

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gašper Cvenkel

Uporaba novih gradnikov iOS 5 skozi primer

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Peter Peer

Ljubljana, 2012

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljane ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil $\text{\LaTeX}$.



Št. naloge: 01858/2012

Datum: 03.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GAŠPER CVENKEL**

Naslov: **UPORABA NOVIH GRADNIKOV IOS 5 SKOZI PRIMER
USE OF NEW COMPONENTS IN IOS 5**

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Identificirajte nove gradnike v iOS 5. Bistvene gradnike primerjajte s konkurenti. Nato uporabite nekatere izmed njih v konkretnem primeru, na katerem razložite tudi značilnosti razvoja v programskem jeziku Objective C, v okolju Xcode. Aplikacijo pripravite za distribucijo preko trgovine App Store.

Mentor:


doc. dr. Peter Peer



Dekan:


prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Gašper Cvenkel,

z vpisno številko 63050019,

sem avtor diplomskega dela z naslovom:

Uporaba novih gradnikov iOS 5 skozi primer

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Peter Peer
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 10.9.2012

Podpis avtorja:

Zahvala

Hvala moji družini za podporo in brce v rit skozi študij. Zahvalil bi se tudi Maruši, ki je pridno prenašala moje negotovanje ob izdelavi diplome in pomankljivo posvečanje časa.

Zahvalil bi se tudi mentorju doc. dr. Petru Peeru, ki mi je s svojimi nasveti in znanjem pomagal izdelati diplomo v obliki kakršna je pred vami.

Vsem še enkrat iskrena hvala.

Maruši...

Kazalo

Povzetek	1
Abstract	2
1 Uvod	3
2 Razvojna platforma	5
3 iOS 5	7
3.1 Povzetek iOS 5	7
3.2 iCloud	8
3.3 Scenarij	10
3.4 Center obvestil	10
4 Primerjava novitet med konkurenti	13
4.1 Apple iCloud, Google Drive, Amazon Cloud Drive, Dropbox in Microsoft SkyDrive	13
4.1.1 Kapaciteta	13
4.1.2 Vrste datotek	15
4.1.3 Enostavnost uporabe	16
4.1.4 Podprte naprave	17
4.1.5 Stroški zakupa prostora	17
4.1.6 Povzetek	18
4.2 Scenarij	20
4.3 Center obvestil	21
5 Applova spletna trgovina	23
5.1 Kaj je Applova spletna trgovina?	23
5.2 Modeli trženja v Applovi spletni trgovini	23
5.2.1 Nakup aplikacije	23
5.2.2 Nakup znotraj aplikacije	23

5.2.3	Reklamni oglasi	24
5.2.4	Kaj zahteva Apple od nas ob objavi?	24
6	Osnove Objective C v Xcode okolju razložene na primeru razvoja aplikacije	27
6.1	Uvod v Objective C	27
6.1.1	Klicanje metod	28
6.1.2	Gnezdenje	28
6.1.3	Večvhodne metode	29
6.1.4	Metode pridobivanja in nastavljanja podatkov	29
6.1.5	Kreiranje objektov	30
6.1.6	Oblikovanje vmesnika razreda	30
6.1.7	Implementacija razreda	32
6.1.8	Metoda init	33
6.1.9	Prikazovanje sporočil v konzoli	33
6.1.10	Poenostavljanje metod pridobivanja in nastavljanja podatkov	34
6.1.11	Kategorije	35
6.2	Razvoj aplikacije	36
6.2.1	Začetne nastavitve projekta	36
6.2.2	Scenarij	38
6.2.3	Core Data	43
6.2.4	Povezovanje nadzornikov pogleda s pripadajočimi datotekami	45
6.2.5	Izbiralec slike	45
6.2.6	Shranjevanje v iCloud	48
6.2.7	Branje iz Core Data	49
6.2.8	Meni akcij	50
6.2.9	Brisanje entitete iz Core Data	51
6.2.10	Generiranje filma	52
6.2.11	Dodajanje spremljevalne glasbe	52
6.2.12	Predvajanje filma	54
6.2.13	Opozorila	55
6.2.14	Nastavitve potrebne za omogočanje nakupov znotraj aplikacije	56
7	Testiranje aplikacije	61
7.1	Popravki aplikacije	61

8 Objavljanje aplikacije v Applovi spletni trgovini	65
9 Zaključek	67
Dodatka	69
A iCloud koda	69
B Metoda dodajanja spremljevalne glasbe k filmu	75
Seznam slik	79
Seznam tabel	81
Literatura	83

Seznam uporabljenih kratic in simbolov

API (angl. Application Programming Interface) – Vmesnik, ki zagotavlja dostop do funkcij operacijskega sistema ali drugih računalniških programov.

ARC (angl. Automatic Reference Counting) – Lastnost prevajalnika, ki nadzira upravljanje s spominom, kar poenostavi razvoj aplikacij.

FPS (angl. Frames Per Second) – Število sličic na sekundo.

IDE (angl. Integrated Development Environment) – Aplikacija s širokim naborom orodij, ki programerjem omogoča lažji razvoj programske opreme.

OS (angl. Operating System) – Operacijski sistem je programska oprema, nujna za delovanje računalnika. Deluje kot vmesnik med uporabnikom in strojno opremo računalnika.

SaaS (angl. Software as a service) – Uporaba programske opreme kot storitve.

SDK (angl. Software Development Kit) – Orodja za razvoj programske opreme.

UI (angl. User Interface) – Uporabniški vmesnik.

Wi-Fi – Tehnologija, ki omogoča elektronskim napravam brezžično izmenjavanje podatkov.

WP (angl. Windows Phone) – Operacijski sistem za mobilnike podjetja Microsoft.

Povzetek

Danes je tehnološki razvoj mobilne telefonije zelo hiter, zato se na trgu pojavlja vse več pametih mobilnih aparatov podprtih z različnimi operacijskimi sistemi, ki pa se tudi razvijajo vzporedno, dobivajo nove funkcije in izboljšave. Zato smo v delu predstavili novosti iOS 5 (iCloud, scenarij, center obvestil), jih primerjali s konkurenti ter predstavili prednosti in slabosti. Na sam nagel razvoj kaže tudi dejstvo, da je že napovedan prihod iOS 6, kljub temu da še ni minilo leto od prihoda njegovega predhodnika. Skozi delo na primeru aplikacije smo spoznali, kako se lotiti projekta izdelave iOS aplikacije, gradnike UV pri iOS in orodja, v katerih izdelujemo sam projekt. Spoznali smo tudi, da iCloud v kombinaciji s Core Data še ni popolnoma zrela tehnologija in da še ni pripravljena na uporabo v produkciji.

Ključne besede:

mobilne aplikacije, novi gradniki iOS 5, Xcode, Objective C

Abstract

Technological development of mobile technology is very fast now days, which contribute to increasing number of smart phones available on the market. Those phones are driven by different operating systems, which are developing, getting new features and upgrades. That is one of the reasons to introduce new features of iOS 5 (iCloud, Storyboard, Notification Center), compare them with competition and present pros and cons. The very rapid development is also reflected by the fact, that it is already announced arrival of iOS 6, although it has not yet passed a year since arrival of his ancestor.

Through the work we have learned how to approach the project development of iOS applications, widgets, UI and tools for building iOS applications. We have also learned that Core Data with iCloud is not mature enough for usage in production environment.

Key words:

mobile application, iOS 5 new features, Xcode, Objective C

Poglavje 1

Uvod

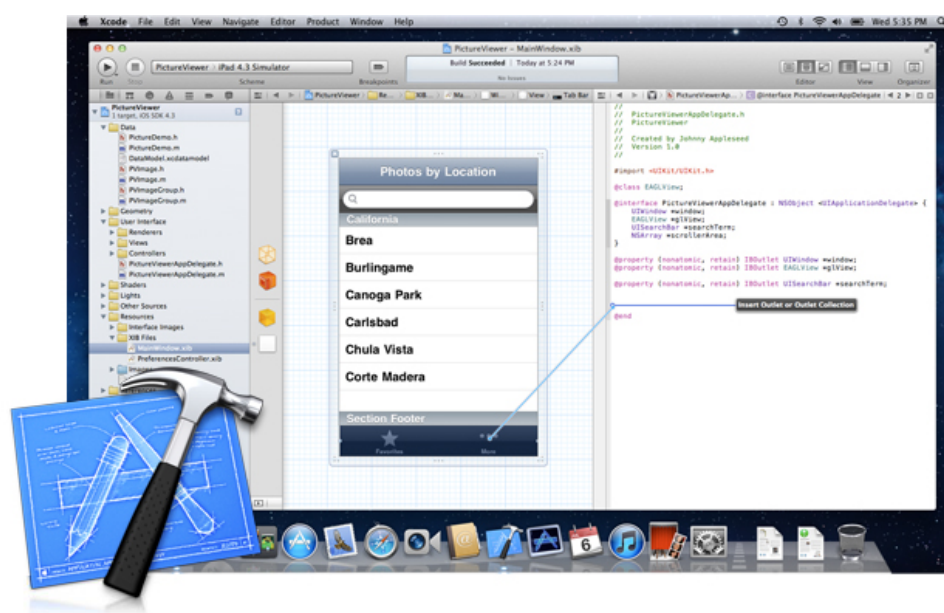
Danes lahko na trgu najdemo že veliko število pametnih mobilnih telefonov, trendi pa nakazujejo, da bo to v prihodnosti še naraslo. Prevladujejo telefoni z nameščenim mobilnim operacijskim sistemom iOS in Android, večji igralec pa postaja tudi operacijski sistem Windows Phone.

Skozi delo bomo primerjali novosti operacijskega sistema iOS 5 s konkurenti, predstavili prednosti in slabosti. Poudarek bomo dali predvsem na oblačne storitve.

V poglavju 2 si bomo pogledali, kaj sploh potrebujemo za razvoj iOS aplikacij. Poglavje 3 nam prikaže novosti iOS 5, v poglavju 4 pa so te novosti primerjane s konkurenčnimi. V poglavju 5 bomo spoznali, kaj je Applova spletna trgovina (angl. App Store) in kakšne vrste nakupov lahko opravljamo v njej. V poglavju 6 bodo razložene osnove programskega jezika Objective C in praktična uporaba novosti prikazana na primeru razvoja aplikacije. V poglavju 7 bomo testirali aplikacijo in dodali nekatere izboljšave. 8 poglavje opisuje, kako naložiti aplikacijo v trgovino, v poglavju 9 pa bomo zbrali še zaključne misli.

Poglavje 2

Razvojna platforma



Slika 2.1: Xcode je močno orodje za razvijanje aplikacij.

Za prikaz delovanja in uporabe novih gradnikov bomo potrebovali računalnik z nameščenim operacijskim sistemom MacOS in razvojnim okoljem Xcode, katerega UV lahko vidimo na sliki 2.1. Razvojno okolje Xcode lahko brezplačno namestimo z Applove spletne trgovine. Ko imamo nameščeno okolje Xcode, lahko razvijamo aplikacije in jih testiramo v simulatorju, v kolikor pa bi želeli razvito aplikacijo testirati na napravi ali jo prenesti v Applovo spletno trgovino, pa se moramo registrirati kot Apple razvijalec (angl. Apple Developer),

s čimer pridobimo dostop do bogate Appleove dokumentacije, baze knjižnic, video posnetkov in ostalih virov. Sama registracija pa ni brezplačna, saj moramo plačati letno članarino v višini 99\$. Ker bomo mi testirali aplikacijo na dejanskih napravah, bomo opravili tudi registracijo, ampak več o tem v poglavju 5.

Xcode je orodje za razvijalce, ki ustvarjajo aplikacije za Appleovo opremo Mac, iPhone, iPod in iPad. Zadnja različica Xcode (4) je optimizirana za lažji in hitrejši razvoj aplikacij, saj so vsa orodja zbrana na enem mestu. Tako je možno razvijanje UV, pisanje kode in razhroščevanje kode v enem samem oknu.

Xcode 4 deluje na operacijskem sistemu OS X Lion+ in vključuje primarno Xcode IDE, inštrumente, simulator za iOS naprave, najnovejše Mac OS X in iOS SDK-je.

Poglavje 3

iOS 5



Slika 3.1: iOS 5 je napredni mobilni operacijski sistem.

3.1 Povzetek iOS 5

iOS je mobilni operacijski sistem, ki je bil razvit pod okriljem družbe Apple Inc. (predstavitveni logo je na sliki 3.1). Uradno je izšel leta 2007 in je bil sprva namenjen samo napravam iPhone in iPod Touch, nato pa so ga priredili tudi za uporabo na iPad-u in Apple TV-ju. Zanj je značilno, da Apple ne dovoljuje nameščanja na drugih platformah kot njegovih. UV temelji na direktni manipulaciji, kjer lahko uporabljamo tudi večdotične poteze (angl. multitouch gestures) [1].

iOS 5 prinaša več kot 200 novih funkcij, ki jih njegovi predhodniki niso imeli [2]. Ker je vseh preveč bomo našteali le najvidnejše:

- Center opozoril (angl. Notification Center) – Vsa obvestila, ki jih prejmete, so zbrana na enem mestu.

- iCloud – Oblačna storitev, kateri težko določimo klasifikacijo, vendar prevladuje mnenje, da spada med SaaS storitve [3, 4].
- Opomniki (angl. Reminders) – Funkcija, ki nam omogoča upravljanje z opravili.
- iMessage – je nova sporočilna storitev, s katero si lahko preko internetnega dostopa pošiljamo neomejeno število sporočil. Sporočila lahko vsebujejo tekste, slike, videe in kontakte.
- Trafika (angl. Newsstand) – Aplikacija, kjer imamo zbrane vse revije, na katere smo naročeni.
- Kamera (angl. Camera) – Z iOS 5 se je tudi drastično izboljšala uporaba vgrajene kamere.
- Brez PC-ja (angl. PC-Free) – Vse začetne namestitve lahko opravimo preko Wi-Fi brez kakršnekoli potrebe po priklopu na računalnik.
- Wi-Fi sinronizacija (angl. Wi-Fi Sync) – Napravo lahko preprosto sinhroniziramo z računalnikom preko Wi-Fi.

3.2 iCloud



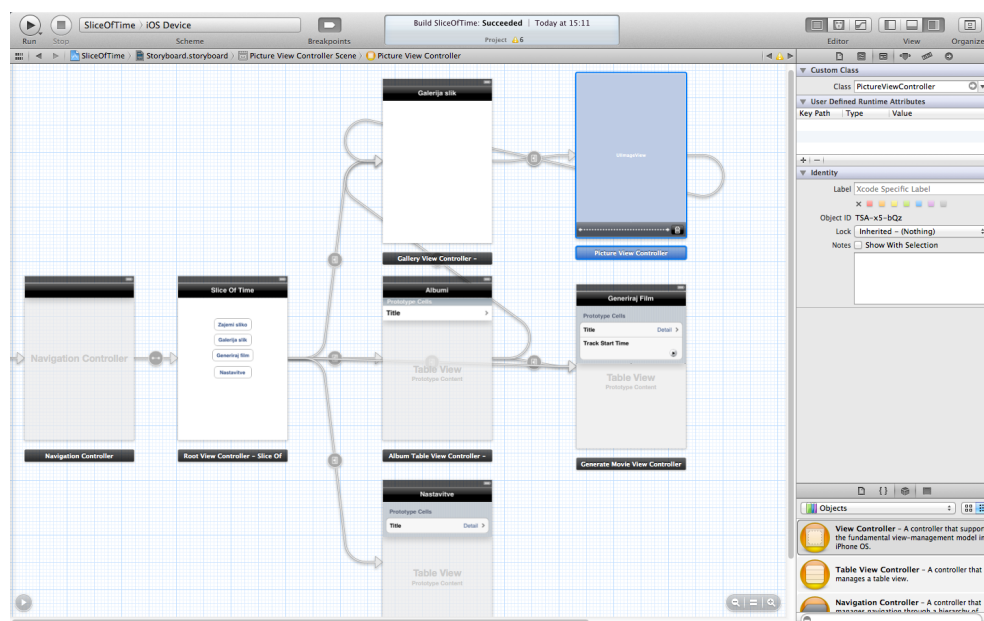
Slika 3.2: iCloud je Applova oblachna storitev.

iCloud, prepoznaven po ikoni na sliki 3.2, je Applova oblachna storitev, ki samodejno povezuje aplikacije in naprave z operacijskim sistemom iOS 5, deloma pa ga lahko uporabljamo tudi na Mac in Windows računalnikih. Steve Jobs je na svoji zadnji predstavitvi obljubil, da bo iCloud storitev, ki “preprosto deluje”. In po začetni namestitvi storitev resnično dobro deluje. Večina iCloud storitev je popolnoma brezplačnih. Paziti moramo le na omejitve prostora na iCloud strežnikih; brezplačno lahko uporabljamo 5GB prostora, za 16EUR na leto

lahko zakupimo dodatnih 10GB, za 32EUR letno dodatnih 20GB, za 80EUR letno pa dodatnih 50GB (avgust, 2012).

Uporaba iCloud-a:

- Koledar, e-pošta, kontakti: iCloud poskrbi za potisk nove pošte prav na vse iCloud naprave. Hkrati iCloud poskrbi tudi za sinhronizacijo koledarjev in kontaktov, tako na mobilnih napravah kot na Mac-u ali PC-ju. Tukaj je še dodatna uporabna funkcija deljenja koledarja z ostalimi iCloud uporabniki, tako da lahko celotna družina uporablja koledar za skupne dogodke.
- Tok slik (angl. Photo Stream): Deljenje fotografij z vsemi napravami, ki so povezane z iCloud-om. Preprosto posnameš fotografijo z eno od naprav in če je funkcija tok slik omogočena, je slika že čez nekaj trenutkov vidna tudi na drugih napravah povezanih z istim iCloud računom.
- Aplikacije, knjige in varnostne kopije: iCloud poskrbi tudi za to, da imajo vse naprave iste aplikacije, knjige (iz aplikacije iBooks) in da se v oblaku redno shranjuje varnostna kopija podatkov in nastavitev za primer, da gre kaj narobe. Novost je tudi, da naprave ni potrebno priključiti na računalnik, saj namestitev lahko opravimo zgolj s povezavo na Wi-Fi omrežje.
- iTunes v oblaku: Ko kupiš glasbo v iTunes Store, imajo vse naprave dostop do te glasbe.
- Dokumenti v oblaku: iCloud zna poleg vsega tega med napravami samodejno prenašati tudi dokumente.
- Funkciji za iskanje prijateljev in naprav (angl. Find My Friends, Find My iPhone): Za vse pozabljive, ki ste nagnjeni k izgubljanju prijateljev ali naprav, bosta ti dve funkciji gotovo zelo uporabni. Možnost iskanja izgubljenega telefona sicer ni nova storitev, iCloud pa sedaj to funkcijo podpira še na drugih napravah, vključno z Mac računalniki (kar je uporabno predvsem na prenosnikih). Uporabno je tudi to, da lahko izgubljeni napravi pošljemo sporočilo ali se odločimo za zaklep in celo brisanje podatkov na daljavo, v primeru, da se najditelj ne odziva na naša prijazna sporočila.



