISupportedInterfaceOrientation“, jehož hodnotou je pole obsahující všechny čtyři typy orientace.

Při změně orientace v iOS 5 je právě zobrazený kontroler upozorněn na změnu orientace v několika krocích zasíláním různých zpráv. V implementacích příslušných metod je možné reagovat na změnu, např. skrytím, nebo zobrazením grafických prvků nebo změnou jejich velikosti. Při implementaci těchto metod není vhodné spouštět kód, který se bude

vykonávat dlouho, protože by animace nebyla plynulá a navíc by mohlo dojít k dočasnému zablokování celého uživatelského rozhraní. Příkladem může být kompletní nahrazení grafických prvků jinými. Následuje sekvence událostí, které nastanou při změně orientace.

1. Je poslána zpráva willRotateToInterfaceOrientation:duration:.
2. Je poslána zpráva viewWillLayoutSubviews, v tuto chvíli je již nastaven nový rozměr.
3. Je poslána zpráva willAnimateRotationToInterfaceOrientation:duration:. Tato metoda je volána z animačního bloku, takže veškeré změny vlastností, které lze animovat, budou animovány.
4. Nyní je spuštěna samotná animace.
5. Jako poslední je poslána zpráva didRotateFromInterfaceOrientation:.

Ještě před tím než nastane proces rotace, je kontroleru zaslána zpráva, zda má změna orientace proběhnout. Implementace, která podporuje pouze portrét mód, vypadá následovně.

```
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)inte
rfaceOrientation
{
    return interfaceOrientation == UIInterfaceOrientationPortrait;
}
```

FishDiary podporuje pouze portrét mód. Avšak při psaní poznámek k vycházce je potřeba povolit i krajinný mód, z důvodu větší uživatelské přívětivosti. V mobilních aplikacích je potřeba minimalizovat uživatelský vstup z klávesnice, protože psaní na malých zařízeních není pohodlné, bohužel to nelze úplně vyloučit. Kvůli odlišné implementaci podpory rotací v iOS 6 je zapotřebí udělat několik úprav.

V konfiguračním souboru je nutné povolit všechny orientace, i když se budou využívat jen v jediném kontroleru. Dále se musí vytvořit potomek UINavigationControlleru, který se bude v celé aplikaci používat, a přetížit metodu shouldAutorotate.

```
-(BOOL)shouldAutorotate
{
    return NO;
}
```

Jako poslední krok je v kontroleru, který spravuje psaní poznámek, povolit všechny orientace.

```
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)toIn
terfaceOrientation
{
    return YES;
}
```

Při animaci velmi komplexní hierarchie grafických prvků se může stát, že animace nebude plynulá. V tomto případě se doporučuje během animace nahradit obrazovku obrázkem, který bude totožný se vzhledem obrazovky po rotaci, a teprve v momentě, kdy je rotace kompletní, uspořádat prvky na obrazovce a odebrat obrázek. Tím se nejnáročnější část kódu spustí až po proběhnutí animace, která bude plynulá, a uživatel si ničeho nevšimne.

4.6 Gesta

Kvůli dotykovým displejům v mobilních zařízeních je nutná podpora gest a jejich obsluha. Systém iOS nabízí několik předdefinovaných gest, které je možno využívat:

- UITapGestureRecognizer – nastává při jednom kliknutí na displej,
- UIPinchGestureRecognizer – nastává při pohybu dvou prstů po displeji, využívá se např. při zvětšování/zmenšování obrázků,
- UIPanGestureRecognizer – nastává při tažení jedním prstem,
- UISwipeGestureRecognizer – nastává při rychlém potažení jedním prstem po obrazovce,
- UIRotationGestureRecognizer – nastává při kruhovém pohybu dvou prstů,
- UILongPressGestureRecognizer – nastává při jednom kliknutí a držení prstu na obrazovce.

Gesta lze ještě rozdělit do dvou skupin:

- samostatné – nastane pouze jedenkrát (např. UITapGestureRecognizer),
- nepřetržité – jejich trvání probíhá v časovém intervalu (např. UIPinchGestureRecognizer).

Každý interaktivní vestavěný grafický prvek podporuje určitá gesta. Mimo to je možné další gesta přidávat. To se hodí zejména v případě, kdy je potřeba vytvořit vlastní grafický prvek.

V aplikaci FishDiary je možné přidat rozpoznávač gest např. k buňce tabulky, která zobrazuje na detailu vycházky popis. Jelikož popis může být dlouhý text a čtení v malé buňce by nebylo pohodlné, je vhodné přidat buňce gesto UIPinchGestureRecognizer, jež umožní uživateli pohybem dvou prstů zvětšit výšku buňky.

Buňka obsahující popis zobrazuje text v grafickém prvku UITextView a v buňce se mohou nacházet i další grafické prvky. Kdyby se rozpoznávač gest přidal k UITextView a uživatel se dotkl prstem na kraji buňky, gesto by nebylo rozpoznáno. Stejně tak, když by se rozpoznávač přidal na samotnou buňku, gesto by neprošlo, pokud by se uživatel dotkl prstem na UITextView. Jako efektivní řešení tohoto problému může být přidání rozpoznávače gest na celou tabulku, tedy UITableView.

```
UIPinchGestureRecognizer *pinchGestureRecognizer =
[[UIPinchGestureRecognizer alloc] initWithTarget:self
action:@selector(handlePinch:)];
[self.tableView addGestureRecognizer:pinchGestureRecognizer];
```

Když je po tomto nastavení rozpoznáno gesto „pinch“ na UITableView, zavolá se metoda handlePinch: a jako parametr jí bude předán objekt typu UIPinchGestureRecognizer, který v sobě zapouzdřuje veškeré užitečné informace o průběhu gesta, např. souřadnice, kde se uživatel dotkl displeje, nebo měřítko, pomocí něhož je možné vynásobit aktuální výšku buňky k dosažení větší nebo menší hodnoty. Rozpoznaná gesta se mohou nacházet v různých stavech, aktuální stav je součástí objektu v parametru a využívá se především u nepřetržitých gest. V metodě handlePinch: je potřeba rozlišit tyto stavy:

- UIGestureRecognizerStateBegan – gesto začalo, vykonává se první akce; je zapotřebí nastavit instanční proměnné, které uchovávají počáteční výšku buňky, cestu k buňce, a nakonec nastavit novou výšku,
- UIGestureRecognizerStateChanged – tento stav značí, že uživatel stále hýbe prsty a gesto pokračuje dál, nastaví se tedy nová výška,
- UIGestureRecognizerStateEnded a UIGestureRecognizerStateCancelled – v tomto stavu stačí vynulovat instanční proměnné, protože gesto už skončilo, uživatel zvedl prsty z displeje.

```
- (void) handlePinch:(UIPinchGestureRecognizer *)pinchRecognizer
{
    if (pinchRecognizer.state == UIGestureRecognizerStateBegan) {
        CGPoint pinchLocation = [pinchRecognizer
locationInView:self.tableView];
        _pinchedIndexPath = [self.tableView
indexPathForRowAtPoint:pinchLocation];
        _initialPinchHeight = [self tableView:self.tableView
heightForRowAtIndexPath:_pinchedIndexPath];
         [self updateForPinchScale:pinchRecognizer.scale
atIndexPath:_pinchedIndexPath];
    } else if (pinchRecognizer.state == UIGestureRecognizerStateChanged)
{
         [self updateForPinchScale:pinchRecognizer.scale
atIndexPath:_pinchedIndexPath];
    }
}
```

Další využití gest se naskýtá v případě, když chce uživatel vybrat fotografii, která bude použita jako náhled ve výpisu všech vycházek. Obrázky jsou zobrazeny v UIButtonu, aby bylo možné při jednoduchém stisknutí přejít do galerie, která umožňuje prohlížení obrázků v plné velikosti. UITapGestureRecognizer již nelze použít, ale lze použít UILongPressGestureRecognizer. Ten umožňuje nastavit časovou prodlevu, po které se má vykonat akce s gestem spojená. Následující kód přidá UILongPressGestureRecognizer k UIButtonu a při rozpoznání zavolá metodu showSetAsCoverAlertView:, které se jako parametr předá objekt typu UILongPressGestureRecognizer.

```
-(void)showSetAsCoverAlertView:(UILongPressGestureRecognizer *)gesture
{
    if (gesture.state == UIGestureRecognizerStateBegan) {
        // process
    }
}
```

4.7 Práce s obrázky

Aplikace FishDiary umožňuje uživateli ukládat k jednotlivým vycházkám fotografie nebo obrázky a jeden z nich je vždy použit jako náhled reprezentující vycházku v celkovém výpisu. Při zobrazení obrázku je nutné ho celý načíst do paměti. Klasická fotografie z iOS zařízení má velké rozlišení, její načtení by zabralo velké množství paměti, která je v mobilních přístrojích omezená, a je potřeba jí šetřit. Nebylo by optimální načítat pro náhled fotografii v původním rozlišení. Ve výpisu jich může být na obrazovce zároveň pět nebo šest a použitá paměť by rostla. Vždy je vhodné načíst co nejmenší obrázek, který je potřeba pro zobrazení. Nabízí se několik možností, jak tento problém vyřešit.

