Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj aplikace pro mobilní operační systém iOS

Sakun Aleksandr

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Aleksandr Sakun

Informatika

Název práce

Vývoj aplikace pro mobilní operační systém iOS

Název anglicky

App development for iOS mobile operating system

Cíle práce

Cílem bakalářské práce je popsat metody a technologie které se používají k vývoji aplikací na mobilní zařízení s použitím programovacího jazyka Swift. Dílčími cíli je představení platformy iOS, vývojových prostředí a programovacích jazyků pro tvorbu aplikací pro tento OS a dále návrh a implementace ukázkové aplikace.

Metodika

Bakalářská práce bude rozdělena na dvě základní části: teoretickou, která bude založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce a praktickou, která spočívá ve vytvoření návrhu architektury, návrhu uživatelského rozhraní a následné implementaci iOS aplikace za použití programovacího jazyka Swift. A také bude použito připojení FireBase pro ukládání a přistupování k souborům.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Mobilní aplikace, iOS, iPhone, Reaktivní programování, Swift, MVVM, RxSwift, FireBase, Xcode

---

Doporučené zdroje informací

App Architecture: iOS Application Design Patterns in Swift Paperback – May 13, 2018 by Chris Eidhof (Author), Matt Gallagher (Author), Florian Kugler (Author)

FireBase dokumentace. [online] Dostupné z: https://firebase.google.com/docs/

LEE, Valentino., Heather. SCHNEIDER a Robbie. SCHELL. Mobile applications: architecture, design, and development [online]. Upper Saddle River, N.J.: Pearson Education, c2004.

RxSwift: Reactive Programming with Swift, Second Edition [online]. raywenderlich.com, c2017 [cit. 2018-03-19]. ISBN 9781942878469.

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 3. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 3. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 03. 2019

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „Vývoj aplikace pro mobilní operační systém iOS" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.03.2019 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Jiřímu Brožkovi, Ph.D., za vedení diplomové práce.

Vývoj aplikace pro mobilní operační systém iOS

Abstrakt

Tato práce se zabývá vývojem mobilní aplikace na platformě iOS při pomoci reaktivního programování. Na základě teoretické části byla vytvořena aplikace. Bylo provedeno porovnání jednotlivých softwarových architektur iOS a vybrána byla architektura MVC (Model View Controller). Dalším úkolem bylo vytvořit logický návrh, s jehož pomocí byla provedena kontrola funkčnosti jednotlivých komponentů. Po vytvoření logického návrhu byl vytvořen grafický návrh. Aplikace byla vytvořena pomocí Apple programu Xcode, k implementaci kódu – OOP jazyk (objektivně orientovaný programovací jazyk) Swift, společně s jeho rozšířením RXSwift, který byl použit pro reaktivní programování. Aplikace poskytuje možnost zaznamenávat všechny a výdaje a také bude rozdělena na různé položky, pomocí kterých bude snadné se orientovat v aplikaci.

Klíčová slova: iOS, MVC, RXSwift, Xcode, Swift, mobilní aplikace, reaktivní programování

App development for iOS mobile operating system

Abstract

This work deals with the development of mobile applications on the iOS platform to help of reactive programming. The basis of the theoretical part was created by the application. I did of comparison of different software architectures on iOS and select MVC architecture (Model View Controller). Furthermore, my task was to create a logical design, which would check the functionality of each component. After creating the logical design, a graphic design was created. The creation of the app were in Apple programs Xcode, and for the implementation of the code – OOP language (object-oriented programming language) Swift, together with its extension RXSwift, which was used for the reactive programming. The application provides the ability to record all your income and expenses, and also will split the different sections to help with navigation in the application.

Keywords: iOS, MVC, RXSwift, Xcode, Swift, mobile application, reactive programming

Obsah

Seznam obrázků

1 Úvod

V dnešní době je téměř každý člověk závislý na telefonu, i kdybychom to chtěli popřít, ale den bez telefonu si nedokážeme představit. Zvyšují se tak nároky lidí na techniku, protože už si zvykli, že to, co už dnes existuje, zítra již nebude stačit a bude to nahrazeno něčím jiným, dokonalejším. Každý den se programátoři snaží zavést něco nového. Existuje řada užitečných aplikací, které nám umožňují komunikovat, sledovat počasí, sledovat novinky, hrát si. Všechny jsou součástí masové spotřeby, protože telefon je mini-počítač, který je velmi kompaktní a který lze mít vždy s sebou, do značné míry nahrazuje počítače a notebooky.

Ze všech těchto důvodů jsem se věnoval vývoji iOS aplikací. Nejsem tak dlouho se zabývám vším, ale uvědomil jsem, že pro mě je to velmi zajímavá věc. Vše, co jsem se naučil, bych chtěl ukázat v této bakalářské práci.

Mým úkolem v této práci je vytvořit aplikaci, která bude počítat příjmy a výdaje, protože v naší době ne každý člověk může utrácet své peníze, aniž by věnoval pozornost jejich množství. Chci vytvořit cenově dostupnou aplikaci, kterou bude moct využít i běžný uživatel. Tato aplikace bude offline, aby se člověk mohl podívat na své náklady a počet zbývajících peněz v každém okamžiku.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem bakalářské práce je popsat metody a technologie, které se používají k vývoji aplikací na mobilní zařízení s použitím programovacího jazyka Swift. Dílčími cíli je představení platformy iOS, vývojových prostředí a programovacích jazyků pro tvorbu aplikací pro tento OS a dále návrh a implementace ukázkové aplikace.

2.2 Metodika

Bakalářská práce bude rozdělena na dvě základní části. Teoretická část bude založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce. Praktická část spočívá ve vytvoření návrhu architektury, návrhu uživatelského rozhraní a následné iOS aplikace za použití programovacího jazyka Swift.

3 Teoretická východiska

3.1 Co je iOS

iOS – je mobilní operační systém od společnosti Apple [1].

3.2 Kompletní historie iOS

Pojem iPhone zná v dnešní době opravdu každý. Součástí iPhonu je operační systém iOS, který konkuruje systémům Android a Windows Phone. Když Steve Jobs představil iPhone v lednu roku 2007, říkal, že se telefony dělí do dvou kategorií: na jedné straně byly výkonné telefony, které ale byly velmi složité, na druhé straně byly mobily snadno ovladatelné, ale jejich funkce byly omezené. Proto se společnost Apple rozhodla vytvořit mobil, který musel změnit všechno. Tento mobil byl mnohem silnější a lepší než ostatní. iPhone, naprosto inovativní ve své době představoval to, co přinutilo jiné velké společnosti věnovat mu pozornost a změnit všechno ve svých technologiích. Dalo by se dokonce říci, že pokud by nebylo společnosti Apple, Android by nebyl tak konkurenční jako teď.

Hlavním operačním systémem iPhone byl vždy iOS (iPhone OS). Popravdě řečeno, hardware iPhonu nebyl nikdy tak dokonalý, ale spolupráce systému s hardwarem dělá iPhone nejvýkonnějším mobilním telefonem na planetě, operační systém tak odstraňuje hardwarové nedostatky.

Během posledních osmi let se iOS změnil a dále vyvíjel. V současné době je iOS opravdu vyspělý a jde o jeden z nejbezpečnějších systémů. V době, kdy byla představena první generace tohoto mobilního operačního systému, neexistovaly žádné aplikace a uživatelé si dlouhé chvíle krátili sledováním videí na YouTube a hraním internetových her. Při pohledu do minulosti nás překvapuje, jak se z jednoduchého operačního systému, ve kterém nebyl multitasking, složky nebo v dnešní době důležité aplikace, se vyvinul jeden z nejlepších operačních systémů [2].

3.2.1 iPhone iOS 1

V lednu 2007 Steve Jobs představil iPhone OS, a změna názvu proběhla až za několik let. I přestože byl iPhone velmi inovativní a progresivní, chyběla mu řada funkcí. Proto nebylo možné v první verzi instalovat aplikace, protože v tuto dobu ještě neexistoval App Store, ale společnost Apple udala správný směr v uživatelském prostředí. Později však

Apple vydal první iOS SDK, čímž změnil svůj názor a v roce 2008 byl spuštěn iTunes App Store [2].

3.2.2 iPhone iOS 2

Ve druhé generaci iPhone OS byla hlavní novinka v tom, že už bylo možné využít App Store, který byl oficiálně spuštěn spolu s vydáním nového systému 8. července 2008. Uživatel už nemusel používat počítač ke stažení aplikace. A ještě v ten den byl představen iPhone 3G [2].

3.2.3 iPhone iOS 3

iPhone OS 3.0 byl doposud jednou z největších aktualizací systému. Tento OS přinesl funkce jako kopírovat, vyjmout a vložit. Byl zlepšen GPS a ještě navíc přidán kompas. Také byla přidána nová funkce: „Find my iPhone" (tato aplikace nám pomáhá najít polohu ztraceného iPhonu) [2].

3.2.4 iPhone iOS 4

V roce 2010 byla představena čtvrtá generace OS, která už nesla název iOS. Přibyl „multitasking", to znamená, že uživatel mohl pracovat s více spuštěnými aplikacemi najednou, také byla k dispozici nová možnost komunikovat pomocí video hovoru přes aplikace FaceTime. Ještě vytvořili iBooks, který umožňuje číst elektronické knihy, a vzdálené přehrávání multimédií AirPlay [2].

3.2.5 iPhone iOS 5

iOS 5.0 nám přinesl přes 200 nových funkcí, což je největší update z roku 2007. Tato generace nabízí úplně novou funkci – Siri, aplikace, která rozpoznává dotazy a výhledová odpovědi na webových stránkách nebo se orientuje v mobilních aplikacích. Další možností je využití služby iMessage, která umožňuje komunikaci mezi zařízeními iOS a prvním cloudovým úložištěm od Apple - iCloud [2].

3.2.6 iPhone iOS 6

V roce 2012 uvedla společnost Apple šestou generaci, která přináší absolutně novou aplikaci pro iPhony – mapy s možností hlasové navigace. Také byl přidán Passbook, do něhož mohou uživatelé ukládat letenky, kupony, lístky do kina a mnoho dalších položek. Nakonec zlepšil Siri [2].

3.2.7 iPhone iOS 7

Sedmá generace systému iOS přináší největší designovou změnu od původního iOS. Úplnou změnou prošlo GUI, nové funkce a celkově byl hezčí systém. Součástí systému byl nový Control Center (tento způsob určuje, jak přepínat celou řadu běžně používaných nastavení). Byl vylepšen multitasking a vytvořen AirDrop. Dále v aplikace „foto" se mohly používat filtry [2].

3.2.8 iPhone iOS 8

V roce 2014 přišla osmá generace iOS 8. V tu dobu Apple přestal pracovat nad GUI, ale prostě se aktivně zapojil různými aplikacemi, což znamenalo zapojení řady vývojářů. A hlavně byl představen nový programovací jazyk – Swift [2].

3.2.9 iPhone iOS 9

V roce 2015 přišel nový iOS 9. Apple se staral o stabilitu celého systému. Díky tomu měl náš iPhone lepší výdrž baterie a novou možnost používat režim „Low Power" (tento režim odstraní z pozadí aktualizace, které nepoužíváme). Apple také vytvořil novou aplikaci „News" (s její pomocí jsme mohli číst různé zprávy). Došlo také ke zlepšení multitaskingu [2].