Slika 3.3: Scenarij.

3.3 Scenarij

Scenarij (angl. Storyboard) je vizualna predstavitev UV iOS. Če pogledamo sliko 3.3, vidimo najprej platno (angl. Canvas), na katerem je več nadzornikov pogleda (angl. View Controller) povezanih med seboj s povezavami (angl. Segue). Vsaka izmed teh povezav pravzaprav predstavlja prehod med dvema nadzornikoma pogleda. Vsak izmed teh nadzornikov lahko vsebuje posamezne gradnike (angl. views), kot so gumbi, labele, tabele, tekstovna polja itd. Tako lahko že na začetku dobimo dober občutek, kako bo izgledal naš UV in ali smo na kaj pozabili. Uporaba scenarija je tudi priporočen način za oblikovanje UV aplikacije.

3.4 Center obvestil

Center obvestil (angl. Notification Center) je funkcija iOS, s katero lahko uporabnik vidi opozorila, ki jih generirajo aplikacije na napravi. Primer za naprave iPod, iPad in iPhone je na sliki 3.4.

Center obvestil je prenovljen sistem za obveščanje v iOS 5. Obvestila lahko pregledamo s potegom navzdol na zgornjem delu zaslona. Klik na eno izmed



Slika 3.4: Center obvestil na napravah iPod, iPad in iPhone.

obvestil odpre aplikacijo, ki je to obvestilo generirala.

Obstajata dve vrsti obvestil: Prvo je reklamna pasica (angl. Banner), ki se prikaže na vrhu zaslona (slika 3.5), drugo pa modro pojavno opozorilo (angl. Alert) (slika 5.1).



Slika 3.5: Reklamna pasica.

Center obvestil ima tudi dva pripomočka, vreme (angl. Weather) in pregled delnic (angl. Stocks), ki prikazujejo vremenske podatke za trenutno uporabnikovo lokacijo in informacije o vseh delnicah, ki jih je uporabnik predhodno izbral za prikaz. Vse aplikacije, ki uporabljajo potisnjena obvestila (angl. Push Notifications), pravzaprav uporabljajo center obvestil na iOS 5.

Za vsako aplikacijo, ki ima možnost generiranja obvestil, lahko v nastavitvah prilagodimo:

- ali dovolimo aplikaciji objavljati obvestila,
- maksimalno število obvestil, ki jih aplikacija lahko objavi,

- izgled obvestila (brez, reklamna pasica ali pojavno opozorilo),
- ali dovolimo aplikaciji prikaz značk (angl. Badge App Icon),
- ali dovolimo prožitev zvoka ob generiranju obvestila,
- in ali je obvestilo vidno na zaklenjenem zaslonu.

Uporabnik se lahko odzove na obvestilo kljub zaklenjenemu zaslonu tako, da potegne po njem od leve proti desni.

Poglavje 4

Primerjava novitet med konkurenti

4.1 Apple iCloud, Google Drive, Amazon Cloud Drive, Dropbox in Microsoft SkyDrive

4.1.1 Kapaciteta

Mesto	Storitev	Zastonjski prostor	Max prostor
1.	Dropbox	18GB	1TB+
2.	Cloud Drive	5GB	1TB
3.	Google Drive	5GB	16TB +Pisarniška orodja +20000 pesmi
4.	SkyDrive	7(25)GB	125GB+Pisarniška orodja
5.	iCloud	5GB	55GB+Pisarniška orodja +katerikoli iTunes nakup

Tabela 4.1: Pregled prostorskih lastnosti oblčnih storitev.

Očitno mesto za začetek primerjave storitev v oblaku je prostorska kapaciteta, vendar ker ima vsaka storitev še dodatne funkcionalnosti, primerjava ni tako enostavna kot se zdi na prvi pogled. Pregled prostorskih lastnosti oblčnih storitev so navedene v tabeli 4.1. S prostorskega vidika so najbolj očitne storitve Amazon Cloud Drive [5], Dropbox [6] in do neke mere SkyDrive [7].

Z Amazon dobiš kolikor prostora hočeš med 5GB in 1TB, odvisno od potreb. Seveda je prvih 5GB prostora na voljo brezplačno.

Prav tako dobiš pri Dropboxu 2GB začetnega zastonjskega prostora, kateri pa lahko naraste celo do 18GB, če se na vaše priporočilo pridruži dovolj novih članov. Prav tako dobiš brezplačen prostor z uporabo njihovih novih funkcij. V primeru, da ustvarite račun za skupine, je maksimalen prostor, ki ga imate na voljo 1TB+.

SkyDrive je Microsoftova storitev, ki nam omogoča brezplačno shranjevanje do 7GB, če pa smo se vpisali pred aprilom 2012 nam je bilo zagotovljenega kar 25GB brezplačnega prostora. K temu prostoru pa je potrebno prišteti še prostor, ki ga porabijo drugi produkti Windows Live, s katerimi urejamo besedila, preglednice, e-maile itd.

Primerjava Google Drive-a [8] z ostalimi storitvami pa je že nekoliko bolj zapletena. Pri njem se dobi 5GB brezplačnega prostora, poleg tega pa še 1GB prostora za Picaso. Poleg tega se v ta prostor ne štejejo še dokumenti (angl. Google Docs) in e-maili. V kolikor je prostora premalo ga lahko dokupimo po Googlovem mesečnem planu tja do vsega 16TB. Poleg tega lahko shranimo še 20.000 pesmi v okviru nove storitve Google Play Music.

Najtežja za primerjavo pa je zagotovo Applova storitev iCloud [9], ki omogoča 5GB brezplačnega prostora za shranjevanje. Kaj pa lahko shranjujemo v iCloud pa je zelo omejeno. Neposrednega dostopa pravzaprav sploh ni, ampak se v iCloud avtomatično sinhronizirajo elektronska pošta, kontakti, koledar, datoteke za obnovitev sistema, pisarniški dokumenti (angl. office documents) in beležke. Poleg tega se v ta prostor shranjujejo tudi slike (omejitev je 1.000 slik), ki pa so shranjene 30 dni, tako da ta prostor ni namenjen stalni shrambi letih. Kar se tiče aplikacij, knjig in glasbe, ki so bile kupljene preko iTunes, jih lahko hranimo neomejeno število, saj se dejansko ne shranjujejo v iCloud, ampak so povezane le z Applovim računom uporabnika. Edina omejitev je 25.000 pesmi, ki jih niste kupili preko iTunes nakupov. Maksimalen prostor, ki ga lahko dodatno zakupimo za iCloud je 5GB.

Če potegnemo črto, moramo reči, da je primerjava po absolutnem prostoru, ki nam ga ponujajo posamezne storitve kar zapletena. Ampak iz vsega lahko izluščimo, da Apple ponuja najmanj zastonjskega prostora in ima tudi najmanjši maksimalen prostor, ki ga lahko zakupimo. Prostorsko gledano Applov iCloud rešujejo bonusi v zvezi z nakupi preko iTunes.

Naslednja primerjava je med Microsoftovo storitvijo SkyDrive in Googlovo storitvijo Google Drive. Za navadnega uporabnika ima rahlo prednost SkyDrive, saj ima nekoliko več zastonjskega prostora, če pa imamo z Googlom sklenjen Google Apps uporabniški račun, pa nam ta omogoča več maksimalno zakupljenega prostora.

Po drugi strani pa je Amazonov način uporabe Cloud Drive-a preprost in

razumljiv, ampak v primerjavi z Dropbox-om še vedno potegne krajšo palico. Dropbox-ov račun za skupine in zmožnost pridobiti kar 18GB brezplačnega prostora je pač boljša ponudba.

4.1.2 Vrste datotek

Mesto	Storitev	Zmožnost shranjevanja
1.	Cloud Drive	karkoli
1.	Dropbox	karkoli
2.	Google Drive	karkoli (maks. velikost datoteke je 10GB)
3.	SkyDrive	karkoli (maks. velikost datoteke je 2GB)
4.	iCloud	karkoli

Tabela 4.2: Pregled datotečnih lastnosti oblačnih storitev.

Kar se tiče raznolikosti datotek imata gladko prednost Amazonov Cloud Drive in Dropbox (tabela 4.2). Nanju lahko nalagamo karkoli digitalnega, ne glede na vrsto in velikost datoteke. Podobno drži tudi za Google Drive, ki pa ima omejitev velikosti datoteke na 10GB, kar je precej, ampak že lahko predstavlja probleme pri velikih HD filmih.

Tehnično gledano lahko shranimo tudi v iCloud karkoli, saj lahko v poštni predal shranimo osnutek s priponkami, v katerih so datoteke z željeno vsebino. Je pa pri tem potrebno omeniti, da lahko s tem hitro zapolnimo prostorske kapacitete, ki so nam na voljo ter imamo tudi omejitev pri velikosti datoteke.

V SkyDrive lahko naložimo karkoli želimo, dokler je datoteka manjša od 2GB. Ima tudi možnost spletnega pregledovalnika slik, s katerim enostavno gledamo slike tudi na spletu. Podobno opcijo ima tudi Google s Picaso. Ne glede na vse pa se storitve, glede na omejitve pri vrstah datotek, ne približajo Amazonu in Dropbox.

4.1.3 Enostavnost uporabe

Mesto	Storitev
1.	iCloud
2.	Dropbox
3.	SkyDrive
4.	Google Drive
5.	Cloud Drive

Tabela 4.3: Pregled enostavnosti uporabe.

Z vidika enostavnosti uporabe (tabela 4.3) so se najbolj potrudili pri Applu, pa čeprav so na račun te enostavnosti prišli na tržišče nekoliko kasneje. Njihov način sinhronizacije vsebine med napravami poteka povsem avtomatsko. Vse datoteke so zbrane na enem mestu, ki ga sploh ni treba posebej urejati, razen zbrisati stare maile in shraniti slike s toka slik. Preprostost temelji na tem, da uporabnik ni zmožen direktno dostopati do njihovega prostora.

Koncept Dropbox-ovega čarobnega žepa (angl. magic pocket) je malo težji za razumevanje in uporabo, ampak ko ga enkrat razumemo, je mogoče še bolj brežhiben kot Applov iCloud. Sinhroniziranje datotek med napravami se izvaja samodejno, vse kar je potrebno storiti je le, da datoteko potegnemo v določeno mapo. Nato je ta datoteka dostopna z vseh naprav na katerih smo namestili Dropbox aplikacijo, do njih pa lahko dostopamo tudi preko brskalnika. Če hočemo posamezne datoteke deliti s prijatelji, jim enostavno dodelimo pravice za dostop do mape. Dropbox pa ima tudi novo storitev Get Link s katero lahko ustvarimo povezavo do posamezne datoteke in jo pošljemo tudi tistim, ki niso uporabniki Dropbox-a ter jim tako omogočimo dostop do nje.

Tako Google kot Microsoft sta se pri razvijanju svojih storitev zgledovala po Dropbox-u. Tako sta pri obeh na voljo tako namizne kot tudi mobilne aplikacije za uporabo storitev ter še dostop do datotek preko brskalnika. Tukaj pa velja omeniti še eno funkcijo SkyDrive, ki nam omogoča dostop do vseh datotek na računalniku z oddaljene lokacije, seveda ob pogoju, da je računalnik prižgan, ima povezavo z internetom in je ta funkcija omogočena.

Amazonov Cloud Drive je najbolj pregleden. Dobiš toliko prostora kolikor ga plačaš, naložiš lahko kar hočeš in vse je shranjeno na enem mestu. Slabost te storitve je, da potrebujemo ločena odjemalca: enega za nalaganje datotek, drugega pa za pretok teh datotek nazaj. Poleg tega ima Amazon enako veliko glasbeno knjižnjico kot iTunes, ampak se morajo vseeno pesmi prenesti na server, namesto da bi se avtomatsko primerjali z obstoječimi pesmimi na serverju.

Posledično nalaganje velike lastne glasbene knjižnice vzame kar nekaj časa.

4.1.4 Podprte naprave

Mesto	Storitev	Podprte naprave
1.	SkyDrive	katerakoli
2.	Dropbox	katerakoli, ni namenske aplikacije za WP
3.	Google Drive	katerakoli, ni namenskega aplikacije za iOS, WP
4.	Cloud Drive	Android, PC, Mac in katerakoli Flash naprava
5.	iCloud	PC, Mac in iOS

Tabela 4.4: Pregled podprtih naprav.

V tej sekciji bomo poskušali odgovoriti, do katere oblačne storitve lahko dostopamo z največ naprav (tabela 4.4). Če pogledamo samo po številkah so zmagovalci zagotovo SkyDrive, Dropbox in Google Drive, saj do njih dostopamo s katerekoli naprave, če imamo le povezavo s spletom. Kar tem trem storitvam manjka so edino namenske aplikacije za nekatere priljubljene platforme (z izjemo SkyDrive).

Za temi tremi, po številu, je Amazonov Cloud Drive, do katerega lahko dostopate na katerikoli napravah, če leta podpira Adobe Flash. Torej ne deluje na iOS in Windows Phone 7.

Na zadnje mesto po številu podprtih naprav pride zagotovo iCloud, saj do te storitve lahko dostopamo v polni meri z Applovih naprav, imamo pa tudi namenski odjemalec za PC v okviru iTunes, tako da lahko dostopamo do slik, videov, glasbe, ipd.

4.1.5 Stroški zakupa prostora

Pri primerjavi cen zakupnih planov pa nas čaka kar nekaj dela, če hočemo narediti ustrezno primerjavo. V tabelah od 4.5 do 4.9 lahko vidimo plane plačevanja kot so navedeni na spletnih straneh ponudnikov storitev (avgust 2012) [5, 6, 7, 8, 9]. Da pa bo primerjava bolj pregledna bomo mi preračunali, koliko bi nas stal zakup 1GB na leto pri vsaki storitvi.

Iz tabel lahko vidimo, da pri nekaterih uporabnikih cena pada z zakupom večje količine prostora. Pri teh bomo upoštevali povprečno vrednost pri 100GB prostora in so vidne v tabeli 4.10.

Tako lahko zaključimo, da se cene ponudnikov zelo razlikujejo, saj je iCloud kar štirikrat dražji kot najcenejša SkyDrive in Cloud Drive. Google Drive

Prostor	Mesečni stroški
25GB	2,49\$
100GB	4,49\$
200GB	9,49\$
400GB	19,49\$
1TB	49,49\$
2TB	99,49\$
4TB	199,49\$
8TB	399,49\$
16TB	799,49\$

Tabela 4.5: Google Drive – pregled mesečnih stroškov.

Prostor	Letni stroški
20GB	10\$
50GB	25\$
100GB	50\$

Tabela 4.6: SkyDrive – pregled letnih stroškov.

jima sicer tesno sledi, medtem ko je Dropbox, po mojem mnenju na račun popularnosti in prepoznavnostni, nekje na sredini med ponudniki. Poleg tega velja še omeniti, da se nam kapaciteta Gmail-a poveča na 25GB ob zakupu dodatnega prostora za Google Drive, Amazon pa nam ponuja 20GB dodatnega prostora na leto, če kupimo pri njih vsaj en glasbeni album.

4.1.6 Povzetek

Amazonov Cloud Drive

Za Cloud Drive zagotovo lahko rečemo, da ima potencial. Ima dobro ponudbo glede prostora in tudi možnost, da dobite 20GB brezplačnega prostora, če ste le vsako leto pripravljeni kupiti glasbeni album iz njihove spletne trgovine. V

Prostor	Mesečni stroški	Letni stroški
100GB	9.99\$	99\$
200GB	19.99\$	199\$
500GB	49.99\$	499\$

Tabela 4.7: Dropbox – pregled mesečnih in letnih stroškov.

Prostor	Letni stroški
20GB	10\$
50GB	25\$
100GB	50\$
200GB	100\$
500GB	250\$
1000GB	500\$

Tabela 4.8: Amazon Cloud Drive – pregled letnih stroškov.

Prostor	Letni stroški
10GB	20\$
20GB	40\$
50GB	100\$

Tabela 4.9: iCloud – pregled letnih stroškov.

ta prostor lahko naložite karkoli hočete in do tega tudi dostopate z velikega števila naprav. Kar se mu šteje v slabost je, da vizualno ni tako privlačen in da nima avtomatične sinhronizacije.

Appleov iCloud

iCloud je eleganten, preprost za uporabo, a zelo nefleksibilen, kar se tiče direktnega dostopa do datotek. Če ste uporabnik Appleovega sistema, enostavno ne morete mimo njega. Od vseh naštetih storitev ima tudi najboljše implementirano upravljanje z glasbo. Če pa niste uporabnik Appleovega sistema, pa se vam ne splača izbrati te storitve.

Mesto	Storitev	Letni strošek 1GB
1.	SkyDrive	0,5\$
1.	Cloud Drive	0,5\$
2.	Google Drive	0,54\$
3.	Dropbox	0,99\$
4.	iCloud	2\$

Tabela 4.10: Pregled stroškov za 1GB na leto.

Dropbox

Dropbox je trenutno najbolj dodelana oblachna storitev te vrste na trgu. Dostopna je takorekoč s katerekoli naprave, ima veliko število namenskih odjemalcev, je elegantna in preprosta za uporabo. Kar mogoče malo razočara je začetni zastonski prostor (2GB), tako da če hočemo več prostora moramo nadlegovati prijatelje, da se pridružijo tej storitvi ali pa plačati kar nekaj denarja.

Google Drive

Google ima veliko število uporabnikov svojih storitev, tako Gmail-a, kot Google+ ipd. Ti uporabniki se bodo prej ali slej srečali z Google Drive-om, pa tudi če ne bo to njihova glavna oblachna storitev za sinhronizacijo. Nakup dodatnega prostora ni najcenejši, tudi podpora napravam (zanekrat) ni najboljša.

Microsoftov SkyDrive

Je enostaven za uporabo, z veliko zastonskega prostora. Škoda je le, da so začetni zastonski prostor zmanjšali iz 25GB na 7GB, saj bi to pomenilo veliko prednost pred konkurenti. Dodali so tudi aplikacijo, ki nudi podporo SkyDrive napravam z nameščenim Android operacijskim sistemom.

4.2 Scenarij

O scenariju smo pisali že v poglavju 3.3. V tej sekciji pa se bomo usmerili na priljubljene SDK-je za razvoj mobilnih aplikacij in poskušali poiskati podobno funkcijo tudi pri konkurentih.

