**EKONOMICKÁ UNIVERZITA V BRATISLAVE**

**FAKULTA HOSPODÁRKEJ INFORMATIKY**

Evidenčné číslo:103004/I/2014/1399147131

**Vývoj mobilnej aplikácie pre platformu iOS určenej na zobrazovanie rozvrhov.**

**Diplomová práca**

**2014 Roman Teličák**

**EKONOMICKÁ UNIVERZITA V BRATISLAVE**

**FAKULTA HOSPODÁRKEJ INFORMATIKY**

**Vývoj mobilnej aplikácie pre platformu iOS určenej na zobrazovanie rozvrhov.**

**Diplomová práca**

**Študijný program:** Manažérske rozhodovanie a informačné technológie

**Študijný odbor:** 6258 Manažérske rozhodovanie a informačné technológie

**Školiace pracovisko:** Katedra aplikovanej informatiky

**Vedúci záverečnej práce:** Ing. JánPittner, PhD.

**Bratislava 2012** **Roman Teličák**

**ZADANIE ZÁVEREČNEJ PRÁCE**

**Čestné vyhlásenie**

**Čestne vyhlasujem, že záverečnú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.**

**Dátum: ......................................................**

(podpis študenta)

Poďakovanie

Touto cestou by som sa chcel poďakovať vedúcemu diplomovej práce Ing. Jánovi Pittnerovi, PhD. za odbornú pomoc, venovaný čas a cenné rady, ktoré mi poskytol pri jej vypracovaní.

ABSTRAKT

TELIČÁK Roman: *Vývoj mobilnej aplikácie pre platformu iOS určenej na zobrazovanie rozvrhov.* Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky, Manažérske rozhodovanie a informačné technológie. Katedra Aplikovanej Informatiky. – Vedúci záverečnej práce: Ing. Ján Pittner, PhD. –Bratislava: FHI EU, 2013/2014, x strán.

Cieľom záverečnej práce je vytvoriť mobilnú aplikáciu pre platformu iOS, ktorá bude slúžiť na zobrazovanie rozvrhov. V práci si priblížime exitujúce systémy na zobrazovanie rozvrhov. Následne si priblížime trh s mobilnými zariadeniami a ich operačnými systémami na Slovensku, a vo svete. Platforma iOS, jej história, vývoj a jazyk použitý na tvorbu softwaru na tejto platforme bude taktiež spomenutý. Nasledovať bude implementácia návrhu novej aplikácie na zobrazovanie rozvrhu. Práca je rozdelená do štyroch kapitol. Obsahuje x grafov, x tabuliek a x obrázkov. Prvá kapitola je venovaná súčasnému stavu doma a v zahraničí, kde sú uvedené momentálne používané systémy určené na zobrazovanie rozvrhov. Priblížime platformu, na ktorej budeme vyvíjať. Aké poskytuje nástroje na tvorbu softwaru, a samozrejme primárny jazyk a jeho syntax na tvorbu algoritmov. V ďalšej kapitole si stanovíme ciele a požiadavky, ktoré by mala nová aplikácia spĺňať. Tieto ciele majú rôzny charakter. Od používateľských, až po technické a implementačné požiadavky. Obsahom tretej kapitoly bude kódovanie samotnej aplikácie. Budeme sa zaoberať parsovaním zdrojových dát z databázových súborov, napĺňaním internej databázy v mobilnom zariadení týmito dátami a nakoniec ich prezentáciou na používateľskom rozhraní. Dôležité implementačné detaily si priblížime konkrétnymi kódmi a vysvetlíme ich. Súčasťou tejto kapitoly budú aj obrazové prílohy vytvoreného používateľského rozhrania. V poslednej kapitole zhodnotíme výsledky našej práce,a či sme naplnili stanovené ciele.

**Kľúčové slová:**

iOs, iPhone, iOS aplikácie, tvorba iOS aplikácií, Objective-C, Xcode, storyboard, web servis, PHP, CodeIgniter, MySQL, SQLite

# **ABSTRACT**

TELIČÁK Roman: Timetable mobile application developing for iOS platform

University of Economics in Bratislava. Faculty of Business Informatics; Department of Applied Informatics. – Supervisor of the bachelor work: Ing. Ján Pittner, PhD. –Bratislava: FHI EU, 2013/2014, x pages.

The goal of thesis is to create a mobile application based on iOS platform, dedicated to display timetables. We will briefly describe existing systems, which display timetables as well. National and global mobile device market and their operating systems will be a topic of discussion. We will talk about iOS as platform, its history and programmatic language used in order to develop software on this platform. Next we will create an timetable iOS application. Thesis has four chapters and cointains of x charts, x tables and x pictures. The first chapter is devoted to the already existing systems for timetables displaying at home and abroad. Facts about iOS platform, which we are about to develop for will be presented. What tools this platform provides for developers to make their coding more efficient. What programmatic language is used on this platform. What kind of syntax does it use. All of these questons will be answered in this chapter as well. In next chapter the goals of this thesis are specified. Those goals have various nature. From user’s requests on interface, to code implementation. The third chapter is about coding exclusively. We will acquire source data by parsing database files, and later we will insert those data to integrated iPhone database. In later phases of application developing we select bunch of those data to present them on user interface. We will analyze and explain important implementation details within the code. There are also figures of user interface included. The last chapter talks about fulfilling goals of thesis. If we have met the requiremetns of thesis.

**Key words:**

iOs, iPhone, iOS applications, iOS applicationsdeveloping, Objective-C, Xcode, storyboard, web service, PHP, CodeIgniter, MySQL, SQLite

Obsah

Úvod

Zobrazenie rozvrhu je jednou z rutinných činností, ktorá čaká študenta na začiatku každého semestra. Túto operáciu vykonávajú pravidelne rádovo tisíce študentov, väčšinou aj opakovane. Z tohto dôvodu sú čas a námaha venovaná optimalizácií procesu vyhľadania a zobrazenia rozvrhu na mieste. S rozširovaním informačných technológií pribúdajú možnosti prístupu k informáciám. Obdobie, kedy jediným prostriedkom prístupu k informáciám uloženým na webe bol počítač (či už klasický stolový, alebo notebook) sú dávno preč. Momentálnym trendom sú mobilné zariadenia, ako smartphony alebo tablety, ktoré sú schopné nielen pristupovať k informáciám z webu, ale v mnohom zastúpia alebo predčia funkcionalitu ponúkanú klasickými počítačmi. Keďže tieto zariadenia sú rozšírené v značnej miere najmä u mladej generácie, je vytvorenie mobilnej aplikácie, ktorá by slúžila študentom pri každodennej dochádzke správnym ťahom. Potenciál využitia takejto aplikácie je vysoký.

Aplikácia ktorá slúži na zobrazovanie rozvrhu (ale aj každá iná) by sa mala snažiť dosiahnuť čo najväčší používateľský komfort. Používateľským komfortom sa rozumie pohodlie používateľa pri interagovaní s aplikáciou, ktoré je ovplyvnené množstvom rôznych faktorov (tieto faktory sú priblížené v nasledujúcej kapitole).

1. Súčasný stav riešenej problematiky doma a v zahraničí

Možnosti aké má súčasný študent Ekonomickej Univerzity v Bratislave, ak chce zistiť svoj rozvrh je pár.

# 1.1 Lookin

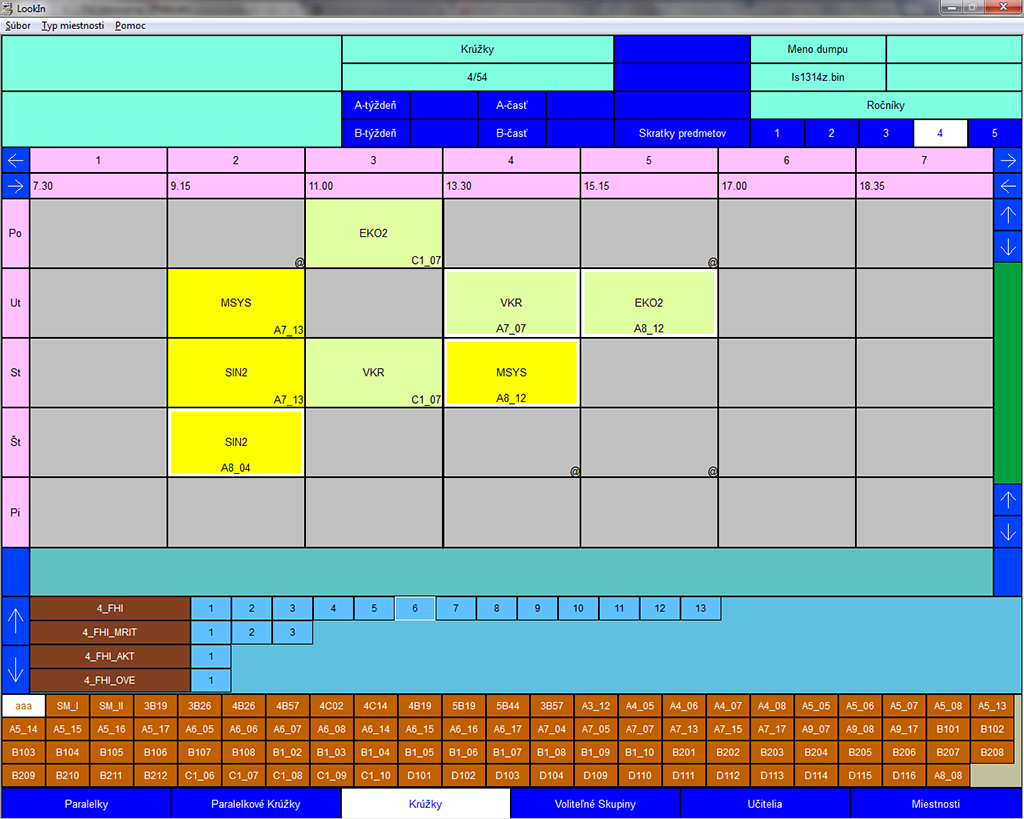
Po zadaní kľúčových slov „euba rozvrh“ do webového vyhľadávača Google ako prvý odkaz smeruje na desktopovú aplikáciu Lookin. Tento program je voľne stiahnuteľný na stránkach Ekonomickej Univerzity v Bratislave v sekcii „Rozvrh výučby“. Návod na nainštalovanie Lookinu je zdokumentovaný v niekoľkostranovom pdf súbore, ktorý je taktiež dostupný na vyššie spomenutej webovej stránke. Tento súbor obsahuje (okrem neproporcionálne roztiahnutých obrázkov, pravopisných chýb a množstva nefunkčných odkazov) krok za krokom postup k úspešnému nainštalovaniu a spusteniu programu Lookin. Keďže program vyžaduje na svoju obsluhu naštudovanie dokumentu, nový používateľ to má náročnejšie.

Po stiahnutí Lookin-u je tento súbor potrebné nakopírovať ho do vhodného priečinku na disku C. To vyžaduje vytvorenie niekoľkých adresárov z toho dôvodu, že tento program používa absolútne adresovanie ciest. Ďalším krokom je stiahnutie databázového súboru z rovnakej stránky, ktorý obsahuje samotný rozvrh pre daný semester. Následne po spustení Lookin-u je potrebné lokalizovať databázový súbor a dať ho načítať. Po tomto kroku sa nám zobrazí rozvrh, ale netreba sa tešiť predčasne, keďže je nevyhnutné krokmi v priloženom návode zvoliť svoju fakultu, ročník, odbor a číslo krúžku. Následným filtrovaním vo výsledkoch si užívateľ dokáže zobraziť aj nepovinné a voliteľné predmety.

Zobrazený rozvrh sa nedá vytlačiť ani exportovať do iných súborov, takže ak si ho chce užívateľ opätovne zobraziť je potrebné prejsť popísanými krokmi znovu (okrem sťahovania a vytvárania cieľových priečinkov samozrejme).

Nie je potrebné pripomínať (keďže cesta ku programu musí byť c:WinRozvrhy/Exe/Lookin.exe), že tento program je dostupný iba pre používateľov operačného systému spoločnosti Microsoft. Takže používatelia iného operačného systému ako Windows si svoj rozvrh nezobrazia.

Študenti túto aplikáciu neobľubujú, ale keďže nemá vhodnú alternatívu, sú naďalej nútení ju používať. Aj napriek tomu, že žijeme v informatizovanej spoločnosti a aplikácie s takouto vysokou využiteľnosťou by sa mali ovládať a vyzerať diametrálne odlišne ako je tomu teraz.



