Razvoj IPHONE aplikacije pomoću programskog jezika Swift

Vuković, Vanja

Sveučilište u Zagrebu
Prirodoslovno-matematički fakultet
Matematički odsjek

Vanja Vuković

RAZVOJ IPHONE APLIKACIJA POMOĆU PROGRAMSKOG JEZIKA SWIFT

Diplomski rad

Zagreb, rujan 2015.

Ovaj diplomski rad obranjen je dana _____ pred nastavničkim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____ .

Potpisi članova povjerenstva:

1. _____

2. _____

3. _____

Sveučilište u Zagrebu
Prirodoslovno-matematički fakultet
Matematički odsjek

Vanja Vuković

RAZVOJ IPHONE APLIKACIJA POMOĆU PROGRAMSKOG JEZIKA SWIFT

Diplomski rad

Voditelj rada:
Prof. dr. sc. Robert Manger

Zagreb, rujan 2015.

SADRŽAJ

UVOD 6

1 **Osnove** 9
1.1. Razvojna okolina, alati, programski jezici i biblioteke 9
1.2. Xcode - korisničko sučelje 13
1.3. Playground 19
1.4. MVC model 20

2 **Razvoj korisničkog sučelja** 22
2.1. Uvod 22
2.2. Storyboard 23
2.3. Pogled 25
2.4. Učitavanje grafičkog sučelja 26
2.5. Dodavanje elemenata korisničkog sučelja 28
2.6. Biblioteka objekata..... 30
2.7. Atributi 32
2.8. Autolayout 33

3 **Swift** 35
3.1. Uvod 35
3.2. Hello world 35
3.3. Osnovna sintaksa 36
3.4. Ključne riječi 38
3.5. Varijable 38
3.6. Konstante 40
3.7. Naredbe za grananje 40
3.8. Petlje 41
3.9. Znakovni niz 46
3.10. Enumeracije i strukture 50
3.11. Klase 51
3.11.1. Sintaksa 51
3.11.2. Varijable članice 52
3.11.2.1. Stored properties 53
3.11.2.2. Lazy stored properties 54
3.11.2.3. Computed properties 55
3.11.3. Metode 56
3.11.4. Nasljeđivanje 58

4 **Primjer aplikacije** 60

Sadržaj

5 **Zaključak** 64

Literatura 65

Sažetak 66

Summary 67

Životopis 68

UVOD

Glavna tema mog diplomskog rada na Prirodoslovno matematičkom fakultetu u Zagrebu su Iphone aplikacije, odnosno njihov razvoj pomoću programskog jezika Swift. Za ovu temu sam se odlučio vrlo lako, bez previše razmišljanja. Razvojem mobilnih aplikacija tržište softvera (*eng. software*) je dobilo novu dimenziju. Otvorilo je vrata razvijačima aplikacija (*eng. developerima*) da na vrlo jednostavan i poprilično jeftin način plasiraju svoje aplikacije krajnjem korisniku što predstavlja veliku motivaciju programerima, nudeći im milijune potencijalnih korisnika putem Appleovog App Storea. Od samog početka tržište mobilnih aplikacija se rapidno širi i napreduje te samim time postaje izazov za programere koji moraju konstantno pratiti razvoj, kako samog tržišta tako i tehnologija. Kako smo na našem fakultetu imali niz kolegija koji se baziraju na objektno orijentiranom programiranju, Iphone aplikacije su mi se pokazale kao idealan način kako znanje stečeno na fakultetu mogu primijeniti u praksi.

Kao alat za razvoj aplikacija sam odabrao Swift koji je također objektno orijentirani programski jezik i koji je jako nov. Kako je i Apple 2014. na Worldwide Developers Conference (WWDC) prilikom prvog predstavljanja Swifta najavio, Swift ima niz prednosti nad Objective-C te je to jedan od glavnih razloga zašto sam ga odabrao kao programski jezik. Kako tema mog diplomskog nije objektno orijentirano programiranje, tako se u diplomskom radu nisam doticao osnova i koncepata koje zastupa objektno orijentirano programiranje. Pretpostavka je da čitatelj posjeduje osnovno znanje vezano uz objektno orijentirano programiranje i njegovu filozofiju te je upoznat s pojmovima kao što je klasa, metoda, objekt i sl.