3.2.10 iPhone iOS 10

V září 2016 přišel iOS 10. V něm se objevily další velké změny, například App Store se stal plnohodnotnou platformou a plně se otevřel vývojářům třetích stran. A Siri už mohla komunikovat s aplikacemi, které nejsou od Applu. Velkých změn se dočkala oblast notifikací, která nyní zobrazuje přímo i fotky či videa [2].

3.2.11 iPhone iOS 11

V roce 2017 vyšel iOS 11. Tady opět došlo k vylepšení App Storu a Siri. Apple přidal technologie rozšířené reality, které zprostředkují hluboké zážitky. iOS 11 je maximálně nastaven na nové iPady a používá na nich Apple tužky. Také byla vytvořena nová aplikace – Soubory [9].

3.2.12 iPhone iOS 12

V roce 2018 přišel iOS 12, který je aktuální dodnes. Nepřinesl nic nového, jenom zlepšení všeho, co už existovalo. Tento systém je mnohem výkonnější a rychlejší [9].

3.3 Architektura iOS

iOS je odlehčenou verzí operačního systému OS X, který používáme v počítačích od společnosti Apple. Tento systém je určen pro mobilní zařízení, to znamená, že nemusí obsahovat veškerou funkcionalitu OS X, ale je tady přidána podpora dotykového ovládání. Základní model architektury systému iOS se skládá ze čtyř vrstev:

- Rozhraní Cocoa Touch.
- Media Layer (Multimédia).
- Core Services (Služby).
- Core OS (Jádro systému) [5].

3.3.1 The Frameworks

Fundament framework implementuje kořenové třídy NSObject, pomocí kterých probíhá definice chování objektu. Spustí implementace třídy, které představují primitivní typy. V základě se všechno spoléhá na zabezpečení objektu, správu souborů a práce s XML dokumenty. Máme možnost využití svých tříd. Základ je založen na Core Foundation framework, který vydává procesní (ANSI C) rozhraní.

Framework UIKit obsahuje různé třídy, a proto tady probíhá budování uživatelských rozhraní programů. V UIKit probíhá spousta widgetů, např. (text, obrázky, tabulky) [5].

3.3.2 Jazyk

Objective-C je původní primární jazyk pro vývoj Cocoa a Cocoa Touch aplikací. Nicméně projekty pro Cocoa a Cocoa Touch aplikací mohou obsahovat C++ a ANSI C kód. Navíc můžete rozvíjet Cocoa aplikace pomocí skriptovacích jazyků, které jsou dodány do Objective-C runtime jako PyObjC a RubyCocoa [5].

3.4 Cocoa Touch

Co to je Cocoa, můžeme pochopit podle definice od společnosti Apple: „Je sada objektově orientovaných frameworků, které poskytují průběh prostředí pro aplikace, jež se spustí na iOS zařízeních. Pomocí Cocoa Touch můžeme budovat svůj software. Kvůli tím frameworkam každý programátor může zvolit libovolnou úroveň pro svou aplikaci [5].

3.5 Media vrstva

Media layer je termín společnosti Apple Inc., který odkazuje na softwarové rámce a technologie umožňující zvukové, vizuální a další multimediální funkce v zařízeních se systémem iOS. Definuje celou multimediální architekturu v mobilních zařízeních a aplikacích s technologií Apple.

Vývojáři softwaru obvykle používají vrstvu média pro pochopení, přístup a vývoj mobilních aplikací a služeb v zařízeních se systémem iOS. Vrstva médií je rozdělena do několika komponent, které jsou specificky klasifikovány pro grafické, zvukové nebo vizuální podpůrné rámce. Grafické rámce umožňují vývojářům vytvářet aplikace, které poskytují grafické rozhraní, animace, čitelnost vstupů / výstupů (I / O) a přístup k nativním vizuálním prvkům zařízení. Audiovizuální rámec umožňuje přehrávání, nahrávání a integraci zvuku v rozvinutých aplikacích (6).

Grafický a zvukový rámec zahrnuje:

Grafický

- Core Graphics Framework.
- Core Animation Framework.
- Otevřete GL.

Zvuk

- Rámec přehrávače médií.
- Otevřete AL rámec.
- Core Audio Framework [6].

3.6 Core services vrstva

Vrstva Core Services je zodpovědná za správu základních systémových služeb, které používají nativní aplikace iOS. Vrstva Cocoa Touch závisí na vrstvě Core Services pro některé její funkce. Vrstva základních služeb také poskytuje řadu nepostradatelných funkcí, jako jsou blokové objekty, Grand Central Dispatch, Purchase In-App a iCloud Storage.

Jedním z nejvítanějších dodatků k vrstvě Core Services je ARC nebo automatické počítání referencí. ARC zjednodušuje proces správy paměti v Objective-C.

Základním rámcem pro aplikace iOS je Fundament framework nebo Fundament for short. Fundament framework je více než sbírka užitečných tříd, jako jsou: *NSArray*,

NSDictionarya, *NSDate*. Fundament poskytuje *NSObject* třídu kořenů, která poskytuje základní rozhraní pro spouštění Objective-C. Dalším důležitým rámcem vrstvy Core Services a úzce svázaným s rámcem fundamentu je rámcová platforma *Core Foundation.* Podobně jako rámec fundamentu umožňuje různým knihovnám a rámcům sdílet data a kód.

Core Foundation má funkci známou jako toll-free bridging (bezplatná přemostění), která umožňuje nahrazení objektů Cocoa za objekty Core Foundation v parametrech funkcí a naopak [6].

3.6.1 Block objekty

Od verze 4.0 je možné používat objekty typu Block. Jedná se o jazykový konstrukt jazyka C, který je možný používat ve stávajícím C nebo Objective-C kódu. Block objekt reprezentuje anonymní funkci a související data. Takový konstrukt je v jiných jazycích často nazýván closure nebo lambda. Block objekty se hodí jako callback [6].

3.6.2 Grand Central Dispatch

Ve verzi 4.0 byla přidána technologie Grand Central Dispatch postavená na BSD, která umožňuje správu úloh v aplikaci. GCD kombinuje asynchronní model programování s vysoce optimalizovaným jádrem a poskytuje tak jednoduchou a zároveň efektivní alternativu k vláknovému programování [6].

3.6.3 Lokační služby

Umožňují sledovat aktuální polohu uživatele. Služby využívají k určení polohy veškerý dostupný hardware (Wi-Fi, telefonní síť, GPS). Aplikace tak mohou uživateli nabídnout data relevantní k jeho poloze (např. nejbližší restaurace apod.) [6].

3.6.4 SQLite

Odlehčená SQL databáze umožňuje ukládání uživatelských dat [6].

3.6.5 Podpora XML

Podpora pro zpracování XML dokumentů [6].

3.7 Vrstva Core OS

Vrstva Core OS poskytuje nízkoúrovňové funkce ostatním technologiím, které jsou na ní postaveny. I když nejsou většinou v aplikacích využívány přímo, velice pravděpodobně je využívají vysokoúrovňové komponenty systému [6].

3.7.1 Accelerate Framework

Poskytuje rozhraní pro práci s matematickými funkcemi (obdoba java.math), velkými čísly, výpočty DSP apod. Výhodou tohoto frameworku oproti vlastní implementaci těchto funkcí je fakt, že v různých verzích iOS určených pro různá zařízení je tento framework optimalizován pro daný hardware [6].

3.7.2 External Accessory Framework

Poskytuje rozhraní pro práci s matematickými funkcemi (obdoba java.math), velkými čísly, výpočty DSP apod. Výhodou tohoto frameworku oproti vlastní implementaci těchto funkcí je fakt, že v různých verzích iOS určených pro různá zařízení je tento framework optimalizován pro daný hardware [6].

3.7.3 External Accessory Framework

Tento framework poskytuje podporu pro komunikaci s externími zařízeními připojenými přes Bluetooth nebo třicetipinový konektor zařízení. Framework také umožňuje získávat informace o dostupném příslušenství a navázat komunikaci [6].

3.7.4 Security Framework

Kromě vestavěných bezpečnostních vlastností iOS je možné využít Security framework, který dokáže zaručit bezpečnost citlivých dat. K dispozici jsou rozhraní pro certifikáty, soukromé a veřejné klíče, generování kryptografických pseudonáhodných čísel apod. Dále je možné ukládat data do zašifrovaného centrálního úložiště svazku klíčů (keychain). V tomto úložišti je navíc možné údaje sdílet mezi aplikacemi (pouze v případě, že je aplikace zkompilována s příslušným nastavením oprávnění) [6].

3.8 Výběr architektury

Pokud vývojář pracuje současně s více věcmi, pak často nemůže udržet přehled nad všemi chybami, takže vaše třídy musí být uloženy v paměti. Pokud je řeč o aplikaci v iOS, pak mohou být špatné příklady architektury:

- Tato třída je podtřída UIViewController.
- Vaše data jsou uložena přímo v UIViewController.
- Vaše UIViews téměř nic dělají.
- Model je hloupá datová struktura.
- Vaše testy jednotky nepokryjí nic [7].

3.8.1 Proč distribuce?

Distribuce udržuje spravedlivé zatížení našeho mozku, zatímco se snažíme zjistit, jak to funguje. Pokud si myslíte, že čím více se vyvíjíte, tím lépe se váš mozek přizpůsobí pochopení složitosti, pak máte pravdu. Ale tato schopnost není lineárně měřitelná. Nejjednodušší způsob, jak porazit složitost, je rozdělit odpovědnosti mezi více subjektů na základě zásady *jednotné odpovědnosti* [7].

3.8.2 Proč je testovatelnost?

Obvykle to není otázka pro ty, kdo již pociťovali vděčnost k jednotkovým zkouškám, které *selhaly* po přidání nových vlastností nebo kvůli refactoringu některých složitostí třídy. Znamená to, že testy *uložily* vývojářům problémy v běhu, což se může stát, když se aplikace nachází na zařízení uživatele a oprava trvá týden, než se dostane k uživateli [7].

3.8.3 Základy MV (X)

- MVC.
- MVP.
- MVVM.
- VIPER.

První tři z nich předpokládají zařazení entit aplikace do jedné ze tří kategorií:

- **Model** – je zodpovědný za doménová data nebo *vrstvu pro přístup k datům*, která manipuluje s třídy **Person** nebo **PersonDataProvider**.
- **View** – je zodpovědný za prezentační vrstvu (GUI).

- **Controller/Presenter/ViewModel** – zprostředkovatel mezi *modelovou* a *prezentační vrstvou*, který je obecně odpovědný za změnu *modelovové* reakce na uživatelské změny v *prezentační* vrstvě a aktualizuje informace *prezentační* vrstvy na základě *modelové* vrstvy.