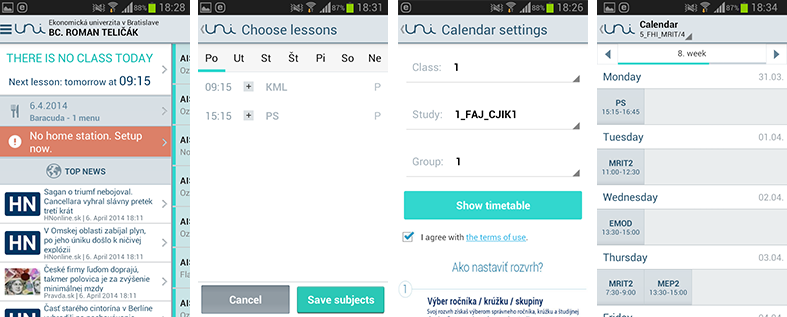
Obrázok 1: Zobrazenie rozvrhu v systéme Lookin

# 1.2 UniApps

Alternatívou Lookinu je komplexná multiplatformová aplikácia UniApps. Po spustení aplikácie je užívateľ vyzvaný na výber univerzity, keďže túto aplikáciu využíva viacero škôl na Slovensku, ale aj v Českej republike. Aplikácia je integrovaná s akademickým informačným systémom a pre jej spustenie je potrebné vygenerovať v AIS v sekcii „Správa používateľov“ kód a následne ho v aplikácií vložiť pre identifikáciu používateľa.

Po prihlásení je potrebné otvoriť ľavé menu a zvoliť položku „Rozvrh“. Zobrazí sa kalendár, ktorý je však prázdny. Vyučujúcimi hodinami ho naplníme tak, že v pravom hornom rohu klikneme na ikonu nastavenia, kde si vyberieme svoj ročník, odbor a skupinu. Zobrazí sa nám kalendár s hodinami, ktoré si pridáme do svojho kalendára kliknutím na plus ikonu, ktorá je umiestnená vedľa každého predmetu. Po ukončení pridávania hodín potvrdíme tento výber tlačidlom hotovo pravom hornom rohu. Ak sa používateľ odhlási z aplikácie a prihlási na inom zariadení, prispôsobený rozvrh mu ostane v platnosti. Aplikácia ponúka okrem zobrazenia rozvrhu aj ďalšiu funkcionalitu ako zobrazovanie headlinov z rôznych médií, plánovanie mhd, menu v jedálňach atď.

Keďže používateľ tejto aplikácie si musí ručne vyselektovať vyučovacie hodiny, ktoré mu prislúchajú, je táto aplikácia bez predchádzajúcej vedomosti o svojom rozvrhu nepoužiteľná. Fakt, že ponúka množstvo informácií rôzneho druhu, je na úkor jej prehľadnosti. Takisto je potrebné spomenúť, že nie všetky možnosti, ktoré nám ponúka sú funkčné, respektíve aktuálne.



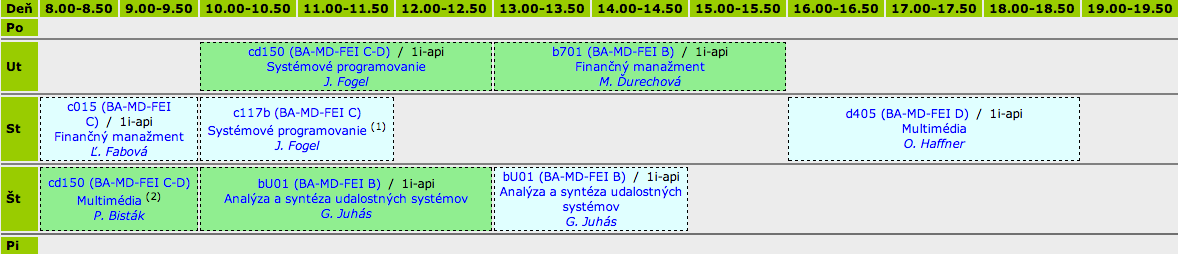
Obrázok 2: Obrazovky z Uniapps aplikácie

# 1.3 Aplikácie na zobrazovanie rozvrhu z iných univerzít

Pre porovnanie som si vybral systém, ktorý používa Slovenská technická univerzita. Ak študent tejto univerzity chce vedieť, aký je jeho osobný rozvrh, musí byť registrovaný v systéme ais. Po prihlásení nájde na svojej domovskej stránke odkaz na „Zobrazenie a tlač rozrvhov“. Po kliknutí na tento link je používateľ odnavigovaný na stránku, kde si môže zobraziť svoj rozvrh, alebo po zadaní príslušných kritérií si zobrazí rozvrh akéhokoľvek iného študenta alebo učiteľa. Je možné si vybrať v akom formáte sa požadovaný rozvrh má otvoriť (HTML, PostScritpt, PDF ...).

Po zobrazení príslušného rozvrhu je možné vidieť detaily jednotlivých hodín, ako názov predmetu, miestnosť a meno vyučujúceho (podobne ako v Lookin). Prednášky su viditeľne odlíšené pomocou rámika okolo jednotlivej hodiny. Po kliknutí na jednotlivú položku, napr. meno učiteľa je používateľ presmerovaný na profil učiteľa, kde môže vykonávať rozličné operácie. Je možné vidieť kontakt na vyučujúceho, aké predmety a kedy vyučuje, jeho/jej publikačnú činnosť, akým prácam je vedúci, resp. oponent a mnohé ďalšie.

Výhodou takéhoto prístupu je, že používateľ si nemusí sťahovať ďalší software, ani databázové súbory každý semester, respektíve ak je zverejnená novšia verzia. Prístup k rozvrhu má priamo vo svojom ais profile, na ktorý sa prihlási z ľubovoľného webového prehliadača.



Obrázok 3: Rozvrh Slovenskej technickej univerzity vo webovom prehliadači

# 

# 1.4 Nová aplikácia

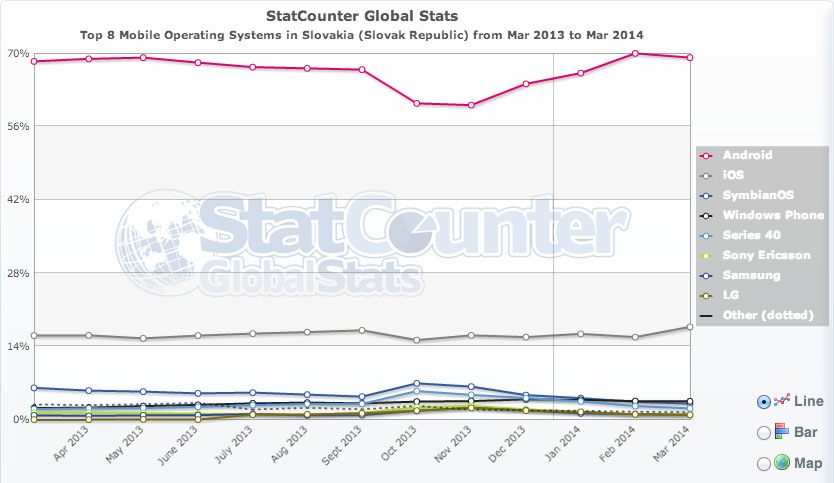
Mojou motiváciou je vytvoriť aplikáciu na prezeranie rozvrhu tak, aby som sa vyvaroval slabých miest vyššie spomenutých riešení a vylepšil ich silné stránky. Aby bolo niečo také vytvoriť je dôležité sa vžiť do úlohy používateľa tejto aplikácie. Čo teda používateľ uprednostňuje a kde sú jeho preferencie pri používaní aplikácií, resp. informačných systémov. Najmä keď cieľovú skupinu používateľov tvorí najmä široká verejnosť, kde počítame s tým, že gramotnosť pri používaní informačných technológií, resp. počítačovej gramotnosti môže byť na nízkej úrovni.

Hlavé pravidlo je: „Nenúť používateľa rozmýšľať!“[[1]](#footnote-1). Používateľ je lenivý tvor, ktorý sa môže v ktoromkoľvek momente ukončiť prácu s aplikáciou a nahradí ju za konkurenčnú. Tomuto sa snažíme predísť za každých okolností. Už na prvej obrazovke po spustení daného softwaru musí byť používateľovi bez dlhšieho zamyslenia jasné čo má robiť, ako pokračovať, aká je jeho úloha. Napríklad: čo má vyplniť do formulára pri registrácií (aké sú povinné a nepovinné polia), vložiť sms kód do textového poľa, alebo ako vykonať jednoduchú akciu akou je vymazanie emailu zo schránky. Na čo sa používajú zaužívané postupy z reálneho života, ako spomínané zmazanie emailu sa vykoná po stlačení ikony s košom. Známy je prípad kedy do rúk dieťaťa vložili tablet s aplikáciou na prezeranie rozprávkových kníh. Aj keď dieťa nikdy predtým takýto nástroj v rukách nedržalo, bez ďalších inštrukcií prirodzene vedelo, ako posúvať zobrazované stránky v aplikácií. Použilo rovnaké gestá ako pri obyčajnej papierovej knihe. Toto je dobrý príklad na test použiteľnosti (tzv. usability testing).

Odliadnuc od testu použiteľnosti, ďalším faktorom komfortu používateľa je čas strávený používaním aplikácie na dosiahnutie konkrétneho výsledku. Tento čas sa návrhári softwaru snažia minimalizovať čo najviac, aby príliš dlhými formulármi a komplikovanou navigáciou používateľovi nekazili dojem a neznechutili jej ďalšie používanie. V našom prípade sa pokúsime znížiť počet klikov potrebných na zobrazenie rozvrhu.

# 1.5 iOS a Android na slovenskom a zahraničnom trhu

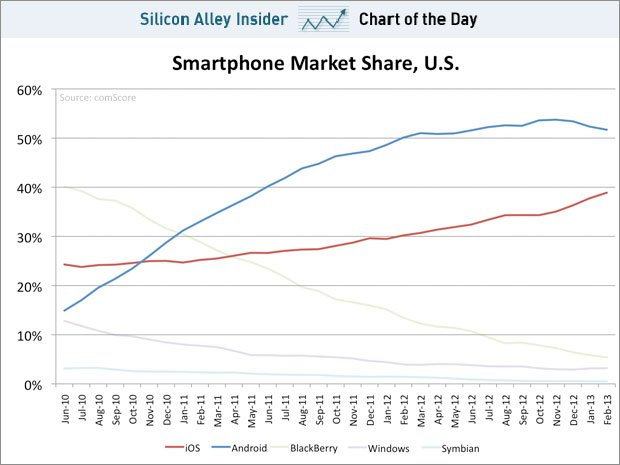
Najnovšia štatistika rozšírenia mobilných operačných systémov na Slovensku hovorí, že majoritný podiel na tomto trhu si s takmer 70% drží operačný systém Android od spoločnosti Google. Nasleduje iOS (mobilný operačný systém inštalovaný v apple produktoch) s 18% a minoritný podiel od jedného do troch percent majú systémy Windows Phone, SymbianOS, Series 40, LG a Sony Ericson (usporiadané zostupne)[[2]](#footnote-2).



Obrázok 4: Porovnanie mobilných operačných systémov na Slovensku

Zdroj: http://gs.statcounter.com/#mobile\_os-US-monthly-201303-201403

Na druhej strane, v Spojených štátoch amerických je podiel iOS a Androidu vyrovnanejší. Operačný systém od Googlu prevyšuje iOS zhruba o necelých 13% a tak dominuje s 51,7 percentami a iOS má 38,9%[[3]](#footnote-3).



Obrázok 5: Porovnanie mobilných operačných systémov v US

Zdroj: <http://www.businessinsider.com/androids-market-share-in-the-united-states-2013-4>

Najväčšou výhodou Androidu oproti iOs je široká škála mobilných zariadení, na ktorých je tento systém nainštalovaný. Keďže zdrojový kód Androidu je vydaný pod licenciou Apache, výrobcovia zariadení, mobilní operátori a vývojári môžu tento kód nielen použiť, ale aj voľne meniť[[4]](#footnote-4). Výrobcovia mobilných zariadení na čele so spoločnosťou Samsung zásobujú trh množstvom telefónov a tabletov, od najlacnejších a najmenej vybavených až po drahé a výkonné. Takýmto prístupom si zabezpečia vedúce postavenie na trhu s mobilnými aplikáciami aj v rozvojových krajinách. Problém Androidu nastáva v jeho aktualizáciách. Keďže každý výrobca si do operačného systému dopĺňa svoje features a výsledkom je fakt, že hoci Google vydá novú verziu Androidu, dostanú sa k nej iba používatelia, ktorí si zakúpia najnovšie zariadenia. Upgradnut Android na starších zariadeniach bez straty záruky je možné iba s dovolením predajcu hardwaru. Nezriedka nastanú situácie, kedy najnovší Android má 5,3%[[5]](#footnote-5) všetkých android zariadení, pričom iOS7 je nainštalovaných na 87% apple zariadeniach.[[6]](#footnote-6)

Na druhej strane, Apple má marketing ako žiadna iná firma na svete. Majitelia svoje produkty milujú,a to najmä pre vertikálnu integráciu, na ktorej funguje všetko takmer dokonale. Zariadenia Apple dokážu osloviť používateľov svojím dizajnom. V tomto ponímaní má značne navrch nad konkurenčným Androidom. Na druhej strane zariadania s iOS si môžu dovoliť ľudia, ktorí nemajú až tak hlboko do vrecka, kedže cena najnovších modelov mobilných telefónov sa pohybuje okolo sumy 700 euro[[7]](#footnote-7).