První způsob je uložení původní fotografie v nativním rozlišení, při načtení zmenšit rozlišení a jako náhled použít fotografii zmenšenou. Tento způsob není vhodný hned ze dvou důvodů. Za prvé je stejně nutné načíst celou původní fotografii do paměti a zmenšit ji a za druhé zmenšení fotografie není zrovna levná operace.

Lepším řešením je ukládat fotografii v původním rozlišení pro zobrazení v prohlížeči a zároveň si obrázek uložit i v menším rozlišení vhodném pro náhled. Při pořízení je fotografie stejně načtena do paměti a v tomto okamžiku je vhodné ji zmenšit a uložit si i menší verzi. Vše je nutné provést na pozadí, aby tato drahá operace neblokovala hlavní vlákno, a tudíž i uživatelské rozhraní.

4.8 Získání GPS souřadnic

K navigačním účelům je v iOS k dispozici celý framework Core Location. Při práci s polohou je potřeba si uvědomit, že tyto operace spotřebovávají velké množství energie a nesprávné použití by mohlo mít vliv na výdrž baterie. Pro určení polohy je totiž nutné komunikovat se všemi vysílači v okolí, přístupnými Wi-Fi body nebo s GPS modulem.

Ne každé zařízení je schopno rozpoznávat polohu. Pokud se vyvíjí aplikace, jejíž funkčnost je závislá na poloze, je vhodné tuto skutečnost definovat v konfiguračním souboru. Zamezí se tím možnost stažení aplikace na starší zařízení, které práci s polohou nepodporují.

Aktuální polohu zařízení lze získávat dvěma způsoby:

- standardní změna polohy – dostupný pro všechny verze iOS,
- výrazná změna polohy – tento způsob je dostupný pro iOS 4 a vyšší.

Před použitím lokačních služeb je nutné ověřit, zda jsou služby dostupné. Vede k tomu několik důvodů:

- uživatel může mít lokační služby zakázané v nastavení,
- uživatel může lokační služby zakázat pro konkrétní aplikaci,
- zařízení může být v letadlovém režimu nebo nemusí mít dostatek energie.

Doporučuje se tedy hned na začátku zavolat třídní metodu locationServicesEnabled třídy CLLocationManager, která vrací datový typ boolean. Pokud se touto metodou neověří dostupnost lokačních služeb, uživateli se zobrazí upozornění, zda nechce lokační služby aktivovat. Avšak většinou má uživatel tyto služby zablokované z nějakého konkrétního důvodu a tak nemusí být vždy uživatelsky přívětivé toto upozornění zobrazovat.

K používání lokačních služeb, ať už standardních nebo výrazných změn polohy, je zapotřebí vytvořit instanci třídy CLLocationManager. I když může v aplikaci existovat více instancí této třídy, doporučuje se vytvořit vždy jen jednu a tu používat všude. Pokud by existovalo více instancí, mohlo by to vést k soupeření o citlivé zdroje, jako je GPS modul a jiné. K tomuto účelu je vhodné použít návrhový vzor „singleton“, který zaručí, že bude vytvořena pouze jedna instance konkrétní třídy v celé aplikaci.

```
@property (strong, nonatomic) CLLocationManager *sharedManager;
+ (CLLocationManager *)sharedManager
{
    static dispatch_once_t once;
    dispatch_once(&once, ^{
        _sharedManager = [[self alloc] init];
    });

    return _sharedManager;
}
```

Tento způsob vytváření singleton instancí je doporučovaný Applem. Funkce dispatch_once zajistí, že blok, který jí je předán ve druhém parametru, bude spuštěn pouze jednou za celý běh aplikace.

Po vytvoření instance se musí provést nastavení tohoto objektu, a to zejména vlastnosti desiredAccuracy, distanceFilter a delegate (bude dostávat aktualizace o změnách polohy). Zatímco distanceFilter pouze udává o kolik metrů musí zařízení horizontálně změnit polohu, aby byla zaslána aktualizace aplikaci, tak desiredAccuracy je více komplexní. Je možné ji nastavit na několik konstant:

- kCLLocationAccuracyBestForNavigation – k určení polohy se použijí všechny dostupné možnosti kombinované s dalšími údaji, jedná se tedy o nejpřesnější možné určení polohy, je však určeno pouze pro navigace a pro použití při zapnutém napájení zařízení,
- kCLLocationAccuracyBest – nejvyšší přesnost,
- kCLLocationAccuracyNearestTenMeters – +- 10 metrů,

- kCLLocationAccuracyHundredMeters – +- 100 metrů,
- kCLLocationAccuracyKilometer – +- 1 kilometr,
- kCLLocationAccuracyThreeKilometers – +- 3 kilometry.

Je zřejmé, že čím vyšší přesnost je nastavena, tím vyšší bude spotřeba energie. Proto je důležité vždy promyslet, jak velká přesnost je pro aplikaci potřeba.

Po nastavení objektu CLLocationManager se může spustit samotné sledování změn. A to posláním instanční zprávy startUpdatingLocation. Jakmile budou dostupná první data, bude upozorněn objekt nastavený jako delegát.

```
- (void)locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation fromLocation:(CLLocation
*)oldLocation
{
    self.trip.latitude = [NSNumber
numberWithFloat:newLocation.coordinate.latitude];
    self.trip.longitude = [NSNumber
numberWithFloat:newLocation.coordinate.longitude];
    [self.locationManager stopUpdatingLocation];
}
```

V aplikaci FishDiary je zapotřebí získat GPS souřadnice pouze jednou pro konkrétní vycházku nebo revír a ty uložit. Proto v metodě delegáta, která je volána, když nastane změna polohy, se zavolá metoda stopUpdating. Pokud by se tato metoda nezavolala, lokační služby by neustále sledovaly polohu a vedlo by to k velké spotřebě energie.

Mimo jiné lze sledovat, když se zařízení přiblíží k určitému bodu nebo regionu. To lze použít např. v případě, když aplikace upozorňuje na významná místa v okolí, kde se zařízení právě nachází.

Doručování změn polohy je možné, i když aplikace běží v pozadí. Toho se využívá např. v aplikacích, které monitorují cestu, po níž se zařízení pohybovalo.

Nakonec je uvedeno pár tipů na co si dávat pozor při práci s lokačními službami kvůli úspoře energie.

- Vypnout lokační služby, když se nepoužívají.
- Používat výrazné změny polohy místo standardních.
- Používat co nejmenší nutnou přesnost určování polohy.

4.9 Zobrazení vycházek a revírů na mapě

Základní implementace map do iOS aplikací je velmi jednoduchá, v Xcode stačí vložit MapViewController, který má jako hlavní okno vestavěnou mapu, objekt typu MKMapView. Tímto krokem je zaručeno, že mapu bude možné zobrazit, pohybovat se

v ní, přiblížit a oddálit. Do mapy je však možné přidávat i vlastní informace a ty zobrazovat různými způsoby.

Při prvním zobrazení mapy je důležité nastavit, jaká část bude viditelná a jak moc bude přiblížená. K tomu slouží vlastnost region třídy MKMapView, která obsahuje strukturu MKCoordinateRegion. Nastavení viditelné části mapy a přiblížení v aplikaci FishDiary:

```
MKCoordinateRegion region;
region.center.latitude = self.customAnnotation.coordinate.latitude;
region.center.longitude = self.customAnnotation.coordinate.longitude;
region.span.latitudeDelta = SPAN_LAT_DELTA;
region.span.longitudeDelta = SPAN_LON_DELTA;
[self.mapView setRegion:region animated:YES];
```

Nejzajímavější částí je „span“ – jedná se o analogii k výšce a šířce obdélníku, ale je specifikován stupni, minutami a vteřinami.

Na mapě je také možné zobrazit aktuální polohu zařízení a to pomocí vlastnosti showsUserLocation, která je typu boolean.

V aplikace FishDiary si uživatelé ukládají dva typy dat, která jsou vhodná zobrazovat na mapě. Jsou to jednotlivé vycházky a revíry, vycházka určuje konkrétní místo na revíru. K zobrazování vlastních bodů v mapě se používají tzv. anotace. Naopak k vyznačení celé oblasti v mapě nebo např. cesty se používají tzv. překrytí.

Aby bylo možné vycházky a revíry na mapě zobrazit, musí aplikace poskytnout dva typy objektů:

- objekt, který přijímá MKAnnotation protokol a zapouzdřuje v sobě data konkrétní anotace – anotační objekt,
- potomka třídy MKAnnotationView (nebo tuto třídu samotnou), která bude použita pro samotné vykreslení na mapě – určuje vizuální podobu.

Anotační objekty jsou malými datovými objekty, které v sobě zapouzdřují data potřebná k zobrazení na mapě, jako jsou GPS souřadnice nebo popisek. Jelikož jsou anotační objekty podmíněny pouze přijetím protokolu, je možné použít jakoukoliv existující třídu v aplikaci, která nese potřebná data. Před zobrazením na mapě je nutné učinit následující kroky.