Vše toto rozdělení entit nám pomáhá:

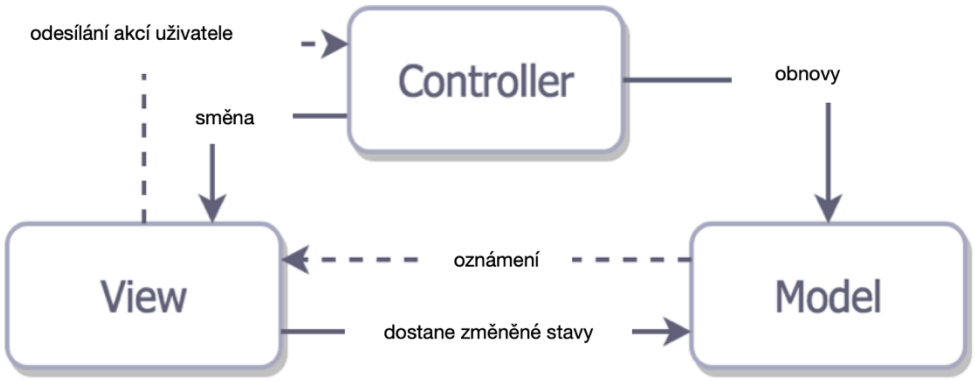
- Lépe jim porozumět.
- Znovupoužitelnost.
- Nezávislé testování [7].

3.8.4 MVC – Model View Controller (Jak to bylo dříve)

Na začátku se podíváme na tradiční verze MVC, dříve než budeme diskutovat o vizi MVC společností Apple.

Takže MVC – jedná se o uživatelské rozhraní (UI). Nemusí být grafický, hlasové ovládání má také dobré. Nelze opominout, že program nemusí mít uživatelské rozhraní, zároveň může mít programovací rozhraní (API), nebo prostě být užitečné.

Obrázek 2:

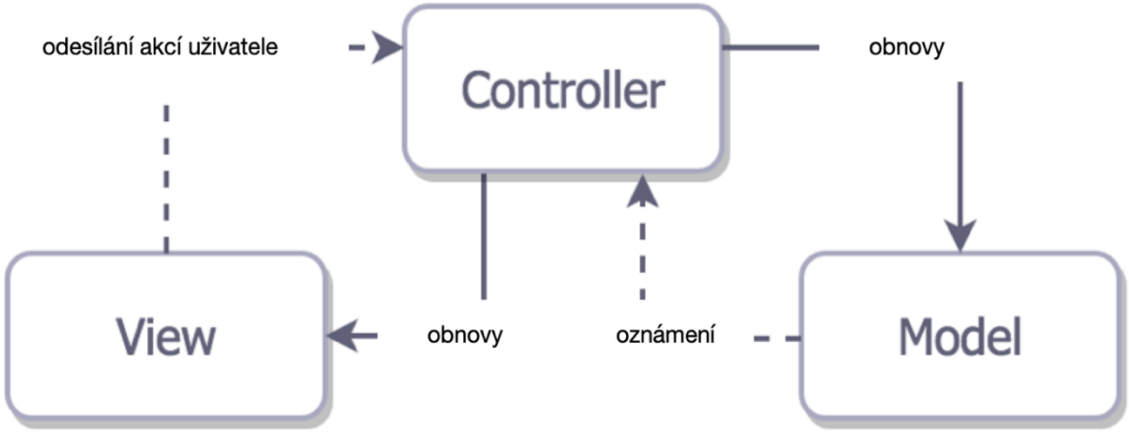


Zdroj: Vlastní zpracování

V tomto případě je **View** zbytečný. Je to prostě poskytnutý **Controller,** jakmile se **Model** změnil. I když je možné realizovat tradiční MVC v iOS aplikaci, nedává to moc smysl vzhledem k architektonickému problému – všechny tři entity úzce souvisí, každá entita ví o dalších dvou. To výrazně snižuje možnost opětovného použití každého z prvků — to není to, co chcete mít ve své aplikaci [7].

3.8.5 MVC od společnosti Apple (Očekávání)

Obrázek 3:

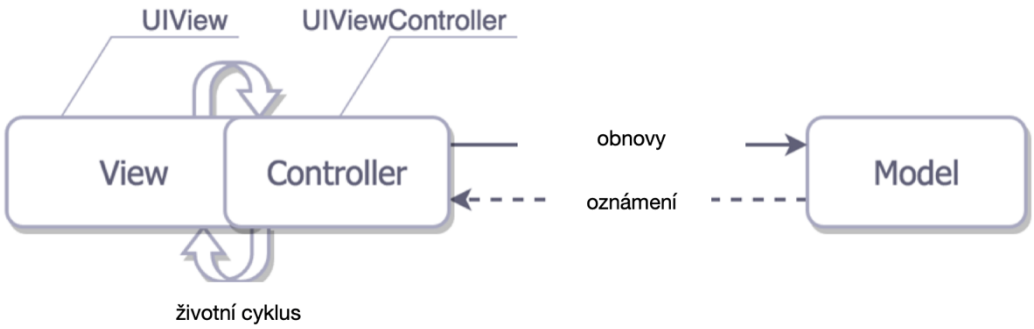


Zdroj: Vlastní zpracování

Controller je prostředníkem mezi **View** a **Model** tak, že o sobě nevědí. Nejméně opakovaně použitelný je **Controller,** což je obvykle pro nás dobře, protože musíme mít místo pro všechnu chytrou obchodní logiku, která neodpovídá **Modelu**. Teoreticky to vypadá velmi jasně, ale máte pocit, že je něco špatně? Dokonce spousta lidí říká **Massive ViewController** (řídicí jednotka s velkým pohledem) místo zkratky MVC. Kromě toho se pro vývojáře iOS stalo důležitým tématem k zobrazení vyčerpání **Contollera**. Co se stane, pokud Apple prostě vezme tradiční MVC a trochu to zlepší [7]?

3.8.6 MVC od společnosti Apple (Realita)

Obrázek 4:



Zdroj: Vlastní zpracování

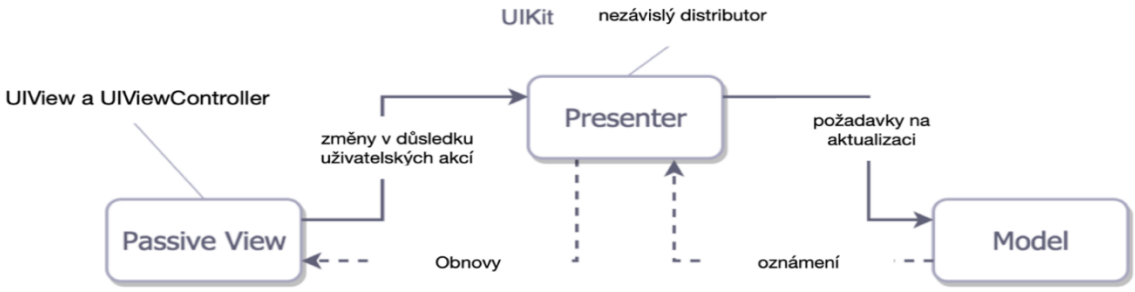
Reálné Cocoa MVC vás podporuje tak, abyste psali ten kód pohledové ovládání (**Controller**), protože jsou řídicí vrstvy úzce propojeny s životním cyklem **View** vrstvy, a proto je velmi těžké říci, že tyto vrstvy jsou oddělené. Ale tady problém vypadá ještě rozsáhleji. **Controller** také slouží jako delegát a zdroj dat pro prezentační vrstvu, obvykle zodpovědný za odesílání a rušení requestů na síť. Můžeme zde říci, že **ViewController** je nejdůležitější částí v architektuře. Po tom všem se může zdát, že Cocoa MVC je poměrně špatná volba vzoru. Ale pojďme to ocenit, pokud jde o známky dobré architektury definované na začátku článku:

1. **Distribuce** – View a Model jsou skutečně odděleny, ale View A Controller jsou úzce spojeny.
2. **Testovatelnost** – kvůli špatné distribuci budete pravděpodobně testovat pouze Model.
3. **Snadné použití** -– nejmenší množství kódu mezi jinými vzory, kromě toho vypadá jasně.

Cocoa MVC – je nejlepší architektonický vzor, pokud jde o rychlost vývoje [7].

3.8.7 MVP – Model View Presenter

Obrázek 5:

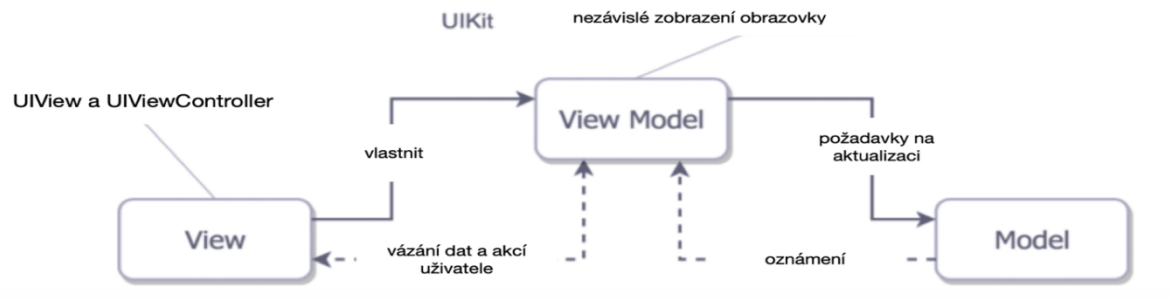


Zdroj: Vlastní zpracování

Co si myslíte? Nevypadá ten model jako model od společnosti Apple? Myslíte, že MVC od Apple je ve skutečnosti MVP? Tam jsou **View** a **Controller** úzce souvisí, a zároveň vystupují jako zprostředkovatel v MVP. **Presenter** - nemá vztah k životním cyklu **ViewControlle**r. **View** může být snadno nahrazen mock objects (objekt-imitace), proto **Presenter** nepoužívá layout-kód, ale je zodpovědný za aktualizaci **View** v souladu s novými daty a stavem [7].

3.8.8 MVVM – Model View ViewModel

Obrázek 6:



Zdroj: Vlastní zpracování

V teorii **Model View ViewModel** vypadá velmi dobře. **View** a **Model** jsou nám již známé jako prostředník. Tento model je velmi podobný MVP, protože zvažuje **ViewContoller** jako **View** a v něm neexistuje úzké spojení mezi **View** a **Modelem**. Jenom to, že dělá vazby (bindings) stejné jako verze MVP, ale není mezi **View** a **Modelem**, určitě mezi **View** a **ViewModel**. Tak co je **ViewModel** v prostředí iOS? Je to nezávislá představa od UIKit pomocí **View**. **ViewModel** způsobuje změny v **Modelu** a už lepší aktualizováno s ním. Mají vazby (bindings) mezi **View** a **ViewModel**, to znamená, že ten první model, který jsme zlepšili, také bude automaticky aktualizován [7].