Prvo poglavlje ovog diplomskog rada nosi naziv “Osnove”. Na početku poglavlja govorit ćemo o osnovnim pojmovima, odnosno programskim jezicima koje imamo na raspolaganju za razvoj, alatu kojim ćemo se koristiti pod nazivom Xcode, najpoznatijim bibliotekama pod nazivom Cocoa i Cocoa Touch, razlici između operativnih sistema OS X i iOS. Na kraju poglavlja govorit ćemo o App Storu kao centralnoj platformi za distribuciju aplikacija te o načinu funkcioniranja istog. Navesti ćemo glavne značajke programskih jezika Objective-C i Swift, dati primjere koda svakog od njih na kojima ćemo pokazati jednostavnost sintakse Swifta nad Objective-C. U drugom potpoglavlju razmatrat ćemo korisničko sučelje programa Xcode, njegove panele, glavne funkcionalnosti i način pokretanja i testiranja aplikacija. U trećem potpoglavlju govorimo o playgroundima koji su došli s verzijom 6 Xcodea. U četvrtom potpoglavlju govorimo o MVC modelu razvoja aplikacija koji je glavna ideja razvoja iPhone aplikacija. Razmatrat ćemo je teoretski i navesti glavne zadaće koji MVC model propisuje za svaki dio aplikacije.

Drugo poglavlje je posvećeno razvoju korisničkog sučelja (*eng. user interface - UI*) aplikacije. U uvodnom dijelu govorit ćemo o načinima na koje možemo kreirati korisničko sučelje, spomenuti Auto layout i prokomentirati neke od dizajnerskih načela o kojima bi trebali voditi računa prilikom dizajniranja korisničkog sučelja. Veliki dio ovog poglavlja posvetili smo Storyboardu koji je vizualna prezentacija korisničkog sučelja te pogledima (*eng. view*) koji su osnovni objekti koje koristimo prilikom implementacije korisničkog sučelja. Uz navedeni objekt proučit ćemo i ostale objekte koji nam stoje na raspolaganju, kako ih dodati u postojeće korisničko sučelje i kako postaviti vrijednosti njihovih atributa. Spomenuti ćemo i hijerarhiju pogleda (*eng. view hierarchy*). Hijerarhija pogleda nam služi za fizičko grupiranje objekata unutar korisničkog sučelja. No, hijerarhija pogleda nam koristi i prilikom samog učitavanja korisničkog sučelja. Na kraju poglavlja bavimo se Auto layoutom, koji nam služi za definiranje položaja objekata u ovisnosti o drugim elementima korisničkog sučelja i položaja uređaja na kojem se korisnik služi aplikacijom.

Treće poglavlje je u potpunosti posvećeno Swiftu kao programskom jeziku, odnosno njegovoj sintaksi. Prilikom pisanja ovog poglavlja uzeli smo pretpostavku da već posjedujemo osnovno znanje objektno orijentiranih jezika, pa tako nismo posebno objašnjavali npr. kako funkcioniraju petlje ili ideju klasa i objekata. Svi primjeri navedeni u četvrtom poglavlju smo pisali unutar Playgrounda. Četvrto poglavlje započinjemo s uvodom, Hello world programom i kratkim opisom Playgrounda. Nakon toga govorimo o osnovnim sintaksnim pravilima Swifta i rezerviranim riječima, varijablama i konstantama. Zatim prelazimo na naredbe za grananje (*eng. if statement*), petlje (*eng. loop*), naredbe (*eng. statement*) i znakovne nizove (*eng. string*). Kod znakovnih nizova dati ćemo primjere kreiranja, konkatenacije, određivanja duljine i prepoznavanja praznih znakovnih nizova te ćemo ostale bitne funkcije vezane uz znakovne nizove navesti i opisati. Najveći dio četvrtog poglavlja posvetit ćemo klasama. Opisat ćemo njihovu sintaksu i varijable članice koje ćemo podijeliti u dvije grupe: Stored properties i Computed properties. Kod Stored propertiesa ćemo spomenuti i jednu podvrstu te grupe varijabla članica pod imenom Lazy stored properties. Nakon varijabla članica govorit ćemo o metodama i nasljeđivanju klasa s čime završavamo četvrto poglavlje.