1. Vytvořit anotační objekt.
2. Definovat MKAnnotationView.
3. Implementovat metodu mapView:viewForAnnotation: v delegátovy mapy.
4. Přidat anotační objekt k mapě pomocí metody addAnnotation: nebo addAnnotations:.

Implementace metody mapView:viewForAnnotation: v aplikaci FishDiary vypadá následovně:

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView
viewForAnnotation:(id<MKAnnotation>)annotation

{
    if ([annotation isKindOfClass:[MKUserLocation class]]) {
        return nil;
    }
    static NSString* TripAnnotationIdentifier =
@"tripsAndLocAnnotationsIdentifier";
    MKPinAnnotationView* pinView = (MKPinAnnotationView *)
    [mapView
dequeueReusableAnnotationViewWithIdentifier:TripAnnotationIdentifier];
    if (!pinView) {
        pinView = [[MKPinAnnotationView alloc]
initWithAnnotation:annotation reuseIdentifier:TripAnnotationIdentifier];
        pinView.animatesDrop = YES;
        pinView.canShowCallout = YES;
        UIButton* rightButton = [UIButton
buttonWithType:UIButtonTypeDetailDisclosure];
        pinView.rightCalloutAccessoryView = rightButton;
    } else {
        pinView.annotation = annotation;
    }
    if ([pinView.annotation isKindOfClass:[TripAnnotation class]]) {
        pinView.pinColor = MKPinAnnotationColorRed;
    } else if ([pinView.annotation isKindOfClass:[WaterLocationAnnotation
class]]) {
        pinView.pinColor = MKPinAnnotationColorGreen;
    }
    return pinView;
}
```

1. Mapa zobrazuje aktuální polohu zařízení také pomocí anotačního objektu, proto je potřeba v prvním kroku zjistit, zda se nejedná o aktuální polohu. Pokud ano, vrátí se nil, čímž se systému dá najevo, že jde právě o aktuální polohu.

2. Jako další krok je potřeba získat objekt typu MKAnnotationView, který tato metoda vrací. Mapa může zobrazovat tisíce různých anotačních objektů. Kdyby se měly alokovat všechny tyto objekty najednou, využití paměti by nebylo enormní. Proto systém alokuje pouze objekty, které jsou právě viditelné na obrazovce, a ty jsou využívány pořád dokola, jen se jim mění parametry. Toho se dosáhne pomocí metody dequeueReusableAnnotationViewWithIdentifier:TripAnnotationIdentifier. Pouze v případě, kdy tato metoda vrátí nil, je nutné alokovat nový objekt.

3. Jako poslední je potřeba rozlišit mezi vycházkou a revírem. K tomu lze využít introspekci. Vycházce se poté nastaví červená barva špendlíku a revíru zelená.

Ještě si lze všimnout nastavení vlastnosti rightCalloutAccesoryView na UIButton. Toto nastavení umožní přechod z konkrétního bodu na mapě na detail vycházky nebo revíru.

4.10 Lokalizace

Největší trh s iOS zařízeními nabízejí Spojené státy a další anglicky mluvící země. Pokud není aplikace cílená pro český trh, vždy je vhodné vydat verzi v angličtině. Zde je seznam prvků, které je nutné zapojit do lokalizace:

- prvky navržené jako UI v grafickém editoru musí být schopné správně zobrazovat různé délky řetězců,
- statické texty ve zdrojovém kódu musí být přeloženy,
- ikony a jiné grafické prvky musí být lokalizovány,
- zvukové soubory (zejména mluvené slovo) musí být nahrány ve všech jazykových mutacích,
- dynamické prvky generované aplikací, jako jsou data, časy, čísla nebo měny, musí být formátovány ve správné lokalizaci,
- kód, který zpracovává text, musí být přizpůsoben pro práci s různými lokalizacemi,
- tabulková data musí být řazena podle správné lokalizace.

iOS má velkou podporu internacionalizace a lokalizace všech výše zmíněných prvků. Někdy to však může znamenat mnoho práce navíc, která se nakonec nemusí vyplatit. Proto je důležité při prvotní analýze rozmyslet to, do kolika jazyků aplikaci lokalizovat.

Lokalizace statických textů ve zdrojovém kódu se provádí pomocí metody NSLocalizedString, která přijímá dva parametry. Prvním je klíč, který určuje lokalizovaný řetězec, a druhým komentář, který specifikuje kontext, ve kterém je řetězec použit, a je užitečný zejména pro překladatele. Samotné překlady se pak nacházejí v tzv. „strings“ souborech, jejichž formát je následující:

```
"SEND_BY_EMAIL" = "Poslat e-mailem";
"MAP" = "Mapa";
"PHOTO_LIBRARY" = "Vybrat z uložených";
```

Problém nastává s lokalizací řetězců, které jsou obsaženy v UI v grafickém editoru, tzv. „storyboardech“. V iOS 5 je téměř nutné mít pro každou jazykovou mutaci zvláštní „storyboard“. Z něho je pak možné pomocí různých nástrojů vygenerovat soubor s řetězci. Avšak při jakékoliv změně rozložení prvků je nutné upravit všechny „storyboardy“, což může být opravdu hodně práce. Toto je i jeden z důvodů proč byl v iOS 6 zaveden „AutoLayout“. Ten umožňuje dokonalé nastavení rozmístění prvků bez nutnosti uvádět pevné rozměry. Tím se zaručí, že vše bude zobrazeno korektně i s variabilní délkou řetězců.

4.11 Tipy pro optimalizaci

Tato kapitola má za úkol shrnout pár tipů, jak se vyhnout problémům s výkonem aplikací všeobecně.

ARC – správa paměti

ARC eliminuje většinu úniků paměti. Bez ARC by se např. v tomto případě musel objekt manuálně uvolnit a na to se může snadno zapomenout.

```
UIView *view = [[UIView alloc] init];
[self addSubview:view];
...
 [view release];
```

ARC obstará uvolnění úplně samo a programátor se může věnovat důležitějším problémům. Nicméně pořád je potřeba si dávat pozor při práci s bloky, „retain cycles“ nebo s frameworky, které poskytují pouze C API.

ReuseIdentifier

Často se stává, že programátor zapomene na tento důležitý identifikátor. Je nutné ho nastavit v editoru grafického rozhraní, poté i ve zdrojovém kódu a používat „deque“ metody u patřičných kolekcí. Bez tohoto přístupu by se alokoval vždy nový objekt (např. pro každý řádek tabulky). Mimo jiné by se spotřebovávalo zbytečně hodně paměti i času procesoru a mohlo by to vést i k narušení plynulosti animací. ReuseIdentifier se týká především UITableView, UICollectionView a anotačních objektů.

Hlavní vlákno

Jak již bylo několikrát řečeno, časově náročný kód se nesmí nikdy spouštět na hlavním vlákně, protože by blokoval uživatelské rozhraní a uživatel by se mohl domnívat, že došlo k pádu aplikace.

Především kód komunikující přes síť je nutné vždy spouštět v pozadí, protože tato komunikace může být extrémně pomalá, zvláště v mobilních zařízeních. Doporučuje se použít GCD nebo NSOperations a NSOperationQueues a spustit dlouho trvající úkol na pozadí. Po dokončení je možné na hlavním vlákně zavolat kód, který aktualizuje uživatelské rozhraní. GCD spolu s bloky tuto práci velmi usnadňuje.

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,
0), ^{
    // časově náročný úkol
    dispatch_async(dispatch_get_main_queue(), ^{
        // aktualizovat UI na hlavním vlákně
    });
});
```

Adekvátní velikost obrázků

Vždy je vhodné zobrazovat obrázek v co nejmenších možných rozměrech, rozlišení. Obrázky je totiž nutné načíst do paměti, kde zabírají hodně místa. Pokud je tedy potřeba zobrazit pouze malý náhled obrázku, je dobrým zvykem použít stejně tak malý obrázek.

Změna velikosti obrázku za běhu aplikace je náročný úkol a jeho vykonání chvíli trvá. Je proto vhodné obrázek zmenšit již při stažení nebo importu nové fotografie a samozřejmě na pozadí, neblokovat hlavní vlákno.

Vhodná kolekce pro uchovávání dat

Základem efektivity aplikací je vybrat správnou kolekci pro ukládání dat.

- Pole – řazené kolekce, rychlé vyhledávání podle indexu, ale pomalé vyhledávání podle hodnoty a zároveň pomalé vkládání/mazání hodnot.
- Slovník – páry typu klíč/hodnota, rychlé vyhledávání pomocí klíče.
- Množina – neřazená kolekce, rychlé vyhledávání podle hodnoty, rychlé vkládání/mazání hodnot.

Lazy loading

Obrazovka se může skládat z mnoha různých grafických prvků, které nemusí být hned všechny viditelné. Některé se zobrazí až po určité vykonané akci. Načíst všechny prvky hned při prvním zobrazení nemusí být efektivní, protože může nastat situace, kdy se daný prvek nezobrazí vůbec. Proto se v iOS velice často používá načítání objektů až ve chvíli, kdy jsou skutečně potřeba.