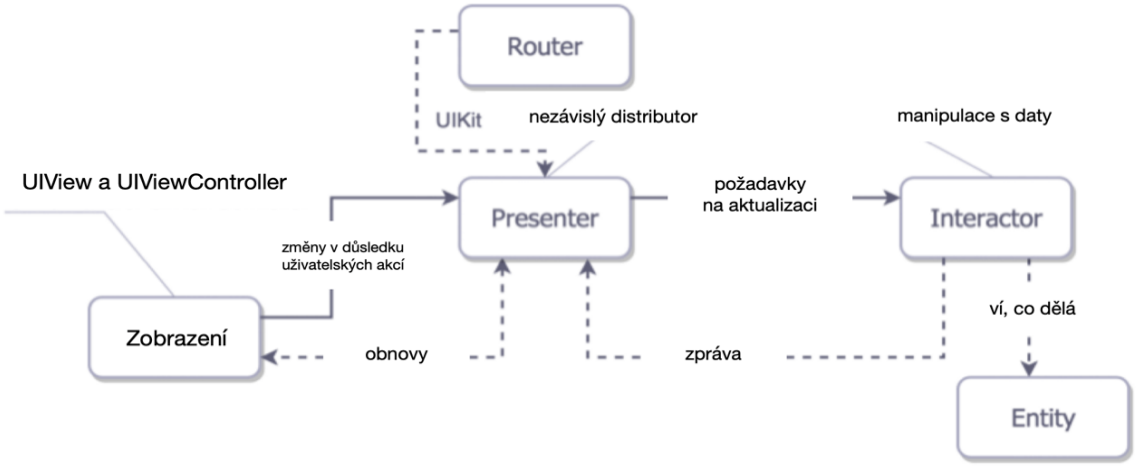
3.8.8.1 Co to jsou vazby (bindings)

Vazby jsou k dispozici „z krabičky" pro vývoj OS X, ale není v arzenálu iOS vývojáře. Jedinou možnost, kterou máme, je napsat svoji vlastní vazbu. To bude fungovat jako určitá nadstavba pro KVO observer, což je spojeno s notifikacemi. To všechno znamená, že nemůžeme použít pro vývoj OS X aplikace. V této době vznikají reaktivní frameworky (RxSwift, ReactiveCocoa) určené pro vývoj iOS, což je pro vývojáře velmi silný nástroj [7].

3.8.9 Viper

Pro nás je tento model velmi zajímavý, protože nepatří do skupiny MV(X). Hlavní jeho výhoda je v tom, že Viper obsahuje pět vrstev architektury místo tří.

Obrázek 7:



Zdroj: Vlastní zpracování

- **Interactor** – obsahuje business logiku, související s údaji (Entities): např. vytváření nových instancí entit nebo jejich získání ze serveru. Pro tyto účely budete používat některé Služby, které vypadají jako vnější závislosti, nikoli součást modulu VIPER.
- **Presenter** – obsahuje business logiku, která je úzce propojená s UI (ale UIKit-nezávislé), způsobuje metody v interactor.
- **Entities** – jednoduché datové objekty, které nemají přístup k datům, protože za to všechno odpovídá interactor.
- **Router** je odpovědný za přechody mezi VIPER-moduly [7].

3.9 Vývojové prostředí iOS

3.9.1 Xcode iOS

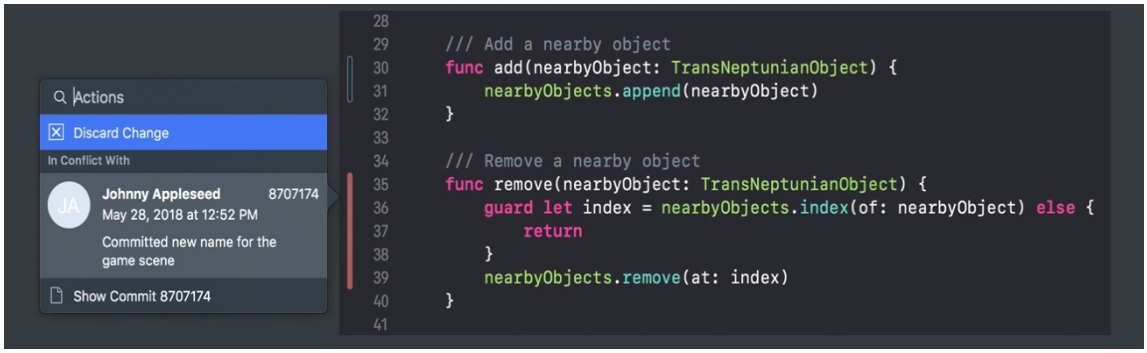
Xcode IDE (Integrované vývojové prostředí) obsahuje vše, co potřebujete k vytvoření úžasných aplikací pro všechny platformy Apple. Teď Xcode a nástroje vypadají skvěle v novém tmavém režimu na macOS Mojave. Zdrojový kód editor umožňuje transformovat nebo refaktorovat kód snadněji a pomáhá rychle získat informace o chybách. Můžete si vytvořit svůj vlastní nástroj s vlastní vizualizací a analýzou údajů. Swift sestavuje software rychleji, pomáhá zajišťovat rychlejší aplikace a vytváří i menší binární soubory. Testovací soupravy jsou mnohokrát rychlejší, pracovat s týmem je jednodušší a bezpečnější. Xcode vám také dává výkonné nástroje pro vytváření své vlastní temné aplikace pro macOS. Interface Builder vám umožní rychle přepínat svůj design a náhled ze světla do tmy. Katalog aktiv dává možnost definovat aktivity a pojmenované

barvy a umožňuje přepnout aplikaci. To všechno se udělá pomocí ovládacích prvků v Xcode, které se vztahují pouze k vaší aplikaci. Není třeba měnit vaše nastavení systému. V Markdown (*lehký značkovací jazyk vytvořený za účelem psaní nejvíce čitelného a snadno upravitelného textu*) soubory, nadpisy, tučný a italic text, odkazy a další formátování jsou okamžitě vykresleny v editoru, jak píšete. Jump Bar vždycky rozumí Markdown struktuře, takže můžete rychle procházet vaše README.md a soubory dokumentace [10].

3.9.2 Psát a používat kód jako profesionál

Hladké animace jsou použity pro skládání kódu, aby se zlepšilo zaměření, nebo když Xcode upozorní na chyby a nabízí opravu. S velkou Markdown podporou bude průvodní dokumentace vypadat skvěle. Použijte příkaz, vyberte symboly nebo celé struktury, s jejichž pomocí můžete transformovat nebo refaktorovat váš Swift, C, C++ a Objective-C kód. Změny kódu jsou zvýrazněny vedle každého řádku, pokud jsou tyto změny vytvořeny lokálně. V okamžiku, kdy zadáte nový řádek kódu, budete vědět, pokud jste vytvořili chybu, můžete rychle klepnout na červený indikátor, abyste získali více informací o chybném kódu.

Obrázek 8:



Zdroj: Vlastní zpracování

Xcode zahrnuje Swift, který sestavuje software rychleji, pomáhá zajišťovat rychlejší aplikace a vytváří méně binárních souborů. Optimalizováno pro nejnovější multi-core Mac hardware, Xcode a Swift pro rychlý vývoj platformy [10].

3.9.3 Swift

Swift je výkonný a intuitivní programovací jazyk pro macOS, iOS, watchOS a tvOS. Psaní Swift kódu je interaktivní, syntaxe je stručný, ale přesto zahrnuje výrazné a rychlé moderní prvky. Swift kód je bezpečný, ale také produkuje software, který běží rychle.

Swift je výsledkem nejnovějších výzkumů programovacích jazyků v kombinaci s desítkami let zkušeností budování Apple platformy. Pojmenované parametry, které byly přeneseny z Objective-C, jsou vyjádřené v čisté syntaxi. Odvodí typy, aby byl kód čistší a méně náchylný k chybám, zatímco moduly vylučují nadpisy a poskytují prostory názvů. Paměť je řízena automaticky a nemusíte psát středníky [9].

3.9.3.1 Důležité funkce Swiftu

- N-tice a více návratových hodnot.
- Generika.
- Rychlé a stručné iterace přes rozsah nebo kolekce.
- Struktury, které podporují metody, rozšíření a protokoly.
- Funkční programovací vzory, například mapy a filtr.
- Nativní zpracování chyb pomocí **try / catch / throw** [9].

3.9.3.2 Výhody Swiftu

- **Playgrounds a REPL v Xcode** – Zadejte řádek kódu a výsledek se ihned zobrazí. Pak se můžete rychle podívat na výsledek po straně vašeho kódu. Výsledek může zobrazovat grafiku, seznamy výsledků nebo grafy a hodnoty v průběhu času. Můžete otevřít Timeline Assistan, který sleduje komplexní pohled a animace, skvělé pro experimentování s novým kódem uživatelského ROZHRANÍ. Svůj kód na hřišti (Playground), jednoduše přesuňte do vašeho projektu.
- **Navrženo pro bezpečnost** – Swift eliminuje celé třídy nebezpečného kódu. Proměnné jsou vždy inicializovány před použitím, pole a celá čísla jsou kontrolovány na přetečení a paměť je řízena automaticky. Syntaxe je naladěn tak, aby bylo snadné definovat svůj záměr – například jednoduchý charakter klíčová slova definovat proměnnou (var), nebo konstantní (let). Dalším bezpečnostním prvkem je, že ve výchozím nastavení Swift objektů nikdy

nemůže být nulový. Ve skutečnosti, Swift bude zastaven, aby se nebyl použít (nil) objekt s chybou. To dělá psaní kódu mnohem čistším a bezpečnějším.

- **Read-Eval-Print-Loop (REPL)** – Na LLDB ladění konzole v Xcode obsahuje interaktivní verzi Swift jazyka. Použití Swift syntaxe hodnotí a komunikuje se spuštěnou aplikací, nebo napsat nový kód. K dispozici v rámci Xcode konzole nebo v Terminálu.
- **Objective-C Interoperability** – Můžete vytvořit zcela novou aplikaci s Swift, nebo začít používat Swift kód a implementovat nové vlastnosti a funkce ve vaší aplikaci. Swift kód má přístup k Objective-C souborů ve stejném projektu, s plným přístupem k vašim Objective-C API.
- **Rychlý a výkonný** – Od začátku byl Swift postaven jako rychlý. Pomocí vysokých výkonů LLVM kompilátoru je Swift kód je transformován do optimalizovaného nativního kódu, který získá co nejvíce z moderního hardwaru. Syntaxe a standardní knihovny byly také naladěny. Swift je nástupce od C a Objective-C jazyky. To zahrnuje nízké úrovně primitiva, jako jsou typy řízení toku a operátorů. To také poskytuje objektově-orientované zvláštnosti, jako jsou třídy, protokoly a generika, což Cocoa a Cocoa Touch vývojáři dosahují výkonu, který požadují [9].

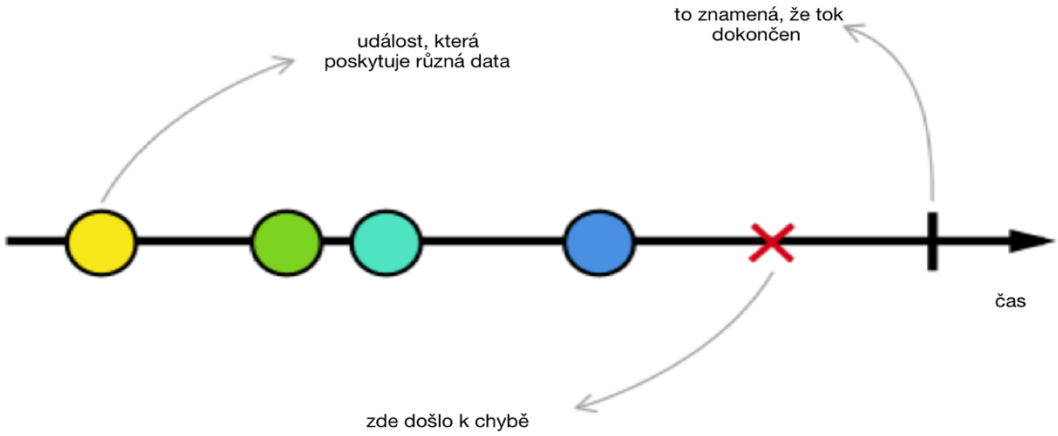
3.9.4 Reaktivní programování (RxSwift)

Reaktivní programování je programování s asynchronními datovými toky. Toky jsou všudypřítomné, cokoli může být tok: proměnné, uživatelské vstupy, vlastnosti, cache, data, struktury.