Tema četvrtog poglavlja je iphone aplikacija pod nazivom “Telefonski imenik” koju smo implementirali. Samu aplikaciju i njen izvorni kod možete pronaći na CD-u priloženim uz ovaj diplomski rad. U samoj aplikaciji smo koristili većinu prethodno spomenutih alata i funkcionalnosti koje nam nude Xcode i Swift. Također smo koristili i neke funkcionalnosti koje nismo spominjali u diplomskom radu pa smo ih u ovom poglavlju spomenuli, ukratko opisali i dali primjer njihove primjene. Ideja ovog poglavlja je pokazati konkretnu primjenu naučenog ali ne i detaljno objašnjavati sam kod aplikacije. Aplikacija “Telefonski imenik” sprema kontakte korisnika, omogućava kreirati grupe kontakta kao i favorite.

U zadnjem poglavlju diplomskog rada iznosimo zaključak. Iznijeti ćemo kritički osvrt na sam diplomski rad, spomenuti ćemo sadržaj koji nismo mogli detaljnije obraditi zbog same širine područja a smatramo ga bitnim. Prokomentirat ćemo dobre i loše strane tehnologije koja je korištena u diplomskom radu te dati osvrt na budućnost i mogući razvoj cijele industrije mobilnih aplikacija.

1. Poglavlje

Osnove

1.1. Razvojna okolina, alati, programski jezici i biblioteke

Krenut ćemo od osnovnih pojmova s kojima ćemo se susretati u ovom diplomskom radu i koji su neophodni za razvoj iPhone aplikacija. Objasnit ćemo za početak pojmove poput XCode, Objective-C i Swift, te Cocoa i Cocoa Touch te dati grube razlike između pojedinih.

Kako bi mogli razvijati iPhone aplikacije potrebne su nam sljedeće tri stavke:

Mac računalo s OS X operativnim sistemom
Xcode
iOS SDK (*eng. software development kit*)

U trenutku pisanja ovog diplomskog rada ne postoji rješenje (službeno izdano od strane Apple) koje bi se omogućio razvoj iPhone aplikacija na računalima koja nemaju OS X operativni sistem. Dakako, na Internetu možete pronaći niz prijedloga i rješenja kako pokrenuti OS X na računalima koja nisu Mac, no službene podrške od strane Applea nema.

Objective-C je objektno orijentirani programski jezik koji se koristi za razvoj programa i aplikacija za iOS i OS X. Osim navedene svrhe ali u puno manjem broju, Objective-C se koristi za razvoj na operativnom sustavu Linux. Objective-C je primarni programski jezik koji koristi Apple za razvoj programa na OS X. Za Objective-C možemo reći da je nastao miješanjem ideje programskog jezika Smalltalk, odnosno njegovom "messaging"-u te programskom jeziku C. Objective-C su primarno razvili Brad Cok i Tom Love u ranim 1980-im u njihovoj tvrtki Stepstone. Obojica su upoznati sa idejom Smalltalka u vrijeme kada su razvijali Objective-C što je utjecalo na njihov razvoj. U novijim verzijama Xcodea (3.1. i kasnijim) defaultni kompajler za Objective-C je Appleov LLVM kompajler. U prethodnim verzijama korišten je gcc (koji je instaliran i u kasnijim verzijama Xcodea i može biti korišten kao kompajler). Kako je Objective-C rađen na C-u, moguće je koristiti kod pisan u C-u unutar Objective-C klase.

Swift je objektno orijentiran programski jezik koji je kreirao Apple za razvoj programa i aplikacija na operativnim sistemima iOS i OS X. Predstavljen je na WWDC-u 2014. godine. Apple je razvio Swift kako bi poboljšao određene dijelove Objective-C. Apple i dalje omogućava razvijačima aplikacija da koriste već postojeće ravojne okvire *(eng. framework)* Cocoa i Cocao Touch te da koriste veliki dio Objective-C unutar Swifta. Glavna mana Objective-C koja je otklonjena Swiftom je otpornost samog programa na pogreške

u kodu, odnosno sigurnost te sažetost. Dolaskom Swifta dobili smo puno intuitivniju i sažetiju sintaksu. Swift kompajlira LLVM kompajler koji dolazi s Xcode 6 te koristi Objective-C *runtime*. Stoga, unutar jednog programa možemo koristiti kod pisan u C-u, Objective-Cu, C++ i Swiftu. Na WWDC, 08.06.2015. objavljeno je da će Swift postati open source te da će podržavati iOS, OS X i Linux.