Nicméně ne vždy je tento způsob tím nejlepším. Vždy záleží na konkrétní situaci. Kdyby zobrazení náročného grafického prvku bylo součástí složitější animace, mohlo by mít toto zpožděné načtení vliv na plynulost animace.

Recyklace „drahých“ objektů

Inicializace objektů, jako jsou např. NSCalendar nebo NSDateFormatter, je časově náročná, avšak při parsování datových zdrojů (XML, JSON) je jejich použití téměř nutné. Z tohoto důvodu je vhodné nevytvářet vždy nové instance, ale zvolit jiný přístup. Je možné si objekt uložit do vlastnosti třídy nebo využít statickou proměnnou. Při použití statické proměnné bude objekt v paměti po celou dobu existence aplikace, což také nemusí být vždy optimální. Řešení vlastností ve spojení s „lazy loadingem“ bude ve většině případů efektivnější.

NSDateFormatter

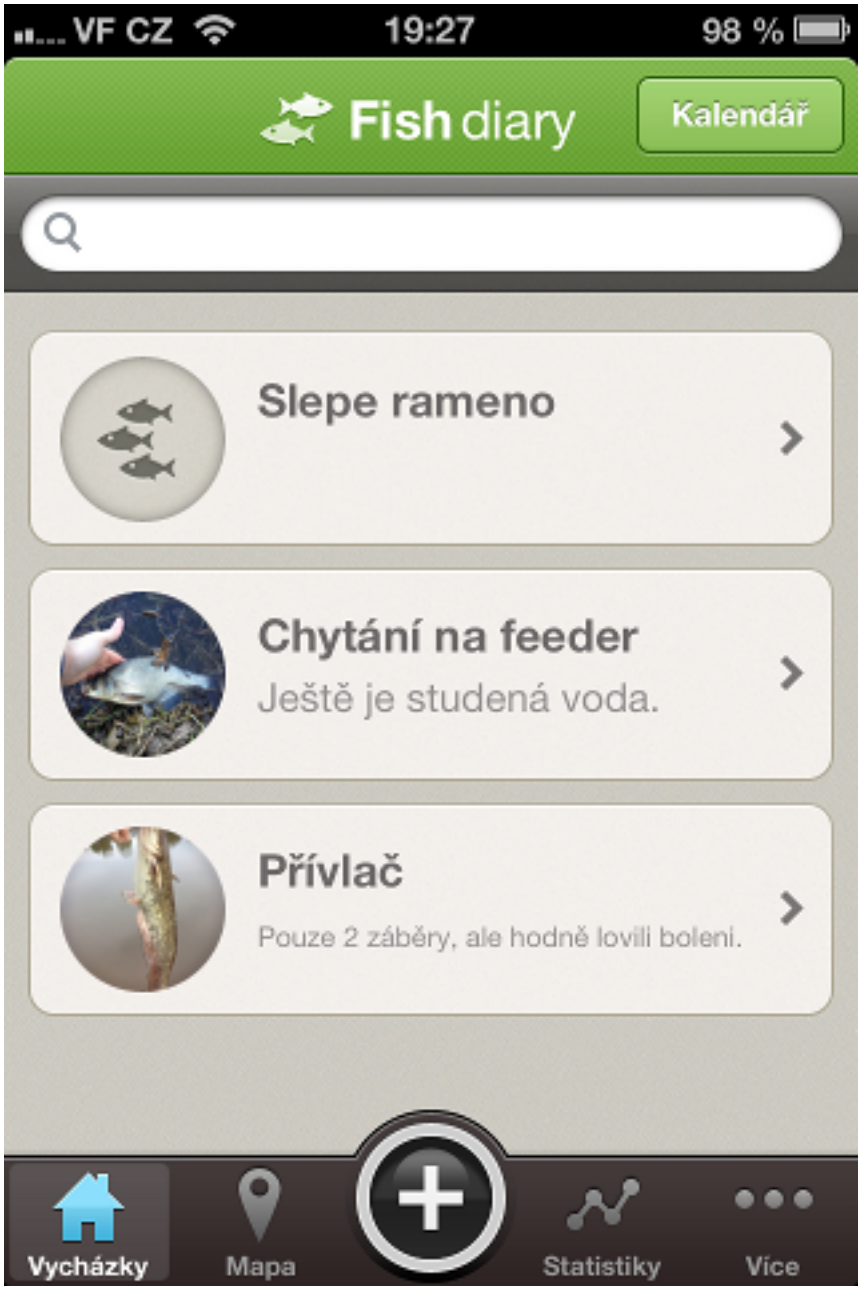
Pokud je potřeba parsovat velké množství dat, je nutné tento objekt recyklovat. Když je zapotřebí ještě větší rychlost je možné použít pro parsování pouze jazyk C.

V iOS je vždy nejvýhodnější pracovat s časovým razítkem namísto řetězce, který udává datum. Vytvoření objektu NSDate z časového razítka může být rychlejší než parsování v jazyce C.

```
- (NSDate *)dateFromTimestamp:(NSTimeInterval)timestamp
{
    return [NSDate dateWithTimeIntervalSince1970:timestamp];
}
```

4.12 Dokončení

Na následujícím screenshotu je ukázána úvodní obrazovka aplikace FishDiary s již uloženými vycházkami.



Obrázek 15 – Úvodní obrazovka aplikace FishDiary

Již během programování aplikace je vhodné se věnovat marketingu a propagaci. Základní kroky, které by měla každá mobilní aplikace následovat, jsou popsány v následující kapitole.

5 Marketing iOS aplikací

Z důvodu propagace aplikace FishDiary bylo rozhodnuto, že k uvedení na trh je nutné vytvořit webovou stránku, vytvořit profily aplikace na sociálních sítích, vytvořit tiskový balíček a oslovit potencionální cílovou skupinu uživatelů. Všechny tyto kroky jsou podrobněji popsány v této kapitole.

Většina aplikací v AppStore je vyvíjena za účelem zisku finančních prostředků. Z tohoto důvodu není samotný vývoj jedinou věcí, která se musí řešit, pokud má být aplikace opravdu úspěšná. Když se podaří vyvinout skvělou aplikaci, ale nikdo o ní nebude vědět, tak je to zbytečná práce. Marketing je tudíž nedílnou součástí celého vývoje a měl by být i součástí úplně prvních kroků analýzy ještě před samotným programováním.

První otázka, kterou si je třeba položit, je kdo bude aplikaci používat. Je tedy zapotřebí určit cílovou skupinu uživatelů a podle toho navrhnout nejen funkce, ale i vzhled aplikace. Je jasné, že lidé kolem dvaceti let mají jiný styl vnímání než lidé nad čtyřicet let. Nejlepší názory a připomínky lze vždy získat přímo od uživatelů. Když je známa cílová skupina, je možné na internetu najít patřičná diskuzní fóra nebo weby a spojit se s lidmi, kteří jsou potencionálními uživateli budoucí aplikace, a zeptat se jich na jejich názor, zkušenosti či požadavky na funkčnost. Samotný vývojář většinou přemýšlí úplně jinak než „obyčejný“ uživatel a znát názory uživatelů ještě před návrhem a implementací může být neocenitelné.

Další neméně důležitou součástí je průzkum trhu, je potřeba udělat rešerši konkurence a analyzovat jejich produkty. V současné době se v AppStore nachází více než 750 000 tisíc aplikací a každým dnem jich spoustu přibývá, těžko tak vývojář přijde s unikátní aplikací, i když i to se může stát. To však vůbec nevadí, každou aplikaci lze vylepšit, třeba přidáním nových funkcí, dokonalým uživatelským prostředím nebo nejlépe kombinací obou možností. Po vyhledání všech nejpoužívanějších aplikací stejného nebo podobného typu se vyplatí prozkoumat každou zvlášť přímo v AppStore. Mezi kritéria, na která je potřeba se zaměřit, patří celkové hodnocení aplikace, funkce, které uživateli nabízí, velikost, kategorie, do které je zařazena, nebo obrázky z aplikace. Avšak jedním z nejdůležitějších kritérií je názor uživatelů, který vývojář určitě musí projít, vyhodnotit a zapracovat do analýzy vlastní aplikace. Samozřejmostí by mělo být stažení/koupě aplikace a vlastní otestování.

Pokud se i v této fázi jeví vlastní aplikace odlišně a lépe než již existující, nic nebrání ve vývoji.

5.1 Webová stránka

Již v začátcích vývoje je vhodné vytvořit jednoduchou webovou stránku, prezentující aplikaci, na které lze zveřejňovat průběh vývoje, nahrávat fotky uživatelského rozhraní – postačí i hotový grafický návrh nebo pouze ukázka nějaké zajímavé části. Takto je možné si získat první fanoušky, kteří budou čekat na vydání aplikace a budou prvními zákazníky.