# 1.6 iOS

iOS (v minulosti iPhone OS) je mobilný operačný systém vyvinutý a distribuovaný spoločnosťou Apple Inc. Tento systém bol na trh uvedený v roku 2007. Pôvodne bol určený pre telefón iPhone. O niekoľko mesiacov neskôr bol rozšírený tak, aby podporoval aj zariadenia iPod Touch (September 2007), nasledovali iPad (Január 2010), iPadMini (November 2012) a z druhej generácie zariadení aj Apple Tv (September 2010). Na rozdiel od Windows Phone (operačný systém pre mobilné zariadenia od spoločnosti Microsoft) a Android (Google), iOS nie je možné nainštalovať na zariadenia iného typu ako Apple. Do októbra 2013 App Store (virtuálny obchod na distribúciu aplikácií postavených na platforme iOS) obsahoval viac ako 1 milión iOS aplikácií, z ktorých polovica bola optimalizovaná pre iPad. Tieto apliácie boli sumárne stiahnuté viac ako 60 miliárd krát. Vo štvrtom kvartáli roka 2012 iOS používalo 21 percent mobilných zariadení na trhu, viedol Android.

Užívateľské rozhranie iOS je založené na koncepte priameho ovládania pomocou dotykových gest. Medzi elementy slúžiace na ovládanie užívateľského rozhrania sú posúvateľné obrazovky, prepínače a tlačidlá. Užívateľ interaguje s aplikáciami pomocou gest, ako sú: posunutie, ťuknutie, stlačenie, z ktorých každé má svoju úlohu v rámci operačné systému iOS. Niektoré aplikácie na detekciu trasenia, alebo rotovania s týmto zariadením používajú akcelerometre vstavané priamo v mobilnom zariadení (výsledkom môže byť otočenie obrazovky z módu portrét na mód krajina).

IOS zdieľa s OS X (UNIX operačný systém určený pre desktopové počítače od Apple) frameworky ako Core Fundation a Fundation. Avšak rozdiel je v UI sade, ktorú používajú. Na rozdiel od iOs, ktoré používa Cocoa Touch, OS X používa Cocoa. Z tohto dôvodu iOS aplikácie nie sú kompatibilné s OS X aplikáciami[[8]](#footnote-8).Hlavné verzie iOS sú vydávané ročne cez iTunes. Aktuálna verzia (7) iOS bola vydaná v 18. septembri 2013[[9]](#footnote-9).

# 1.7 iOS SDK (Software development kit)

V praxi iOS SDK reprezentuje množinu nástrojov určených vývojárom softwaru a aplikácií na platforme iOS. Obsahuje nástroj na odladenie programu (debugger), knižnice, frameworky, simulátory zariadení (iPad, iPhone rôzne verzie), dokumentáciu a ukážky kódu. Pri počiatočnom zverejnení iOS (január 2007) nebolo dovolené tretím stranám vytvárať aplikácie pre túto platformu. Steve Jobs tento krok odôvodnil tak, že developeri môžu vytvárať webové aplikácie, ktoré by vypadali a chovali sa ako iPhone aplikácie. Napriek tomu už 17. októbra toho istého roku Jobs oznámil na oficiálnom blogu Apple, že SDK bude sprístupnené tretím stranám vo februári nasledujúceho roku. SDK bolo vydané 6. Marca 2008 a umožňuje vývojárom vytvárať aplikácie pre iPhone a iPodTouch, a taktiež ich testovať na simulátore daného zariadenia. Ak chce vývojár svoju aplikáciu vložiť do vlastného zariadenia, musí zaplatiť poplatok za tzv. iOS developer program, ktorý je vo výške 99 amerických dolárov za rok. Developermi vytvorené aplikácie sú zákazníkom distribuované prostredníctvom App store. Podľa dohody, s ktorou musí vývojár súhlasiť, ak chce distribuovať svoju aplikáciu, ak niekto zakúpi danú aplikáciu 70% z ceny ide vývojárovi a 30% si berie App store.

Obsah SDK pozostáva z nasledovných častí:

* Cocoa Touch
* Media
* Core Services
* Mac OS X Kernel

**Cocoa touch** je UI (user interface) framework. Je postavený na Mac OS X Cocoa API (application programming interface) a ako on, je primárne napísaný v jazyku Objective-C. Tak ako Cocoa, aj Cocoa touch je založený na MVC (model view controller) software architektúre. Medzi jeho hlavné funkcie patria:

* Core animácie
* Multitasking
* Rozpoznávanie gest[[10]](#footnote-10)

Medzi **Media** patria napríklad OpenAL, Quartz, OpenGL ES a ďaľšie technológie. OpenAL (open audio library) je multiplatformové audio API. Jeho úlohou je efektívne rendrovanie multikanálového trojdimenzionálneho priestorového zvuku. Quartz sa skladá z dvoch technológií a to Quartz 2D a Quartz Compositor. Tie sú zodpovedné za 2D renderovanie a Quartz Compositor posiela inštrukcie grafickej karte. Quartz je často zamieňaný ako synonymum Core graphics. OpenGL ES je subset OpenGL API na rendrovanie počítačovej grafiky, najmä 2D a 3D grafiky použitej počítačovými hrami.

Vrstva **Core services[[11]](#footnote-11)**obsahuje základné systémové servisy pre aklikácie. Kľučové sú Core foundation a Foundation frameworky, ktoré používajú všetky aplikácie. Táto vrstva poskytuje funkcionalitu ako lokácia, iCloud, sociálne siete a networking. Do tejto vrstvy spadá aj ARC (automatic reference counting) technológia, ktorá zjednodušuje process manažovania životného cyklu objektu v Objective-C. Ďalšou užitočná technológioa zahrnutá v Core services je SQLite, ktorá umožňuje vložiť do aplikácie lightweight SQL databázu bez toho aby bežala na databázovom servri. Z aplikácie ju možno jednoducho ovládať pomocou SQLite dotazov.

Mac OS X Kernel (XNU)[[12]](#footnote-12)je jadro operačného systému zodpovedné za manažment súborov na disku, prideľovanie energie ostatným komponentom, bezpečnosť, TCP/IP protokol a iné. Pôvodne bol XNU vyvinutý spoločnosťou NeXT pre operačný systém NeXTSTEP. Bolo to hybridné jadro, ktoré sa skladalo z Mach kernel (vytvorený v Carnegie mellon University pomocou 4.3BSD komponentov) a Objective-C API, ktoré slúžilo na tvorbu ovládačov, nazývaný aj Driver Kit. Po tom ako Apple kúpil NeXT, oba komponenty boli vylepšené, a ako celok sa využíva ako jadro operačných systémov.

# 1.8. Objective - C[[13]](#footnote-13)

Objective-C je primárny programovací jazyk, ktorý sa používa pri tvorbe softwaru pre operačné systémy OS X a iOS. Vyvinul sa z C a Smalltalk jazyka, po druhom menovanom podedil objektovo orientovaný princíp. Objective-C prevzal syntax, primitívne dátové typy a postupnosť tvorby programu od C. Syntax pre objektovo orientovanú funkcionalitu jazyka je implementáciou Smalltalk štýlu správ (messaging). Keďže Objective-C je tenká vrstva postavená na C, je možné kompilovať ľubovoľný C program s Objective-C kompilátorom. Takisto môže programátor vložiť kus C kódu do Objective-C triedy. Jazyk bol vytvorený v osemdesiatych storočiach minulého storočia v spoločnosti NeXT, ktorá ho zvolila za hlavný jazyk pre svoj operačný systém NeXTSTEP, ktorý je predchodca dnešných OS X a iOS. Zdrojové súbory napísané v tomto jazyku mjú .m súborový suffix, a hlavičkové súbory majú .h príponu, takisto ako v C jazyku. Takisto ako C, aj Objective-C používa pri deklarácií svojich premenných (objektov) tzv. pointre (ukazovatele)[[14]](#footnote-14). Ukazovateľ je premenná, ktorej hodnota je adresa v pamäti. Pomocou pointrovej aritmetiky môžeme manuálne manipulovať s touto hodnotou a tak manažovať využitie pamäte pomocou umiestňovania premenných na presné miesta v pamäti. Napríklad jazyk Java neumožňuje programátorovi priamo narábať s pamäťou. Keďže pointer je iba konkrétna adresa v pamäti, môže „ukazovať“ na rôzne dátové typy, ako integer, float, struct, alebo dokonca aj na funkciu

Špecifikom tohto jazyka sú tzv. messages (správy). Objective-C ako objektovo orientovaný jazyk je založený na posielaní správ inštanciám objektov. Vývojár v tomto jazyku nevolá metódy nad inštanciami tried, namiesto toho im posiela messages (správy). Rozdiel medzi týmito dvomi konceptmi je v tom, ako je vykonaný kód, ktorý je referencovaný danou správou, alebo metódou.

V objektovo orientovaných jazykoch, ktoré využívajú metódy (napr. C++), je táto metóda viazaná kompilátorom na nejaký kus kódu, ktorý je definovaný v triede, z ktorej je objekt inštancovaný. V tomto prípade je jasné už od kompilácie (compile-time), aký kus kódu sa zavolá. Interne sa v takýchto jazykoch pristupuje k metódam a funkciám ako „offset“. Tento „offset“ znamená, že ak sa posunieme v pamäti od daného počiatočného bodu o istý počet jednotiek pamäte, nájdeme začiatok hľadanej funkcie. Výhoda tohto postupu je rýchlosť, naopak nevýhodou je, že je menej flexibilný ako nasledovný postup.

V Objective-C je to inak[[15]](#footnote-15). V tomto prípade si Objective-C v runtime drží zoznam všetkých funkcií a metód, o ktorých vie. Každý záznam v tomto zozname pozostáva z dvoch záznamov. Prvý hovorí o názve metódy (známy ako „selector“ metódy), a druhý je adresa v pamäti, kde sa táto metóda/funkcia nachádza. Volanie metódy nad objektom je v tomto prípade odlišné. Keď sa kód kompiluje, kompilátor preloží kód [anObject doMethod:aParameter]; do objc\_msgSend(anObject, @selector(doMethod:), aParameter);(zjednodšene povedané, reálne to funguje komplexnejšie, všetko je vysvetlené v Apple dokumentácií zameranej na runtime[[16]](#footnote-16)). Funkcia objc\_msgSend() robí nad zoznamom metód tzv. dynamický lookup, kedže pozná jej názov (@selector(doMethod:)), ide po jednotlivých položkách zoznamu a hľadá zhodu. Toto umožňuje programátorom robiť veľmi zaujímavé veci, akomodifikácia tohto listu. Napríklad navzájom vymeniť hodnoty v zozname a tak namiesto toho, aby selektor A vykonal kód A, selektor A môže namiesto toho ukázať kód, na ktorý ukazuje B selektor. Nevýhoda takéhoto prítupu je, že je pomalší oproti predchádzajúcemu (v praxi ide o nanosekudny).

1.8.1 Základná syntax jazyka Objective-C - Triedy

Väčšina objektovo orientovaných jazykov, používa triedy, ktoré slúžia na zapuzdrenie (encapsulate) dát a taktiež poskytuje prístup k nim. Objective-C trieda obsahuje inštančné premenné, metódy a/alebo properties (vlastnosti). Syntax Objective-C použitá pri deklarácií rozhrania triedy vypadá nasledovne[[17]](#footnote-17):

@interface SimpleClass : NSObject

@end

Tento ukážkový kód deklaruje triedy SimpleClass, ktorá dedí od NSObject triedy. Verejné properties (vlastnosti) a správanie triedy je definované v @interface deklarácií. V tomto príklade vidíme, že okrem superclass (trieda od ktorej naša trieda SimpleClass dedí, teda NSObject) nie je deklarované nič iné. Tým pádom inštancie triedy SimpleClass budú ponúkať rovnakú funkcionalitu ako inštancie triedy NSObject.