Pri razvoju za mobilno platformo Android sem bil popolnoma neuspešen in nisem našel podobne funkcije za razvoj UV. Pri razvoju aplikacij za Windows Phone 7 sem bil prav tako neuspešen pri iskanju konkurenčne rešitve. Tako lahko zaključimo, da je scenarij funkcija, ki nam zelo skrajša čas potreben za razvoj aplikacije in da podobne funkcije lahko tudi kaj kmalu pričakujemo pri konkurentih, v kolikor želijo privabiti še več razvijalcev za njihove platforme.

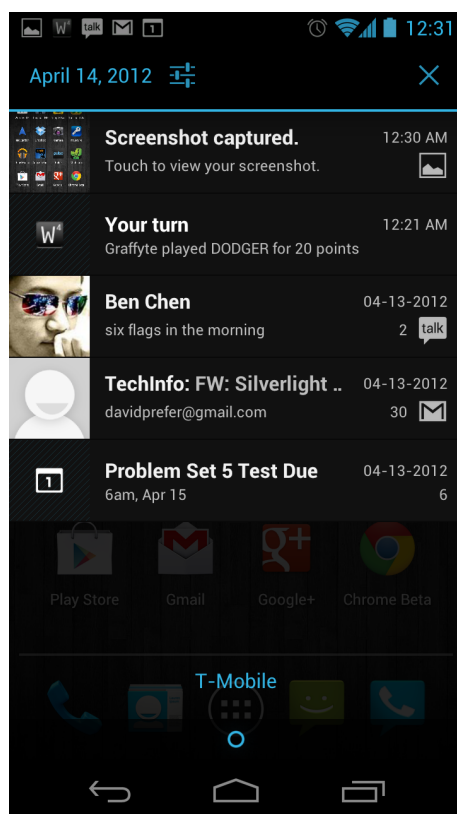
Ker pa smo želeli tudi preveriti, kako nov je koncept scenarija, smo preverili, ali obstaja podobna funkcionalnost mogoče pri razvoju stacionarnih aplikacij, pa smo bili prav tako neuspešni. Funkcionalnosti ni vključene celo v Xcode pri izdelavi aplikacij za MacOS, kjer bi jo najprej pričakovali.

4.3 Center obvestil

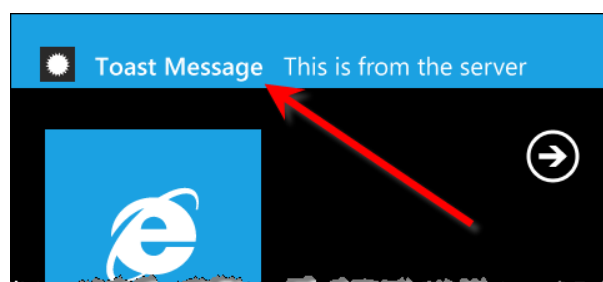
Če malo pobrskamo po spletu, je odprtih veliko tem, kako je Apple ukradel koncept centra obvestil (slika 3.4) Androidu (slika 4.1) in po vsem napisanem moram reči, da je to zelo verjetno [10, 11]. Ni pa ostalo samo pri tem, da ga je ukradel, ampak ga je na nekaterih področjih tudi izboljšal.

Pa poglejmo prednost, ki jo ima Android še vedno pred iOS. Največja prednost je, da ves čas lahko vidimo, katera aplikacija nam je poslala obvestilo s prikazano ikono v statusni vrstici. Na obvestila se odzovemo pri obeh v centru obvestil, do katerega dostopamo tako, da potegnemo od zgornjega dela ekrana navzdol. Prednost, ki jo ima iOS pa je prikaz novega obvestila, v katerem tudi vidimo predogled letega, in tudi, da lahko dostopamo do obvestil iz zaklenjenega zaslona. Pri iOS lahko tudi v nastavitvah nadziramo, katera aplikacija lahko pošilja obvestila in koliko jih lahko pošlje. Tudi sam UV je pri iOS-u neprimerno bolj dodelan.

Windows Phone na tem področju koraka daleč zadaj, saj nima sorodnega centra, kjer bi se zbirala nova obvestila. Ko se generira novo obvestilo se pokaže zapečeno obvestilo (angl. Toast Notification) (slika 4.2), ki pa po nekaj sekundah izgine in ni vidno nikjer več. Edini indikator, da imamo obvestilo je številka na ikoni aplikacije, ki je obvestilo generirala. Je pa napovedan center obvestil s prihodom Window Phone 8.



Slika 4.1: Center obvestil Android.



Slika 4.2: Primer zapečenega obvestila pri WP7.

Poglavje 5

Applova spletna trgovina

5.1 Kaj je Applova spletna trgovina?

Applova spletna trgovina (angl. App Store) je spletno mesto, kjer najdemo aplikacije za naprave iPhone, iPod, iPad in Mac. Nekatere aplikacije so na voljo brezplačno, nekatere so plačljive. Podrobnosti o modelih trženja v Applovi spletni trgovini si bomo pogledali v naslednji sekciji. Načini plačila za Slovenijo so kreditne kartice Mastercard, Visa in American Express.

5.2 Modeli trženja v Applovi spletni trgovini

5.2.1 Nakup aplikacije

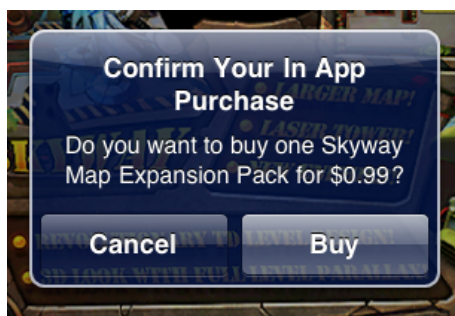
Najbolj osnoven model trženja je direktni nakup aplikacije (angl. App Purchase). Pri tem modelu v Applovi spletni trgovini poiščemo aplikacijo, ki je plačljiva in takoj plačamo zanjo.

5.2.2 Nakup znotraj aplikacije

Veliko aplikacij je brezplačnih (ali poceni) in nudijo dodatne vsebine ali funkcionalnosti v obliki nakupa, ki ga opravite znotraj aplikacije (angl. In-App Purchases). Primer je na sliki 5.1.

Obstajajo štiri različne vrste nakupov znotraj aplikacije[12]:

- Neporabljiv nakup: Nakup se opravi enkrat in se ga lahko prenese na več naprav z istim iTunes računom. Primer: nakup bonus ravni za igro ali dodaten vodič po mestu.



Slika 5.1: Obvestilo o nameri nakupa znotraj aplikacije.

- Porablјiv nakup: Nakup, ki ga ne moremo prenesti med napravami in mora biti opravljen vsakič, ko želimo neko stvar znotraj aplikacije. Primer: dodatno zdravje, denar pri igranh.
- Naročnine: So enkratne storitve, ki jih moramo znova plačati, ko enkrat potečejo in jih tudi ne moremo prenašati med napravami. Primer: enomesečna naročnina na časopis.
- Samodejno obnavlјujoče naročnine: So storitve, ki nam omogočajo, da se nakup izvede na določen časovni razmak. So prenoslјive med napravami. Primer: Tedenska naročnina na časopis, mesečna naročnina na revijo.

5.2.3 Reklamni oglasi

Obstaja pa tudi model, kjer aplikacijo nudimo brezplačo, a v zameno za to uporabniku prikazujemo reklamne oglase (angl. iAd). Ko uporabnik klikne na ta oglas, se ga preusmeri iz trenutne aplikacije v Applovo spletno trgovino ali brskalnik Safari, odvisno od oglasa. Ponavadi imajo aplikacije s tem modelom prodaje tudi možnost nakupa znotraj aplikacije, ki oglase odstrani.

5.2.4 Kaj zahteva Apple od nas ob objavi?

Da do objave sploh lahko pridemo, je predpogoj imeti eno izmed kreditnih kartic Mastercard, Visa ali American Express, saj ni mogoč drug način plačila 99\$, da se registriramo kot iOS razvijalec. Registriramo se na Applovi spletni strani [13] s klikom na gumb Enroll Now, kjer nas vodič vodi skozi postopek registriranja. Ker pa prihajamo iz Slovenije, je potrebno omeniti, da Apple zahteva pošiljanje nekaterih podatkov preko faksa. Ko pošljemo faks, naj bi nanj odgovorili v 5-tih delovnih dneh, kar je v mojem primeru tudi držalo. V

odgovor pošljejo mail, ki potrjuje nakup in vas napoti v iTunes Connect, kjer urejate vse podatke v zvezi z računom. Obvezno moramo izpolniti nekatere podatke pod zavihkom "Contracts, Tax, and Banking", odvisno od tega, kako želimo prodajati aplikacijo. V kolikor jo želimo dajati zastonj, nam ni potrebno storiti ničesar več. Mi pa bomo z našo aplikacijo poskusili tudi nekaj zaslužiti, tako da bomo izpolnili tudi potrebne obrazce za "iOS Paid Applications".

Na tem mestu pa je potrebno razmisliti, kateri model trženja bi bil najbolj ustrezen. Lahko bi se odločili, da bi bila aplikacija plačljiva, ampak bi s tem odvrnili kupce, ki niso prepričani, da jo želijo kupiti ne da bi jo preizkusili. Tako bi bilo potrebno še dodati okrnjeno (angl. Lite) verzijo naše aplikacije, ki nekaterih funkcij ne bi imela.

Druga možnost pa bi bila uporaba nakupov znotraj aplikacije. To bi pomenilo, da ima uporabnik pravzaprav celotno aplikacijo, vendar ima zaklenjene določene funkcije, ko pa opravi nakup, se te funkcije odklenejo. Ker je v začetku aplikacija brezplačna, potencialni kupec ne bo dolgo razmišljal o namestitvi, zato se bomo tudi pri izdelavi naše aplikacije odločili za to možnost. Nastavitve potrebne za implementacijo si bomo pogledali v podpoglavju 6.2.14.

Poglavje 6

Osnove Objective C v Xcode okolju razložene na primeru razvoja aplikacije

Za prikaz novosti v iOS 5 bomo razvili aplikacijo SliceOfTime. Aplikacija bo omogočala:

- zajemanje slik,
- prikaz zajetih slik,
- generiranje filma iz zajetih slik, kjer bo uporabnik imel možnost izbire FPS in spremljevalne glasbe,
- nastavitev časa in pogostosti proženja obvestila, ki uporabnika opolni, da je potrebno zajeti sliko,
- sinhronizacija podatkovne baze v iCloud,
- objavljanje generiranega filma v socialnih omrežjih Facebook in Youtube ter izvoz v galerijo slik naprave.

V tem diplomskem delu bomo predpostavili, da je bralec že seznanjen z osnovami programskega jezika C.

6.1 Uvod v Objective C

Podrobnejšo obrazložitev lahko najdemo v [14, 15].

6.1.1 Klicanje metod

Osnovna sintaksa za klic metode na nekem objektu je:

```
[object method];  
[object methodWithInput:input];
```

Metode lahko vračajo tudi vrednost:

```
output = [object methodWithOutput];  
output = [object methodWithInputAndOutput:input];
```

Pokličemo lahko tudi metode posameznih razredov. Na tak način tudi ustvarjamo objekte. V spodnjem primeru kličemo metodo razreda `NSString`, ki nam vrne nov `NSString` objekt:

```
id myObject = [NSString string];
```

Id tip pomeni, da se spremenljivka `myObject` lahko nanaša na katerikoli tip objekta, tako da dejansko razred in metode, ki jih objekt implementira, niso znane dokler ne prevedemo aplikacije.

V prejšnjem primeru je očitno, da je tip objekta `NSString`, zato lahko tip spremenimo:

```
NSString* myString = [NSString string];
```

`MyString` je sedaj `NSString` spremenljivka, tako da nas bo prevajalnik opozoril, če bomo na njej poskušali uporabiti metodo, ki je `NSString` ne podpira.

Če pozorno pogledamo zapis, opazimo zvezdico desno od tipa objekta. Vse Objective C objektne spremenljivke uporabljajo kazalce. Id tip pa je že predefiniran kot kazalec, zato tukaj ni potrebe po zvezdici.

6.1.2 Gnezdenje

Pri mnogih jezikih bi gnezdenje metod ali funkcij izgledalo takole:

```
function1(function2());
```

To pomeni, da je rezultat `function2` posredovan kot vhod `function1`. V Objective C pa gnezdene metode zgledajo tako:

```
[NSString stringWithFormat:[prefs format]];
```

Pri Objective C se izogibamo gnezdenju več kot dveh klicev v eni sami vrstici, saj z lahkoto postane neberljiva.

6.1.3 Večvhodne metode

Nekatere metode zahtevajo več vhodnih vrednosti. Pri Objective C je ime metode lahko razdeljeno na več segmentov. Večvhodna metoda z dvema segmentoma izgleda takole:

```
-(BOOL)writeToFile:(NSString *)path atomically:(BOOL)
    useAuxiliaryFile;
```

Klic te metode pa bi izgledal takole:

```
NSString *myData=@"Zapis";
BOOL result = [myData writeToFile:@"/tmp/log.txt" atomically:NO
    ];
```

6.1.4 Metode pridobivanja in nastavljanja podatkov

Vse spremenljivke pri Objective C so privatne, tako da moramo uporabiti metode pridobivanja in nastavljanja podatkov (angl. Accessors), da pridobimo ali nastavimo vrednost v večini primerov. Kako te metode umestimo v razred si bomo pogledali v podpoglavju 6.1.6. Obstajata dve sintaksi. Prva izgleda takole:

```
[photo setCaption:@"Day at the Beach"];
output = [photo caption];
```

Koda druge vrstice ne bere direktno vrednosti spremenljivke, ampak pravzaprav kliče metodo z imenom `caption`. V večini primerov ne dodajamo "get" predpone pri metodah pridobivanja podatkov (angl. getters) v Objective C.

Kadar opazimo kodo znotraj oglatih oklepajev, pošljamo sporočilo objektu.

Pična sintaksa (angl. dot syntax) za metode pridobivanja in nastavljanja podatkov je novost pri Objective C 2.0, ki je del Mac OS X 10.5, in izgleda takole:

```
photo.caption = @"Day at the Beach";
output = photo.caption;
```

V projektu lahko uporabimo katerikoli stil, ampak je lepa praksa, da smo konsistentni in ne uporabljamo obeh v istem projektu. Sintakso s piko se uporablja samo pri metodah pridobivanja in nastavljanja podatkov in ne pri splošno namenskih metodah.

6.1.5 Kreiranje objektov

Imamo dva glavna načina kreiranja objektov. Prvega smo spoznali že prej:

```
NSString* myString = [NSString string];
```

Ta je bolj priročen avtomatski način. V tem primeru se ustvarja objekt, ki sprosti spomin avtomatsko. V mnogih primerih pa je potrebno uporabiti ročni način:

```
NSString* myString = [[NSString alloc] init];
```

Ta je gnezden klic metode. Najprej se kliče `alloc` metoda razreda `NSString`. To je klic na nižjem nivoju, ki rezervira spomin in instancira objekt. Kasneje moramo ta spomin spostiti ročno:

```
[myString release];
```

Ob tem pa je vredno opozoriti, da ima iOS 5 nov način urejanja spomina (ARC), pri katerem ni potrebno več ročno sproščati objektov, kakor je bilo to potrebno prej.

Drugi del je klic `init`, ki inicializira nov objekt. `Init` implementacija ponavadi naredi osnovno inicializacijo, kot je ustvarjanje instančne spremenljivke. Podrobnosti tega so nam nepoznane kot uporabniku tega razreda. V nekaterih primerih pa lako uporabimo druge možnosti `init` metode, ki imajo tudi vhodne spremenljivke:

```
NSNumber* value = [[NSNumber alloc] initWithFloat:1.0];
```

6.1.6 Oblikovanje vmesnika razreda

Sintaksa Objective C za ustvarjanje razreda je zelo preprosta. Tipično jo najdemo v dveh delih:

- Vmesnik razreda je shranjen v `ClassName.h` datoteki in definira instance spremenljivk in javnih metod.
- Implementacija je v `ClassName.m` datoteki in vsebuje dejansko kodo za metode najavljene v `ClassName.h` datoteki. Pogosto vsebuje tudi privatne metode, ki niso na voljo uporabnikom razreda.

Tukaj je primer vmesnika razreda. Ime razreda je `Photo`, zato je datoteka imenovana `Photo.h`:

```
#import <Cocoa/Cocoa.h>

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
@end
```

Najprej uvozimo `Cocoa.h`, da vključimo vse splošne razrede za Cocoa aplikacijo. `#import` direktiva vključuje avtomatski mehanizem, ki nam preprečuje uvažanje iste datoteke večkrat.

`@interface` je deklaracija razreda `Photo`. Dvopičje specificira nadrazred, v našem primeru `NSObject`.

Znotraj zavrtih oklepajev sta dve instančni spremenljivki: `caption` in `photographer`. Obe sta tipa `NSString`.

Na koncu `@end` označi konec deklaracije razreda.

Pa pogledjmo, kako bi na istem primeru dodali še metode. Dodali bomo metode za pridobivanje in nastavljanje podatkov:

```
#import <Cocoa/Cocoa.h>

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
//metodi za pridobivanje podatkov
- (NSString*) caption;
- (NSString*) photographer;
instance methods
//metodi za nastavljanje podatkov
- (void) setCaption: (NSString*)input;
- (void) setPhotographer: (NSString*)input;

@end
```

Kot smo že omenili, ponavadi pri teh metodah izpuščamo predpono "get". Pomišljaj (-) pred imenom metode pomeni, da je to instanca metode (angl.

instance method). Če je pred imenom znak plus (+) pa pomeni, da je to metoda razreda (angl. class method). Pa si pogledjmo kako deklariramo metodo razreda in instanco metode:

```
@interface MyClass : NSObject
```

```
+ (void)aClassMethod;
- (void)anInstanceMethod;
```

```
@end
```

Nato ju lahko uporabljamo takole:

```
[MyClass aClassMethod];
```

```
MyClass *object = [[MyClass alloc] init];
[object anInstanceMethod];
```

V oklepajih navedemo željen tip vrednosti, ki jo metoda vrne. Metode za nastavljanje podatkov ne vračajo nobene vrednosti, zato jih deklariramo kot void.

6.1.7 Implementacija razreda

Implementacija razreda z metodami za pridobivanje podatkov:

```
#import "Photo.h"
```

```
@implementation Photo
```

```
//metodi za pridobivanje podatkov
```

```
- (NSString*) caption {
    return caption;
}
```

```
- (NSString*) photographer {
    return photographer;
}
```

```
//metodi za nastavljanje podatkov
```

```
- (void) setCaption: (NSString*)input
{
```

```
        caption = input;
    }

    - (void) setPhotographer: (NSString*)input
    {
        photographer = input;
    }

@end
```

Ta del kode se začne z `@implementation` in imenom razreda ter se konča z `@end`. Vse implementacije metod so med tema dvema vrsticama.