Tímto způsobem je však možná i komunikace s uživateli během samotného vývoje a možnost učení se z poskytnutých postřehů.

Technické řešení této stránky se může obejít bez nutných finančních prostředků. Lze využít „open-source“ CMS systémy, jako je Wordpress nebo další. Na Wordpress je dostupných zdarma nebo za pár dolarů na internetu mnoho šablon určených právě pro mobilní aplikace.

Pro aplikaci FishDiary byla vytvořena webové stránka[3], která je poháněna open-source CMS systémem Wordpress na nějž byla nasazena šablona, kterou lze stáhnout zdarma a je přímo určena pro propagaci mobilních aplikací.

5.2 Sociální sítě

Dalším způsobem, jak propagovat aplikaci zdarma, jsou sociální sítě. Vždy se vyplatí vytvořit stránku na Twitteru i na Facebooku a zveřejňovat zde veškeré důležité milníky ve vývoji aplikace. Avšak je potřeba si dávat pozor, aby každý příspěvek nesl hodnotné informace pro odběratele a aby příspěvky nebyly příliš časté. Mohlo by to potom vést k obtěžování odběratelů.

Sociální sítě jsou zároveň jedním z nejlepších kanálů, jak komunikovat s uživateli. Jak již bylo řečeno, vždy je důležité komunikovat se zákazníky a snažit se řešit jejich připomínky, vyplatí se to.

Vytvoření krátkého videozáznamu může být jedním z nejdůležitějších faktorů v představování aplikace uživatelům ještě před zveřejněním aplikace samotné. Na videu uživatelé nejlépe uvidí stěžejní funkce v akci. Toto už však není tak jednoduché jako tvorba webové stránky nebo založení stránek na sociálních sítích. Je nutné video vhodným způsobem natočit a lákavě sestříhat, což nemusí být v silách vývojáře. Kvalitně zpracovaný krátký videozáznam může aplikaci přidat velkou hodnotu.

Již během vývoje aplikace FishDiary byl vytvořen profil na Facebooku[4], na kterém byly prezentovány postřehy z vývoje a screenshoty z aplikace ještě před jejím vydáním.

Stejný princip byl aplikován na Twitteru, kde byl také založen profil aplikace FishDiary[5], na kterém byly zveřejňovány postřehy z vývoje a těsně před publikací aplikace v AppStore příspěvky gradovaly.

5.3 Internetová fóra

Již při prvotní marketingové analýze by měl vývojář najít patřičná internetová fóra, zaregistrovat se a začít komunikovat s potencionálními uživateli. Je vhodné si uchovávat seznam vytvořených vláken, pravidelně se na ně vracet a komunikovat s přispěvovateli. Do příspěvků je také velmi prospěšné přidávat důležité milníky z vývoje.

[3] http://fishdiaryapp.com
[4] https://www.facebook.com/FishDiaryApp
[5] https://twitter.com/FishDiaryApp

Z důvodu propagace aplikace FishDiary směrem k cílové skupině budoucích potencionálních uživatelů, bylo vytvořeno několik vláken na rybářských fórech, kde lze diskutovat s uživateli a vyhodnocovat jejich cenné připomínky.

5.4 Specializované PR magazíny

Recenze aplikace ve specializovaných magazínech určitě nejsou k zahození. Jedním z nejlepších způsobů recenze je nalákat velké portály, které se zabývají Apple technologiemi, aby samy vytvořily recenzi nové aplikace. Toto však není lehké a povede se to pouze výjimečným aplikacím na trhu.

Některá média jsou ochotna zveřejnit krátké představení aplikace, ale většinou nemají prostředky na to psát recenzi sama. Proto se vyplatí sepsat krátké představení a rozeslat ho emailem co nejvíce magazínům. Některé se určitě chytnou a recenzi zveřejní. Právě k tomuto způsobu propagace aplikací slouží portál prMac[6]. Stačí se zaregistrovat, vybrat si vyhovující placený program a služba se sama postará o zveřejnění recenze na stovkách webů.

K propagaci aplikace FishDiary v PR magazínech bylo využito placené služby prMac, které se postarala o publikaci tiskového balíčku na více než 1000 specializovaných magazínech.

5.5 Promo kódy

Po schválení aplikace v AppStore je možné si v iTunesConnect vygenerovat tzv. promo kódy. Pro každou verzi aplikace je možné použít 50 promo kódů, které mají platnost čtyři týdny. Pomocí promo kódu je možné aplikaci z AppStore stáhnout zdarma, i když je placená. K marketingovým účelům jsou promo kódy nepostradatelné. Při zasílání recenze PR magazínům, jak je zmíněno v předchozí podkapitole, se vyplatí připojit i promo kód, aby si recenzent mohl aplikaci zdarma stáhnout a vyzkoušet. Případně lze vytvořit nějaké typy soutěží, kde první tři příčky získají promo kód.

5.6 Tiskový balíček

Jedná se o kolekci materiálů, která ulehčuje recenzentům psaní článků. Hlavní smysl tiskového balíčku je poskytnout recenzentům co nejpřehlednější souhrn informací, které mohou použít při psaní recenze a nemusí je nikde hledat. Většinou se vytváří „zip“ archiv, který obsahuje:

- popis aplikace a jejích vlastností,
- užitečné odkazy jako webovou stránku, twitter a facebook stránku nebo odkaz na videozáznam z aplikace,
- kontakty na vývojáře,

[6] Dostupný z http://prmac.com

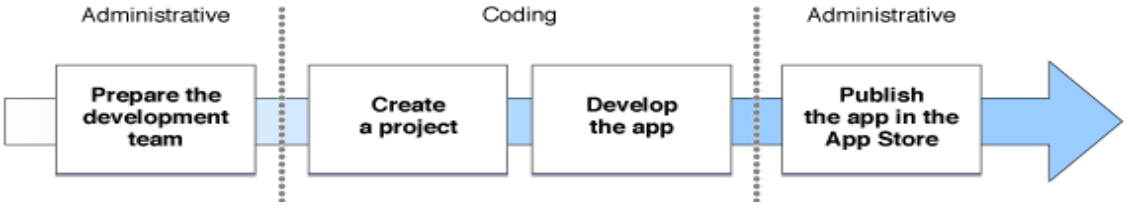
- grafiku – ikony, loga nebo fotky z aplikace.

Na tiskovém balíčku je potřeba si dát záležet, protože mnoho recenzentů pouze kopíruje text a publikuje ho na webu.

Po dokončení základního marketingu a propagace aplikace je nutné publikovat samotnou aplikaci v AppStore. Tento proces je popsán v následující kapitole.

6 Zveřejnění aplikace v AppStore

Vývoj iOS aplikací sestává z několika částí. Poslední částí je publikace aplikace v AppStore. Nutné kroky publikace aplikace jsou popsány v této kapitole.



Obrázek 16 – diagram vývoje iOS aplikací, zdroj [2]

Důležitou součástí je volba ceny aplikace. Již při počáteční analýze FishDiary bylo rozhodnuto, že aplikace bude placená. AppStore umožňuje jako nejnižší cenu zvolit $1. Z dostupných studií a analýz vyplývá, že iOS uživatelé jsou ochotni za kvalitu platit, a proto může být malá cena někdy kontraproduktivní. U kvalitní aplikace nemusí mít vývojaří strach zvolit vyšší cenu, proto byla počáteční cena FishDiary stanovena na $2, avšak v budoucnu se počítá s navýšením na $4.

6.1 Podepisování zdrojových kódů

Každá iOS aplikace, která má být nahrána do AppStore nebo spuštěna na fyzickém zařízení, musí být kryptograficky podepsána. Během vývoje aplikace je nutné ji testovat mimo simulátoru také na skutečném iOS přístroji. Některé funkce (gyroskop, iCloud) nelze v simulátoru testovat a nastávají i případy, kdy se aplikace chová odlišně v simulátoru nebo na skutečném zařízení. Proto je nevyhnutelné aplikaci testovat na skutečných přístrojích. iOS neumožňuje nahrávání vlastní aplikace do přístrojů, aniž by byla podepsaná pomocí vývojového certifikátu. Stejně tak nahrání do AppStore je podmíněno podepsáním aplikace pomocí distribučního certifikátu a zároveň není možné aplikace nahrát do AppStore bez fyzického iOS zařízení.

Níže je uveden seznam kroků, které jsou nutné k tomu, aby bylo možné aplikaci spustit na vlastním iOS zařízení. Všechny kroky lze provést v provizním portálu, do kterého se získá přístup při registraci do vývojového programu Applu.

Seznam kroků:

1. zaregistrování vlastních iOS zařízení do provizního portálu,
2. vytvoření explicitní ID aplikace,
3. použití vytvořené ID aplikace k vytvoření vývojového provizního profilu,
4. importování profilu do Xcode.

6.2 iTunes Connect

Před nahráním aplikace do AppStore je nutné vytvořit samotnou aplikaci ve webovém rozhraní iTunes Connect. Je potřeba vyplnit název aplikace, popis, klíčová slova a připojit potřebné grafické prvky, které jsou popsány v následující kapitole.