Objekty majú často properties, ku ktorým majú prístup ostatné objekty (verejný prístup). Ak sa rozhodneme navrhnúť triedu, ktorá má za úlohu uchovať informácie o človeku, je prirodzené, že jednotlivé inštancie tejto triedy budú obsahovať properties ako meno a priezvisko danej osoby. Takže:

@interface Person : NSObject

@property NSString \*firstName;

@property NSString \*lastName;

@end

V tomto prípade trieda Person deklaruje dve verejné properties, ktoré sú inštanciami NSString triedy. Obe tieto properties sú Objective-C objektmi, preto pred ich názvami sa nachádza hviezdička, ktorá indikuje, že sa jedná o C pointre (ukazovatele), ktoré boli priblížené v predchádzajúcom texte. Takéto deklarovanie je výraz ako každé iné v jazyku C, preto musí byť ukončené bodkočiarkou. K takto definovanej triede môžeme pripájať aj iné informácie okrem mena a priezviska, napríklad rok narodenia. Túto property môžeme deklarovať nasledovne:

@property NSNumber \*yearOfBirth;

Toto však nie je najoptimálnejšie riešenie, keďže potrebujeme uložiť iba jednoduchú číselnú hodnotu. V tomto prípade bude vhodnejšie použiť jeden s primitívnych dátových typov, ktorý nám poskytuje jazyk C, ako integer:

@property int yearOfBirth;

1.8.2. Základná syntax jazyka Objective-C – Medódy

Väčšina tried, okrem toho, že obsahujú rôzne properties, taktiež poskytujú „správanie“ (behaviour) tried. Takéto „správanie“ je implementované pomocou metód, ktoré umožňujú posielanie správ jednotlivým objektom. Objective-C software je zostavený z množstva objektov, ktoré sú vzájomne prepojené a práve vzájomným posielaním správ je umožnená komunikácia medzi nimi. Metódy v tomto jazyku sú konceptuálne podobné štandardným funkciám v jazyku C a iných programovacích jazykoch a vypadá nasledovne:

void SomeFunction();

Ekvivalent tejto metódy v Objective C vypadá takto:

-(void)someMethod;

V tomto prípade metóda nemá žiadne parametre. Kľúčové slovo z jazyka C void, ktoré je v zátvorkách hovorí, že metóda po svojom vykonaní nevráti žiadnu hodnotu. Znamienko – pred metódou naznačuje, že táto metóda je môže byť zavolaná iba nad objektom, ktorý je inštanciou danej triedy (tzv. inštančná metóda). Ďalším typom metód je metóda triedy. Je odlíšená znamienkom pred jej deklaráciou (má znamienko +). Takáto metóda nie je volaná nad objektom, ale nad triedou samotnou.

Syntax metód, ktoré obsahujú parametre je diametrálne odlišný od typických C funkcií. Jazyk C definuje parametre funkcie v zátvorkách:

void SomeFunction(SomeType value);

V Objective-C sú parametre metódy súčasťou názvu danej metódy použitím dvojbodky.

- (void)someMethodWithValue:(SomeType)value;

Takisto ako návratová hodnota, aj typ parametra je špecifikovaný v zátvorkách, rovnako ako v jazyku C. Ak metóda obsahuje viac parametrov, syntax je takisto značne odlišný od C. Viaceré parametre sú v C funkcií definované v zátvorke, oddelené čiarkami (parameter pozostáva z názvu premennej a jej dátového typu). V Objective-C metóda, ktorá má dva parametre vypadá takto:

-(void)someMethodWithFirstValue:(SomeType)value1 secondValue:(AnotherType)value2;

V tomto prípade value1 a value2 sú názvy parametrov, ktoré sú použité v implementácií na prístup k hodnotám dodaných do metódy. V prípade, že táto metóda bola zavolaná, k samotným parametrom sa pristupuje ako k obyčajným premenným. Je dôležité poznamenať, že v jazyku Objective-C je poradie parametrov nutné dodržať, takže metóda musí byť volaná tak, ako bola deklarovaná. V podstate secondValue: časť deklarácie metódy je aj časťou názvy metódy.

someMethodWithFirstValue:secondValue:

Takéto pomenovanie metód a jej prametrov je jeden z princípov, ktorý prispieva k veľmi dobrej čitateľnosti jazyka Objective-C. Implementácie tried a ich interné správanie (dvojité zdrojové súbory pre každú triedu)

V prípade, že je pre triedu vytvorené rozhranie, vrátane properties a metód, ktoré budú použité pre verejný prístup, je potrebné naprogramovať správanie tried. Ako bolo už skôr spomínané, rozhranie triedy je uložené v súbore, ktorý je práve na to určený, často označovaný za header file(hlavičkový súbor). Tento súbor ma príponu .h. Druhý súbor je určený pre implementáciu Objective-C triedy, tento súbor ma príponu .m. Keďže rozhranie je definované v inom súbore, ako implementácia, je potrebné povedať kompilátoru, aby najprv prečítal rozhranie, až následne implementáciu triedy. Toto je zariadené Objective-C preprocesing directive (predprocesovým príkazom) #import. Tento príkaz je podobný k #include z jazyka C, no v Objective-C je potrebné sa uistiť, že daný hlavičkovú súbor je počas kompilácie importnutý iba raz. Ďaľšim rozdiel je v syntaxi: #include je ukončený bodkočiarkou, zatiaľ čo #import nie je ukončený žiadnym znakom. Základná syntax pre implementáciu triedy (doteraz sme popisovali iba rozhranie triedy) je takáto:

#import "XYZPerson.h"

@implementation XYZPerson

@end

Ak deklarujeme metódy v rozhraní triedy (súbor s .h príponou), je potrebné ich implementovať v tomto súbore (.m súbor). Povedzme, že jednoduché rozhranie triedy s jednou metódou je nasledovné:

@interface XYZPerson : NSObject

- (void)sayHello;

@end

Implementácia tohto rozhrania je:

#import "XYZPerson.h"

@implementation XYZPerson

- (void)sayHello {

NSLog(@"Hello, World!");

}

@end

V tomto príklade používame funkciu NSLog(), ktorá vypíše vložený parameter do konzoly. Táto funkcia je podobná printf() funkcií z jazyka C a jej parametre môžu byť viaceré, no prvý z nich musí byť Objective-C string. Implementácia Objective-C metód je podobná definícií funkcií v C jazyku, aj tu sa používajú kučeravé zátvorky, v ktorých je vložený relevantný kód. Okrem toho, meno metódy musí byť identické svojmu prototypu, a parametre a návratový typ sa musia zhodovať. Objective-C podedilo case sensitivity od C, takže metódu:

- (void)sayhello {

}

by kompilátor považoval za úplne odlišnú od sayHello metódy. Vo všeobecnosti mená metód by mali začínať s malým písmenom. Konvencia v jazyku Objective-C je používať mená metód, ktoré ju popisujú viac (z toho rezultujú dlhšie názvy metód, často zložené z viac ako štyroch slov), na rozdiel od typických funkcií v jazyku C. A metóda obsahuje viac slov, je konvenciou použiť camel case (každé nové slovo začína novým písmenom), následne sú metódy čitateľnejšie. Použitie bielych znakov (whitespaces) je v Objective-C flexibilné. Je na developerovi, či odsadí kód medzerami, alebo tabulátorom. Taktiež je zvykom dávať ľavú zátvorku, ktorá otvára kód, dávať na nový riadok.

# 1.9 Xcode

Xcode je vývojové prostredie (IDE – integrated development enviroment), ktoré obsahuje súpravu nástrojov, ktoré pomáhajú pri vývoji softwaru pre OS X a iOS. Xcode bol vyvinutý Applom ako nástroj, ktorý má (ako všetky Apple produkty) poskytnúť čo najväčší komfort pre svojich používateľov, v tomto prípade vývojárov. Keďže obsahuje množstvo komponentov potrebných pri vývoji programov. Aj napriek komplexnosti je narábanie s týmto produktom jednoduché a prirodzené. Uvedený na trh bol v rolu 2003, aktuálna verzia je 5.1. a je zdarma dostupný na Mac App Store pre operačné systémy Mac OS X Lion a OS X Mountain Lion. Registrovaným developerom je umožnené sťahovať predchádzajúce verzie, respektíve beta verzie tohto softwaru. Ako už bolo spomenuté, Xcode sa skladá z viacerých častí ako:

* editor zdrojového kódu
* organizátor zdrojových súborov
* konzola na debuggovanie
* iOS simulátor
* interface builder
* a množstvo ďalších

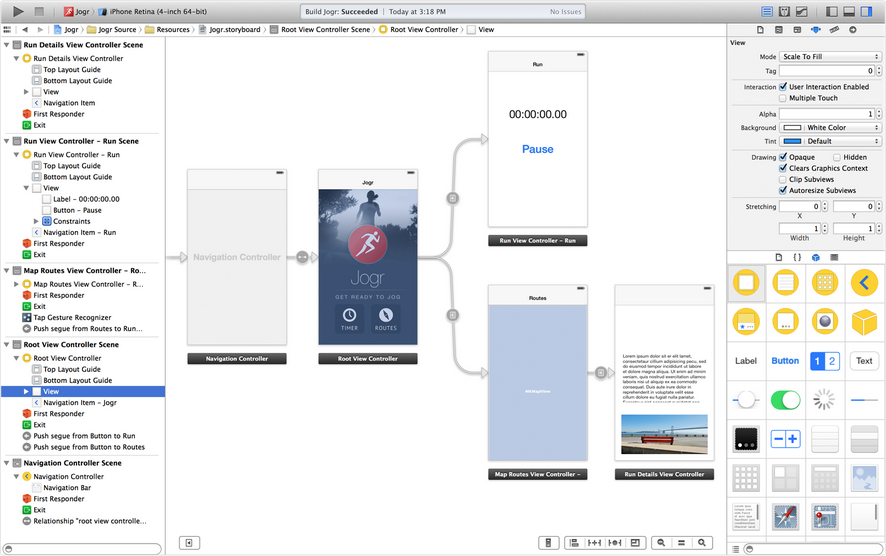
Ako každé IDE, aj Xcode poskytuje editor zdrojového kódu, ktorý poskytuje už štandardnú funkcionalitu, ako zvýrazňovanie kódu, kolapsovanie kusov kódu, inteli sense (automatické dopĺňanie kódu pri jeho písaní, zvýrazňovanie syntaktických chýb ...). Takisto podporuje aj automatické formátovanie a odsadzovanie kódu, dokončovanie zátvoriek a mnohé ďalšie.

Organizátor súborov zdrojového kódu sa nachádza na ľavej strane okna, kde je zobrazený stromovou štruktúrou. Developer pri vytváraní nových zdrojových súborov sa môže ich vkladať do priečnikov a tak výrazne sprehľadniť ich organizáciu. Sú tu zobrazené okrem samotných zdrojových súborov napísaných vývojárov, aj napríklad tzv. Supporting files (obrázky, plist súbory, main.m ...), testy aplikácie a použité frameworky.

Použitím debuggovacej konzoly môžeme jednoducho kód ladiť pomocou tzv. breakpointov alebo logov, ktoré môžu obsahovať hodnoty premenných, dotiahnuté dáta sql dotazmi a iné.

iOS simulátor nám ponúka náhľad na samotnú vyvíjanú aplikáciu, ako bude vypadať na samotnom zariadení. Je na výber množstvo rôznych zariadení, na ktorom si môžeme program spustiť, či už rozdielne verzie iPhonov a iPodov. Nezabudlo sa ani na odlišné rozlíšenie (retina, nonretina) a architektúru (32 bit a 64 bit) týchto prístrojov. Interface builder umožňuje vývojárovi jednoducho poskladať aplikáciu z ponúkaných komponentov (views, tab bary, tlačidlá, sliders) a následne v kóde implementovať ich funkcionalitu.

Xcode obsahuje aj ďalšie nástroje nápomocné nástroje ako Log Navigator (nápomocný pri testovaní aplikácie), Assistant editor (zobrazenie viacerých zdrojových súborov naraz). Všetky tieto nástroje sú ľahko nastaviteľné podľa preferencií každého vývojára, či už pomocou klávesových skratiek, gest alebo množstva pluginov dostupnými na webe.



Obrázok 6: Xcode a jeho storyboard nástroj

Zdroj: <https://developer.apple.com/xcode/ide/>

2. Cieľ práce

Primárnym cieľom je vytvorenie aplikácie postavenej na platforme iOS pre zobrazovanie rozvrhu, ktorá by slúžila študentom Ekonomickej univerzite v Bratislave. Požiadavky na samotnú aplikáciu sú:

* jednoduché ovládanie
* minimálna záťaž pre preprocesor a pamäť zariadenia
* prístup k samotnej aplikácií čo najnekomplikovanejší
* príťažlivý grafický dizajn pre používateľa
* aktualizácie raz za semester
* podpora aj pre staršie verzie iOS
* offline aplikácia

Pod jednoduchým ovládaním aplikácie sa rozumie čas, námaha, počet ktorou, ktoré musí používateľ vynaložiť, na dosiahnutie cieľa, ktorý si stanovil pred spustením aplikácie. V našom prípade hovoríme o zobrazení konkrétneho rozvrhu, pre daného študenta v daný deň. Táto požiadavka patrí medzi základné, ktoré by mala vytvorená, ale aj každá iná aplikácia primárne spĺňať. Keďže o počítačovej gramotnosti medzi budúcich používateľov nevieme veľa, musíme dbať na to, aby aj tí menej zdatní používatelia nemali problém aplikáciu používať. Samozrejmé je, že nie je potrebné prečítať manuál, ktorý by popisoval kroky, ako s ňou pracovať. Užívateľ už na prvej obrazovke musí vedieť, ako narábať s aplikáciou, respektíve musí mu byť jasný workflow, ktorý vedie k zobrazeniu rozvrhu. Používateľ je lenivý tvor, musíme mu vyjsť v ústrety a vyvarovať sa situácií, kedy by mu došla trpezlivosť a nahradil by našu aplikáciu, za konkurenčnú.