6.1.8 Metoda `init`

Vedno lahko tudi spišemo `init` metodo, v kateri nastavimo začetne vrednosti spremenljivk:

```
- (id) init
{
    if ( self = [super init] )
    {
        [self setCaption:@"Default Caption"];
        [self setPhotographer:@"Default Photographer"];
    }
    return self;
}
```

Pri tem delu kode nas lahko malo zmede tretja vrstica, je pa pravzaprav klic nadrazredu, da naredi lastno inicializacijo. `If` stavek samo preveri ali je bila inicializacija uspešna preden poskuša nastaviti vrednosti spremenljivk.

6.1.9 Prikazovanje sporočil v konzoli

Prikazovanje sporočil v konzoli je preprosto. Pravzaprav je `NSLog()` funkcija skoraj identična C-jevski funkciji `printf()`. Pa si pogledjmo, kako bi izpisali različne vrednosti:

```
NSLog(@"The current date and time is: %@", [NSDate date]); //
    izpis datuma
NSLog(@"Log int number: %d", intNumber); //izpis integer stevila
```

```
NSLog(@"Log double number: %f", doubleNumber); //izpis double
    stevila
NSLog(@"Log double number: %.2lf", doubleNumber); // izpis
    double stevila z natancnostjo dveh decimalnih stevil
```

Na tak način lahko izpišemo posamezne vrednosti objekta v konzoli. Več o možnostih izpisa se nahaja na [16].

6.1.10 Poenostavljanje metod pridobivanja in nastavljanja podatkov

Ko smo napisali metodi za nastavljanje in pridobivanje podatkov, smo opazili, da so precej enostavne in da bi jih lahko posplošili. Posplošimo jih s pomočjo Objective C funkcionalnosti imenovane lastnosti (angl. properties), ki nam generira metodi za nastavljanje in pridobivanje podatkov. `Photo.h` datoteka z uporabo programskega stavka lastnosti bi izgledala takole:

```
#import <Cocoa/Cocoa.h>

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
@property (retain) NSString* caption;
@property (retain) NSString* photographer;

@end
```

`@property` je direktiva Objective C, ki deklarira metodi za pridobivanje in nastavljanje podatkov. Beseda `retain` v oklepajih pomeni, da naj metode za nastavljanje podatkov zadržijo vhodno vrednost, nadaljevanje vrstice pa specificira tip in ime.

Implementacijski razred bi po poenostavljanju izgledal tako:

```
#import "Photo.h"

@implementation Photo

@synthesize caption;
@synthesize photographer;

@end
```

Ob tem je potrebno omeniti, da lahko še vedno deklariramo metode za pridobivanje in nastavljanje podatkov, v katerih lahko naredimo spremembe. Prevajalnik bo zaznal, katera metoda je implementirana in katera ni ter sam dodal manjkajoče.

6.1.11 Kategorije

Kategorije (angl. categories) so ena bolj uporabnih funkcionalnosti Objective C. Kategorije nam omogočajo, da dodajamo metode že obstoječemu razredu ne da bi ustvarjali podrazrede ali da bi vedeli podrobnosti njegove implementacije.

Ta uporabnost je posebej izrazita, ko želimo dodati metode že vgrajenim objektom. Če želimo dodati metodo vsem instancam tipa `NSString` v naši aplikaciji, samo dodamo kategorijo. Na primer, če želimo dodati metodo tipa `NSString`, da bi določili ali je vsebina URL naslov, bi to izgledalo takole:

```
#import <Cocoa/Cocoa.h>
```

```
@interface NSString (Utilities)
- (BOOL) isURL;
@end
```

Opazimo, da je deklaracija podobna deklaraciji razreda. Razlika je, da ne navedemo nadrazreda in da je ime kategorije v oklepajih, v tem primeru `Utilities`.

Implementacija razreda pa bi izgledala takole (sama implementacija je samo primer, toliko da pokažemo sam koncept):

```
#import "NSString-Utilities.h"

@implementation NSString (Utilities)

- (BOOL) isURL
{
    if ( [self hasPrefix:@"http://"] )
        return YES;
    else
        return NO;
}

@end
```

Sedaj lahko to metodo uporabljamo na kateremkoli objektu tipa `NSString`. Primer uporabe bi bil:

```
NSString* string1 = @"http://pixar.com/";
NSString* string2 = @"Pixar";

if ( [string1 isURL] )
    NSLog (@"string1 is a URL");

if ( [string2 isURL] )
    NSLog (@"string2 is a URL");
```

Ob tem je treba omeniti še, da kategorije ne morejo imeti instančnih spremenljivk. Velja pa, da jih lahko uporabljamo za spremembo že obstoječih metod razredov. Ob tem pa velja velika previdnost, saj sprememba metod vpliva na instance v celotni aplikaciji.

6.2 Razvoj aplikacije

6.2.1 Začetne nastavitve projekta

Ker bo naš projekt imel podporo za hranjenje podatkov v iCloud, moramo narediti nekaj predhodnih korakov, preden začnemo z izdelavo projekta v Xcode.

Prvi korak je ustvarjanje identifikacijske številke aplikacije (angl. App ID), ki ima omogočeno tudi podpora za iCloud. Da bi storili to, moramo obiskati center razvijalcev iOS (angl. iOS Developer Center) [17] in se prijaviti v iOS-ov portal (angl. iOS Provisioning Portal), kjer urejamo certifikate, naprave in identifikacijske številke aplikacije (prehodno se moramo registrirati kot uradni Apple razvijalec in plačati registracijo v vrednosti 99\$, kar smo si podrobneje ogledali v poglavju 5). Na tem mestu ustvarimo novo identifikacijsko število aplikacije, podobno kot je na sliki 6.1.

Ko smo ustvarili identifikacijsko številko aplikacije, moramo omogočiti še podporo iCloud-u. To storimo tako, da med identifikacijskimi številkami aplikacij poiščemo pravkar ustvarjeno in kliknemo gumb Configure (slika 6.2).

Med naštetimi nastavitvami za aplikacijo imamo tudi opcijo Enable for iCloud. To opcijo omogočimo (glej sliko 6.3) in potrdimo nastavitve.

Naslednji korak je ustvarjanje profila (angl. Provisioning Profile) za identifikacijsko številko aplikacije, ki smo jo pravkar ustvarili. To prav tako storimo v iOS-evem portalu pod zavihkom Provisioning. Kliknemo gumb New Profile. Nato se nam pojavi okno, ki ga izpolnimo z željenimi podatki podobno, kot je na sliki 6.4. Ko potrdimo nastavitve se nam med Development Provisioning Profiles pojavi tudi profil za našo aplikacijo, ki pa ima status Pending. Nekaj

The screenshot shows the 'Provisioning Portal' interface. On the left is a sidebar with navigation links: Home, Certificates, Devices, App IDs (selected), Pass Type IDs, Provisioning, and Distribution. The main area is titled 'Create App ID' and has two tabs: 'Manage' and 'How To'. The form contains three sections:

- Description:** A text input field with the value 'Diplomsko delo Slice Of Time'. A note below states: 'You cannot use special characters as @, &, *, " in your description.'
- Bundle Seed ID (App ID Prefix):** A text input field with the value '34NVC5BKJ9'. A note below states: 'Your Team ID (34NVC5BKJ9) will be used as the App ID Prefix.'
- Bundle Identifier (App ID Suffix):** A text input field with the value 'com.cvenko.SliceOfTime'. An example below shows 'com.domainname.appname'.

 At the bottom right are 'Cancel' and 'Submit' buttons.

Slika 6.1: Ustvarjanje nove identifikacijske številke aplikacije.

The screenshot shows the configuration page for the App ID '34NVC5BKJ9.com.cvenko.Sli...'. The title is 'Diplomsko delo Slice Of Time'. Below the title is a list of features and their status:

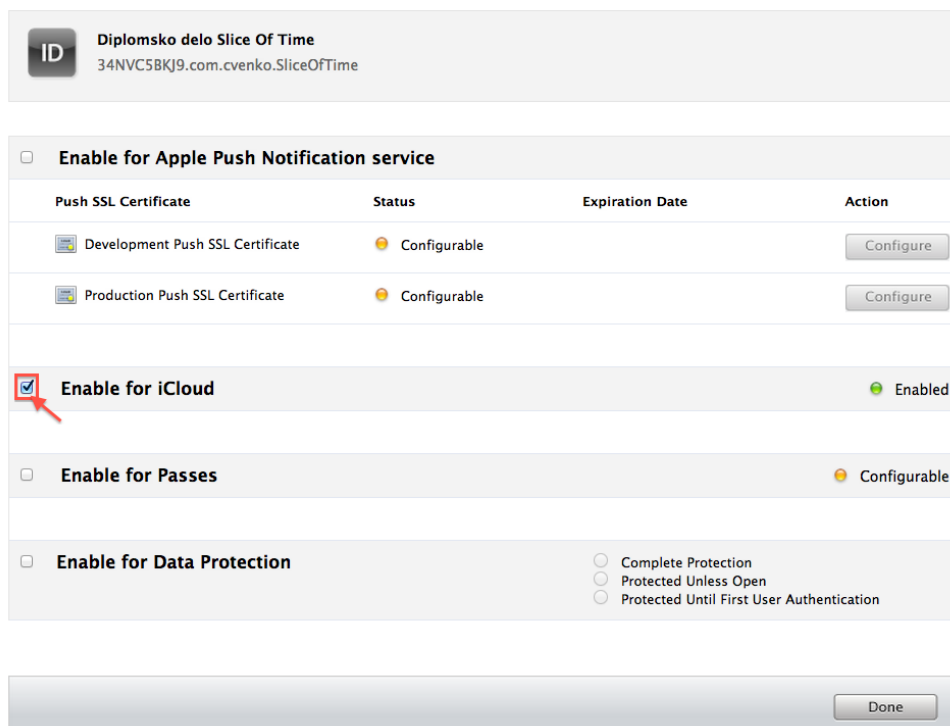
- Passes: Configurable
- Data Protection: Configurable
- iCloud: Configurable
- In-App Purchase: Enabled
- Game Center: Enabled
- Push Notification: Configurable

 To the right of each feature is a corresponding icon (orange circle for Configurable, green circle for Enabled). A red box highlights the 'Configure' button next to the 'Passes' status.

Slika 6.2: Konfiguracija nove identifikacijske številke aplikacije.

časa počakamo, osvežimo stran in ko je status profila Active, ga prenesemo na naš računalnik. Dvokliknemo preneseni profil, ki bi se nam moral avtomatično prenesti v Xcode-ov Organizer. Ko uspešno izvedemo vse korake, bi moral Xcode-ov Organizer izgledati tako, kot je prikazano na sliki 6.5.

Naslednji korak pa je ustvarjanje novega projekta v Xcode-u. To storimo tako, da izberemo File/New/Project. Ker bo naš projekt uporabljal Core Data (več o tem v poglavju 6.2.3), moramo izbrati med predlogami Empty Application. Kliknemo gumb Next, ki nam odpre naslednje okno z možnostmi nastavitve projekta (slika 6.6). Nastavimo ime projekta (v našem primeru SliceOfTime), Company Identifier mora biti enak tistemu, ki smo ga uporabili pri ustvarjanju identifikacijske številke aplikacije, Class Prefix je lahko polju-ben ali prazen, označuje pa, kako se bodo začnjala imena razredov v našem projektu. Ker bomo razvijali aplikacijo za iPhone, izberemo Device Family



Slika 6.3: Omogočanje iCloud podpore.

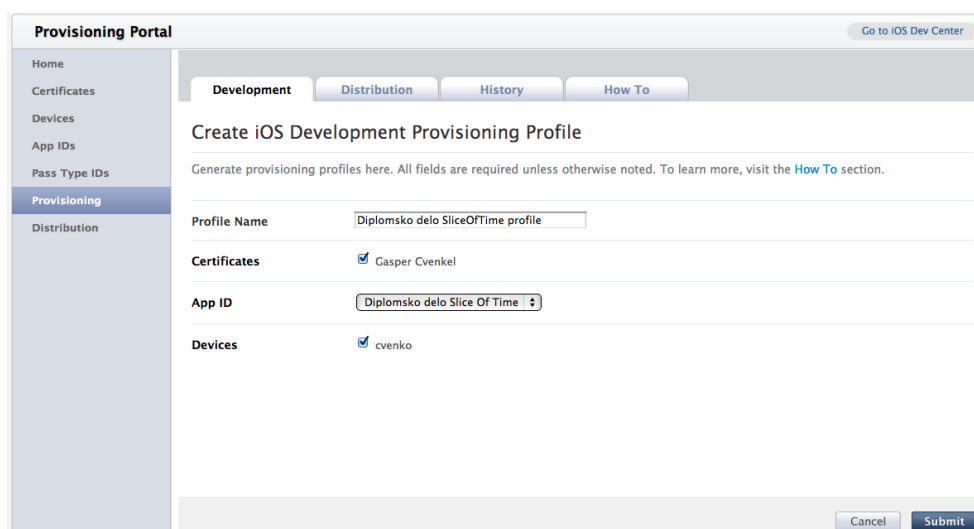
iPhone ter obkljukamo opciji Use Core Data in Use Automatic Reference Counting (ARC). Potrdimo in izberemo, kam želimo shraniti naš projekt.

V samem projektu pa je potreben še dodaten korak, da omogočimo iCloud funkcije. To storimo tako, da v projektnem navigatorju (angl. Project Navigator) izberemo naš projekt, in nato Targets SliceOfTime. Pod zavihkom Summary se pomaknemo čisto na dno, kjer je opcija Enable Entitlements, ki jo omogočimo. Ko jo omogočimo, nam Xcode avtomatično izpolni ostala polja (glej sliko 6.7).

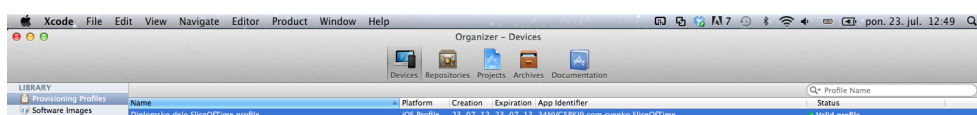
Tako smo končali začetne nastavitve projekta, sedaj pa začnimo z izdelavo UV.

6.2.2 Scenarij

Ker smo na začetku izbrali opcijo Empty Application, moramo v projekt dodati še scenarij. To najenostavneje naredimo tako, da v projektnem navigatorju z desno tipko kliknemo na naš projekt in izberemo New File. Pojavi se nam okno, v katerem na levi strani izberemo zavihek User Interface in nato izberemo



Slika 6.4: Obrazec za izdelavo profila.

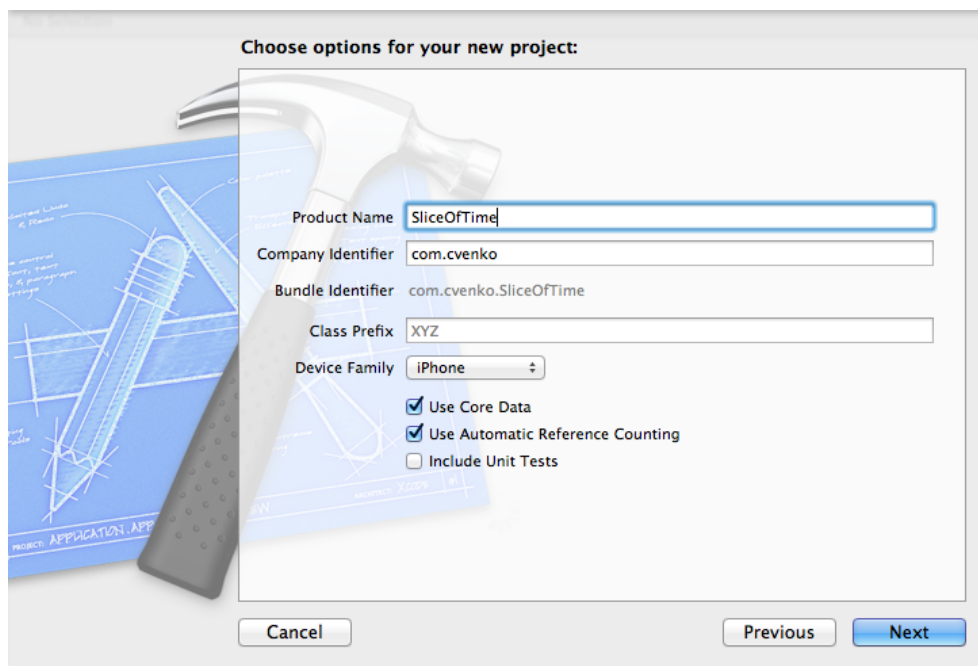


Slika 6.5: Organizer s prikazanim veljavnim profilom.

predlogo Storyboard. Potrdimo, izberemo Device Family iPhone, potrdimo, še enkrat potrdimo mesto shranjevanja in scenarij je dodan. Pred nami se odpre prazno platno scenarija. Da pa bomo ustvarjeni scenarij tudi dejansko uporabljali, ga moramo še izbrati kot glavni scenarij (angl. Main Storyboard). To storimo tako, da izberemo naš projekt, nato pod Targets SliceOfTime in v zavihu Summary v okencu Main Storyboard izberemo pravkar ustvarjenega (slika 6.8).

Sedaj pa bomo tudi prvič posegli v kodo. Ker smo scenarij dodali naknadno, moramo še v AppDelegate.m spremeniti metodo `application:didFinishLaunchingWithOptions` in sicer tako, da pobrišemo vse razen `return YES`. Ker na tem mestu lahko tudi preverimo ali smo pravilno nastavili delovanje iCloud-a bomo tej metodi dodali tudi kodo za preverjanje pravilne nastavitve iCloud. Celotna metoda izgleda takole:

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
    NSURL *ubiq = [[NSFileManager defaultManager]
```



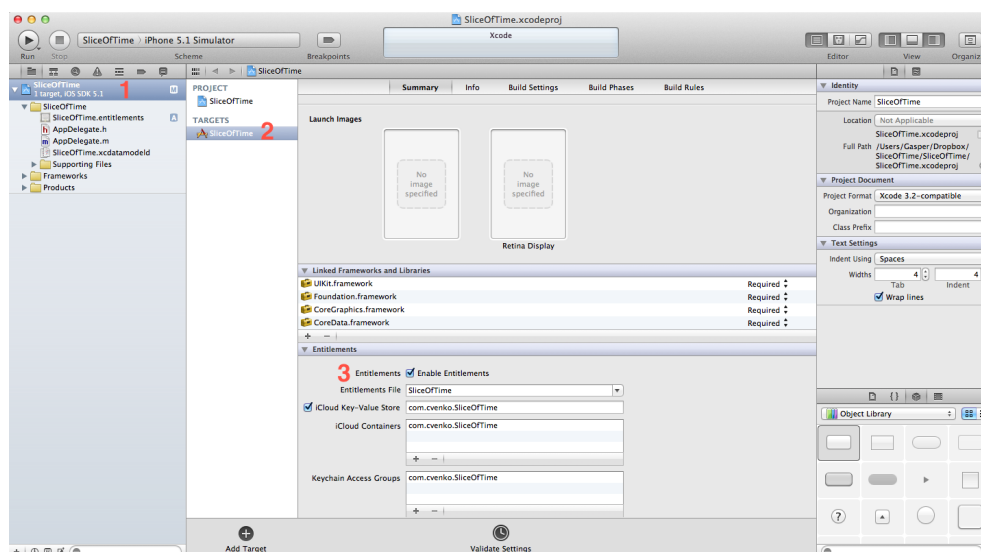
Slika 6.6: Okno z nastavitvami projekta.