Základním jazykem je angličtina, ale všechna pole, včetně grafických prvků, lze lokalizovat do jazyků, které jsou podporovány AppStorem. Pokud aplikace podporuje angličtinu a němčinu a uživatel si ji zobrazí v německém AppStore, uvidí německou lokalizovanou verzi a naopak. Bohužel podpora AppStore pro češtinu zatím chybí (to neznamená, že aplikace nelze do češtiny lokalizovat).

6.3 Vyžadované grafické prvky

iOS aplikace jsou na displeji, tzv. „springboardu“, reprezentovány názvem, ale hlavní prvek určující konkrétní aplikaci je ikona, kterou musí povinně disponovat všechny iOS aplikace. Mimo ikony jsou vyžadovány i jiné grafické prvky shrnuté v následující tabulce včetně rozměrů.

Tabulka 2 – Rozměry ikon a obrázků v pixelech

Popis	Rozměr pro Iphone5	Rozměr pro iPhone s vysokým rozlišením	Rozměr pro iPhone	Rozměr pro iPad s vysokým rozlišením	Rozměr pro iPad
Ikona aplikace	114 x 114	114 x 114	57 x 57	144 x 144	72 x 72
Ikona zobrazovaná v AppStore	1024 x 1024	1024 x 1024	512 x 512	1024 x 1024	512 x 512
Obrázek zobrazený při spuštění aplikace	640 x 1136	640 x 960	320 x 480	1536 x 2008 (portrét) 2048 x 1496 (krajinný)	768 x 1004 (portrét) 1024 x 748 (krajinný)
Ikona pro spotlight a nastavení	58 x 58	58 x 58	29 x 29	100 x 100 (spotlight) 58 x 58 (nastavení)	50 x 50 (spotlight) 29 x 29 (nastavení)

Mimo tuto grafiku, která se definuje přímo v Xcode je nutné poskytnout alespoň jeden a nanejvýše pět obrázků z aplikace, opět v různých rozměrech pro různá zařízení (iPhone5/iPhone/iPad).

6.4 Nahrání aplikace do AppStore

Veškeré nahrávání probíhá automaticky z Xcode. Pokud je aplikace připravená v iTunes Connect a v Xcode je importovaný distribuční provizní profil, stačí provést následující kroky:

1. v Xcode, v menu Product spustit položku „Archive“ – vytvoří kompletní archiv, který může být nahrán do AppStore,
2. spustit Xcode Organizer a pod položkou „Archives“ vybrat právě vytvořený archiv,
3. stisknout tlačítko „Validate“ – mimo jiné ověří, zda je kompletně vytvořená aplikace v iTunes Connect,
4. stisknout tlačítko „Distribute“ – zajistí nahrání aplikace do AppStore ke schválení.

6.5 Stavy aplikace

Každá iOS aplikace musí projít celým životním cyklem, který zahrnuje kontrolu aplikace a zdrojových kódů zaměstnanci Applu. Aplikace může být buď schválena, nebo zamítnuta. Pokud je schválena, je následující den dostupná v AppStore. V opačném případě obdrží vývojář vyjádření, z jakého důvodu byla aplikace odmítnuta a má možnost aplikaci upravit a znovu potvrdit ke schválení. Tento proces se může opakovat i několikrát. Apple vývojářům nabízí tzv. „Human interface guidelines“[7], kde lze nalézt doporučené implementace všech grafických iOS prvků a další typy k vývoji. Jako další musí aplikace splňovat podmínky definované v dokumentu „AppStore Review Guidelines“[8].

Tabulka 3 – Stavy aplikace v iTunes Connect

Název stavu	Popis
Prepare For upload	První status aplikace po vytvoření v iTunes Connect.
Waiting For upload	Po vyplnění všech metadat v iTunes Connect je nutné přepnout aplikaci do tohoto stavu, aby bylo možné nahrát archiv.
Waiting For review	Aplikace čeká ve frontě na schválení.
In Review	Aplikace je právě kontrolována zaměstnanci Applu.
Pending Contract	Aplikace je schválena, ale vývojář nemá ještě vytvořené nebo schválené všechny potřebné smlouvy.
Upload Received	Tento stav nastává bezprostředně po nahrání aplikace z Xcode.
Pending Developer Release	Aplikace je schválena, ale publikace v AppStore byla vývojářem nastavena na pozdější datum.
Processing For AppStore	Probíhá zpracování pro AppStore a do 24 hodin bude aplikace připravena k prodeji.
Pending Apple Release	Publikace je pozdržena Applem, např. kvůli tomu, že požadovaná verze iOS ještě není veřejně vydána.
Ready For Sale	Aplikace je dostupná v AppStore.
Rejected	Aplikace byla zamítnuta.

[7] https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html

[8] https://developer.apple.com/appstore/resources/approval/guidelines.html

Název stavu	Popis
Metadata Rejected	Metadata aplikace nesplňují podmínky.
Removed From Sale	Aplikace byla stažena z AppStore.
Developer Rejected	Vývojář odstranil aplikaci z čekání na kontrolu.
Developer Removed From Sale	Vývojář stáhl aplikace z AppStore.

6.6 Stav – Ready for sale

Jakmile je aplikace schválena, je možné ji začít prodávat. Počty stažení je třeba sledovat a vyhodnocovat z nich závěry. Taktéž se začnou objevovat hodnocení uživatelů, nebo neočekávané chyby v aplikaci, které je nutné řešit. Z tohoto důvodu existuje několik aplikací, které tuto práci značně usnadňují. Nejdůležitější aplikace pro správu iOS aplikací jsou popsány v následující kapitole.

7 Nástroje pro sledování aplikace

Ač se může zdát, že po zveřejnění aplikace v AppStore práce končí, opak je pravdou. V tuto chvíli se začne objevovat kritika od uživatelů, která nebývá vždy úplně konstruktivní a objektivní. Vždy je potřeba veškerou odezvu sledovat a pokud to lze, na připomínky reagovat. Je důležité mít vše pod kontrolou a měřit užitečná data. Téměř zajisté se objeví nějaké bugy, které se uživatelům jeví jako pády aplikace a to není nikdy příjemné. Bez externích nástrojů není téměř možné pády aplikace sledovat. Naštěstí je k dispozici mnoho nástrojů ulehčující tuto práci.

7.1 Testflight

Apple umožňuje vývojářům testovat vlastní aplikace na maximálně 100 iOS zařízeních, které musí být zaregistrovány v provizním portálu. Reálnou aplikaci je nutné testovat, avšak dříve byl celkem problém aplikaci dostat na zařízení běžných uživatelů (dále jen testerů). V Xcode se musí vygenerovat a exportovat distribuční archiv, který se musí přes iTunes synchronizovat s připojeným zařízením. Tento postup je pro testera zbytečně složitý a každý ho nemusí zvládnout. Navíc s každou opravnou verzí ho musí absolvovat znovu a znovu.

Z tohoto důvodu vznikla služba Testflight[9], která výše zmíněný postup značně ulehčuje. Nejsložitější kroky provede pouze jednou vývojář, zatímco testeři se pouze zaregistrují, po potvrzení jejich zařízení vývojářem otevřou odkaz z emailu a aplikace se sama stáhne a nainstaluje. Pomocí této služby je i možné najít ochotné a zkušené testery, pokud vývojář nemá své vlastní. Jedna z funkcí je i rozhraní pro komunikaci s testery. Navíc kompletní služba je zdarma.

7.2 Crashlytics

Občasné bugy a pády aplikace jsou běžné, dá se říct, že žádnou aplikaci se na poprvé nepodaří vyvinout úplně bez chyb. Je však nutné učinit taková opatření, aby se pádům aplikace vždy předcházelo. Jedním z nejužitečnějších nástrojů je Crashlytics[10], který přehledně seskupuje všechny crash reporty z pádů aplikace, které nastanou. Vývojář tak může reporty analyzovat a chyby opravit.