Je potrebné aplikáciu optimalizovať, aby pamäť, respektíve výpočetné kapacity, ktoré využíva boli čo najmenšie. Neželaným efektom by bolo, keby spustenie našej aplikácie spôsobovalo spomalenie zariadenie, v horšom prípade jeho mrznutie. Dosiahnuť čo najlepšiu efektivitu a výkon aplikácie je možné viacerými spôsobmi. Medzi najčastejšie patrí optimalizácia algoritmov, z ktorých sa tento software skladá. Vyvarovať situácie, kedy by sa pamät zapĺňala zbytočnými premennými, triedami, alebo keby sa čas procesoru mrhal na vykonávanie neoptimálnych cyklov atď. Zefektívniť aplikáciu nám pomôže Xcode. Ten obsahuje množstvo nástrojov, pomocou ktorých monitorujeme alokovanie pamäte, využitie pamäte v čase, diagnostikovanie množstva energie, ktoré je spotrebované na beh aplikácie a mnohé ďalšie[[18]](#footnote-18).

Distribuovanie aplikácie je ďalším krokom, ktorým je potrebné sa zaoberať pri tvorbe iOS aplikácií. V tomto nám pomôže App Store, ktorý je centralizovaným úložiskom aplikácií založených na iOS platforme. Všetky aplikácie, ktoré sa v App Store boli prezreté samotnými pracovníkmi tejto spoločnosti, konkrétne zdrojové kódy týchto aplikácií. Tento proces sa skladá z viacerých krokov, cez ktoré musí aplikácia prejsť, aby bola umiestnená v App Store. Až potom majú milióny používateľov zariadení od Apple možnosť si túto aplikáciu stiahnuť a začať používať. Fakt, že cieľovými používateľmi sú študenti, ktorí vo väčšine nepracujú a tak ich disponovanie s finančnými prostriedkami je značne obmedzené. Je potrebné stanoviť minimálnu cenu, respektíve poskytnúť túto aplikáciu zdarma.

Dizajn predáva. O tomto fakte sa presvedčili nielen tvorcovia aplikácií pre rôzne platformy, ale aj každý, kto si prezrie rebríček najúspešnejších aplikácií. Aj aplikácie orientované na jednu úlohu (ako tá naša), môžu byť komerčne úspešné, stačí na to príťažlivý zovňajšok, ktorý lahodí oku používateľa. Momentálnym trendom dizajnu je tzv. flat (plochý) dizajn, ktorý sa vyznačuje minimalistickým prístupom v používateľskom rozhraní. V praxi to znamená, že sú odstránená 3d efekty (tiene, textúry rôznych typov), gradienty (farba prechádza simultánne z jedného odtieňa do druhého) z komponentov, z ktorých je tvorené UI. Flat design sa sústreďuje na použitie jednoduchých elementov (jednoduché geometrické tvary), písmo a na tzv. flat (pastelové) farby. Vďaka tomuto trendu, sú používateľské rozhrania jednoduché, prehľadné, nezaťažujú tak výpočtovú techniku (rýchlejšie sa načítavajú) a sú responzívne (dobre zobrazovateľné na rôznych veľkostiach obrazoviek). V tomto duchu by sme rád vytvoril aj našu aplikáciu.

Podmienka, aby aplikácia fungovala aj offline (teda bez priameho prístupu zariadenia k internetu) je logická, keďže rozvrh sa mení iba raz za semester. Z toho vyplýva, že je nutné aktualizovať databázu raz za semester. Tento proces vykoná aplikácia sama, bez toho aby používateľ inicializoval tento krok. Ak je dostupný nový rozvrh, aplikácia automaticky nahradí starú databázu za aktuálnu a je možné ju ďalej používať.

Pri vývoji je nutné myslieť aj na staršie verzie iOS ako 7. Napriek tomu, že podiel používateľov neaktuálnej verzie operačného systému je malý (okolo 10%), je na mieste umožniť používanie tejto aplikácie aj pre týchto používateľov.

3. Metodika práce a metódy skúmania

V tejto kapitole uvediem kroky a postupy, ktoré som zvolil pri vývoji aplikácie na zobrazovanie rozvrhov pre aplikáciu na platforme iOS. Taktiež technológie a nástroje použité v procese tvorby si v tejto kapitole nájdu miesto.

Prvým problémom, na ktorý je potrebné upriamiť pozornosť sú dáta, ktoré reprezentujú jednotlivé rozvrhy. Potrebné je zvoliť čo najmenej bolestné riešenie. Je nemysliteľné, aby sa tieto získali ručným prechádzaním rozvrhu v Lookine a ich zapisovaním do konkrétnej dátovej štruktúry (napr. xml), odkiaľ by dáta boli zobrazované v aplikácií. Taktiež je potrebné myslieť sa na repetitívny charakter tejto operácie, keďže rozvrh sa mení pravidelne raz za semester. Automatizovanie tohto procesu je správny krok. Jedným zo zdrojov takýchto dát je databázový súbor, ktorý je vkladaný do programu Lookin. Tento súbor je zverejnený na oficiálnych stránkach Ekonomickej univerzity v Bratislave, a je aktualizovaný každý semester. Nevýhodou tohto súboru je jeho čitateľnosť, keďže bol vytvorený pre číťanie v Lookine. Charakter tohto súboru je .bin a čítať je ho možné iba ak developer pozná jeho štruktúru. Program v ktorom sa rozvrh tvorí má dva výstupné súbory. Jeden z nich je už spomenutý .bin súbor, ktorý je nečitateľný. Druhým je .par súbor, ktorý je čítateľný človekom. Prvým krokom je parsovanie.par súboru a získané dáta vkladať do databázy, z ktorej je možné pomocou dotazov skladať jednotlivé rozvrhy pre daný ročník, fakultu, odbor a číslo krúžku.

# 3.1 Štruktúra a parsovanie .par databázového súboru

Databázový súbor obsahuje kompletné informácie, ktoré sú potrebné na zostavenie rozvrhu pre celú univerzitu. Jednotlivé informácie sú rozdelené do blokov, z ktorých má každý svoj nadpis, ktorý začína bodkou a nasledujú veľké písmená. Za ním nasledujú dáta ktoré patria k tomuto bloku. Nasledujúcimi riadkami priblížim, aké dáta súbor poskytuje a zároveň ich využijem vo svojej aplikácií a v akom formáte sú v súbore uložené. Súbor je identifikovaný, pre aký semester a rok je relevantný.

.OBDOBIE

Letný semester 2013/2014

Aké je rozloženie vyučovacích hodín v danom semestri.

.HODINY

1 7.30 - 9.00

2 9.15 - 10.45

3 11.00 - 12.30

...

Zoznam všetkých katedier na univerzite. Tieto záznamy slúžia ako cudzí kľúč k zoznamu učiteľov, keďže každý učiteľ prislúcha konkrétnej katedre (ak nie je známa katedra, pod ktorú učiteľ patrí, je mu v databáze priložená Neznama\_katedra). Každý záznam obsahuje názov a skratku danej katedry.

.KATEDRY

Neznama\_katedra KAAA 728 0

Katedra\_ekonomickej\_teorie KET 675 9

Katedra\_hospodarskej\_politiky KHP 662 9

Katedra\_verejnej\_spravy\_a\_regionalneho\_rozvoja KVSRR 674 0

Katedra\_socialneho\_rozvoja\_a\_prace KSRP 679 0

Katedra\_financii KF 692 0

...

Nasleduje zoznam miestností, kde sa konajú prednášky/cvičenia. Každý záznam obsahuje názov miestnosti a jej kapacitu.

.MIESTNOSTI

U SM\_I 80

U SM\_II 175

U 3B19 20

Ďalším v poradí je zoznam učiteľov. Každý záznam obsahuje meno a priezvisko učiteľa (v niektorých prípadoch iba priezvisko), skratku katedry, ku ktorej patrí a jeho titul.

.UCITELIA

Adamcova\_Livia KJaT Prof : U

Adamcova\_Silvia KJaT Mgr : U

Adamik\_ KBMF PhD : U

Adamkova\_Hana KMZ Mgr : U

Nasleduje zoznam fakúlt, odborov a krúžkov. Tento zoznam je rozdelených do jednotlivých študijných rokov. Z týchto údajov sa dajú zistiť nasledovné informácie:

* zoznam fakúlt a ich skratiek
* zoznam odborov danej fakulty, ich skratky, počet študentov v jednotlivých odboroch
* počet krúžkov daného odboru

Formát dát:

.PARALELKY

1\_rocnik 1

$ Narodohospodarska\_fakulta NHF

$$ Financie\_bankovnictvo\_a\_investovanie FBI 174/1-9

$$ Poistovnictvo POI 31/1-2

$$ Narodne\_hospodarstvo NH 74/1-4

$$ Ludske\_zdroje\_a\_socialny\_manazment LZSM 30/1-2

$$ Verejna\_sprava\_a\_regionalny\_rozvoj VSRR 17/1

$$ Ekonomicka\_teoria\_a\_ekonomicka\_zurnal ETZ 14/1

$ Obchodna\_fakulta OF

$$ Podnikanie\_v\_obchode\_a\_CR POCR 197/1-10

...

Posledným, najdôležitejším a najdlhším zoznamom je zoznam predmetov. Len pre zaujímavosť tento zoznam obsahuje zhruba 4 900 riadkov. Z týchto záznamov sa dajú získať údaje:

* pre aký ročník, fakultu, odbor je predmet určený
* či je predmet povinný alebo voliteľný
* názov, kód, katedra a vyučujúci predmetu
* výmera predmetu (počet prednášok a cvičení)
* jednotlivé záznamy o vyučovacích hodinách hovoria: či sa jedná o cvičenie alebo prednášku, kde a kedy sa daná výuka koná, kto ju učí, pre ktorý krúžok je relevantná

.PREDMETY

1\_NHF 15101: KF: S V | Financie\_a\_mena FM Zubalova\_Alena 1/0 197 ;0

# 1 2 3 4 5 6 7 8 9 10 11

~ 692 0

$1 B107 2 5 0 Zubalova\_Alena :F

1\_NHF 15101/1: KF: S V | Financie\_a\_mena FM Belickova\_Kornelia 1/0 143 ;1

# 12 13 14 15 16 17 18 19

~ 692 0

$1 D112 2 5 0 Belickova\_Kornelia :F

1\_NHF\_FBI 11215: KET: S V | ET\_Makroekonomia\_I ET\_MA1 Lisy\_Jan 1/1 116 ;2

# 1 2 3 4 5 6

~ 675 9

$1 D114 2 2 0 Lisy\_Jan

&1 B206 2 6 1 0

&1 D203 2 4 2 0

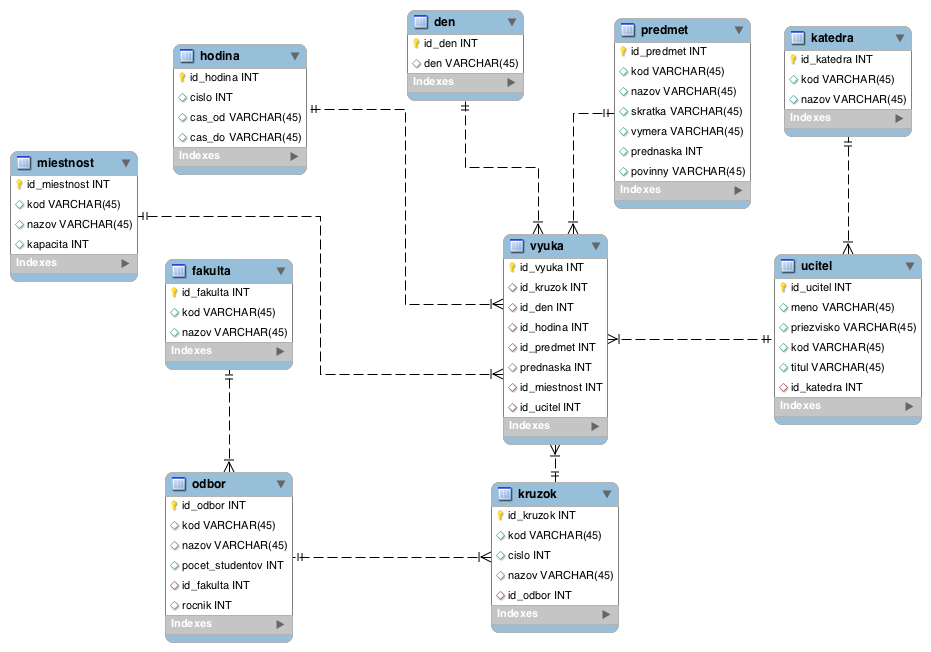
&1 D103 2 1 3 0

&1 D107 3 1 4 0

&1 D108 3 3 5 0

&1 D202 3 3 6 0

Prvým krokom bolo zvoliť databázovú technológia, ktorú by som použil na uloženie parsovaných dát. Rozhodol som sa pre MySql. MySql je druhá najpoužívanejšia[[19]](#footnote-19) open-source relačná databáza vytvorená spoločnosťou Oracle Corporation. Túto technológiu som z volil z toho dôvodu, že už s ňou mal skúsenosti, a takisto zavážil fakt, že dokáže spolupracovať s PHP jazykom, ktorý použil v nasledujúcich fázach projektu. Následne bolo nutné navrhnúť entitno relačný model, ktorý by zobrazoval štruktúru plánovanej databázy. Mal by obsahovať tabuľky a atribúty tak, aby sa využil informačný potenciál, ktorý databázový súbor poskytoval. Databázový model som navrhol pomocou softwaru MySQLWorkbench.