Tok je sled probíhajících událostí v určitém čase. Tok může vydávat tři různé věci: hodnotu (nějakého typu), chybu nebo „dokončeno" signál. Domnívají se, že „dokončení" se koná například tehdy, když aktuální okno nebo zobrazení obsahující toto tlačítko je uzavřeno. Zachycují tyto vysílané události pouze asynchronně tím, že definují funkci, která se spustí, když se hodnota provádí. Další funkce při chybě a poslední funkce, když je „dokončeno". Někdy lze tyto poslední dva vynechat a můžete se soustředit jen na vymezení funkce pro hodnoty.

Tok může být použit jako vstup do dalšího. Ještě více toků může být použito jako vstupy pro další toky. Můžete sloučit dva toky a filtrovat je. Můžete mapovat datové hodnoty z jednoho toku do druhého, nového.

Obrázek 9:



Zdroj: Vlastní zpracování

RxSwift – je verze Swift s reaktivním rozšířením, která je napsaná na něm samotném [11].

4 Vlastní práce

4.1 Logický návrh

Designu jde o pochopení podstaty vašeho produktu, jeho funkčnosti a také o navrhování produktů užitečných pro lidi. Za tímto účelem bylo vybráno šest hlavních bodů:

1. User-flow (vývojový diagram aplikace) byly navrženy pro každou obrazovku.
2. Nakresleny náčrty obrazovek (wireframes).
3. Vybrány vhodné vzory a barevné palety.
4. Vytvořeny prototypy a design (mock-ups).
5. Sestaven interaktivní prototyp aplikace a lidé ji ocenili.
6. Poslední retušování prototypu bylo provedeno a obrazovky vylepšeny tak, aby byly všechny připraveny k vývoji.

4.1.1 User-flow

Je velmi užitečné, protože poskytuje logickou představu o tom, jak by aplikace měla fungovat a vyřešit problém.

Obvykle se user-flow skládá ze tří typů tvarů:

1. Obdélníky – slouží k zobrazení obrazovky.
2. Kosočtverce – používá se pro podmínky (například stisknutí tlačítka přihlášení, zoom).
3. Šipky – spojují obrazovky a podmínky.

4.1.2 Náčrtky obrazovek (Wireframes)

To jsou v podstatě rychlé osnovy aplikace. Náčrt, schéma, kde budou umístěny obrázky, zkratky, tlačítka. Jedná se o hrubý náčrt toho, jak by měla aplikace fungovat.

4.1.3 Vzory a barevné palety

Barva – je důležitý aspekt designu jako celku. Například první věc, kterou uživatelé vidí, je ikona aplikace, poté startovní okno, které již může vytvořit dobrý, nebo špatný dojem o aplikaci.

4.1.4 Prototypy a design

Jedná se o poslední položku, která již staví kompletní model aplikací a konečný výběr barev pomocí různých komentářů třetích stran. Po dokončení této části můžete již začít pracovat v Xcode.

4.2 Grafický návrh

Grafický návrh je vytvořen na základě požadavků různých lidí, kteří rozhodli, jakou barvu by chtěli vidět v této aplikaci. Byla poskytnuta volba čtyř různých barev, dvě světlé a dvě tmavé. Všichni se přiklonil k jasným barvám, s nimiž už bylo možné pracovat a ukázat jim dvě různá rozvržení. Nakonec byla vybrána na pozadí písková barva, konkrétně RGB (255, 216, 160) (#FFD8A0 v hexadecimálním zápisu).

Na první stránce, kterou uživatel vidí při spuštění (Lauch Screen), byla zvolena obrazovka s růžovou barvou a bílým textem, ve kterém je napsán název aplikace „finanční kniha" a tlačítko s nadpisem „start". Rovněž byl přidán Navigation bar (navigační lišta) u všech ViewControllerů, který je v azurové barvě, konkrétně RGB (0, 240, 254) (#00F0FE v hexadecimálním zápisu). Kombinace jasných, ale současně klidných barev je ideální volbou pro design.

Následující obrazovky již plně odpovídají logickému designu, který byl popsán v kapitole výše. Všechny další obrazovky a prvky používají čtyři různé barvy: černou, bílou, světle růžovou a středně světlý odstín zelené. Černá barva se používá pro všechny nadpisy a tlačítka zpět, růžová, světlezelená barva a bílá pro normální zvýraznění pozadí.

4.3 Implementace

4.3.1 IDE Xcode

Integrované vývojové prostředí (IDE) společnosti Apple, která poskytuje vývojářům nástroje pro tvorbu aplikací pod iPhone, iPad, Mac, Apple Watch a Apple TV. Poslední verze Xcode 10 je k dispozici ke stažení zdarma. Xcode se spouští pouze na počítačích s operačním systémem OS X (iMac, Macbook a Mac Mini).

V Xcode IDE používá systém sdílení dat aplikace Model View Controller pro segmentaci jednotlivých vrstev aplikace,takže je snadnější provádět změny kódu. Například vrstva UI je rozdělena nástroji jako nový "„Interface Builder", s jeho pomocí můžete umístit vizuální ovládací prvky na obrazovce. Auto Layout umožňuje dynamicky

řídit prezentaci objektů pro obrazovky různých velikostí. Pomocí „Storyboard" jsou pohodlně umístěny obrazovky aplikace. Režim „Preview" rychle ukáže, jak vypadají obrazovky aplikace. Žádný z těchto nástrojů se nedotýká softwarového kódu, který vytvoříte.

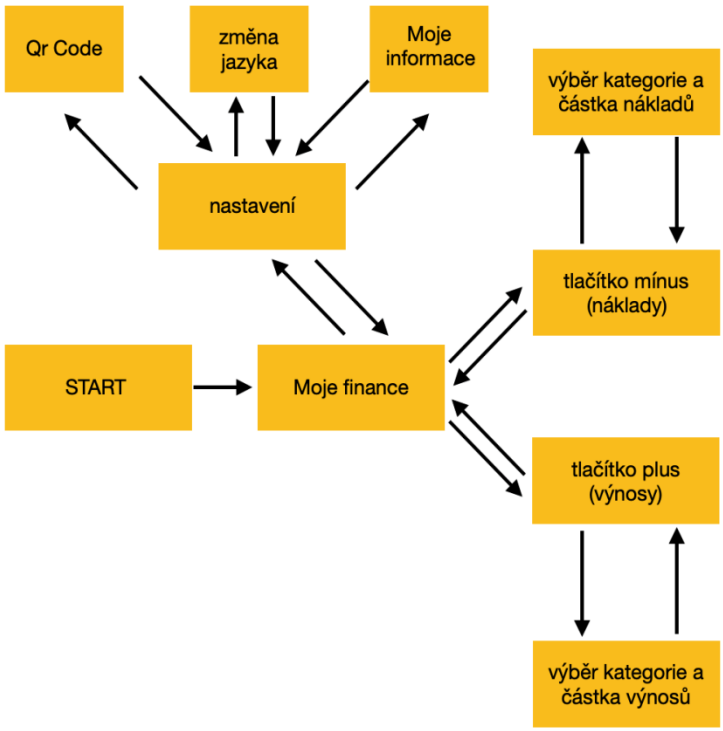
Xcode 10 je radikálně rychlá verze, obsahuje téměř vše, co potřebujete k vývoji aplikací pro všechna zařízení Apple,zejména nové editorské rozšíření. Volba „Runtime Issues" upozorňuje na chyby, které Xcode automaticky detekuje. „Thread Sanitizer" sleduje změny dat a další chyby. Kontrola rozhraní provádí „View Debugger″ – aktualizační nástroj s vysokou vizuální přesností. „Memory Debugger" upozorňuje na „úniky paměti" a skryté chyby.

S Xcode mohou pracovat individuální vývojáři. Programový kód je ověřen v úložišti Git. Podporovány jsou koncepty kontinuální integrace a testovací nástroje.

4.3.2 Struktura architektury

Pro aplikaci byla zvolena architektura MVC.

Obrázek 10:



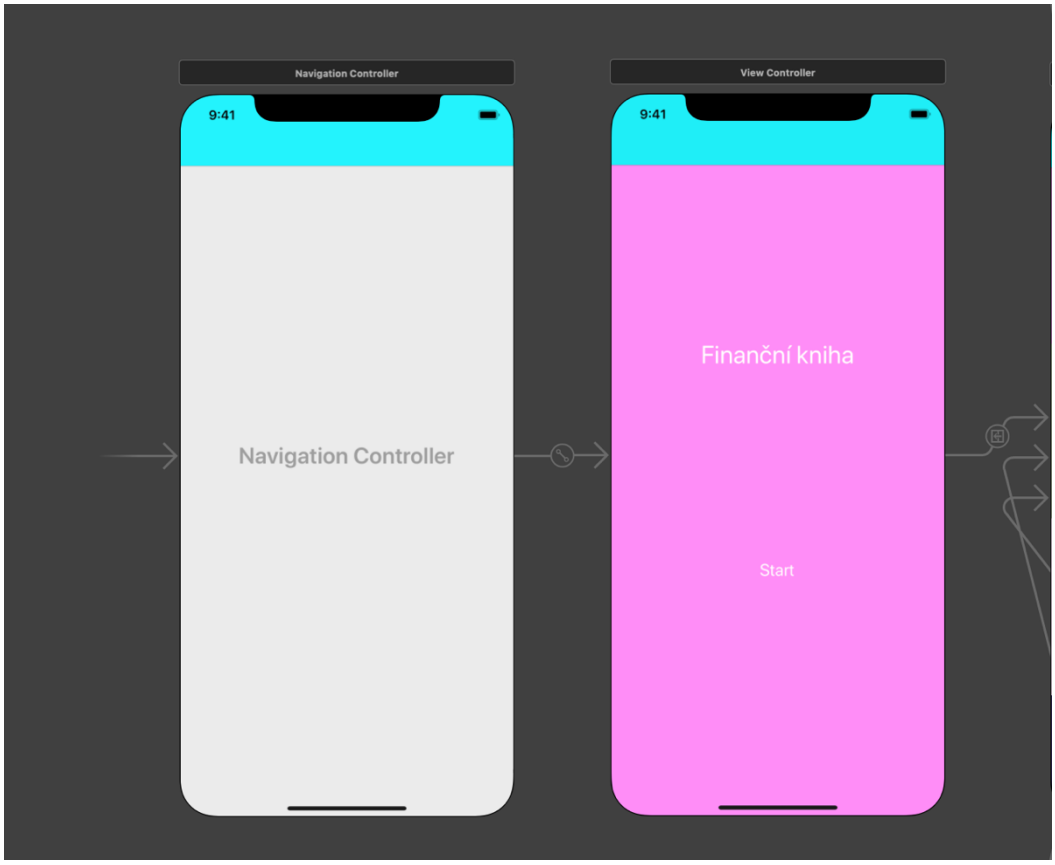
Zdroj: Vlastní zpracování

Na diagramu výše můžeme vidět obrazovky aplikace a přechody mezi nimi. Jsou spojeny pomocí tlačítek a hlavním „Navigation Controller".