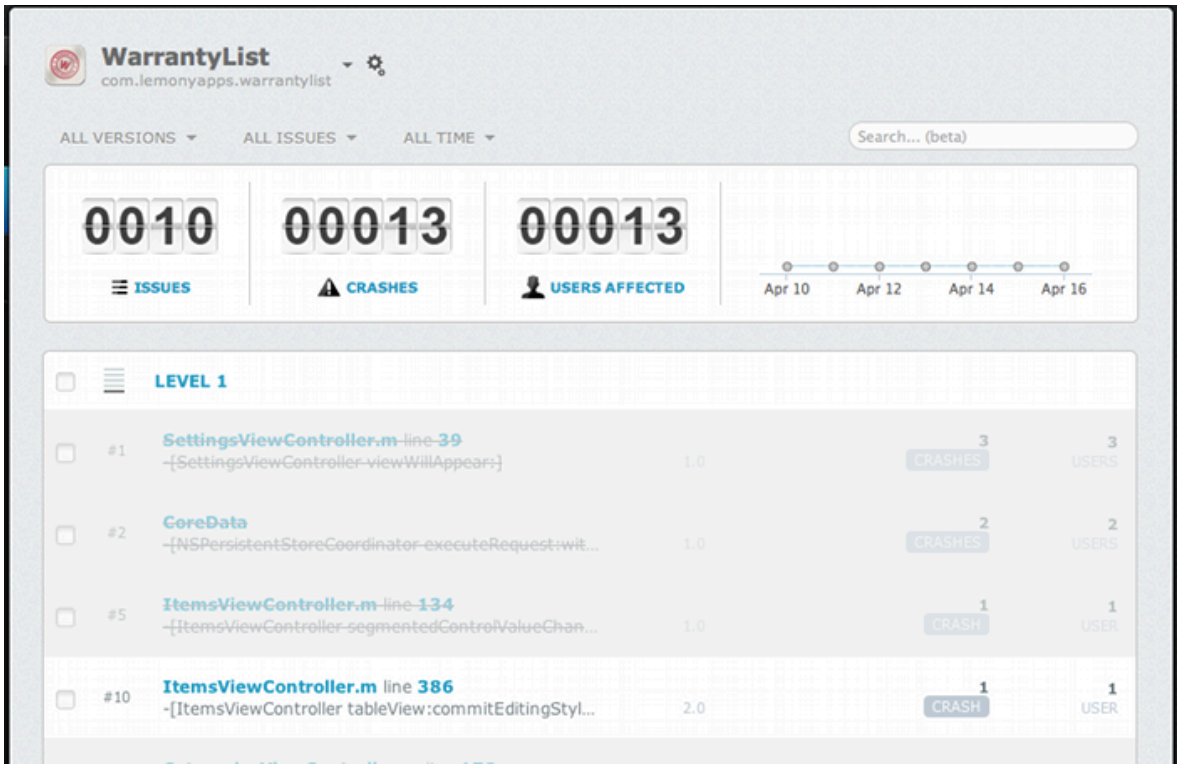
Na rozdíl od jiných externích frameworků je implementace Crashlytics do vlastní aplikace extrémně jednoduchá, následující seznam popisuje nutné kroky:

- zaregistrovat se na portálu Crashlytics,
- nainstalovat aplikaci Crashlytics pro Mac OS X,

[9] https://testflightapp.com
[10] http://crashlytics.com

- spustit průvodce a následovat posloupnost kroků, vše je provedeno automaticky, jen je potřeba vložit jeden řádek kódu, který průvodce sám vygeneruje, do delegátu aplikace.

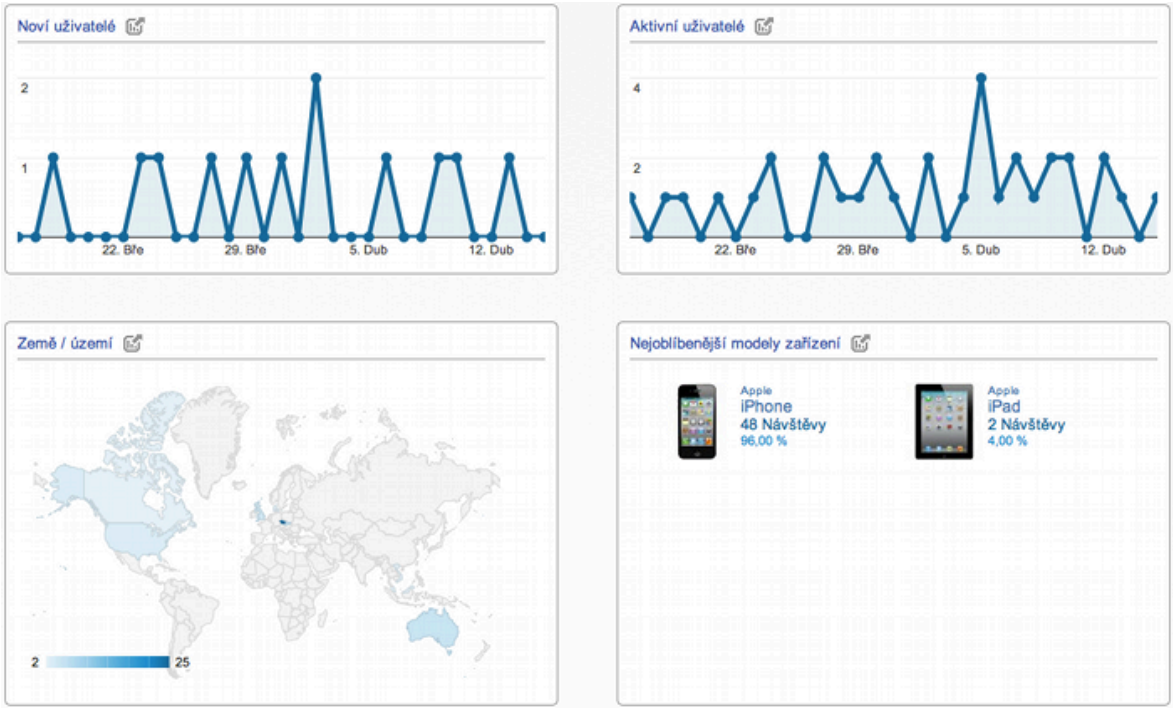
Po těchto krocích je implementace hotová a Crashlytics začne sbírat všechny crash reporty. Ve webovém rozhraní jsou reporty seskupené podle aplikací, je totiž možné používat jeden účet na sledování více aplikací. Dále jsou reporty seskupené podle typu chyby, tzn. že totožné chyby nejsou zobrazeny každá zvlášť, ale zobrazují se jako jeden řádek v tabulce a je zobrazen jejich počet. Jednotlivé reporty je možné označovat jako vyřešené, nebo jako otevřené, což také značně usnadňuje orientaci.



Obrázek 17 – Rozhraní Crashlytics

7.3 Google analytics

Celkem zajímavou statistikou může být počet každodenních spuštění aplikace nebo počet vykonání určité akce uživatelem. Toto lze sledovat pomocí nástroje Google analytics pro mobilní aplikace. Monitorují se jak noví uživatelé, tak spuštění aplikace stávajícími uživateli, včetně zařízení, které bylo použito (iPhone/iPad).



Obrázek 18 – Google analytics pro mobilní aplikace

Jako alternativu lze použít nástroj Flurry[11], který umožňuje více pokročilé možnosti sledování.

7.4 Nástroje pro sledování prodejů/stažení

Celkový přehled o počtu stažení aplikace a celkové utržené finanční částce je velmi důležitý. Apple nabízí pouze jednoduchý nástroj, který je součástí iTunes Connect, ale data nejsou archivována historicky, což je problém.

iTunes Connect

Základním nástrojem pro správu aplikací přímo u Applu je webová aplikace iTunes Connect. Nabízí i jednoduché rozhraní s přehledem počtu stažení aplikací. S velkým počtem aplikací v AppStore však není možné, aby Apple uchovával historicky všechna data. Z tohoto důvodu jsou data k dispozici pouze pár týdnů nazpět a archivace je ponechána na vývojářích.