Obrázok 7: Zobrazenie entito relačného diagramu navrhovanej databázy

Ako je evidentné zo schémy, je tu jedna dominantná tabuľka, z ktorej smerujú cudzie kľúče do ostatných tabuliek. Skoro všetky atribúty tabuľky Výuka sú cudzími kľúčmi. Táto tabuľka je faktová a ostané tabuľky sú dimenzionálne. Takáto schéma sa nazýva star scheme (hviezdicová schéma), kde v strede je faktová tabuľka a na ňu nadväzujú dimenzionálne tabuľky. Star schéma je špeciálnym prípadom snowflake schémy a je viac efektívna pri vykonávaní jednoduchých dotazov. Takáto databáza je v vysoko normalizovaná, keďže redundancia dát je minimálna.

Do takto pripravenej databázy môžeme začať nalievať dáta z .par súboru. Používateľské rozhranie je navrhnute jednoducho. Obsahuje formulár s inputom, kde sa uploadne .par súbor a tlačidlo, ktoré tento súbor odošle na server, kde sa sparsuje a dáta sa vložia do databázy. Používateľovi príde do prehliadača o úspešnom alebo neúspešnom vykonaní akcie. Na splnenie tohto scenára je potrebné napísať server skript, ktorý sparsuje a dáta vloží ich do databázy. Tento skript vytvorím použitím PHP jazyka a frameworku CodeIgniter.

PHP (hypertext preprocessor) je jazyk, ktorý sa používa na tvorbu serverových skriptov. S jeho pomocou sa dajú tvoriť dynamické a interaktívne webové aplikácie. PHP je široko rozšírený a je efektívnou alternatívou ku konkurencií, ako napríklad ASP od Microsoftu. CodeIgniter je „ľahký“ php framework, poskytuje množstvo knižníc, helprov, preddefinovaných tried, ktoré umožňujú písať php skripty jednoduchšie (na menej riadkov) a tým pádom efektívnejšie vyvíjať aplikácie a celkovo zrýchliť tento proces.

# 3.2 Popis algoritmu na parsovanie doumentu

Na začiatku celého procesu sa obsah uploadnutého súboru načíta do poľa, každý riadok osobitne je jeden index v poli. Ďalším krokom je odstránenie prázdnych riadkov v súbore, teda prázdnych stringov v na jednotlivých pozíciách v poli.

// read file to array and trim it

$file\_content = file($file, FILE\_IGNORE\_NEW\_LINES | FILE\_SKIP\_EMPTY\_LINES);

Nasleduje cyklus, ktorý iteruje po indexoch v poli a hľadá ich konkrétne hodnoty, ako napr. .HODINY, .KATEDRY, .MIESTNOSTI … Povedzme, že sa pri iterovaní narazí na obsah poľa na pozícií n je string .UCITELIA. Následne sa z tohto poľa vyberajú všetky hodnoty a vkladajú sa do osobitného poľa, až kým sa nenarazí na na string, ktorý začína bodkou a nasledujú veľké písmená (to značí, že už sme v inej skupine údajov), teda:

// while not match uppercase word begining with .

while (!preg\_match("/^\.[A-Z]+$/", $file[$row])) {

array\_push($items, $file[$row]);

$row++;

}

Výsledkom je nové pole, ktoré obsahuje záznamy o učiteľoch, ktoré ďalej spracujeme. Pokračujeme s novovytvoreným poľom, a to tak, že cez neho iterujeme a po jednom zázname vkladáme postupne do databázy. Pomocou regulárneho výrazu si zo záznamu vyselektujeme hodnoty, ktoré nás zaujímajú, teda: meno, priezvisko, katedru a titul učiteľa.

preg\_match("/([a-zA-Z]+)\_([a-zA-Z]\*)\s([a-zA-Z]+)\s([a-zA-Z]+)\s:\s([A-Z])/", $teachers[$i], $teachers\_matches)

Kde premenná $teachers[$i] obsahuje riadok zo zdrojového súboru o vybranom učiteľovi, a premenná $teachers\_matches obsahuje vyselektované hodnoty podľa regulárneho výrazu, takže napríklad záznam

Adamkova\_Hana KMZ Mgr : U

bude vložený do poľa, ktoré vypadá nasledovne:

[0] => Adamkova\_Hana KMZ Mgr : U

[1] => Adamkova

[2] => Hana

[3] => KMZ

[4] => Mgr

[5] => U

Ďalej potrebujeme zistiť hodnotu id\_katedra, ktorej stĺpec kod ma hodnotu KMZ

$q = $ci->db->query("SELECT id\_katedra from katedra WHERE kod = '$teachers\_matches[3] '");

$institute = $q->row();

V tomto momente už máme k dispozícií všetky dáta potrebné na vloženie kortéže do tabuľky ucitel, a teda:

// teacher table structure: id, id\_katedra, kod, priezvisko, meno, titul

$teacher = array(

"id\_katedra" => $institute->id\_katedra,

"kod" => $teachers\_matches[1]."\_".$teachers\_matches[2],

"priezvisko" => $teachers\_matches[1],

"meno" => $teachers\_matches[2],

"titul" => $teachers\_matches[4]

);

a následne:

$ci->db->insert('ucitel', $teacher);

kde $ci je CodeIgniter premenná, cez ktorú sa pristupuje k všetkej funkcionalite, ktorú tento framework ponúka. Obdobne by sme parsovali aj ostatné dáta do databázy. Niektoré entity, ako napríklad zobrazený príklad s tabuľkou ucitel sú relatívne jednoduché. No sú tu aj tabuľky, ktorých napĺňanie je komplikovanejšie a veľmi vetvené. Toto vetvenie rezultuje z rôznych situácií, ktoré môžu nastať pri vkladaní danej entity. Napr.

1\_NHF 15101: KF: S V | Financie\_a\_mena FM Zubalova\_Alena 1/0 197 ;0

# 1 2 3 4 5 6 7 8 9 10 11

$1 B107 2 5 0 Zubalova\_Alena :F

V tomto príklade záznamu predmetu vidíme, že po názve fakulty nenasleduje žiadne odbor. To značí, že tento predmet je určený všetkým odborom z danej fakulty a roku. Písmeno S za skratkou katedry hovorí, že ide o predmet povinný a čísla za mriežkou hovoria, že tento predmet je pre krúžky 1 – 11. Nasledujúci riadok indikuje čas a miesto konania výuky tohto predmetu, a teda jedná sa o prednášku, ktorá trvá jednu vyučovaciu hodinu ($1), je v miestnosti B107 v utorok (2) a trvá od 15:15 – 16:45 (5).

# 3.3 Tvorba web servisov

Web servis je metóda komunikácie medzi dvomi elektronickými zariadeniami cez sieť[[20]](#footnote-20). V našom prípade chceme dáta uložené v MySql databáze odoslať cez sieť do mobilného zariadenia na platforme iOS a tam dáta príslušne spracovať (vložiť ich do internej databázy zariadenia). V tomto prípade využijeme charakter ponúkaný web servismi, keďže táto technológia umožňuje komunikovať aj medzi dvomi inak odlišnými systémami. Software systém, ktorý požaduje dáta (v našom prípade iPhone) sa nazýva požadovateľ servisu (service requester), a software systém, ktorý túto požiadavku spracuje a poskytne relevantné dáta ako odpoveď sa nazýva poskytovateľ servisu (service provider). Keďže tieto dva softwarové produkty, môžu byť postavené na rozličných programovacích jazykoch, je potrebné abstrahovať od týchto jazykov a nájsť metódu komunikácie, ktorá na týchto jazykoch nie je závislá[[21]](#footnote-21). Väčšina typov softwaru vie interpretovať XML tagy, takže web servisy môžu použiť XML súbory na prenos dát.

V našom prípade však za prenosový formát zvolíme JSON, keďže skript v PHP jazyku vie jednoducho transformovať dáta z klasických PHP objektov a polí do JSON štruktúry, a následne ich poslať cez sieť ako odpoveď požadovateľovi servisu. A na druhej strane iOS platforma poskytuje funkcie na spracovanie tejto odpovede a ďalšie jej spracovanie. JSON[[22]](#footnote-22) (Javascript object notation) je spôsob zápisu dát (formát dát) nezávislý na počítačovej platforme, určený na prenos dát, ktoré môžu byť organizované v poliach, alebo agregované v objektoch. Vstupom je ľubovoľná dátová štruktúra (číslo, reťazec, boolean, objekt alebo nich zložené pole), výstupom je vždy reťazec. Zložitosť hierarchie vstupnej premennej teoreticky nijak obmedzená.

Je potrebné vytvoriť servisy, ktoré by preniesli dáta uložené v MySql databáze do iPhone. To znamená, že pre každú tabuľku vytvoríme web service, ktorý následne požadovateľ servisu zavolá, a ako odpoveď mu prídu všetky dáta z danej tabuľky. Požadovateľ zavolá servis v tvare URL adresy. Na strane poskytovateľa webovej služby musíme zabezpečiť, aby dáta boli vystavené na rôznych URL adresách. Uvedieme príklad s vystavením dát z tabuľky ucitel na určitú URL adresu:

// UCITEL

function getTeacher() {

$result = [];

$query = $this->db->get('ucitel');

foreach ($query->result() as $row){

array\_push($result, $row);

}

echo json\_encode($result);

}

Ak požadovateľ servisu zavolá request s URL adresou

<http://localhost:8888/DP/web-app/svc/getTeacher>

Vykoná sa horeuvedený kód, ktorý vykoná dotaz nad databázou a z tabuľky ucitel vyberie všetky záznamy spolu s ich atribútmi, transformuje ich do JSON formátu a pošle požadovateľovi servisu (URL adresa je iba ilustračná, pre vývojárske účely). Odpoveď na takéto volanie je:

[

{"id\_ucitel":"1",

"id\_katedra":"1",

"kod":"neznamy\_ucitel",

"priezvisko":"Ucitel",

"meno":"Neznamy",

"titul":null,

},

{"id\_ucitel":"2",

"id\_katedra":"30",

"kod":"Adamcova\_Livia",

"priezvisko":"Adamcova",

"meno":"Livia",

"titul":"Prof"

},

… ]

To, čosa zobrazilo vo webovom prehliadači je len časť odpovede, ktorá prišla na základe volania. Viditeľná časť sa nazýva message body (telo správy). Časť odpovede, ktorá je viditeľná napr. pomocou vývojárskych nástrojov v prehliadači sa nazývame message header (hlavička správy). Message header obsahuje informácie o požiadavke na server, alebo odpovedi zo servra, alebo o správe v tele správy. Tieto informácie sú odlišné, vzhľadom na to, či sa vzťahujú na požiadavku, respektíve na odpoveď zo servra. Message header obsahuje informácie o požadovanej URI (unified resource indentifier), čas a dátum zaslania požiadavky, alebo odpovede, host, content-type a mnoho ďalších. Najdôležitejšou informáciou v hlavičke odpovede je http kód, môže nadobúdať rôzne hodnoty od 100 - 505[[23]](#footnote-23). Medzi najznámejšie patria 200 - OK, 404 - Not Found, 500 - internal server error.

Obdobne ako sme vytvorili servis na vypísanie tabuľky učiteľov, môžme implementovať aj ostaté servisy. Takto pripravené servisy budeme volať z iOS aplikácie a vkladať ich do internej databázy zariadenia. Nemohli sme jednoducho celú databázu preniesť do iPhone, keďže toto zariadenie nepodporuje MySql ale SQLite technológiu.