4.3.3 Začátek a popis aplikace

Úplně první, co bylo uděláno je práce s první obrazovkou a připojením k ní „Navigation Controller", s jehož pomocí byly spojeny všechny obrazovky.

Obrázek 11:



Zdroj: Vlastní zpracování

4.3.4 Navigation Controller (Navigační ovládač)

UINavigationController – třída vysoké úrovně abstrakce, obsahuje hierarchii ostatních ovladačů (obrazovky/UIView. Obsahuje navigační lištu „UINavigationBar". Navigační lišta, která se zobrazí na obrazovce a odpovídajícím způsobem změní obsah dané lišty v závislosti na aktivním ovladači.

V každém okamžiku můžeme z aktivního regulátoru získat libovolnou požadovanou jednotku:

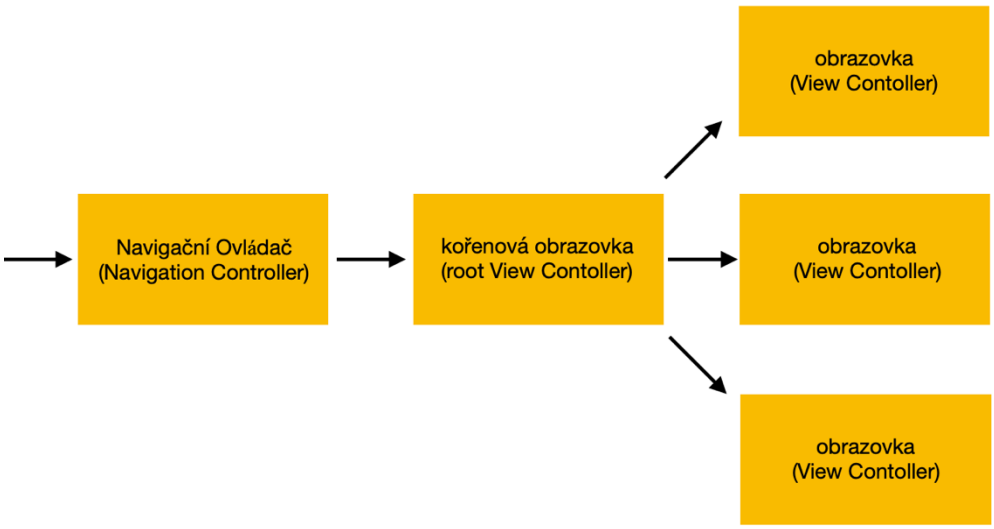
Obrázek 12:

```
self.navigationItem
self.navigationController.navigationBar
```

Zdroj: Vlastní zpracování

Hierarchická struktura:

Obrázek 13:



Zdroj: Vlastní zpracování

Obrázek 14:

```
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)

    // nastavení pomocí navigační lišty
    let titleLabel = UILabel()
    titleLabel.text = "Moje finance"
    titleLabel.textColor = UIColor.black
    titleLabel.font = UIFont.systemFont(ofSize: 25)
    titleLabel.font = UIFont.boldSystemFont(ofSize: 25)
    navigationItem.titleView = titleLabel
```

Zdroj: Vlastní zpracování

Výše uvedený kód ukazuje, jak naprogramovat změnu názvu v navigační liště. Přidání názvu, změna velikosti a barvy. V tomto případě se používá třída UILabel – zobrazení, které zobrazuje jeden nebo více řádků pouze pro čtení textu, se často používá ve spojení s ovládacími prvky a popisuje jejich účel.

Obrázek 15:

```
navigationItem.setHidesBackButton(true, animated: false)
let button  = UIButton(type: .system)
button.setImage(, for: .normal)
navigationItem.rightBarButtonItem = UIBarButtonItem(customView: button)
button.tintColor = UIColor.black
button.addTarget(self, action: #selector(goToNewViewController), for: .touchUpInside)
```

Zdroj: Vlastní zpracování

Zde je rovněž práce s „Navigation bar", ale už jde o nastavení tlačítka, která jsou viditelná pouze při spuštění aplikace. Kompletní dekorace + obrázek. V tomto případě se používá třída UIButton – ovládací prvek, který spustí vlastní kód v reakci na uživatelské interakce. Zde je také přechod na jinou obrazovku pomocí identifikátoru, který odpovídá názvu a používá volič.

Zde byl představen základní kód z „Navigation bar", který byl použit v dalších obrazovkách, protože jeho hlavní funkcí je být v horní části obrazovky azvýraznit název a tlačítka, s jehož pomocí se pohybujeme v aplikaci.

4.3.5 ViewController (VC)

Objekt, který spravuje zobrazení hierarchie pro vaše UIKit aplikace (UIViewController). V UIViewController třída definuje chování, které je společné pro všechny obrazovky. Cíle VC:

1. Aktualizace obsahu obvykle v reakci na změny na podkladová data.
2. Reagovat na uživatelské interakce s výhledem.
3. Změna velikosti zobrazení a řízení rozložení celkového rozhraní.
4. Koordinace s jinými předměty v aplikaci.

4.3.5.1 Storyboard

Storyboardy byly zavedeny jako způsob, jak předělat design rozhraní pro iOS. V době, kdy Apple nepřidali příliš mnoho funkcí, které nebyly k dispozici s XIBs, ale v následujících prezentacích Apple, již byly vynalezeny užitečné funkce: vodítka rozložení.

Všechny nové iOS projekty mají alespoň jeden Storyboard, jenž je připraven k použití: Main.storyboard. Uvnitř si můžete vytvořit hodně rozhraní, z nichž každé představuje jeden View Controller ve vaší aplikaci. Pak můžete navrhnout v „segue" – přechod mezi VC – vše bez jediného řádku kódu.

4.4 Testování aplikace

Testování aplikace bylo provedeno u šesti uživatelů, pomocí kterých byly nalezeny chyby, nedostatky, problémy. To celé bylo opraveno a vylepšeno, například velikosti tlačítek a přizpůsobení pro všechny nejnovější iPhones.

5 Závěr

V dnešní době jsou mobilní aplikace prostě nenahraditelné, protože 90 % lidí má mobily. Téměř veškerá současná technika je s nimi spojena a je vyvíjen nekonečný počet aplikací, s jejichž pomocí můžeme snadněji ovládat techniku. Například můžeme ovládat ledničku, vysavač, vytápění domu prostřednictvím telefonu atd. Každý člověk používá mobil pro různé záležitosti, zcela nahrazuje poznámkový blok s perem.

Proto jsem vytvořil aplikaci, ve které můžete zaznamenat všechny své příjmy a náklady, protože téměř nikdo nebude zapisovat všechno na papír, nebo si to nebude pamatovat. Pro začátek jsem studoval a napsal celý grafický a logický design, vybral architekturu a napsal jsem aplikaci. Pro to všechno byla použita nejnovější verze Xcode 10.1 a programovací jazyk Swift 4.2.

Všechny použité barvy v aplikaci byly vybrány s pomocí hodnocení uživatelů a jejich komentářů. Při práci jsem se setkal s mnoha problémy, které byly vyřešeny a opraveny.

Vyvinutá aplikace diplomové práce se nevyhnula bez chyb, tak hlavní chyba v adaptací na jiné zařízení, protože příliš hodně malých ikon, které je těžké nastavit velikost. Ale to je jediný problém, který jsem nedokázal vyřešit.

Výhodou této aplikace je, že osoba bude schopna ušetřit peníze, protože bude znát každou vynaloženou korunu. To vše obecně ušetří čas a nervy člověka, jelikož uživatel bude vždy schopen kontrolovat a znát svůj rozpočet. Tato aplikace má pohodlné a snadné ovládání, takže ji pochopí každý uživatel, který ji uvidí poprvé. Pohodlnost, barvy, jednoduchost – v této aplikaci jsou shromážděné všechny tyto vlastnosti.

6 Seznam použitých zdrojů

1. IT slovník (CZ). iOS. *It-slovnik.cz* [online]. © 2018 [cit. 2018–11-23]. Dostupné z: https://it-slovnik.cz/pojem/ios
2. GREBEŇ, D. Kompletní historie iOS: od prvního iPhonu až po iOS 9. *Letemsvetemapplem.eu* [online]. © 6. 3. 2016 [cit. 2018–11-23]. Dostupné z: https://www.letemsvetemapplem.eu/2016/03/06/kompletni-historie-ios/
3. Apple Inc. O aktualizacích iOS 11. *Apple.com* [online]. © 2018 [cit. 2018-10-26]. Dostupné z: https://support.apple.com/cs-cz/HT208067
4. Apple Inc. iOS 12: Dokážeš divvy. *Apple.com* [online]. © 2018 [cit. 2018-11-29]. Dostupné z: https://www.apple.com/cz/ios/ios-12/
5. Pbw(CZ). Architektura iOS. *Pbwcz.cz* [online]. © 2018 [cit. 2018-08-01]. Dostupné z: http://www.pbwcz.cz/iOS/Architektura%20bezpecnosti%20iOS/architektura_ios.html
6. Apple Inc. Developer: Cocoa (Touch). *Apple.com* [online]. © 2018 [cit. 2016-02-08]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html
7. ORLOV, B. iOS Architecture Patterns. *Medium.com* [online]. © 28.11.2015 [cit. 2018-04-12]. Dostupné z: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52
8. ALESSI, P. *Vývoj her pro iPhone a iPad.* Brno: Zoner Press. ISBN 978-80-7413-199-8.
9. Apple Inc. Developer: Swift. *Apple.com* [online]. © 2019 [cit. 2019-01-20]. Dostupné z: https://developer.apple.com/swift/
10. Apple Inc. Developer: Xcode. *Apple.com* [online]. © 2019 [cit. 2019-01-20]. https://developer.apple.com/xcode/
11. PILLET, F. et al. *RxSwift: Reactive Programming with Swift* 2nd ed. New York: Razeware LLC, 2017. ISBN 9781942878469.
12. EIDHOF, C. App Architecture: iOS Application Design Patterns in Swift. Scotts Valley, CA: CreateSpace Independent Publishing Platform, 2018. ISBN 978-1719030250.