```
URLForUbiquityContainerIdentifier:nil];  
if (ubiq) {  
    NSLog(@"iCloud access at %@", ubiq);  
} else {  
    NSLog(@"No iCloud access");  
}  
return YES;  
}
```

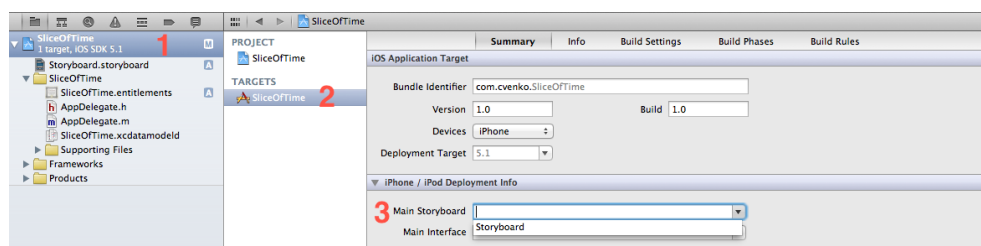
Pa prvič zaženimo aplikacijo. Aplikacijo moramo zagnati na napravi (glej sliko 6.9), saj iPhone Simulator nima podpore za iCloud.

Če vse deluje kot bi morale, bi morali dobiti podoben izpis v konzoli:

```
2012-07-23 17:04:47.276 SliceOfTime[6216:707] Failed to  
    instantiate the default view controller for  
    UIStoryboardFile 'Storyboard' - perhaps the designated  
    entry point is not set?  
2012-07-23 17:04:47.676 SliceOfTime[6216:707] iCloud access at  
    file:///localhost/private/var/mobile/Library/Mobile%20  
    Documents/34NVC8BKJ9~com~cvenko~SliceOfTime/
```



Slika 6.7: Nastavitve projekta Xcode za podporo iCloud-u.

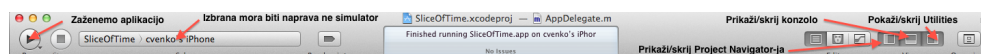


Slika 6.8: Izbira glavnega scenarija.

Prvi del naj vas ne skrbi, saj pomeni samo to, da je naš scenarij še prazen, pomemben je samo drugi del, ki nam pove, da smo uspešno nastavili dostop do iCloud-a.

Pa se lotimo izdelave scenarija. Nekaj osnovne teorije o scenariju smo že podali v poglavju 3.3, sedaj pa pogledjmo, kako z njim delamo praktično.

Scenarij nam vzame kar nekaj zaslonskega prostora, zato nam pride prav, da skrijemo stvari, ki jih ne bomo uporabljali z uporabo ustreznih gumbov (glej sliko 6.9). Ko imamo tako pripravljen prostor, s pomočjo povleci in spusti

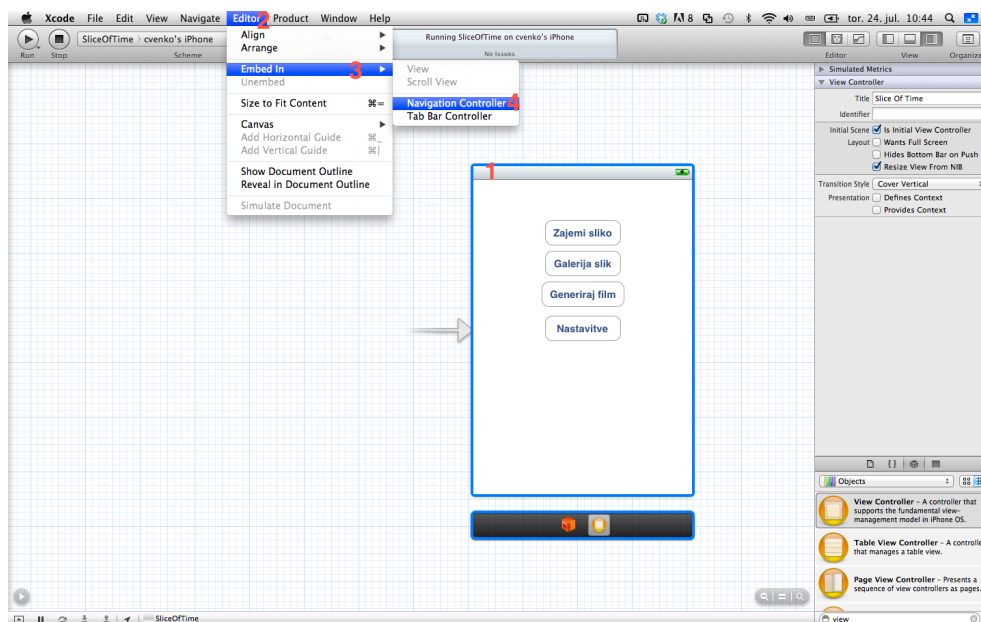


Slika 6.9: Opravilna vrstica.

metode dodajamo gradnike na risalno ploščo. Najprej bomo dodali nadzornika pogleda. Poiščemo ga v spodnjem desnem okencu imenovanem Object Library, povlečemo na risalno ploščo in spustimo. Za preizkus ali vse deluje tako kot mora, lahko povlečemo nanj tudi gumb ali labelo ter zaženemo aplikacijo, ki bi nam morala prikazati gumb oziroma labelo na napravi. Če imamo prikazan gumb in imamo v konzoli ustrezen izpis, smo uspešno nastavili vse začetne nastavitve projekta, ki jih potrebujemo.

Kakšen bo UV pa je potrebno dobro premisliti, saj nam bo to privarčevalo veliko časa in nepotrebnega kodiranja. Tukaj nam scenarij pomaga v veliki meri, saj vidimo, kako bo UV izgledal in deloval na enem platnu.

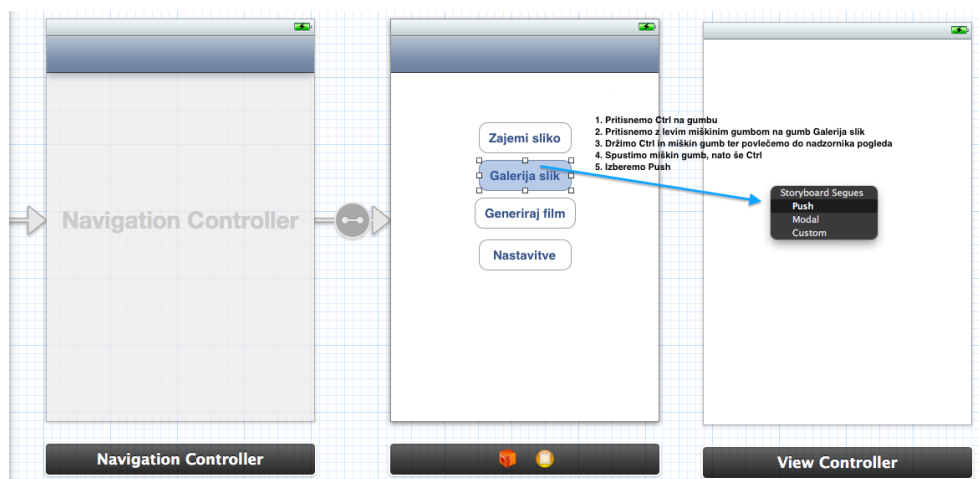
Sedaj bomo dodali gumb z ustreznimi naslovi ter celoten nadzornik pogleda dodali v nadzornika navigacije (angl. Navigation Controller), kot prikazuje slika 6.10, da bomo lažje navigirali med nadzorniki pogleda. Nadzornika navigacije si najlažje predstavljamo kot sklad nadzornikov pogleda, kamor se ti v vrstnem redu nalagajo po vrsti prikazovanja. Tako vedno vemo kateri je prejšnji in naslednji nadzornik pogleda.



Slika 6.10: Dodajanje nadzornika navigacije.

Kar nam še ostane, da bomo poznali osnove delovanja scenarija, je povezovanje med nadzorniki pogleda. Najlažje bo, če si pogledamo dodajanje povezave na primeru slike 6.11. Kot lahko vidimo, najprej pritisnemo tipko Ctrl in nato kliknemo na element, katerega klik bo sprožil prehod. Držimo Ctrl

in levo tipko miške ter vlečemo do nadzornika pogleda, ki ga želimo prikazati. Spustimo levo tipko miške in Ctrl, pojavi se nam okno, kjer izberemo opcijo Push.



Slika 6.11: Povezovanje nadzornikov pogleda.

Po istem postopku dodamo še nadzornike pogleda za generiranje filma, ki ima še povezavo naprej na nadzornika pogleda, kjer bomo imeli možnost ogleda generiranega filma in objavljanje na socialnih omrežjih ter nadzornika pogleda za nastavitve. S tem je osnova za naš UV zaključena (končni scenarij je prikazan na sliki 3.3).

6.2.3 Core Data

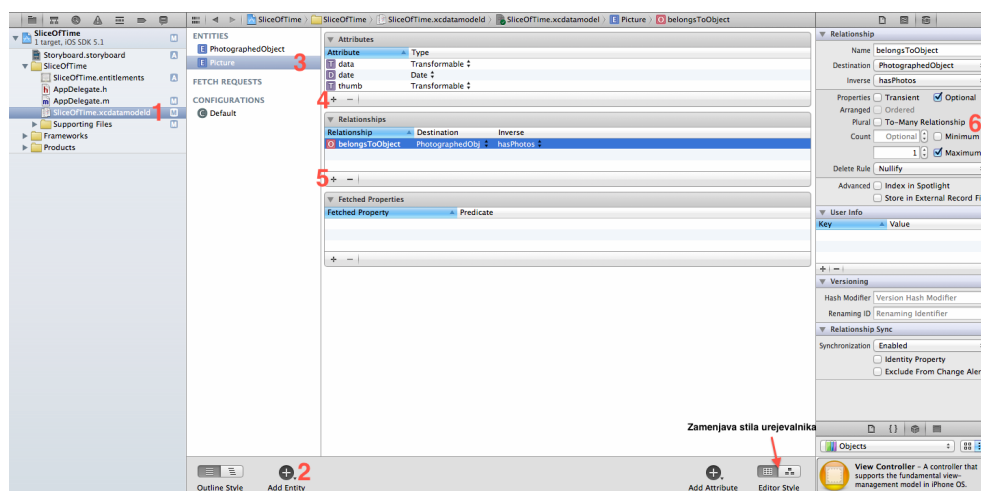
Core Data so pri Applu poimenovali tehnologijo shranjevanja podatkov. Je objektno-orientirana baza podatkov, ki je podprta z liteSQL [18, 19]. Ima lastnosti kot so:

- spremljanje sprememb in možnost razveljavljanja,
- povezave med objekti,
- leno nalaganje podatkov (angl. faulting),
- razvrščanje podatkov s filtri,
- napredno povpraševanje.

Core Data smo omogočili že na začetku, ko smo ustvarjali projekt. Tako nam je Xcode prihranil kar nekaj dela, saj je avtomatično generiral kodo za upravljanje s podatki. Koda se nahaja v AppDelegate-u. Preostane nam še, da ustvarimo podatkovni model.

V projektnem navigatorju Xcode izberemo datoteko SliceOfTime.xcdata-modeld (datoteka je bila generirana avtomatično, ko smo omogočili Core Data). Naš podatkovni model bo preprost: dve entiteti, ki bosta med seboj povezani. Dodamo prvo entiteto s klikom na gumb Add Entity in jo poimenujemo `PhotographedObject`. Ta entiteta bo imela le en atribut `ObjectName` tipa `String` in eno povezavo na drugo entiteto, ki jo poimenujemo `Picture`. S klikom na `+` v sekciji Attributes dodamo entiteti `Picture` attribute `data` in `thumb` tipa `Transformable`, ki jih bomo uporabljali za hranjenje slik in pripadajočih ikon ter atribut `date` tipa `Date`. Ostane nam še, da dodamo povezavo med entitetama. V sekciji Relationship kliknemo znak `+` (izbrano imamo entiteto `Picture`), povezavo poimenujemo `belongsToObject` in kot Destination navedemo `photographedObject`. Nato izberemo še `photographedObject`, kliknemo znak `+`, povezavo poimenujemo `hasPhotos`, kot Destination pa navedemo `Picture`. Tukaj moramo tudi označiti možnost povezave To-Many Relationship (pomembno je, da to storimo samo pri tej povezavi). Lahko omenimo, da imamo tudi grafično možnost urejanja Core Data s klikom na ikono Editor Style. Rezultat in razlaga ikon je vidna na sliki 6.12. Pomembno si je tudi zapomniti, da kadarkoli bomo spreminjali podatkovni model naknadno, moramo aplikacijo zbrisati iz naprave in ponovno naložiti, saj se bo drugače aplikacija sesula zaradi neujemanja podatkovnega modela. (Sam podatkovni model smo spremenili pri optimizaciji aplikacije, da pa bo bralcu prihranjeno dodatno delo, lahko že v začetku zgradi tak podatkovni model, kot je opisano v poglavju 7.1).

Da bi enostavneje uporabljali entitete in njihove attribute v kodi, bomo sedaj generirali podrazrede za entitete. To storimo tako, da označimo obe entiteti, v naslovni vrstici izberemo Editor in nato Create NSManagedObject Subclass. Xcode nam tako zgenerira potrebne datoteke, vendar je tukaj potrebno opozoriti na hrošča, ki se pojavi zaradi povezav med entitetami. Ne bomo se spuščali v podrobnosti, povedali bomo le, da ga odpravimo tako, da ponovno generiramo podrazrede in kliknemo v redu, ko nam Xcode ponudi možnost zamenjave s starimi. Tako smo si pripravili Core Data za uporabo.



Slika 6.12: Predstavitev podatkov Core Data.

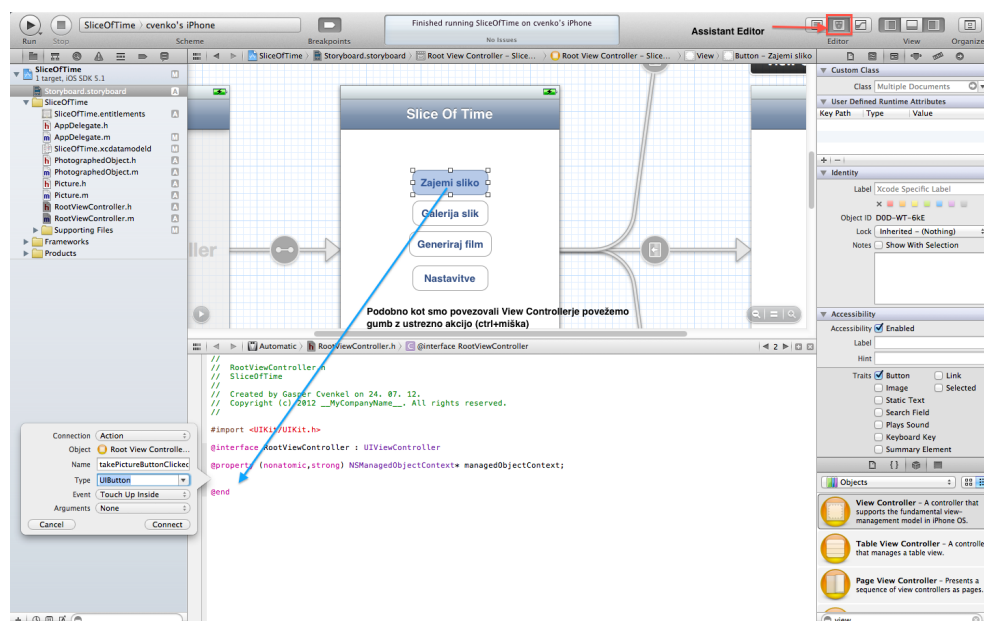
6.2.4 Povezovanje nadzornikov pogleda s pripadajočimi datotekami

Na scenariju smo si pripravili nadzornike pogleda, sedaj pa jih moramo povezati z ustreznimi datotekami, ki jih moramo še ustvariti. To storimo podobno kot smo ustvarili scenarij (glej podpoglavje 6.2.2), le da izberemo razdelek Cocoa Touch in nato Objective C Class. Poimenovali ga bomo `RootViewController` in bo tipa `UIViewController`. Ostale možnosti pustimo prazne. Xcode nam tako zgenerira `RootViewController.h` in `.m` datoteki, ki sta vidni v navigatorju projekta. Preostane nam še, da se premaknemo na scenarij in prvi nadzornik pogleda povežemo s temi datotekami. Kliknemo na nadzornik pogleda in v zavihku Identity Inspector (zavihek je desno zgoraj tretji po vrsti) v razdelku Custom Class izberemo `RootViewController`. Podobno dodamo še razrede za ostale nadzornike pogleda.

6.2.5 Izbiralec slike

Ker bo prvi gumb služil zajemu slike, bomo na tem mestu spoznali `UIImagePickerController`. Tako bomo videli, da je delo z zajemom slik pri iOS res minimalno in elegantno rešeno. Prej pa moramo še povedati, katera akcija naj se proži ob kliku na gumb. V tem primeru nam delo olajša Assistant Editor (glej sliko 6.13).

Xcode nam zopet olajša delo in nam tudi zgenerira ustrezno kodo v `.m`



Slika 6.13: Povezovanje gumba z akcijo.

datoteki. Pa vključimo v to akcijo UIImagePickerController. Najprej nekaj popravkov v .h datoteki. Dodamo delegata UIImagePickerControllerDelegate (delegat je kazalec na metode in si bomo njegovo delovanje pogledali kar na tem primeru). Ustvarimo še spremenljivko picker tipa UIImagePickerController. Koda v .h datoteki izgleda takole:

```
#import <UIKit/UIKit.h>

@interface RootViewController : UIViewController <
    UIImagePickerControllerDelegate>{
    UIImagePickerController *picker;
}
- (IBAction)takePictureButtonClicked;