# 3.3 Tvorba iOS aplikácie

3.3.1 Konštrukcia SQLite databázy

Predtým ako začnem programovať samotnú iOs aplikáciu je potrebné vytvoriť databázu, ktorú bude toto zariadenie používať. Ako už bolo spomenuté, nemôžeme použiť už vytvorenú MySql databázu, keďže iOS zariadenia takýto typ databáz nepodporujú. Je potrebné vytvoriť novú databázu postavenú na SQLite technológií, ktorá bude mať rovnakú štruktúru ako MySql databáza. SQLite databáza je predinštalovaná na operačných systémoch MAC OS X, takže ju bez problémov môžeme začať používať. Vytvoríme ju pomocou CLI (command line interface), teda príkazmi v príkazovom riadku (aplikácia tohto typu má v MAC OS X názvom Terminal). SQLite databáza sa vytvorí jednoduchým príkazom sqlite3 <názov databázy>

$ sqlite3 Rozvrh.db

SQLite version 3.7.13 2012-07-17 17:46:21

Enter ".help" for instructions

Enter SQL statements terminated with a ";"

sqlite>

Pomocou príkazu create table[[24]](#footnote-24)<názov tabuľky> (<názov atribútu><dátový typ><iné atribúty>, ...). Medzi <iné atribúty> patrí primárny alebo cudzí kľúč. V praxi to vypadá nasledovne:

create table ucitel(id\_ucitel integer primary key, id\_katedra integer, kod varchar(45), meno varchar(45), priezvisko varchar(45), titul varchar(10));

Rovnakým princípom vytvoríme ostatné tabuľky. Koniec práce s SQLite príkazovým riadkom ukončíme príkazom .exit. Takto vytvorená databáza sa nám uloží na disk a v neskorších fázach implementácie iOS aplikácie ju nakopírujeme priamo do nej. Pripravená SQLite databáza neobsahuje okrem štruktúry nič, nie sú v nej uložené žiadne dáta. Tie budeme do nej vkladať práve z web servisov.

3.3.2 Založenie projektu v Xcode a príprava SQLite databázy

Na začiatku je potrebné vytvoriť v Xcode nový projekt. Vyplníme formulár s rôznymi poľami, ako Product Name, Organization Name atď. Dôležité je zvoliť minimálne trojpísmenový Class prefix, pre identifikáciu nami vytvorených tried. Dvojpísmenové class prefixy sú rezervované pre Apple. V našom prípade sme zvolili ERO - Euba ROzvrh. Taktiež je potrebné špecifikovať pre aké zariadenie bde aplikácia, v našom prípade vyberieme iPhone. Naša aplikácia bude používať SQLite databázu, takže po vytvorení projektu, je potrebné túto technológiu „nalinkovať“. Jednoducho tak môžeme spraviť v nastaveniach aplikácie, konkrétne na tabe General v sekcií Linked Frameworks and Libraries pridáme nový záznam a z ponuky vyberieme libsqlite3.dylib. Je potrebné spomenúť, že pri vývoji aplikácie použijeme storyboard, kde poskladáme rozhranie aplikácie z dostupných komponentov. Následne komponentom priložíme kód, ktorý sa bude starať o zmenu stavov týchto komponentov.

Na začiatku nás budú zaujímať súbory main.c a EROAppDelegate.h. Tieto dva súbory obsahuje každý novovytvorený projekt v Xcode. Úlohou main.c je okrem iného, inštancovať EROAppDelegate triedu a tak spustiť aplikáciu. Keďže sa EROAppDelegate trieda inštancuje medzi prvými, tu musíme implementovať zapojenie databázy do našej aplikácie. Prvým krokom je nakopírovanie SQLite databázového súboru priamo do Xcode, konkrétne do Supporting Files priečinka. Ďalším krokom je nakopírovanie databázy z Supporting Files do Documents priečinka na iPhone. Do takto uloženej databázy budeme vkladať dáta, ktoré získame volaním web servisov a taktiež budeme nad touto databázou vykonávať dotazy, ktorých výsledky budeme prezentovať na používateľskom rozhraní priamo v telefóne.

Musíme sa uistiť, aby sa operácie kopírovania vykonala iba raz. Teda ak už na tomto mieste databáza nachádza, nekopírovať. To vykonáva metóda -(void) createAndCheckDatabase:

-(void) createAndCheckDatabase {

BOOL databaseAlreadyExits;

NSFileManager \*fileManager = [NSFileManagerdefaultManager];

databaseAlreadyExits = [fileManager fileExistsAtPath:self.databasePath];

if (databaseAlreadyExits) {return;}

NSString \*databasePathFromApp = [[[NSBundlemainBundle] resourcePath] stringByAppendingPathComponent:self.databaseName];

[fileManager copyItemAtPath:databasePathFromApp toPath:self.databasePatherror:nil];

[EROUtilityfillDatabase];

NSLog(@"Database copied from bundle path");

}

Teda ak sa v iPhone documents priečinku databáza nachádza, tak sa ukončí vykonávanie metódy. Inak databáza nakopíruje a následne sa zavolá +(void) fillDatabase: class metóda, ktorá zabezpečí naplneni databázy

3.3.3 Napĺňanie SQLite databázy

Napĺňanie databázy ma na starosti trieda EROUtility, ktorá volá metódy z dvoch ďalších tried (EROWebService a ERODatabaseAccess), ktoré jej v tomto procese asistujú. Ako bolo spomenuté, celý proces začína volaním metódy +(void) fillDatabase:, ktorá v podstate iba inicializuje napĺňanie všetkých tabuliek.

+ (void) fillDatabase {

NSLog(@"importing initialized ...");

[selfpopulateFaculties];

[selfpopulateLessons];

[selfpopulateInstitutes];

[selfpopulateGroups];

[selfpopulateRooms];

[selfpopulateDepartments];

[selfpopulateSubjects];

[selfpopulateTeachers];

[selfpopulateLectures];

[selfpopulateDays];

}

Každá z týchto metód vykonáva v podstate to isté, len s inými parametrami a manipuluje s inými tabuľkami. Na metóde +(void) populateFaculties: si demonštrujeme proces volania servisov a následné ukladanie dát. Metóda je implementovaná nasledovne:

+(void) populateFaculties {

[[EROWebServicesharedInstance] getFacultiesWithSuccess:^(id responseObject) {

NSArray \*array = (NSArray \*)responseObject;

NSMutableArray \*facultiesArray = [NSMutableArrayarrayWithArray:array];

[ERODatabaseAccessinsertFacultiesFromArray:facultiesArray];

dispatch\_async(dispatch\_get\_main\_queue(), ^{});

} failure:^(NSError \*error) {

NSLog(@"Error ocured while populating faculties: %@", error);

}];

}

Ako je evidentné táto metóda ma dva callbacky (časti). Prvá je getFacultiesWithSuccess, ktorá sa zaoberá pozitívnym scenárom, a teda že dáta prídu v poriadku. Druhá je failure časť, ktorá sa vykoná ak príde zo strany servera negatívna odpoveď. Ak sa teda naplní negatívny scenár, do konzoly sa vypíše príslušná chybová hláška. Tieto callbacky implementované pomocou Objective-C objektu (začína znakom ^, za ktorým pokračuje kód uzavretý v zátvorkách). Zjednodušene povedané, táto metóda vykonáva dve podstatné akcie, a to:

* zavolá metódu (void)getFacultiesWithSuccess:(EROWebServiceSuccess)success failure:(EROWebServiceFailure)failure; ktorá sa zavolá v prípade, že pozitívnej odozvy zo servra, a reprezentuje web service call (zavolanie web servisu)
* následne výsledky z tejto metódu predstavujú parameter druhej metódy, a to +(void)insertFacultiesFromArray:(NSMutableArray\*)facultiesArray, ktorá je vloženie týchto dát do korešpondujúcej tabuľky v databáze

Priblížme si metódu, ktorá je zodpovedná za volanie servisu

- (void)getFacultiesWithSuccess:(EROWebServiceSuccess)success failure:(EROWebServiceFailure)failure {

NSURL \*baseURL = [EROUtilitygetWebServicePath];

NSURL \*facultyURL = [baseURL URLByAppendingPathComponent:@"getFaculty"];

NSURLRequest \*request = [NSURLRequestrequestWithURL:facultyURL];

[NSURLConnectionsendAsynchronousRequest:request queue:[NSOperationQueuemainQueue] completionHandler:^(NSURLResponse \*response, NSData \*data, NSError \*error) {

if (response) {

NSError \*e;

id result = [NSJSONSerializationJSONObjectWithData:data options:NSJSONReadingMutableContainerserror:&e];

if (success) {

success (result);

}

} else {

if (failure) {

failure (error);

}

}

}];

}

Na začiatku metódy sú vytvorené url adresy, na ktoré sa bude posielať požiadavka na dáta. Konkrétne baseUrl premenná obsahuje základnú adresu (v našom prípade <http://localhost:8888/DP/web-app/svc>) až premenná obsahuje kompletnú url, určenú na odoslanie (<http://localhost:8888/DP/web-app/svc/getFaculty>). Takto pripravenej URL sa vytvorí inštancia triedy NSURLRequestpomocou factory metódy requestWithURL. Vytvorený request sa odošle prostredníctvom metódy sendAsynchronousRequest, kde v callbacku tejto funkcie príde odpoveď (response), dáta a error. Ak sa naplní pozitívnu scenár, a teda prídu dáta, pridané sú do success parametra metódy, inak sa pridajú do failure parametra. Aby sme si to ujasnili, ak niekto zavolá metódu:

-(void)getFacultiesWithSuccess:(EROWebServiceSuccess)success failure:(EROWebServiceFailure)failure, v prvej časti tejto metódy implementuje prevzatie dát, a v druhej časti zobrazenie chybovej hlášky (pomocou Objective-C bloku).

Ďalším krokom je vloženie získaných dát do konkrétnej tabuľky v databáze. Túto úlohu má na starosti metóda +(void)insertFacultiesFromArray, ktorá má jeden parameter a to pole, ktoré obsahuje fakulty, ktoré chceme vložiť do databázy.

+(void)insertFacultiesFromArray:(NSMutableArray\*)facultiesArray {

FMDatabase \*db = [FMDatabasedatabaseWithPath:[EROUtilitygetDatabasePath]];

[db open];

[db executeUpdate:@"DELETE FROM fakulta"];

for (id faculty in facultiesArray) {

[db executeUpdate:@"INSERT INTO fakulta (id\_fakulta, kod, nazov) VALUES (?,?,?);", [faculty objectForKey:@"id\_fakulta"], [faculty objectForKey:@"kod"], [faculty objectForKey:@"nazov"]];

}

[db close];

NSLog(@"faculties done");

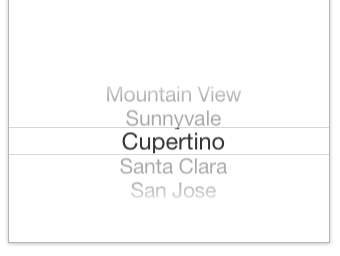
}

Pri operáciách s databázou som použil FMDB[[25]](#footnote-25) knižnicu, ktorá je tzv. obalom (wrapper) nad SQLite technológiou. Príkazy tejto knižnice sú prekladané do samotných SQLite dotazov, takže pri narábaní s databázou developer nemusí toľko písať, ako keby použil SQLite príkazy. Na začiatku algoritmu je vyžiadaná cesta k databáze, následne je otvorenie spojenie s databázou. Pred tým ako začneme vkladať dáta, je potrebné sa uistiť ze tabuľka je prázdna. Následne pomocou for cyklu sú vytvárané záznamy s požadovanými atribútmi. Na konci sa uzavrie spojenie s databázou a vypíše sa správa do konzoly o uložení fakúlt.

Takýmto spôsobom zavoláme všetky web servisy a poskytnuté dáta uložíme do databázy. Po tomto kroku môžeme dáta prezentovať používateľovi na používateľskom rozhraní.

3.3.4 Prezentovanie dát na užívateľskom rozhraní

Predtým ako zobrazíme zoznam predmetov, musíme zistiť zaradenie daného študenta, tj. fakultu, ročník, odbor a číslo krúžku. Na základe týchto informácií vytvoríme sql dotaz, ktorého výsledkom budú záznamy reprezentujúce rozvrh pre daného študenta. Tu je potrebné sa zamyslieť, ako tieto od užívateľa získať. Jedna alternatíva je, aby užívateľ vyplnil formulár zložený zo štyroch (fakulta, ročník, odbor, krúžok) polí a ručne vyplnil ich obsah pomocou virtuálnej klávesnice na obrazovke zariadenia. Toto riešenie môžeme zavrhnúť nielen z toho dôvodu. že je časovo náročné, ale aj preto, že sa užívateľ môže pri vkladaní ľahko pomýliť, respektíve nevie v akom formáte má dané dáta zadávať (či do poľa ročník má napísať číslo, alebo slovo ...). Najschodnejším riešením je zobraziť dáta, z ktorých si používateľ vybere jemu vyhovujúce. iOS vývojárska knižnica ponúka komponent, ktorý je ideálny pre naše požiadavky: UIPickerView.



Obrázok 8: UIPickerView komponent vložený v iPhone simulator v Xcode

Zdroj: Xcode

UIPickerView sa skladá z komponentov, z ktorých používateľ vyberá hodnoty dotykovými gestami, konkrétne sa ho prstom dotkne a posunie ho vo vertikálnom smere. My takéto komponenty budeme potrebovať štyri.