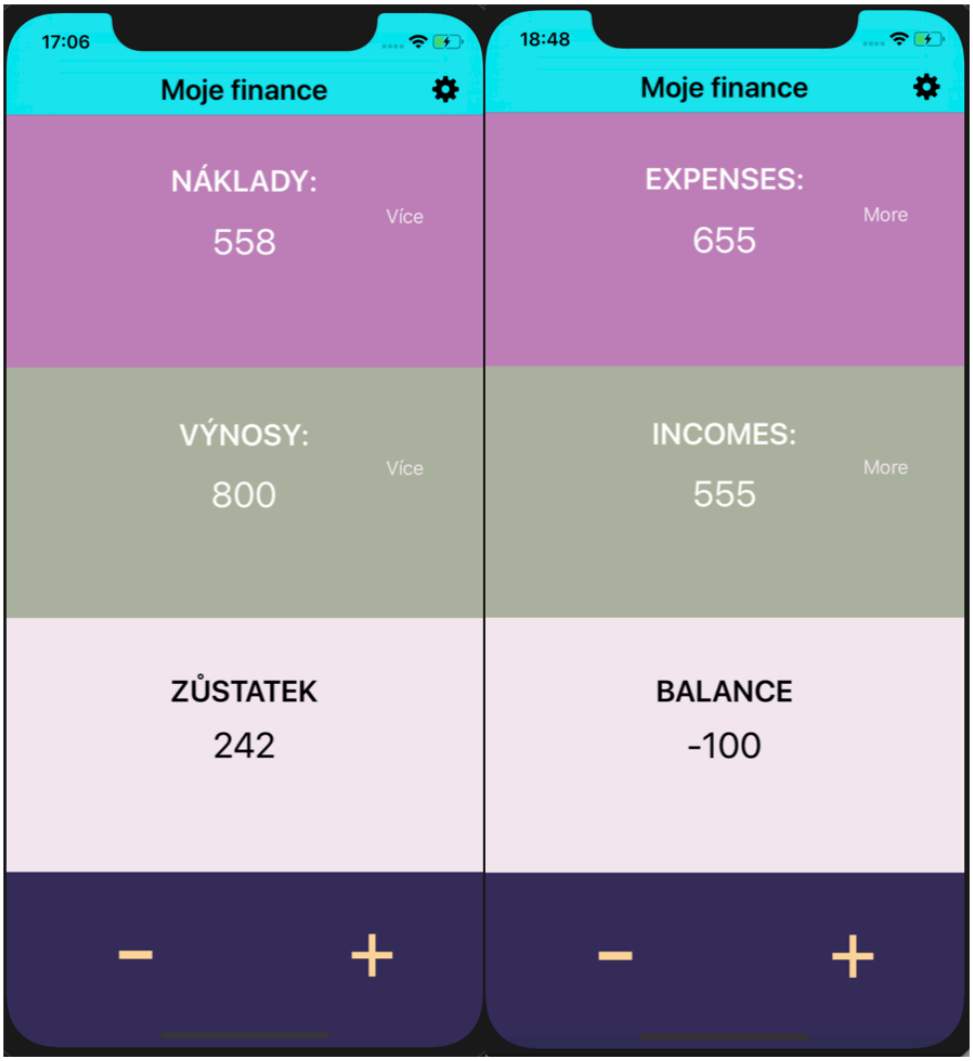
7 Přílohy

Příloha č. 1

Příloha č. 2

```
import UIKit


class ViewController: UIViewController {
    private func setupNavigationBarItems(){

    }
    override func viewDidLoad() {
        super.viewDidLoad()
        setupNavigationBarItems()
    }
}
```

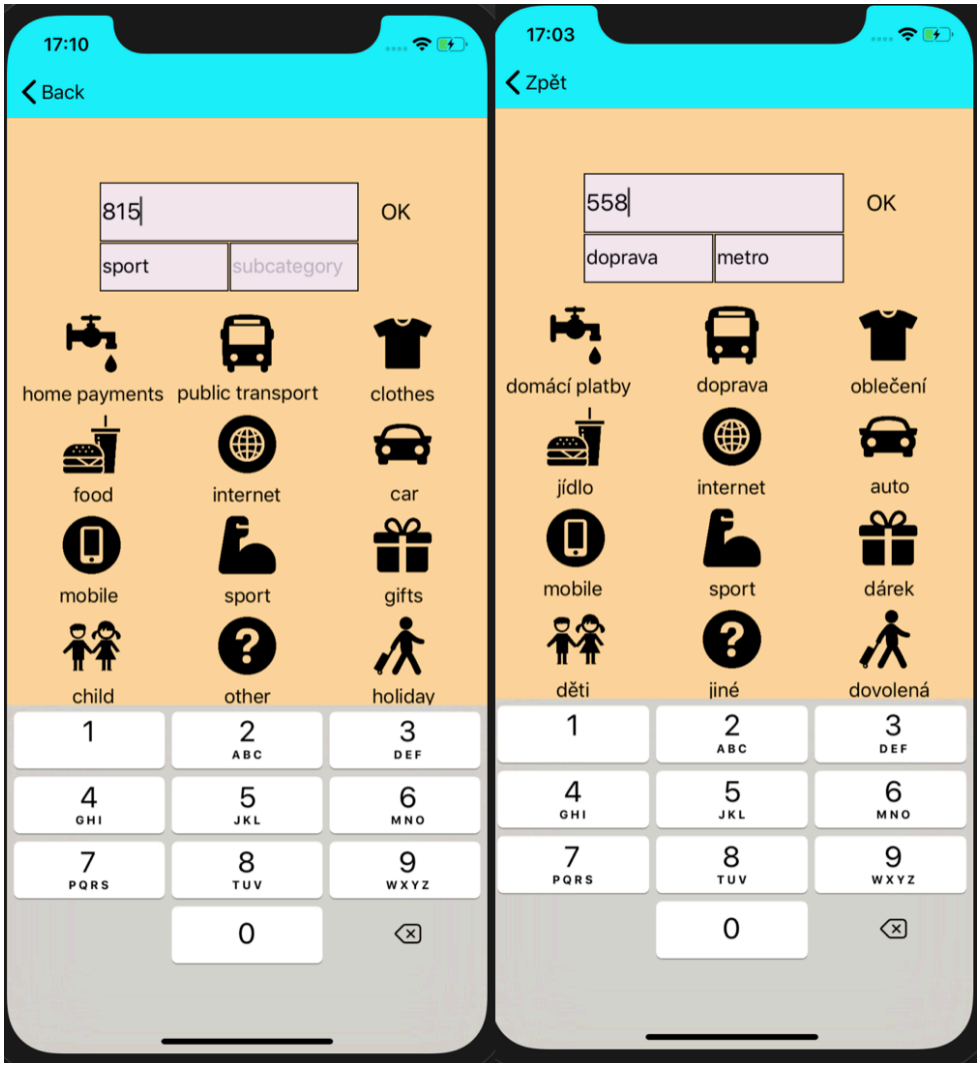
Příloha č. 3


17:06
Moje finance
NÁKLADY:
558
Více
VÝNOSY:
800
Více
ZŮSTATEK
242
18:48
Moje finance
EXPENSES:
655
More
INCOMES:
555
More
BALANCE
-100

Příloha č. 4

```
    override func viewDidLoad() {
       super.viewDidLoad()
    // zaznamenávám třídu, kterou jsem již vytvořil
    let manager = NumManager.default
    displayOne.text = manager.one.description
    displayTwo.text = manager.two.description

        guard let num1 = Int(displayOne.text!), let num2 = Int(displayTwo.text!) else {
            return
        }
       let sum = num2 - num1
       displayThree.text = sum.description
    }
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        // nastavení pomocí navigační lišty pro název
        let titleLabel = UILabel()
        titleLabel.text = "Moje finance"
        titleLabel.textColor = UIColor.black
        titleLabel.font = UIFont.systemFont(ofSize: 25)
        titleLabel.font = UIFont.boldSystemFont(ofSize: 25)
        navigationItem.titleView = titleLabel

      //nastavení pomocí navigační lišty pro tlačítko
        navigationItem.setHidesBackButton(true, animated: false)
        let button  = UIButton(type: .system)
        button.setImage(♨, for: .normal)
        navigationItem.rightBarButtonItem = UIBarButtonItem(customView: button)
        button.tintColor = UIColor.black
        button.addTarget(self, action: #selector(goToNewViewController), for: .touchUpInside)
}


    // umožňuje zobrazit na jiné obrazovce částku, kterou jsme zadali pomocí ID
    @IBAction func pushButton(_ sender: Any) {

        guard Int(displayOne.text!) != nil else {
            return
        }
        performSegue(withIdentifier: "displayLabel", sender: nil)
    }

    //přechod na jinou obrazovku pomocí ID
    @objc func goToNewViewController() {
        performSegue(withIdentifier: "goToNewViewController", sender: self)
    }
}
```

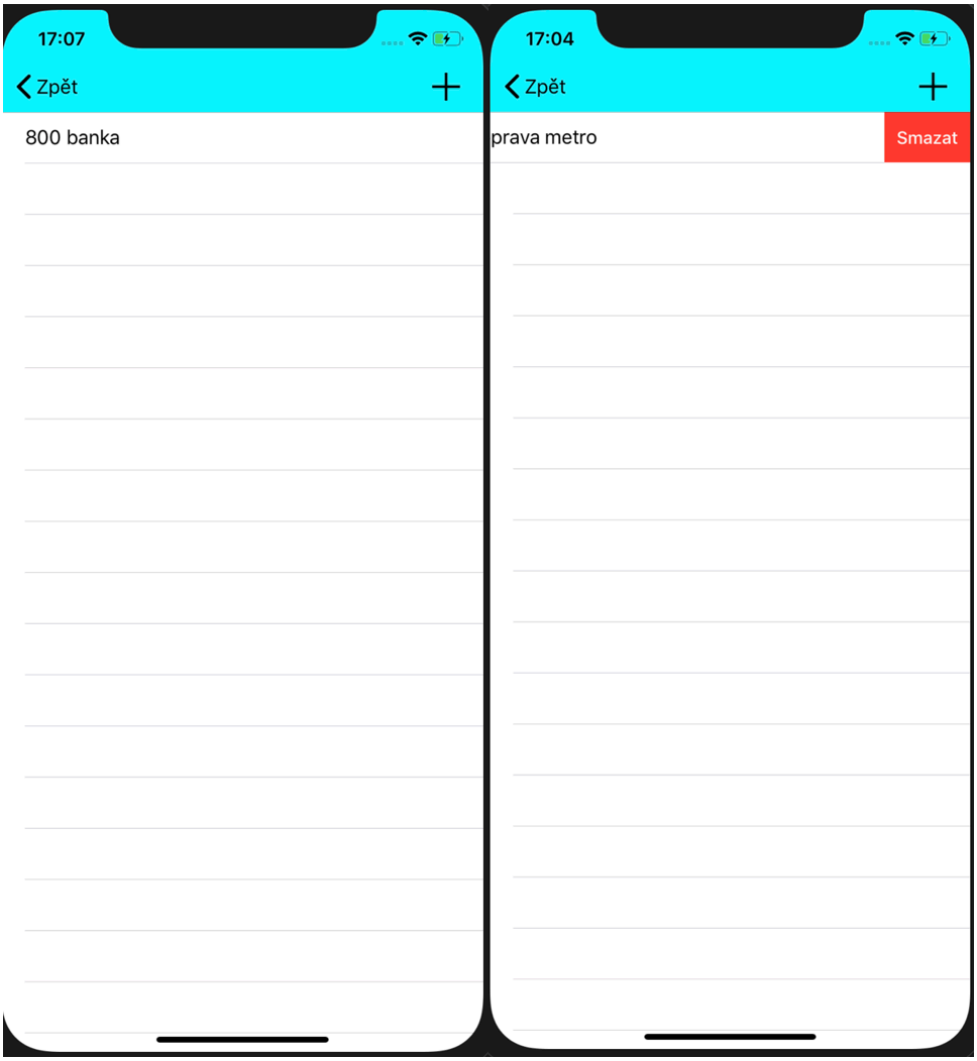
Příloha č. 5



Příloha č. 6

```
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

      numOne.becomeFirstResponder()

    }


    @IBAction func textButton(_ sender: AnyObject){
        textField.text = arrayOfDate[sender.tag]
    }


    @IBAction func pushButton(_ sender: Any) {
        guard let one = Int(numOne.text!) else { return }
        let manager = NumManager.default
        manager.one += one
        performSegue(withIdentifier: "nextOne", sender: nil)
        array.append(numOne.text! + " " + textField.text! + " " + difText.text!)

         }
}
```