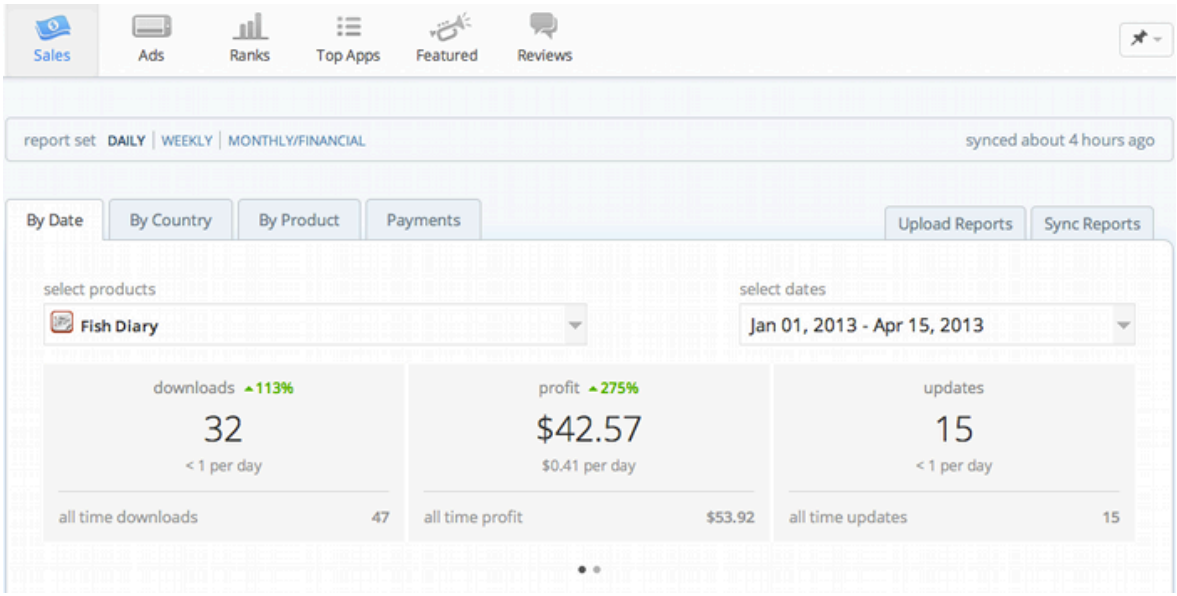
[11] http://www.flurry.com

Obrázek 19 - iTunes Connect

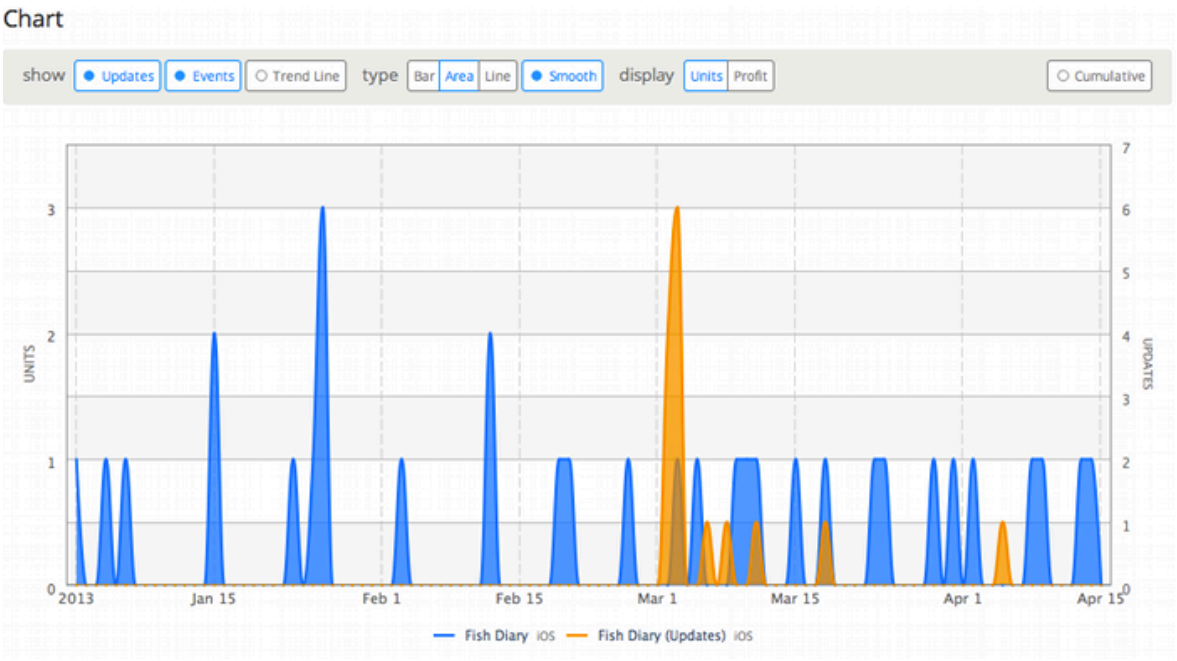
AppFigures

Velmi zdařilým nástrojem pro kompletní analýzu počtu stažení a vydělaných financí je webová aplikace AppFigures. Stačí zadat přihlašovací informace do iTunes Connect a aplikace si sama každý den stáhne nová data a přehledně je zobrazuje. Následující seznam shrnuje nejdůležitější funkce:

- automatický import dat,
- automatické zobrazování ranků v různých lokalizacích AppStoru,
- vizualizace dat v grafech, mapách a tabulkách,
- detailní reporty podle zemí, dat nebo aplikací,
- jednoduchý export dat,
- API.



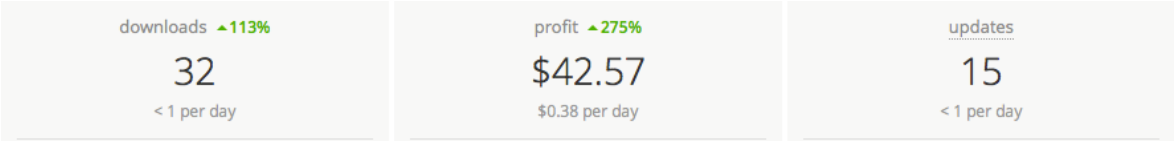
Obrázek 20 – AppFigures – základní statistika



Obrázek 21 – AppFigures – grafy

8 Vyhodnocení výsledků aplikace po čtyřech měsících v AppStore

Aplikace FishDiary vygenerovala za čtyři měsíce pouze 32 stažení.



Obrázek 22 – FishDiary – počet stažení

Největší trh s iOS aplikacemi se nachází ve Spojených státech a logicky bylo tedy nejvíce stažení právě ze Států, na druhém místě je Austrálie.

Country	Profit	Downloads	Updates	$
United States	$12.60	9	2	0
Australia	$10.53	8	7	0
United Kingdom	$5.61	4	0	0
Vietnam	$4.20	3	0	0
Slovakia	$2.83	2	1	0
Lebanon	$1.40	1	0	0
Malaysia	$1.40	1	0	0
Canada	$1.38	1	0	0
Denmark	$1.37	1	0	0
South Africa	$1.25	1	2	0
Czech Republic	$0.00	1	2	1
Netherlands	$0.00	0	1	0

Obrázek 23 – FishDiary – počet stažení podle států

Počet stažení je velmi malý. Při rozboru se nabízí několik možností, proč to tak je.

1. Marketingu v zahraničí, zejména ve Spojených státech, bylo věnováno malé úsilí a aplikace nebyla téměř vůbec propagována.

2. Aplikace má sice jedinečný přístup k ukládání vycházek a úlovků, ale postrádá jakoukoliv přidanou hodnotu.

3. Původní smysl aplikace je osobní zápisník, což je možná chyba, protože dnešní uživatelé chtějí sdílet data a komunikovat s ostatními rybáři.

Jako největší problém se tedy jeví nedostatečný marketing, propagace aplikace na zahraničních trzích. Ačkoli se to nemusí na první pohled zdát, ale marketing mobilních iOS aplikací je velmi komplikovaný. Přístupů existuje mnoho a ne vždy se podaří zvolit ten správný.

Závěr

V práci se povedlo vytvořit kompaktní aplikaci pro rybáře, která plní plnohodnotnou funkci osobního zápisníku. Aplikace podporuje verzi iOS 5 a vyšší a je přizpůsobena i nejnovějšímu modelu iPhonu5. Byla úspěšně schválena a publikována v AppStore, avšak výsledky prodejů nenaplnily očekávání, která byla stanovena v prvotní analýze.

I přesto se nabízí několik možností, jak aplikaci rozšířit a zvednout tak celkovou rentabilitu. Již v prvotní analýze byl jeden z požadavků vytvořit rybářskou sociální síť. Tento požadavek byl nakonec z funkčnosti odebrán, a to zejména kvůli vysokým nákladům na čas a finance. Nicméně aplikace je navržena tak, aby nebylo příliš komplikované tuto funkčnost implementovat, a není vyloučeno, že se tak v budoucnosti nestane.

Jako další krok ve vývoji by bylo vhodné vytvořit přidanou hodnotu, kterou ostatní aplikace podobného typu nenabízejí. Pomocí metod data miningu je možné vytvořit databázi několika desítek tisíc rybářských revírů na celém světě a nabídnout je uživatelům aplikace. Z průzkumu je známo, že by uživatelé ocenili funkci zobrazení revírů v jejich okolí podle GPS souřadnic. Další užitečnou přidanou hodnotou by mohla být databáze ryb, rybářského náčiní nebo nejpoužívanějších uzlů.

Všechna předchozí navrhovaná vylepšení směřují k povýšení aplikace na sociální síť a sběr uživatelských dat, které jsou nejcennějším zdrojem informací. Tento krok by samozřejmě také vedl k nutné změně finanční politiky. Bylo by nutné publikovat aplikaci zdarma, což nebyl původní záměr. Avšak i aplikace tohoto typu lze monetizovat. Jednou z možností je zobrazení baneru s reklamou s možností placeného odstranění.

Literatura a zdroje

[1] KOCHAN, Stephen G. *Objective-C 2.0: Výukový kurz programování pro Mac OS X a iPhone*. Brno: Computer press, a. s., 2010. ISBN 9788025126547.

[2] APPLE INC. IOS Developer Library. 2007.
Dostupné z: http://developer.apple.com/library/ios/.

[3] SADUN, Erica. The iOS 5 Developer's Cookbook: Core Concepts and Essential Recipes for iOS Programmers [online]. Third Edition. 2009 [cit. 2012-10-10]. ISBN 9780321832078. Dostupné z: http://www.amazon.com.

[4] *CS 193P iPhone Application Development* [online]. 2013 [cit. 2013-04-04]. Dostupné z: http://www.stanford.edu/class/cs193p/cgi-bin/drupal/.

[5] RAZEWARE LLC. *Ray Wenderlich - Tutorials for iPhone / iOS Developers and Gamers* [online]. c 2012 [cit. 2013-04-14].
Dostupné z: http://www.raywenderlich.com