@end
```

Z vključitvijo UIImagePickerControllerDelegate-a v naš razred omogočimo zaznavanje dogodkov, ki bodo povezani z našim picker-jem. Vsak delegat ima na voljo več metod, ki jih lahko kliče ob prožitvi določenega dogodka. Nekatere metode so obvezne in jih moramo implementirati, ampak večina je neobveznih. V našem konkretnem primeru delegat lahko kliče me-

todi `imagePickerController:didFinishPickingMediaWithInfo:` in `imagePickerControllerDidCancel:`. Prva se kliče, ko uspešno kliknemo gumb Use, ko smo zajeli sliko, druga pa kadar kliknemo gumb Cancel. Več o delegaciji lahko preberemo v [20].

Pa si pogledjmo še implementacijski del našega razreda:

```
- (IBAction)takePictureButtonClicked {
    picker = [[UIImagePickerController alloc] init]; //
    inicializacija
    picker.delegate = self; //nastavitev delegata na ta razred
    if([UIImagePickerController isSourceTypeAvailable:
        UIImagePickerControllerSourceTypeCamera])
        picker.sourceType = UIImagePickerControllerSourceTypeCamera;
    [self.navigationController presentViewController:picker
        animated:YES];//aktiviramo kamero
}
```

To je vse kar je potrebno storiti za zajem slike. Ko pa bomo končali z zajemom slike, bomo verjetno z njo želeli kaj storiti. Tako bo ob koncu zajema slike delegat poklical metodo `-(void)imagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:(NSDictionary *)info`, kjer bomo obdelovali zajeto sliko. Na tem mestu bi bilo primerno implementirati metodo za spreminjanje velikosti slik, saj slika originalne velikosti zasede preveč prostora, upočasni delovanje, generirani film pa bi bil prevelik za uporabo. Ta metoda za samo izvedbo diplome ni bistvena in ne bo opisana.

Tako zajeto sliko bomo sedaj shranili v prej pripravljeno podatkovno bazo. Ker Core Data uporabljajo metode `NSManagedObjectContext` je prav, da povemo tudi nekaj malega o tem. Te metode si lahko predstavljamo kot vmesnik za dostopanje do baze. Kadarkoli bomo hoteli kaj dodati, vzeti, brisati iz baze bomo uporabili te metode. Te metode je za nas že avtomatično general Xcode v AppDelegat-u, ko smo izbrali na začetku, da bomo uporabljali Core Data. Sedaj moramo le prenesti oziroma povezati, da bodo imeli dostop do teh metod tudi v `RootViewController`-ju. V njem najprej deklariramo lastnost `@property (nonatomic, strong) NSManagedObjectContext* context;`, nato pa v `AppDelegate.m` s spodnjimi vrsticami (damo jih v metodo `application didFinishLaunchingWithOptions`) nastavimo to lastnost:

```
UINavigationController *navigationController = (
    UINavigationController *)self.window.rootViewController;
RootViewController *RVC = (RootViewController *)
    navigationController.topViewController;
```

```
RVC.managedObjectContext = self.managedObjectContext;
```

Sedaj bomo lahko dostopali do teh metod tudi v RootViewController-ju. Pa pogledjmo, kako shranimo zajeto sliko v Core Data. Najprej ustvarimo objekt `Picture`, ki ga ustavimo v prej ustvarjen `context`, nato mu določimo attribute in shranimo. Potrebna koda:

```
Picture *pic=(Picture *)[NSEntityDescription
    insertNewObjectForEntityForName:@"Picture"
    inManagedObjectContext:context];
pic.date=[NSDate date];
pic.data=resizedImage;
NSError *error;
if (![context save:&error]) {
    NSLog(@"Prislo je do napake: %@", [error localizedDescription
    ]);
}
```

Ta koda nam bo shranila zajeto sliko in ji dodala datum oziroma bo javila napako, če je med shranjevanjem prišlo do napake.

6.2.6 Shranjevanje v iCloud

Uspešno smo shranili sliko v Core Data. Sedaj pa moramo shraniti te podatke v iCloud. Zato bomo morali poseči v metode AppDelegate-a, ki nam jih je Xcode avtomatično generiral za uporabo Core Data. Vmesnik AppDelegate-a je že ustrezen za delo z iCloud-om in ne rabi dodatnih sprememb.

V implementacijski datoteki `.m` pa bomo storili kar nekaj sprememb, ki so opisane v kodi, ki je zaradi preglednosti dodana kot priloga A.

Pri izdelavi teh metod smo se zgledovali predvsem po Applovem primeru na spletni strani Apple Developer Forum [21] in Applovih videih Adopting iCloud Storage [22] (na voljo so nam samo kot registriranim razvijalcem). Ta primer smo še prilagodili A, da bo deloval tudi, če uporabnik onemogoči iCloud v nastavitvah, oziroma če iCloud ni dostopen.

Sedaj bodo te metode same poskrbele za sinhronizacijo med napravo in iCloud-om, brez kakršne koli dejavnosti uporabnika. Tako bomo imeli v iCloud-u vedno shranjeno podatkovno bazo, do katere bomo lahko dostopali z različnih naprav.

6.2.7 Branje iz Core Data

Sliko smo shranili v Core Data, sedaj pa je čas, da jo tudi preberemo nazaj. To bomo storili za prikaz galerije zajetih slik. V datoteki GalleryViewControler, ki jo ustvarimo enako kot smo RootViewController, bomo prebrali zajete slike v NSArray, z dodano kodo v metodo viewDidLoad:

```

NSEntityDescription *entityDesc = [NSEntityDescription
    entityForName:@"Picture" inManagedObjectContext:
    managedObjectContext];

NSFetchRequest *request = [[NSFetchRequest alloc] init];
[request setEntity:entityDesc];

NSError *error;
picturesArray = [managedObjectContext executeFetchRequest:
    request error:&error];
NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
    initWithKey:@"date" ascending:YES];

picturesArray=[picturesArray sortedArrayUsingDescriptors:[
    NSArray arrayWithObject:sortDescriptor]];

```

Koda ne potrebuje dodatnega komentarja, saj so že imena metod in spremenljivk dovolj zgovorna.

Da pa bi bila galerija čimbolj podobna Applovi, se bomo poslužili trika in sicer bomo slike dodajali na UIScrollView kot gumbe, ki bodo imeli za ikone zajete slike. To kodo bomo dodali v GalleryViewController.m datoteko v metodo viewDidLoad:

```

UIScrollView *view = [[UIScrollView alloc] initWithFrame:[[
    UIScreen mainScreen] bounds]];
view.backgroundColor=[UIColor whiteColor];
Picture *pic;
int row = 0;
int column = 0;
for(int i = 0; i < picturesArray.count; ++i) {
    pic=[picturesArray objectAtIndex:i];
    UIImage *thumb = pic.thumb;
    UIButton * button = [UIButton buttonWithType:
        UIButtonTypeCustom];
    button.frame = CGRectMake(column*79+4, row*79+4, 75, 75);

```

```
[button setImage:thumb forState:UIControlStateNormal];
[button addTarget:self action:@selector(buttonClicked:)
  forControlEvents:UIControlEventTouchUpInside];
button.tag = i;
[view addSubview:button];
if (column == 3){
    column = 0;
    row++;
}else {
    column++;
}
}
[view setContentSize:CGSizeMake(320, (row+1) * 80 + 10)];
self.view = view;
```

Ko sedaj zaženemo projekt in kliknemo na gumb Galerija slik, bi morali imeti prikazane ikone zajetih slik. Na tem mestu se implementira še akcija, ki se proži s klikom na sliko in nam prikaže sliko v originalni velikosti. V zgornji kodi lahko tudi opazimo, da smo ustvarili gumb programsko in ne z dodajanjem na scenariju. Tako bomo ob kliku na eno izmed slik klicali metodo `buttonClicked`, v kateri se implementira prikaz slike, sama implementacija pa je prepuščena bralcu. Ker mora vsak uporabnik imeti tudi možnost brisanja slike, bomo prikazali brisanje iz Core Data in da se bomo čim bolj približali Applovi galeriji, bomo pogledali še, kako uporabljati `UIActionSheet` (glej sliko 6.14).

6.2.8 Meni akcij

Meni akcij (angl. Action Sheet) je meni, ki prikazuje možnosti akcij, ki jih dobimo ob kliku na nek gumb. V našem primeru kliknemo na gumb izbriši in se nam pojavi meni akcij kot na sliki 6.14.

Vse kar moramo narediti za njegovo uporabo je, da v `.h` datoteki dodamo delegata `UIActionSheetDelegate` in v `.m` datoteki sledečo kodo:

```
UIActionSheet *popupQuery = [[UIActionSheet alloc] initWithTitle
 :@"" delegate:self cancelButtonTitle:@"Preklici "
  destructiveButtonTitle:@"Zbrisi sliko" otherButtonTitles: nil
  ];
popupQuery.actionSheetStyle = UIActionSheetStyleBlackOpaque;
[popupQuery showInView:self.view];
```



Slika 6.14: Meni akcij.

`UIActionSheetDelegate` glede na našo odličitev kliče metodo `-(void)actionSheet:(UIActionSheet *)actionSheet clickedButtonAtIndex:(NSInteger)buttonIndex`, kjer izvedemo brisanje slike (`if(buttonIndex==0)`) oziroma ničesar, če uporabnik klikne gumb `Prekliči` (`if(buttonIndex==1)`).

6.2.9 Brisanje entitete iz Core Data

Brisanje je resnično preprosto in ga izvedemo z eno vrstico kode. Ko brisanje izvedemo moramo spremembe še shraniti:

```
[managedObjectContext deleteObject:clickedImage];  
NSError *error;  
[managedObjectContext save:&error];
```

6.2.10 Generiranje filma

Ustvarimo `GenerateMovieViewController` in ga povežemo s pripadajočim nadzornikom pogleda na scenariju. Dodamo gumb, ki bo ob kliku prožil akcijo generiranja filma. Za generiranje filma si bomo pomagali z že pripravljenim razredom, ki ga najdemo s pripadajočimi navodili in primerom uporabe na: <https://github.com/unixpickle/VideoExporter.git>.

Na tem mestu je smiselno dati uporabniku možnost, da si sam izbira hitrost prehoda med sličicami (FPS) in spremljevalno glasbo. Zajem hitrosti prehoda med sličicami naj bralec implementira sam, mi pa bomo prikazali, kako uporabniku omogočiti izbiro spremljevalne glasbe.

6.2.11 Dodajanje spremljevalne glasbe

Da bi dodali spremljevalno glasbo, jo moramo najprej izbrati. Nato uvozimo `<MediaPlayer/MediaPlayer.h>` knjižnjico. Dodati moramo tudi `MPMediaPickerControllerDelegate-a`. Vse to storimo seveda v .h datoteki, ki zgleda takole:

```
#import <UIKit/UIKit.h>
#import <MediaPlayer/MediaPlayer.h>
#import "VideoExporter.h"

@interface GenerateMovieViewController : UIViewController <
    MPMediaPickerControllerDelegate, VideoExporterDelegate>
{
    VideoExporter * exporter;
    NSArray * picturesArray;
    NSString * rawMoviePath;
    MPMoviePlayerController * player;
    NSURL *audioURL;
    AVMutableVideoComposition *mutableVideoComposition;
    MPMusicPlayerController *soundPlayer;
}

@property (nonatomic,strong) NSManagedObjectContext*
    managedObjectContext;
@property (nonatomic,strong) AVPlayer* AVPlayerForSong;
- (IBAction)chooseAudioButtonClicked:(id)sender;
- (IBAction)generateMovieButtonClicked:(id)sender;
```

@end

V .m datoteki pa imamo nekoliko več dela. Najprej zopet prožimo akcijo, na primer s klikom na gumb Izberi glasbo. Ta akcija bo klicala metodo, v kateri inicializiramo in nastavimo vrednosti MPMediaPickerController-u. Koda je sledeča:

```
MPMediaPickerController *picker = [[MPMediaPickerController alloc
    ] initWithMediaTypes: MPMediaTypeMusic];
picker.delegate = self;
picker.allowsPickingMultipleItems = NO;
picker.prompt = @"Izberi spremljevalno pesem";
[self presentViewController: picker animated: YES];
```

Uporabniku se tako pojavi vmesnik, kjer izbira željeno glasbo. Ko izbere, delegat kliče metodo – (void) mediaPicker: (MPMediaPickerController *) mediaPicker didPickMediaItems: (MPMediaItemCollection *) mediaItemCollection. V tej metodi dostopamo do določenih lastnosti izbrane glasbe. Poglejmo, kako bi dostopali do naslova, kjer je shranjena, naslova pesmi, izvajalca in dolžine:

```
MPMediaItem *clickedSong = (MPMediaItem *) [mediaItemCollection.
    items objectAtIndex: 0];
NSURL *potDoPesmi= [clickedSong valueForKeyProperty:
    MPMediaItemPropertyAssetURL];
NSString *songTitle=[clickedSong valueForKeyProperty:
    MPMediaItemPropertyTitle];
NSString* songArtist=[clickedSong valueForKeyProperty:
    MPMediaItemPropertyArtist];
float songDuration=[[clickedSong valueForKeyProperty:
    MPMediaItemPropertyPlaybackDuration] floatValue];
```

Ko imamo vse to določeno, kličemo metodo za generiranje filma, ko je film zgeneriran pa kličemo še metodo za dodajanje glasbe h generiranemu filmu. Ker je sama metoda dodajanja glasbe precej obsežna je dodana kot priloga B.

Ko imamo tako zgeneriran film z avdio spremljavo, nam preostane le še, da ga predvajamo in ponudimo uporabniku možnosti izvoza. V našem primeru bo to izvoz v Picture Gallery, na Youtube in Facebook. Pa pogledjmo, kako bi predvajali zgeneriran film.

6.2.12 Predvajanje filma

Tako kot ponavadi bomo najprej začeli pri .h datoteki in delegatu: dodamo delegata `MPMediaPickerControllerDelegate` in spremenljivko `MPMoviePlayerController *player`. To so vse spremembe v .h datoteki:

```
#import <UIKit/UIKit.h>
#import <MediaPlayer/MediaPlayer.h>

@interface PlayMovieViewController : UIViewController<
    MPMediaPickerControllerDelegate>{
    MPMoviePlayerController * player;
}

@property (weak, nonatomic) IBOutlet NSURL *urlOfMovie;

@end
```

V .m datoteki pa sledeča koda inicializira predvajalnik in ga pripravi na predvajanje ter predvaja:

```
player = [[MPMoviePlayerController alloc] initWithContentURL:
    urlOfMovie];
player.controlStyle = MPMovieControlStyleFullscreen;
[player setFullscreen:YES];
[player prepareToPlay];
[player.view setFrame:self.view.bounds];
[self.view addSubview:player.view];
player.shouldAutoplay = YES;
```

To so osnovne nastavitve predvajalnika, sedaj pa se moramo še prijaviti na obvestila, ki jih proizvaja naš predvajalnik, da bomo lahko zaznali določene spremembe. Na obvestila se je najbolje prijavljati v metodi `viewWillAppear`, odjavljati od njih pa v metodi `viewWillDisappear`. Metodi bi ob prijavi na obvestilo, da se je končalo predvajanje posnetka, izgledali takole:

```
-(void)viewWillAppear:(BOOL)animated{
    [[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(playerDone) name:
    MPMoviePlayerPlaybackDidFinishNotification object:nil];
}

-(void)viewWillDisappear:(BOOL)animated{
    [[NSNotificationCenter defaultCenter] removeObserver: self];
```

```
}
```

Metoda prijavi trenutni nadzornik pogleda na obvestilo `MPMoviePlayerPlaybackDidFinishNotification`. Ko bo predvajalnik proizvedel to sporočilo, se bo klicala metoda `playerDone`, v kateri izvedemo željene funkcije, ko se predvajanje konča. V našem primeru bomo izvedli prehod na naslednji nadzornik pogleda, kjer bomo imeli naštete možnosti izvoza filma. Metoda `playerDone` izgleda takole:

```
-(void)playerDone{
    [self performSegueWithIdentifier:@"showExportOptions" sender:
    self];
}
```

`showExportOptions` je ime prehoda, ki smo ga določili na scenariju na povezavi med `playMovieViewController` in `exportViewController`.

Da pa uporabnik naše aplikacije ne bo pozabil zajeti slike ob določenem času, si bomo še pogledali, kako nastaviti opozorila.

6.2.13 Opozorila

Za opozarjanje uporabnika, naj zajame novo sliko, bomo uporabljali `UILocalNotification`. Prej moramo zopet narediti nek vmesnik, s katerim bo uporabnik lahko določil, kdaj naj prejema opozorila oziroma kako pogosto. Implementacijo zajemanja teh podatkov naj bralec izvede sam, mi bomo prikazali le, kako se uporablja `UILocalNotification`:

```
UILocalNotification *localNotif = [[UILocalNotification alloc]
    init];
localNotif.fireDate = itemDate;//ura, ko naj se prozi opozorilo
localNotif.timeZone = [NSTimeZone defaultTimeZone];
//nastavitev ponavljanja
if([repeatString isEqualToString:@"Vsak dan"])localNotif.
    repeatInterval=NSDayCalendarUnit;
if([repeatString isEqualToString:@"Vsak teden"])localNotif.
    repeatInterval=NSWeekCalendarUnit;
if([repeatString isEqualToString:@"Vsak mesec"])localNotif.
    repeatInterval=NSMonthCalendarUnit;
// Tekst opozorila
localNotif.alertBody = @"Cas za novo fotografijo!";
// naslov gumba opozorila
localNotif.alertAction = @"View";
```

```
// nastavitve opozorilnega zvoka
localNotif.soundName = UILocalNotificationDefaultSoundName;
localNotif.applicationIconBadgeNumber = 1;
//opozorilo nastavimo kot aktivno
[[UIApplication sharedApplication] scheduleLocalNotification:
    localNotif];
```

Ker pa bo uporabnik seveda spreminjal te nastavitve, je potrebno tudi vedeti, kako ta opozorila izključiti. To storimo zelo enostavno:

```
[[UIApplication sharedApplication] cancelAllLocalNotifications];
```

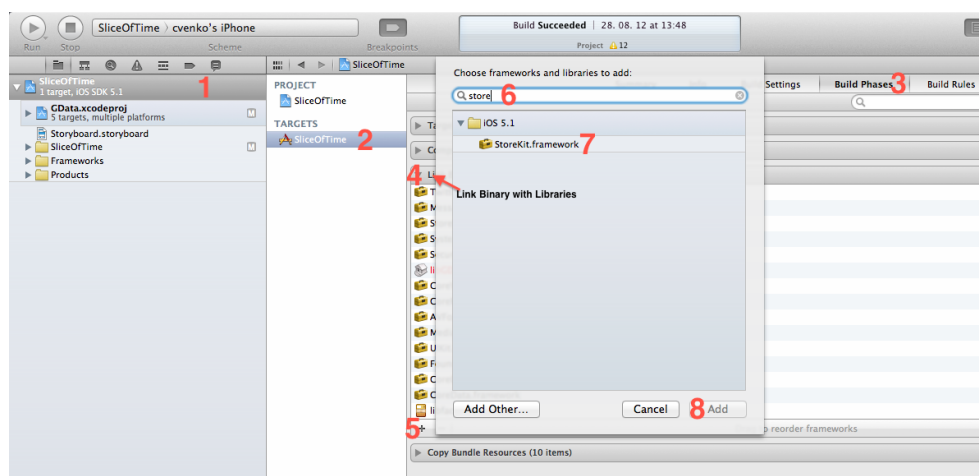
Sedaj se nam bodo prikazovala opozorila (slika 3.5), ko bo to izbral uporabnik ter se dodajala v center obvestil.

6.2.14 Nastavitve potrebne za omogočanje nakupov znotraj aplikacije

Vse nastavitve bomo izvajali v iTunes Connect, do kamor že znamo dostopati. Torej logiramo se v iTunes Connect, in kliknemo Manage Your Applications in nato Add New App. Na tem mestu moramo izpolniti ime aplikacije, SKU number (izmislimo si jo sami in je namenjena lažjemu pregledu v primeru, da imamo veliko število aplikacij) ter izberemo Bundle ID, ki smo ga ustvarili v profilu. Ostale podatke si zaenkrat izmislimo (pravih še nimamo), da nam portal ne javlja nobene napake več. Na ta način smo ustvarili mesto za aplikacijo, kjer bomo urejali vse podatke v zvezi z njo. Razlog, da smo se tega lotili sedaj pa je, da če hočemo napisati kodo za nakupe znotraj aplikacije, moramo najprej te produkte, ki bodo stvar nakupa, ustvariti.

Sedaj poiščemo in kliknemo gumb Manage In-App Purchases in nato Create New. Na voljo imamo več tipov nakupov znotraj aplikacije, mi pa se bomo odločili za Non-Consumable, saj bomo želeli imeti le opcijo za odklep celotne aplikacije (omogočanje izvoza na Facebook, Youtube v Photo Library in slikanje več objektov hkrati). Nato sledimo vodiču in dodamo nakup znotraj aplikacije. Vodič je sicer trivialen razen dela, ko morate izpolniti del Hosting Content with Apple. To je že nova funkcija iOS 6, ki je mi ne bomo uporabljali, zato tukaj izberemo "No". V iTunes Connect smo končali, sedaj pa se lotimo kodiranja.

Najprej moramo vključiti v naš projekt StoreKit.framework (slika 6.15), ter ga uvoziti na mesto, kjer ga bomo potrebovali (v našem primeru exportViewController).



Slika 6.15: Dodajanje framework-a.

Ko uporabnik želi opraviti nakup, na primer s klikom na gumb Kupi celotno verzijo, bomo najprej preverili ali ima uporabnik omogočene nakupe znotraj aplikacij v nastavitvah s sledečo kodo v metodi, ki jo kliče klik na gumb:

```
if ([SKPaymentQueue canMakePayments]) {
    // Display a store to the user.
    [self buy:nil];
} else {
    UIAlertView *noStore=[[UIAlertView alloc] initWithTitle:@"
    "Nakupi znotraj aplikacije onemogoceni" message:@"Prosimo
    omogocite nakupe v nastavitvah" delegate:self
    cancelButtonTitle:@"OK" otherButtonTitles: nil];
    [noStore show];
}
```

Ko smo preverili ali je nakup mogoč, moramo zahtevati podatke o nakupih znotraj aplikacije, ki smo jih ustvarili v iTunes Connect. Z naslednjo metodo zahtevamo podatke o našem nakupu:

```
- (void) requestProductData{
    SKProductsRequest *request= [[SKProductsRequest alloc]
    initWithProductIdentifiers: [NSSet setWithObject:
    kMyFeatureIdentifier]];
    request.delegate = self;
    [request start];
}
```

Ko prejmemo podatke, delegat kliče sledečo metodo:

```
- (void)productsRequest:(SKProductsRequest *)request
    didReceiveResponse:(SKProductsResponse *)response{
    products=response.products;//products so tipa NSArray
    inAppProduct=[products objectAtIndex:0];// inAppProduct je
    tipa SKProduct
    // Populate your UI from the products list.
    // Save a reference to the products list.
}
```

Preden nadaljujemo, moramo še sebe razglasiti kot opazovalca sprememb transakcije. To moramo storiti takoj po zagonu aplikacije:

```
exportViewController *observer = [[exportViewController alloc]
    init];
[[SKPaymentQueue defaultQueue] addTransactionObserver:observer];
```

Ta opazovalec bo klical sledečo metodo kadarkoli se bo zgodila sprememba v transakciji:

```
- (void)paymentQueue:(SKPaymentQueue *)queue updatedTransactions
    :(NSArray *)transactions{
    for (SKPaymentTransaction *transaction in transactions) {
    switch (transaction.transactionState){
        case SKPaymentTransactionStatePurchased:
            [self completeTransaction:transaction];
            break;
        case SKPaymentTransactionStateFailed:
            [self failedTransaction:transaction];
            break;
        case SKPaymentTransactionStateRestored:
            [self restoreTransaction:transaction];
        default:
            break;
        }
    }
}
```

Ko je nakup zaključen, delegat kliče naslednjo metodo:

```
- (void) completeTransaction: (SKPaymentTransaction *)
    transaction{
```

```

    [self recordTransaction:transaction];
    [self provideContent:transaction.payment.productId];
    // Remove the transaction from the payment queue.
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction
    ];
}

```

Moramo pa implementirati še metode, v primeru da se transakcija ne izvede:

```

- (void) failedTransaction: (SKPaymentTransaction *)transaction{
    if (transaction.error.code != SKErrorPaymentCancelled) {
        // Optionally, display an error here.
    }
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction
    ];
}

```

Upoštevati moramo tudi možnost, da je uporabnik že opravil nakup, a ta zaradi različnih razlogov (npr. ponovna namestitev) ni viden. Zato moramo tudi implementirati metodo za ta primer:

```

- (void) restoreTransaction: (SKPaymentTransaction *)transaction
{
    [self recordTransaction: transaction];
    [self provideContent: transaction.originalTransaction.payment.
        productId];
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction
    ];
}

```

Ko imamo vse to pripravljeno, moramo samo še ustvariti nakup in ga dodati v vrsto:

```

SKPayment *payment = [SKPayment paymentWithProduct:inAppProduct
    ];
[[SKPaymentQueue defaultQueue] addPayment:payment];

```

Ostane nam samo še, da implementiramo metodo, ki opravi spremembe v naši aplikaciji po nakupu:

```

- (void)provideContent:(NSString *)productId {
    [[NSUserDefaults standardUserDefaults] setBool:TRUE forKey:
        productId];
}

```

```
[[NSUserDefaults standardUserDefaults] synchronize];

[self viewDidLoad]; //ponovno naložimo trenutni viewController
    v katerem preverjamo status nakupa v NSUserDefaults in
    omogocamo/onemogocamo funkcije na podlagi te vrednosti
}
```

Seveda pa je po vsem tem treba preizkusiti ali nakup deluje tako kot bi moral. To storimo tako, da ustvarimo testne uporabnike v iTunes Connect. V iTunes Connect kliknemo Manage Users, nato pa Test Users, izpolnimo s poljubnimi podatki ter se na testni napravi odjavimo s pravim uporabniškim imenom (Settings/Store). Nato zaženemo aplikacijo in poskusimo opraviti nakup, kjer vnesemo testne podatke.

Podrobnosti metod, ki so bile predstavljene v tem podpoglavju so na voljo v Apple-ovi dokumentaciji [23].

Poglavje 7

Testiranje aplikacije

Aplikacija na prvi pogled deluje počasi, tako da bo vsekakor potrebna optimizacija. Vzrok počasnega delovanja je predvsem iskati v izogibanju hrošču pri uporabi iCloud-a v kombinaciji s Core Data. Namreč, če smo shranjevali spremembe v Core Data le ob izhodu iz aplikacije, je bilo lahko teh sprememb veliko in če je uporabnik zopet zagnal aplikacijo preden so se podatki sinhronizirali, je prihajalo do napak, celo do izgube vseh podatkov. V iskanju rešitev smo naleteli na kar nekaj razvijalcev s podobnimi problemi, ki so na koncu debate zaključili, da uporaba Core Data z iCloud še ni zrela za produkcijo [21, 24, 25, 26]. Tako prva verzija naše aplikacije še ne bo imela podpore za sinhronizacijo z iCloud-om. Ampak ker bi uporabnik vendarle želel prenesti slike v novejšo napravo, mu bomo to omogočili tako, da bomo aplikaciji omogočili deljenje datotek z iTunes, kjer bo lahko uporabnik prenesel bazo na računalnik, nato pa jo ponovno naložil na novo napravo. V kolikor pa bo uporabnik za nastavitve nove naprave uporabil varnostno verzijo, mu bodo slike že na voljo.

7.1 Popravki aplikacije

Prvi popravek bo omogočanje uporabniku dostop do datotek preko iTunes. To storimo tako, da v projektnem navigatorju poiščemo datoteko s končnico .plist (v našem primeru je to SliceOfTime-Info.plist) in jo izberemo. Pojavi se nam okno z nastavitami projekta. Desni klik na prazen prostor nam odpre možnosti, kjer izberemo Add Row in to novo vrstico izpolnimo z Application supports iTunes file sharing (glej sliko 7.1).

Naslednji korak je onemogočanje iCloud. To storimo enostavno tako, da odkljukamo Enable Entitlements (spomin, kako to naredimo, si osvežimo na

sliki 6.7). Torej bo imela naša aplikacija od sedaj naprej samo lokalno shranjevanje.

Že samo po teh korakih je moč opaziti hitrejše delovanje aplikacije. Ker pa nam sedaj ni treba paziti na hrošča, ki se pojavlja zaradi sinhronizacije z iCloud-om, lahko pohitrimo še zajemanje in brisanje slike tako, da odstanimo shranjevanje in pustimo samo dodajanje v `NSManagedObjectContext`. Shranili bomo samo takrat, ko bo aplikacija prešla v ozadje ali pa se bo končala. Ker pa sliki, ki jo zajamemo, tudi spremenimo velikost, nam tudi tukaj za nekaj časa zamrzne UV. Tako bi bilo pametno shranjevanje in spreminjanje velikosti slik dati v drugo nit. Tukaj bomo spoznali novost iOS 5, ki nam pri Core Data pomaga z nitmi. V `rootViewController`-ju, kjer končamo z zajemanjem slike, bomo vstavili sledečo kodo:

```
NSManagedObjectContext *child = [[NSManagedObjectContext alloc]
    initWithConcurrencyType:NSPrivateQueueConcurrencyType];
[child setParentContext:self.managedObjectContext];
[self.managedObjectContext performBlock:^(
    UIImage *resizedImage=[[UIImage alloc] init];
    if(image.size.height>image.size.width){//portrait mode
        resizedImage=[self resizeImage:image newSize:CGSizeMake(720,
            960)];
    }else {
        resizedImage=[self resizeImage:image newSize:CGSizeMake(960,
            720)];
    }
    UIImage *thumbImage=[self resizeImage:image newSize:CGSizeMake
        (75, 75)];
    Picture *pic=(Picture *)[NSEntityDescription
        insertNewObjectForEntityForName:@"Picture"
        inManagedObjectContext:self.managedObjectContext];
    pic.date=[NSDate date];
    pic.data=resizedImage;
    pic.thumb=thumbImage;
    pic.belongsToObject=defaultAlbum;

    NSError *error = nil;
    [child save:&error];
}];
```

Tukaj pravzaprav ustvarimo `NSManagedObject`, ki se ne bo izvajal v glavni

ni, zato UV ne bo blokirano.

Po teh spremembah je delovanje aplikacije, sploh zajemanje slik, zelo hitro.

Opazimo tudi, da nam UV zamrzne pri kliku na gumba Galerija in Generiraj Film. Tukaj bi lahko blokado UV povzročala dva dela kode in sicer zajemanje slik iz Core Data ali pa razvrščanje teh slik po datumu (kodi se nahajata v `viewDidLoad` metodi). Z merjenjem časa izvajanja kode smo ugotovili, da blokado povzroča razvrščanje slik po datumu. Pri generiranju filma bi se tej zamrznitvi lahko izognili, saj razvrščenih slik ne potrebujemo takoj in bi razvrščanje dali v drugo nit. Ker pa se pri prikazu Galerije ne moremo izogniti takojšnjemu razvrščanju, bomo morali poiskati drugačno rešitev. Vseeno bomo najprej prikazali, kako bi dali razvrščanje slik v drugo nit. Na tem mestu bomo uporabili iOS-ov Grand Central Dispatch, ki je gonilna sila večnitnosti pri iOS:

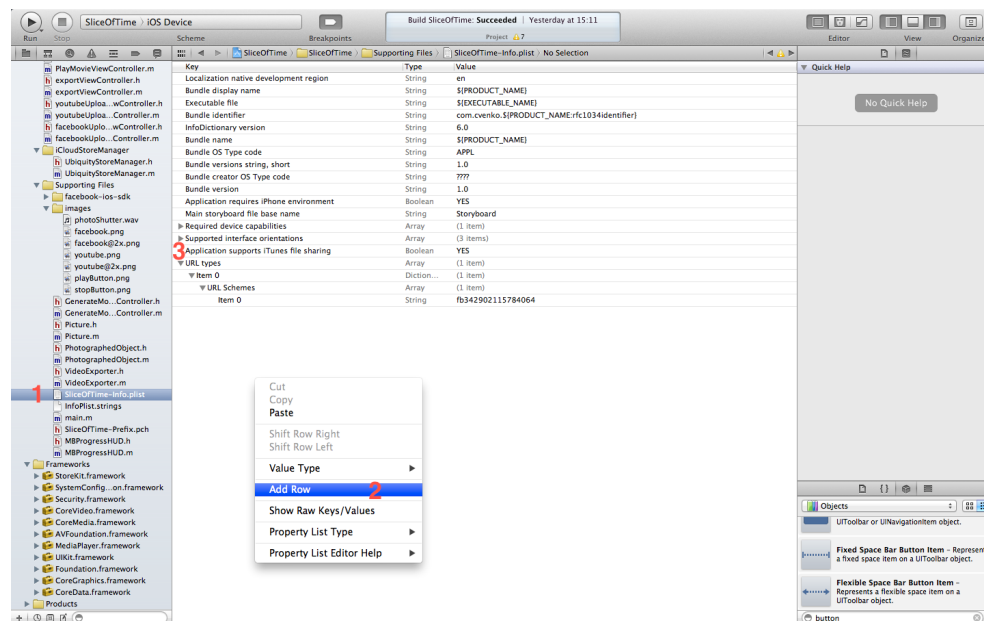
```
dispatch_async(dispatch_get_global_queue(
    DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    picturesArray=[setOfPictures
        sortedArrayUsingDescriptors:[NSArray arrayWithObject:
            sortDescriptor]];
});
```

To bi odpravilo težavo pri zamrznitvi UV pri kliku na gumb Generiraj Film.

Ker pa to ni ustrezna rešitev pri prikazu Galerije, bomo morali poiskati drugačno rešitev. Po temeljitem brskanju po spletu smo našli pravilo čez palec o shranjevanju podatkov v Core Data [27]:

- <100 kb lahko hranimo kot atribut v entiteti,
- <1 mb hranimo kot ločeno entiteto in imamo relacijo nanjo,
- >1 mb hranimo na disku in v Core Data shranimo pot do teh podatkov.

Če se zgledujemo po tem pravilu, smo naredili že v začetku napako, ko smo podatke slike shranili kar kot atribut entiteti `Picture`. V našem primeru bi tako morali dodati entiteti `Picture` relacijo `hasData`, ki kaže na novo entiteto `PictureData`, ki ima en atribut `data` tipa `Transformable`. Seveda moramo ustrezno popraviti tudi shranjevanje, brisanje in dostopanje do slike skozi celoten projekt. Ko naredimo te popravke opazimo, da nam UV ne zamrzuje nikjer več, celo pri galeriji smo dosegli znatno izboljšanje.



Slika 7.1: Dodajanje opcije deljenja datotek z iTunes.

Poglavje 8

Objavljanje aplikacije v Applovi spletni trgovini

O Applovi spletni trgovini smo že govorili v poglavju 5. V tem poglavju pa si bomo pogledali, kaj vse potrebujemo za objavljanje in sam postopek objavljanja v njej.

Najprej moramo ustvariti distribucijski profil (angl. Distribution Provisioning Profile), ki ga ustvarimo na identičen način, kot smo ustvarili razvijalski profil (glej poglavje 6.2.1), le da vse urejamo pod zavihkom Distribution. Naslednji korak naredimo v iTunes Connect, kjer moramo pripraviti aplikacijo na oddajo v spletni trgovini. Pravzaprav smo storili že skoraj vse potrebno, ko smo pripravljali nakupe znotraj aplikacije, preostane nam le še, da kliknemo View Details (znotraj Manage My Applications), kjer še zadnjič pregledamo nastavitve, nato pa kliknemo Ready To Upload Binary. Naslednje korake lahko preskočimo, saj bomo naložili aplikacijo v spletno trgovino kar iz Xcode. Vidimo, da smo dobili status aplikacije Waiting for upload. Pa naložimo še naše delo, da zaključimo proces. Vrnimo se v Xcode, se prepričamo, da smo izbrali distribucijski profil, kliknemo Product in nato Archive. Xcode nam tako zgenerira arhiv aplikacije, ki ga bomo naložili v spletno trgovino. Kliknemo Distribute, izberemo opcijo Submit to iOS App Store, vpišemo naše uporabniško ime in geslo, kliknemo Next in naše delo je na tem mestu končano. Sedaj bomo čakali od 5 do 10 dni, da Apple odobri aplikacijo in da bo vidna v Applovi spletni trgovini.

Poglavje 9

Zaključek

Tekom izdelave diplomske naloge smo spoznali, da je iOS verjetno trenutno res najbolj dodelan mobilni OS, čeprav mu je Android tesno za petami. WP je precej nov, zato mu manjka še nekaj časa, da bo nadoknadil zaostanek za konkurentoma.

Pri primerjavi oblačnih storitev smo spoznali, da imamo na voljo kar nekaj ponudb, za katero pa se je najbolje odločiti pa je odvisno od potreb uporabnika. Osebno zaenkrat koristim možnosti iCloud-a in Dropbox-a, v kolikor pa bi presegel zastojne kapacitete, bi se verjetno odločil za katero izmed preostalih opcij.

Med razvojem aplikacije smo se veliko naučili o samem postopku izdelovanja aplikacij za iOS. Naleteli smo tudi na nepričakovane težave z integracijo iCloud-a v našo aplikacijo. Ker je na spletu splošno mnenje, da iCloud tehnologija v kombinaciji s Core Data še ni zrela [21, 24, 25, 26], bomo z integracijo počakali še nekaj časa, vsekakor pa bo to velik doprinos k aplikaciji v prihodnosti.

Hkrati pa je ta diploma tudi dokaz, kako hitro se stvari v računalništvu spreminjajo, saj je napovedana že nova verzija iOS 6 s par novostmi, ki bi prišle prav tudi pri izdelavi trenutnega projekta. Te novosti so izboljšano delovanje kamere in integracija Facebook-a v sam OS. Naštejmo še nekatere druge pričakovane novosti iOS 6 [28]:

- Phone App – Phone App bo imel nove funkcije, kot so zavrnitev klica s pošiljanjem SMS-a, npr. "Imam delo", možnost, da vam naprava generira opozorilo, da pokličite zavrnjen klic, ko zapustite trenutno lokacijo.
- FaceTime preko 3G omrežja – Do sedaj je bil FaceTime na voljo samo preko Wi-Fi omrežja.

- Izboljššan brskalnik Safari.
- Maps – Maps bo popolnoma prenovljena in ne bo več temeljila na podpori kart iz Google Maps.
- Siri – Siri bo imela nove in prenovljene funkcije.
- Passbook – Virtualna denarnica za vstopnice, koncerte, kartice zvestobe itd.

Ker pa je pri aplikacijah vedno prostor za izboljšave, pogledjmo, katere so še dodatne funkcije, ki bi jih naša aplikacija lahko podpirala v prihodnosti:

- podpora iCloud,
- podpora večim socialnim omrežjem,
- uvoz predhodno zajetih slik (ne z našo aplikacijo),
- izboljšava algoritma za generiranje filma,
- lokalizacija – Prilagoditev aplikacije več jezikom.

Dodatek A

iCloud koda

Sledeča koda ja narejena po primeru na spletni strani [21]. V osnovi je identična, dodana je le še koda, ki omogoča uporabniku lokalno shranjevanje podatkov, kadar iCloud ni na voljo. V diplomski nalogi ta dodatek umeščamo v poglavje 6.2.6.