-(NSInteger)numberOfComponentsInPickerView:(UIPickerView \*)pickerView {

// faculty, year, department, group number

return4;

}

Takisto musíme špecifikovať koľko záznamov bude každý komponent obsahovať. Toto vyjadríme nasledovnou metódou:

-(NSInteger)pickerView:(UIPickerView \*)pickerView numberOfRowsInComponent:(NSInteger)component {

switch (component) {

case0:

returnself.faculties.count;

case1:

returnself.years.count;

case2:

returnself.departments.count;

case3:

returnself.groups.count;

}

return0;

}

Podľa toho, o ktorý komponent sa jedná, priradí sa mu konkrétny počet zobrazovaných záznamov. Kde výraz self.faculties.count vyjadruje počet prvkov v poli self.faculties. Zápisself.faculties.count je ekvivalentný so zápisom [self.faculties count];. Potrebné je špecifikovať, čo má daný komponent zobraziť na konkrétnej pozícií:

-(NSString \*)pickerView:(UIPickerView \*)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component {

switch (component) {

case0:

return [self.facultiesobjectAtIndex:row];

case1:

return [self.yearsobjectAtIndex:row];

case2:

return [self.departmentsobjectAtIndex:row];

case3:

return [self.groupsobjectAtIndex:row];

}

return0;

}

Ako samotný názov hovorí, výsledkom tejto metódy je zobraziť názov pre daný riadok v konkrétnom komponente. Jednoducho sa pre každý komponent zobrazí reťazec uložený v korešpondujúcom poli na pozícií rovnakej, ako je poradie riadku v komponente (row). Taktiež bolo potrebné myslieť na aktualizovanie obsahu jednotlivých komponentov. Myslí sa tým situácia, keď si používateľ zvolil napríklad fakultu FHI, ku nej daný ročník, odbor a číslo krúžku, no náhle zmenil fakultu na FPM, bolo potrebné aktualizovať obsah ostaných komponentov, keďže FPM fakulta má iné odbory a tie majú iné čísla krúžkov. Takže keď sa zmení obsah jedného komponentu, aktualizujú sa aj ostatné komponenty podľa neho. Takto zabezpečíme, aby dotaz ktorý sa eventuálne vykonal, bol korektný.



Obrázok 9: Domáca obrazovka s UIPickerView, ktorý je naplnený relevantnými dátami

Zdroj: Xcode

Keď si používateľ zvolí konkrétne hodnoty z komponentov, potvrdí túto voľbu tlačidlom umiestneným v dolnej časti obrazovky. V tom momente sa vezmú hodnoty z UIPickerView a podľa nich sa vyskladá dotaz, ktorý vráti kompletný rozvrh pre dané hodnoty. Metóda, ktorá zabezpečí dodanie dát na základe parametrov je + (NSMutableArray \*) getLessonsWithFacultyCode:(NSString \*) facultyCode year:(NSString \*) year departmentCode: (NSString \*) departmentCode groupNumber:(NSString \*) groupNumber;Po jej vykonaní, je užívateľ presmerovaný na obrazovku, kde je zobrazený zoznam týchto predmetov.

+(NSMutableArray \*)getLessonsWithFacultyCode:(NSString \*)facultyCode year:(NSString \*)year departmentCode:(NSString \*)departmentCode groupNumber:(NSString \*)groupNumber {

NSMutableArray \*lectures = [[NSMutableArrayalloc] init];

FMDatabase \*db = [FMDatabasedatabaseWithPath:[EROUtilitygetDatabasePath]];

[db open];

FMResultSet \*result = [db executeQuery:@"select id\_vyuka … from vyuka … where kruzok.cislo = (?) AND odbor.kod = (?) AND odbor.rocnik = (?) AND fakulta.kod = (?) order by den.id\_den, hodina.id\_hodina", groupNumber, departmentCode, year, facultyCode];

while ([result next]) {

[selfbuildLectureWithResult: result];

NSDictionary \*lecture = [selfbuildLectureWithResult:result];

[lectures addObject:lecture];

}

[db close];

return lectures;

}

Na začiatku metódy je alokovaná a inicializovaná premenná typu NSMutableArray, do ktorej budeme vkladať výsledok z SQLite dotazu. Nasleduje naviazanie kontaktu s databázou a vykonanie dotazu (pre skrátenie boli menej dôležité čast dotazu nahradené tromi bodkami). Takto získané záznamy boli v cykle while vložené do spomenutého poľa, ktoré je po uzatvorení databázového spojenia odoslané tomu, kto túto metódu zavolal. Výsledkom je zoznam výuky prezentovanej užívateľovi vo formáte riadkov v tabuľke. Jednotlivé záznamy sú farebne odlíšené na základe dňa, v ktorom sa konajú. Najdôležitejšie metódy implementované pre zobrazenie zoznamu:

- (NSInteger)numberOfSectionsInTableView:(UITableView \*)tableView {

// Return the number of sections.

return1;

}

- (NSInteger)tableView:(UITableView \*)tableView numberOfRowsInSection:(NSInteger)section {

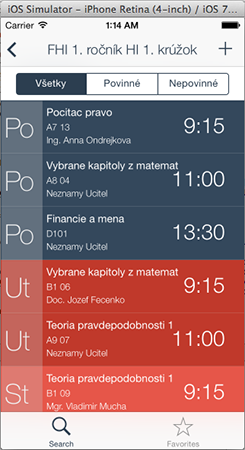
// Return the number of rows in the section.

returnself.lecturesArray.count;

}

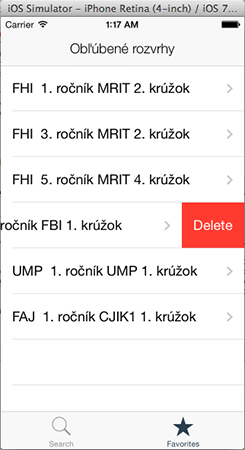
Prvá metóda hovorí o tom, koľko sekcií bude zoznam obsahovať. V našom prípade nám postačí iba jedna. Výstupom druhá metódy je počet riadkov, ktoré sa majú v zozname zobraziť, čo je dané veľkosťou poľa, v ktorom sú uložené výuky.

Keďže som chcel poskytnúť viac prehľadnosti v tomto zozname, vložil som do tohto listu komponent UISegmentedControl, ktorý poskytuje možnosť filtrovať výuku podľa toho, čí sa jedná o výuku predmetu povinného alebo voliteľného. Tento komponent pozostáva z troch tlačidiel, ktoré zobrazujú buď výuku všetkých predmetov, alebo je možné zvoliť zobraziť výuku pre povinné, respektíve voliteľné predmety. Tento komponent je uložený nad samotným zoznamom a je vždy, teda aj keď používateľ scrolluje (prezerá aj položky uložené v nižších častiach zoznamu) zoznam.



Obrázok 10: Zoznam výuky pre dané kritériá

Na základe cieľov, ktoré sme si stanovili na začiatku tvorby aplikácie, chceme používateľovi umožniť čo najjednoduchší prístup k svojmu rozvrhu. Pri prvom spustení aplikácie musí informácie o svojom štúdiu ručne pomocou komponentu UIPickerView zadať. No chceme sa vyvarovať tejto akcie pri každom ďalšom zapnutí aplikácie. Preto som implementoval funkcionalitu tzv. obľúbených rozvrhov. Keď sa užívateľ ocitne na zozname rozvrhu, ktorý si zvolil, v pravom hornom rohu sa nachádza tlačidlo plus, po ktorého stlačení sa daný zoznam výuky uloží do poľa obľúbených rozvrhov. Tým pádom pri každom ďalšom zapnutí aplikácie, má užívateľ prístup k tomuto rozvrhu na dva dotyky. Najprv si zobrazí v tab bare zoznam obľúbených rozvrhov dotykom na ikonu hviezdy a následne ďalším dotykom zvolí, ktorý zoznam chce vidieť. Tento zoznam je editovateľný, takže ak sa používateľ rozhodne odstrániť niektorý rozvrh, vykoná tak dotykom na konkrétny riadok a posunie tento riadok doľava. Zobrazí sa červené tlačidle, po ktorom stlačení sa daný záznam zmaže. Takisto musíme myslieť na to, aby nebolo možné vkladať do obľúbených rozvrhov jeden a ten istý. Toto sme zabezpečili tak, že ak už zoznam je uložený v obľúbených, tlačidlo je neaktívne a po jeho stlačení sa nevykoná žiadna akcia. Táto neaktivita je používateľovi zobrazená tak, že plus tlačidlo je sivej farby (štandardná farba pre neaktívne komponenty).



Obrázok 11: Demonštrovanie mazania v zozname obľúbených predmetov

Implementácia mazania záznamu zo zoznamu obľúbených:

- (void)tableView:(UITableView \*)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath \*)indexPath

{

if (editingStyle == UITableViewCellEditingStyleDelete) {

// Delete the row from the data source

[self.searchCriterionArrayremoveObjectAtIndex:indexPath.row];

[EROScheduleSearchCriteriontransformToDictionaryAndUpdateUserDefaultsWithCriteriaArray:self.searchCriterionArray];

[tableView deleteRowsAtIndexPaths:@[indexPath]withRowAnimation:UITableViewRowAnimationFade];

}

Daný záznam je potrebné vymazať na rôznych miestach. Najskôr z poľa, ktorý je dátovým zdrojom pre zobrazenie záznamov, potom z tabuľky samotnej (vizuálne efekt). Následne treba aktualizovať globálny objekt NSUserDefaults.

Keďže nie všetky dostupné informácie o výuke sú prezentované v riadku zoznamu, bolo potrebné implementovať funkcionalitu, ktorá by používateľovi umožňovala prezrieť detail danej výuky. Túto akciu je možné spustiť dotykom na konkrétny riadok v zozname, následne je používateľ presmerovaný na ďalšiu obrazovku, kde je zobrazený detail výuky. Späť na zoznam sa môže vrátiť tlačidlom v navigátor bare, ktoré je umiestnené v ľavom hornom rohu.

-- vlozit obrazok s detailom vyuky

1. Don't Make Me Think: A Common Sense Approach to Web Usability (http://books.google.sk/books/about/Don\_t\_Make\_Me\_Think.html?id=g1QBFJxB\_eEC&redir\_esc=y) [↑](#footnote-ref-1)
2. Zdroj: http://style.hnonline.sk/digital-132/android-drvi-apple-aj-ostatnych-ma-najvacsi-podiel-559366 [↑](#footnote-ref-2)
3. http://gadgets.ndtv.com/mobiles/news/android-retains-us-smartphone-market-lead-in-q4-ios-grows-slightly-comscore-479686 [↑](#footnote-ref-3)
4. Zdroj: http://www.apache.org/licenses/LICENSE-2.0 [↑](#footnote-ref-4)
5. http://developer.android.com/about/dashboards/index.html?utm\_source=ausdroid.net [↑](#footnote-ref-5)
6. https://developer.apple.com/support/appstore/ [↑](#footnote-ref-6)
7. Zdroj: http://www.alza.sk/search.htm?exps=iphone [↑](#footnote-ref-7)
8. https://developer.apple.com/technologies/mac/cocoa.html [↑](#footnote-ref-8)
9. http://en.wikipedia.org/wiki/IOS\_version\_history [↑](#footnote-ref-9)
10. https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html [↑](#footnote-ref-10)
11. https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/CoreServicesLayer/CoreServicesLayer.html [↑](#footnote-ref-11)
12. https://developer.apple.com/library/mac/documentation/Darwin/Conceptual/KernelProgramming/Architecture/Architecture.html [↑](#footnote-ref-12)
13. https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html [↑](#footnote-ref-13)
14. http://baddotrobot.com/blog/2012/10/06/sending-messages-vs-method-invocation/ [↑](#footnote-ref-14)
15. http://davedelong.tumblr.com/post/58428190187/an-observation-on-objective-c [↑](#footnote-ref-15)
16. https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ObjCRuntimeRef/Reference/reference.html [↑](#footnote-ref-16)
17. https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/DefiningClasses/DefiningClasses.html#//apple\_ref/doc/uid/TP40011210-CH3-SW1 [↑](#footnote-ref-17)
18. https://developer.apple.com/library/mac/documentation/developertools/conceptual/instrumentsuserguide/Introduction/Introduction.html [↑](#footnote-ref-18)
19. Prvá je SQLite databáza, ktorá je vložená do každého iPhone a Android zariadenia, taktiež do webových prehliadačov Chrme a Firefox. Iba v druhom kvartáli 2013 bolo vyrobených 213 miliónov smartphono, z čoho 200 miliónov boli zariadenia s iOS a Androidom. [↑](#footnote-ref-19)
20. http://en.wikipedia.org/wiki/Web\_service#Web\_API [↑](#footnote-ref-20)
21. http://en.wikipedia.org/wiki/Web\_service#Explanation [↑](#footnote-ref-21)
22. http://www.copterlabs.com/blog/json-what-it-is-how-it-works-how-to-use-it/ [↑](#footnote-ref-22)
23. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html [↑](#footnote-ref-23)
24. https://www.sqlite.org/sqlite.html [↑](#footnote-ref-24)
25. https://github.com/ccgus/fmdb/ [↑](#footnote-ref-25)