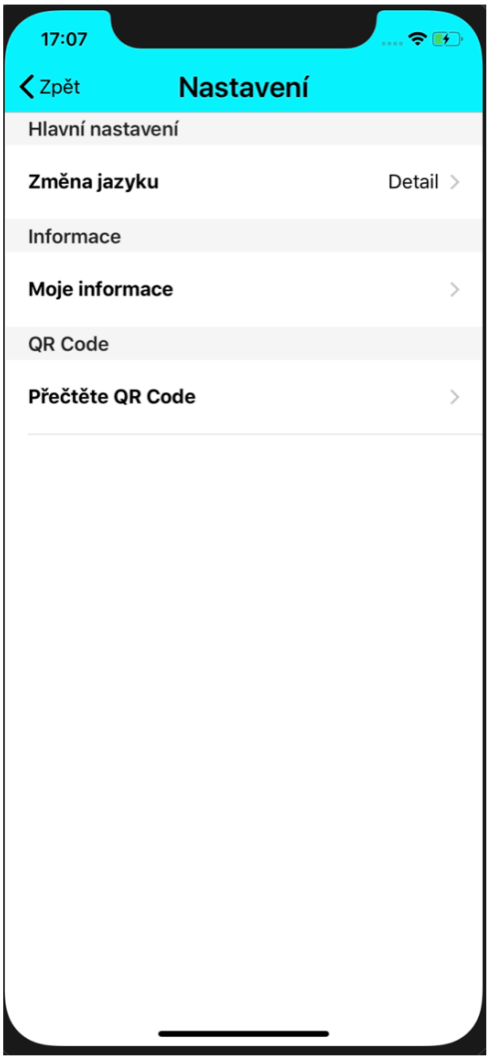
Příloha č. 7



Příloha č. 8

```
import UIKit

var array = [String]()

class displayViewController: UIViewController, UITableViewDelegate,UITableViewDataSource {

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return array.count
    }
    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell: UITableViewCell = tableView.dequeueReusableCell(withIdentifier: "cell")! as UITableViewCell

        cell.textLabel?.text = array[indexPath.row]

        return cell
    }
    override func viewDidLoad() {
        super.viewDidLoad()
    }
    func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
        guard editingStyle == .delete else {return}
        array.remove(at: indexPath.row)
        tableView.deleteRows(at: [indexPath], with: .automatic)
    }
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        let button  = UIButton(type: .system)
        navigationItem.rightBarButtonItem = UIBarButtonItem(customView: button)
        button.tintColor = UIColor.black
        button.setImage(, for: .normal)
        button.addTarget(self, action: #selector(selectOne), for: .touchUpInside)
        }

    override func viewWillAppear(_ animated: Bool) {

        myTableView.register(UITableViewCell.self, forCellReuseIdentifier: "cell")
        myTableView.reloadData()
    }
    @objc func selectOne() {
        performSegue(withIdentifier: "selectOne", sender: self)
    }

    @IBOutlet var myTableView: UITableView!

}
```

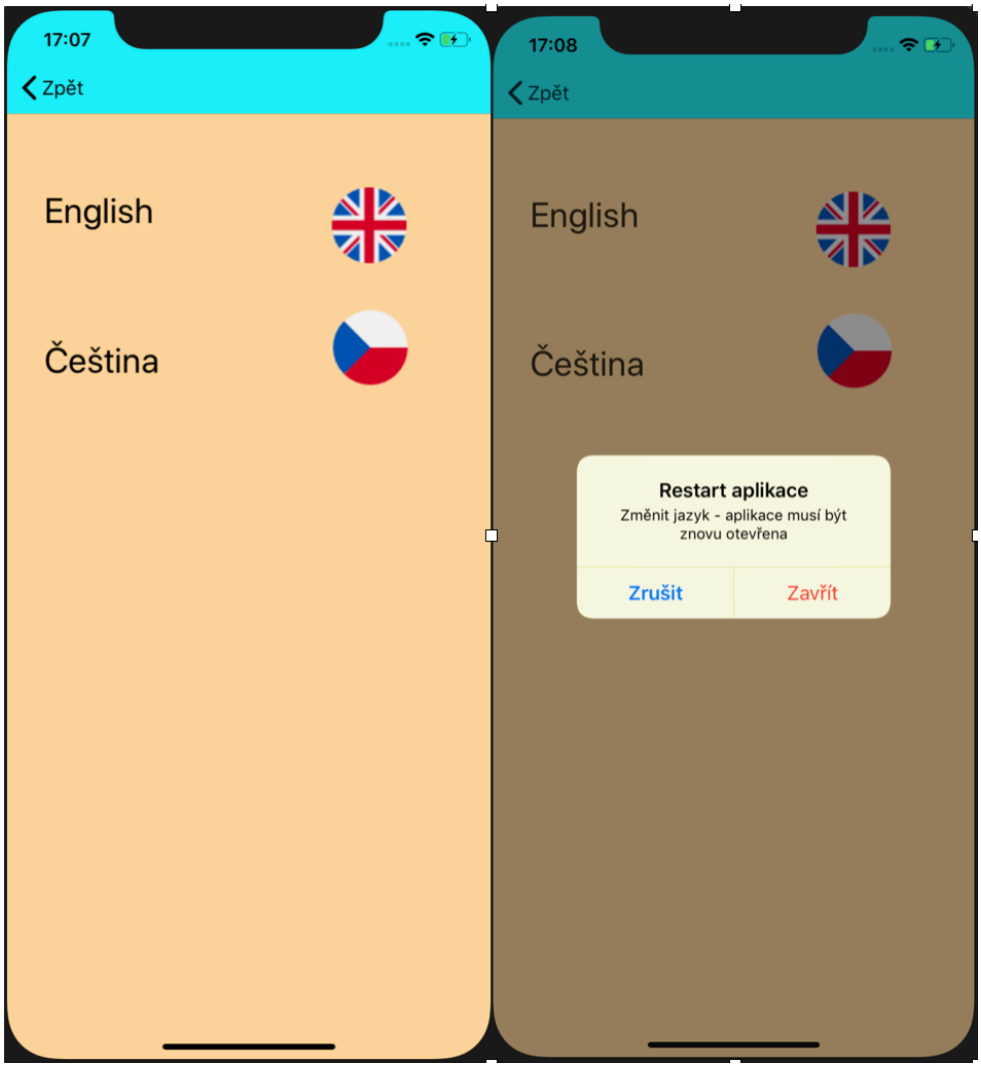
Příloha č. 9


17:07
Zpět
Nastavení
Hlavní nastavení
Změna jazyku
Detail
Informace
Moje informace
QR Code
Přečtěte QR Code

Příloha č. 9 (implementace QR Code)

```
import UIKit
import AVFoundation

class QRScannerController: UIViewController, AVCaptureMetadataOutputObjectsDelegate {

    @IBOutlet weak var frame: UIImageView!

    //zobrazuje video uživateli, když chce použít fotoaparát pro qr code
    var video:AVCaptureVideoPreviewLayer!


    override func viewDidLoad() {
        super.viewDidLoad()
        //vytvoření relace
        let session = AVCaptureSession()
        //identifikovat zachycovací zařízení
        guard let captureDevice =  AVCaptureDevice.default(for: .video) else { return }
        let input: AVCaptureDeviceInput

        do
        {
            input = try AVCaptureDeviceInput(device: captureDevice)
            session.addInput(input)
        }
        catch
        {
           print("ERROR")
        }
        let output = AVCaptureMetadataOutput()
        session.addOutput(output)
        output.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)
        output.metadataObjectTypes = [.qr]

        video = AVCaptureVideoPreviewLayer(session: session)
        video!.frame = view.layer.bounds
        view.layer.addSublayer(video)
        self.view.bringSubviewToFront(frame)
        session.startRunning()
        }
    func metadataOutput(_ output: AVCaptureMetadataOutput, didOutput metadataObjects: [AVMetadataObject], from connection:
        AVCaptureConnection) {
        if metadataObjects.count != 0{
            if let object = metadataObjects[0] as? AVMetadataMachineReadableCodeObject{
                if object.type == .qr{
                    let alert = UIAlertController(title: "QR Code", message: object.stringValue, preferredStyle: .alert)
                    alert.addAction(UIAlertAction(title: "Retake", style: .default, handler: nil))
                    alert.addAction(UIAlertAction(title: "Copy", style: .default, handler: { (nil) in
                        UIPasteboard.general.string = object.stringValue
                }))
             present(alert, animated: true, completion: nil)
            }
        }
```

Příloha č. 10


17:07
Zpět
English
Čeština
17:08
Zpět
English
Čeština
Restart aplikace
Změnit jazyk - aplikace musí být znovu otevřena
Zrušit
Zavřít

Příloha č. 11

```swift
import UIKit

class ChangeLanguageViewController: UIViewController {

    @IBOutlet weak var EngButton: UIButton!
    @IBOutlet weak var CzButton: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    @IBAction func EnClick(_ sender: Any) {
       self.changeToLang("en")
        }

    @IBAction func CzClick(_ sender: Any) {
        self.changeToLang("cs")
    }


 private func changeToLang(_ langCode: String){
    if Bundle.main.preferredLocalizations.first != langCode{
//    if NSLocale.preferredLanguages.first != langCode{
        let confirmAlertCtrl = UIAlertController(title: NSLocalizedString("restartTitle", comment: ""), message:
            NSLocalizedString("restart", comment: ""), preferredStyle: UIAlertController.Style.alert)

        let confirmAction = UIAlertAction(title: NSLocalizedString("close", comment: ""), style: .destructive){ _ in
            UserDefaults.standard.set([langCode], forKey: "AppleLanguages")
            UserDefaults.standard.synchronize()
            exit(EXIT_SUCCESS)
            }
        confirmAlertCtrl.addAction(confirmAction)

        let cancelAction = UIAlertAction(title: NSLocalizedString("cancel", comment: " "), style: .cancel, handler: nil)
        confirmAlertCtrl.addAction(cancelAction)

       present(confirmAlertCtrl, animated: true, completion: nil)

    }
  }
```

Příloha č. 12 (lokalizace aplikace)