```
/*this takes the
    NSPersistentStoreDidImportUbiquitousContentChangesNotification
    and transforms the userInfo dictionary into something that
    -[NSManagedObjectContext
    mergeChangesFromContextDidSaveNotification:] can consume then
    it posts a custom notification to let detail views know they
    might want to refresh. The main list view doesn't need that
    custom notification because the NSFetchedResultsController is
    already listening directly to the NSManagedObjectContext*/
- (void)mergeiCloudChanges:(NSNotification*)note forContext:(
    NSManagedObjectContext*)moc {
    [moc mergeChangesFromContextDidSaveNotification:note];

    NSNotification* refreshNotification = [NSNotification
    notificationWithName:@"RefreshAllViews" object:self userInfo
    :[note userInfo]];
    [[NSNotificationCenter defaultCenter] postNotification:
    refreshNotification];
}

/**
Returns the managed object context for the application.
```

```

If the context doesn't already exist, it is created and bound
to the persistent store coordinator for the application.
*/
- (NSManagedObjectContext *)managedObjectContext {
    if (__managedObjectContext != nil) {
        return __managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self
        persistentStoreCoordinator];

    if (coordinator != nil) {
        NSManagedObjectContext* moc = [[NSManagedObjectContext alloc]
            initWithConcurrencyType: NSMainQueueConcurrencyType];

        [moc performBlockAndWait:^(
            [moc setPersistentStoreCoordinator: coordinator];
            [[NSNotificationCenter defaultCenter] addObserver:self
                selector: @selector (mergeChangesFrom_iCloud:) name:
                NSPersistentStoreDidImportUbiquitousContentChangesNotification
                object:coordinator];
        )];
        __managedObjectContext = moc;
    }
    return __managedObjectContext;
}

// NSNotifications are posted synchronously on the caller's
// thread
// make sure to vector this back to the thread we want, in this
// case
// the main thread for our views & controller
- (void)mergeChangesFrom_iCloud:(NSNotification *)notification {
    NSLog(@"Merging in changes from iCloud...");
    NSManagedObjectContext* moc = [self managedObjectContext];
    [moc setMergePolicy:NSMergeByPropertyObjectTrumpMergePolicy];
    [moc performBlock:^(
        [moc mergeChangesFromContextDidSaveNotification:notification
    ];

```

```

        NSNotification* refreshNotification = [NSNotification
notificationWithName:@"SomethingChanged" object:self userInfo
:[notification userInfo]];
        [[NSNotificationCenter defaultCenter] postNotification:
refreshNotification];
    }];
}

/**
Returns the managed object model for the application.
If the model doesn't already exist, it is created by merging
all of the models found in the application bundle.
*/
- (NSManagedObjectModel *)managedObjectModel{
    if (__managedObjectModel != nil){
        return __managedObjectModel;
    }
    __managedObjectModel = [NSManagedObjectModel
mergedModelFromBundles:nil];
    return __managedObjectModel;
}

- (NSPersistentStoreCoordinator *)persistentStoreCoordinator{
    if((__persistentStoreCoordinator != nil)) {
        return __persistentStoreCoordinator;
    }

    __persistentStoreCoordinator = [[NSPersistentStoreCoordinator
alloc] initWithManagedObjectModel: [self managedObjectModel
]];
    NSPersistentStoreCoordinator *psc =
        __persistentStoreCoordinator;

    // Set up iCloud in another thread:
    dispatch_async(dispatch_get_global_queue(
DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
// ** Note: if you adapt this code for your own use, you MUST
change this variable:

```

```

NSString *iCloudEnabledAppID = @"34NVC8BKJ9.com.cvenko.
    SliceOfTime";
// ** Note: if you adapt this code for your own use, you
    should change this variable:
NSString *dataFileName = @"SliceOfLife.sqlite";
// ** Note: For basic usage you shouldn't need to change
    anything else
NSString *iCloudDataDirectoryName = @"Data.nosync";
NSString *iCloudLogsDirectoryName = @"Logs";
NSFileManager *fileManager = [NSFileManager defaultManager];
NSURL *localStore = [[self applicationDocumentsDirectory]
    URLByAppendingPathComponent: dataFileName];
// NSURL *storeUrl = [[self applicationDocumentsDirectory]
    URLByAppendingPathComponent: @"SliceOfTime.sqlite"];

NSURL *iCloud = [fileManager URLForUbiquityContainerIdentifier
    :nil];

if (iCloud) { //Preverimo ali nam prejsnja vrstica vrne
    rezultat, v kolikor je nil iCloud ne deluje
    NSLog(@"iCloud is working");
    NSURL *iCloudLogsPath = [NSURL fileURLWithPath:[iCloud path
    ] stringByAppendingPathComponent: iCloudLogsDirectoryName]];

    if([fileManager fileExistsAtPath:[iCloud path]
    stringByAppendingPathComponent: iCloudDataDirectoryName] ==
    NO) {
        NSError *fileSystemError;
        [fileManager createDirectoryAtPath:[iCloud path]
        stringByAppendingPathComponent: iCloudDataDirectoryName]
        withIntermediateDirectories:YES attributes:nil error:&
        fileSystemError];
        if(fileSystemError != nil) {
            NSLog(@"Error creating database directory %@",
            fileSystemError);
        }
    }
}

```

```

    NSString *iCloudData = [[[iCloud path]
stringByAppendingPathComponent: iCloudDataDirectoryName]
stringByAppendingPathComponent: dataFileName];

    NSMutableDictionary *options = [NSMutableDictionary
dictionary];
    [options setObject:[NSNumber numberWithBool:YES] forKey:
NSMigratePersistentStoresAutomaticallyOption];
    [options setObject:[NSNumber numberWithBool:YES] forKey:
NSInferMappingModelAutomaticallyOption];
    [options setObject:iCloudEnabledAppID forKey:
NSPersistentStoreUbiquitousContentNameKey];
    [options setObject:iCloudLogsPath forKey:
NSPersistentStoreUbiquitousContentURLKey];

    [psc lock];
    [psc addPersistentStoreWithType:NSSQLiteStoreType
configuration:nil URL:[NSURL fileURLWithPath: iCloudData]
options:options error:nil];
    [psc unlock];
}
else { //Dodan del//Ta del smo dodali za lokalno shranjevanje
podatkov
    NSLog(@"iCloud is NOT working - using a local store");
    NSMutableDictionary *options = [NSMutableDictionary
dictionary];
    [options setObject:[NSNumber numberWithBool:YES] forKey:
NSMigratePersistentStoresAutomaticallyOption];
    [options setObject:[NSNumber numberWithBool:YES] forKey:
NSInferMappingModelAutomaticallyOption];

    [psc lock];
    [psc addPersistentStoreWithType:NSSQLiteStoreType
configuration:nil URL:localStore options:options error:nil];
    [psc unlock];
} //Dodan del- konec//

dispatch_async(dispatch_get_main_queue(), ^{

```

```
        [[NSNotificationCenter defaultCenter] postNotificationName:@  
        "RefetchAllDatabaseData" object:self userInfo:nil];  
    });  
    return __persistentStoreCoordinator;  
}
```

Dodatek B

Metoda dodajanja spremljevalne glasbe k filmu

Sledeča metoda doda filmu glasbeno spremljavo. Kot vhodna parametra prejme pot (angl. file path) do filma tipa NSString in pot, kamor naj se zapiše film z dodano glasbeno podlago. To metodo smo omenili v poglavju 6.2.11.

```
-(void) addAudioToFileAtPath:(NSString *) filePath toPath:(  
    NSString *)outFilePath  
{  
    NSError * error = nil;  
    AVMutableComposition * composition = [AVMutableComposition  
        composition];  
    AVURLAsset * videoAsset = [AVURLAsset URLAssetWithURL:[NSURL  
        fileURLWithPath:filePath] options:nil];  
    AVAssetTrack * videoAssetTrack = [[videoAsset  
        tracksWithMediaType:AVMediaTypeVideo] objectAtIndex:0];  
    AVMutableCompositionTrack *compositionVideoTrack = [  
        composition addMutableTrackWithMediaType: AVMediaTypeVideo  
        preferredTrackID: kCMPersistentTrackID_Invalid];  
  
    [compositionVideoTrack insertTimeRange:CMTimeRangeMake(  
        kCMTimeZero,videoAsset.duration) ofTrack:videoAssetTrack  
        atTime:kCMTimeZero error:&error];  
  
    CMTime audioStartTime = CMTimeMakeWithSeconds(0, 1);  
    CMTime audioEndTime=CMTimeMakeWithSeconds((picturesArray.count  
        /[FPS intValue])+2, 1);
```

```

AVURLAsset * urlAsset = [AVURLAsset URLAssetWithURL:audioURL
options:nil];

AVAssetTrack * audioAssetTrack = [[urlAsset
tracksWithMediaType:AVMediaTypeAudio] objectAtIndex:0];
AVMutableCompositionTrack *compositionAudioTrack = [
composition addMutableTrackWithMediaType: AVMediaTypeAudio
preferredTrackID: kCMPersistentTrackID_Invalid];
[compositionAudioTrack insertTimeRange:CMTimeRangeMake(
audioStartTime,audioEndTime) ofTrack:audioAssetTrack atTime:
kCMTimeZero error:&error];

AVAssetExportSession* assetExport = [[AVAssetExportSession
alloc] initWithAsset:composition presetName:
AVAssetExportPresetHighestQuality];
assetExport.videoComposition = mutableVideoComposition;
assetExport.outputFileType =AVFileTypeQuickTimeMovie;
assetExport.outputURL = [NSURL fileURLWithPath:outFilePath];
[assetExport exportAsynchronouslyWithCompletionHandler:^(void
) {
switch (assetExport.status){
case AVAssetExportSessionStatusCompleted:
NSLog(@"Export Complete");
dispatch_async(dispatch_get_main_queue(), ^{
[self playVideo:assetExport.outputURL];
});
break;
case AVAssetExportSessionStatusFailed:
NSLog(@"Export Failed");
NSLog(@"ExportSessionError: %@", [assetExport.error
localizedDescription]);
//export error (see exportSession.error)
break;
case AVAssetExportSessionStatusCancelled:
NSLog(@"Export Failed");
NSLog(@"ExportSessionError: %@", [assetExport.error
localizedDescription]);
//export cancelled
break;
}
}];

```

```
    }  
  }];  
}
```


Slike

2.1	Xcode je močno orodje za razvijanje aplikacij	5
3.1	iOS 5 je napredni mobilni operacijski sistem	7
3.2	iCloud je Applova oblačna storitev	8
3.3	Scenarij	10
3.4	Center obvestil na napravah iPod, iPad in iPhone	11
3.5	Reklamna pasica	11
4.1	Center obvestil Android	22
4.2	Primer zapečenega obvestila pri WP7	22
5.1	Obvestilo o nameri nakupa znotraj aplikacije	24
6.1	Ustvarjanje nove identifikacijske številke aplikacije	37
6.2	Konfiguracija nove identifikacijske številke aplikacije	37
6.3	Omogocanje iCloud podpore	38
6.4	Obrazec za izdelavo profila	39
6.5	Organizer s prikazanim veljavnim profilom	39
6.6	Okno z nastavitvami projekta	40
6.7	Nastavitve projekta Xcode za podporo iCloud-u	41
6.8	Izbira glavnega scenarija	41
6.9	Opravična Vrstica	41
6.10	Dodajanje nadzornika navigacije	42
6.11	Povezovanje nadzornikov pogleda	43
6.12	Predstavitev podatkov Core Data	45
6.13	Povezovanje gumba z akcijo	46
6.14	Meni akcij	51
6.15	Dodajanje framework-a	57
7.1	Dodajanje opcije deljenja datotek z iTunes	64

Tabele

4.1	Pregled prostorskih lastnosti oblčnih storitev	13
4.2	Pregled datotečnih lastnosti oblčnih storitev	15
4.3	Pregled enostavnosti uporabe	16
4.4	Pregled podprtih naprav	17
4.5	Pregled mesečnih stroškov Google Drive	18
4.6	Pregled letnih stroškov SkyDriveDrive	18
4.7	Pregled mesečnih in letnih stroškov Dropbox	18
4.8	Pregled letnih stroškov Amazon Cloud Drive	19
4.9	Pregled letnih stroškov iCloud	19
4.10	Pregled stoškov za 1GB na leto	19

Literatura

- [1] (2012) iOS - Wikipedia, The Free Encyclopedia. Dostopno na:
<http://en.wikipedia.org/wiki/IOS>
- [2] (2012) iOS Features. Dostopno na:
<http://www.apple.com/ios/features.html>
- [3] (2012) Is Apple's iCloud considered a SaaS or IaaS application. Dostopno na:
<http://www.focus.com/questions/apples-icloud-considered-saas-iaas/>
- [4] (2012) Cloud Computing Today. Dostopno na:
<http://cloud-computing-today.com/2011/06/13/apples-icloud-takes-cloud-computing-beyond-iaas-paas-and-saas-trinity/>
- [5] (2012) Amazon Cloud Drive. Dostopno na:
https://www.amazon.com/gp/feature.html?ie=UTF8&docId=1000796931&ref_=cd_lm_rd_fp
- [6] (2012) Dropbox. Dostopno na:
<https://www.dropbox.com/>
- [7] (2012) SkyDrive. Dostopno na:
<https://skydrive.live.com/>
- [8] (2012) Google Drive. Dostopno na:
<https://drive.google.com/start?authuser=0#home>
- [9] (2012) iCloud. Dostopno na:
<https://www.icloud.com/>
- [10] (2012) How Apple stole notifications from Android and beat them with iOS 5, Jonathan S. Geller. Dostopno na:

<http://www.bgr.com/2011/06/13/how-apple-stole-notifications-from-android-and-beat-them-with-ios-5/>

- [11] (2012) Apple iOS 5: Notification Center - Is it Similar to Android. Dostopno na:
<http://www.youtube.com/watch?v=WidJIe-ktT0>
- [12] (2012) iTunes Store: About In-App Purchases. Dostopno na:
<http://support.apple.com/kb/ht4009>
- [13] (2012) Apple Enroll. Dostopno na:
<https://developer.apple.com/programs/ios/>
- [14] (2012) Learn Objective C. Dostopno na:
http://cocoadevcentral.com/d/learn_objectivec/
- [15] Stephen G. Kochan, *Programming in Objective-C Fourth Edition*, Addison-Wesley, 2012, part 1.
- [16] (2012) String Format Specifiers. Dostopno na:
<https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/Strings/Articles/formatSpecifiers.html>
- [17] (2012) Member Center. Dostopno na:
<https://developer.apple.com/membercenter/index.action>
- [18] (2012) Core Data. Dostopno na:
<http://developer.apple.com/library/mac/#documentation/cocoa/Conceptual/CoreData/cdProgrammingGuide.html>
- [19] Shawn Welch, *iOS 5 Core Frameworks Develop and Design*, Berkeley: Peachpit Press, 2012, pog 2.
- [20] Matt Neuburg, *Programming iOS 5, Second Edition*, Beijing, Cambridge, Farnham, Koln, Sebastopol, Tokyo: O'Reilly, 2011, pogl. 11.
- [21] (2012) iPhoneCoreDataRecipes Sample Code. Dostopno na:
<https://devforums.apple.com/thread/126670?tstart=0>
- [22] (2012) iOS Development Videos. Dostopno na:
<https://developer.apple.com/videos/ios/>

- [23] (2012) In-App Purchase Programming Guide. Dostopno na:
<https://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/StoreKitGuide/AddingaStoretoYourApplication/AddingaStoretoYourApplication.html>
- [24] (2012) Partial solution for iCloud and Core Data Ubiquitous update bug. Dostopno na:
http://www.freelancemadscience.com/fmslabs_blog/2012/3/30/partial-solution-for-icloud-core-data-ubiquitous-update-bug.html
- [25] (2012) Using Core Data with iCloud. Dostopno na:
<https://goddess-gate.com/dc2/index.php/post/452>
- [26] (2012) Core Data in iCloud, Tim Roadley. Dostopno na:
<http://timroadley.com/2012/04/03/core-data-in-icloud/>
- [27] (2012) Core Data - Storing Images (iPhone). Dostopno na:
<http://stackoverflow.com/questions/2090028/core-data-storing-images-iphone>
- [28] (2012) iOS 6 Preview. Dostopno na:
<http://www.apple.com/ios/ios6/>