Sintaksa Swifta je dizajnirana kako bi se izbjegle pogreške u kodu od strane samih programera. Ako dođe do sintaksne greške, greška će unutar editora Xcodea biti lakša za uočiti. Također, programer je primoran pisati puno manje koda za razliku od Objective-C. Primjera radi, u Swiftu ne moramo pisati ";" na kraju svake linije koda. Otklonu grešaka puno pridonosi playground koji je uveden sa Xcodom 6, no o njima ćemo detaljnije pisati u sljedećim poglavljima.

Kako bi dobili uvid koliko je Swift pregledniji i intuitivniji od Objective-C, pogledajmo niže navedeni kod.

Objective-C

```
//Non-Mutable Dictionary
    NSDictionary * myFixedDictionary = @{@"key1":@"This is value1",@"key2":@"This is value2"};

    // Mutable Dictionary
    NSMutableDictionary * myFlexibleDictionary = [[NSMutableDictionary alloc]init];
    //Set Object using Old Syntax
    [myFlexibleDictionary setObject:@"This is value1" forKey:@"key1"];
    //Set Object using New Syntax
    [myFlexibleDictionary setObject:@"This is value2" forKey:@"key2"];

    NSLog(@"myFixedDictionary: %@",myFixedDictionary);
    NSLog(@"myFlexibleDictionary: %@",myFlexibleDictionary);

    //Non-Mutable Array
    NSArray * myFixedArray = [[NSArray alloc]initWithObjects:@"Object1", @"Object2",nil];

    //Mutable Array
    NSMutableArray * myFlexibleArray = [[NSMutableArray alloc]init];
    //Add Object using Old Syntax
    [myFlexibleArray addObject:@"Object1"];
    //Add Object using New Syntax
    myFlexibleArray[1] = @"Object2";

    NSLog(@"myFixedArray: %@",myFixedArray);
    NSLog(@"myFlexibleArray: %@",myFlexibleArray);
```

Swift

```
//Constant Dictionary (Almost similar with Non-Mutable Dictionary)

    let myFixedDictionary = ["key1":"This is the value1","key2":"This is value2"]

    //Variable Dictionary (Almost similar with Mutable Dictionary)
    var myFlexibleDictionary = [String : String]()
    myFlexibleDictionary["key1"] = "This is the value1"
    myFlexibleDictionary["key2"] = "This is the value2"

    println("myFixedDictionary: \(myFixedDictionary)")
    println("myFlexibleDictionary: \(myFlexibleDictionary)")

    //Constant Array (Almost similar with Non-Mutable Array)
    let myFixedArray: [String] = ["Object1", "Object1"]

    //Variable Array (Almost similar with Mutable Array)
    var myFlexibleArray = [String]()
    myFlexibleArray.append("Object1")
    myFlexibleArray.append("Object2")

    println("myFixedArray: \(myFixedArray)")
    println("myFlexibleArray: \(myFlexibleArray)")
```

Xcode je razvojna okolina (*eng. integrated development environment - IDE*) koju koristimo za razvoj iPhone aplikacija. Xcode sadrži source editor, kompajler, emulator, razvojne okvire i još puno elemenata koja će nam olakšati razvoj iPhone aplikacija. U principu, od alata za razvoj potreban nam je samo XCode (pod pretpostavkom da grafičke elemente imamo već napravljene u nekom od programa kao Photoshop i sl.). Detaljnije o sučelju Xcodea ćemo govoriti u sljedećim poglavljima.

Možemo koristiti neki drugi IDE za razvoj, no kako Apple ima striktna pravila kojima regulira strukturu aplikacije koje mogu biti objavljene na App Storeu, preporučljivo je koristiti Xcode jer sam automatski otklanja potencijalne neregularnosti koje bi mogle prouzročiti probleme. Xcode je besplatan te se može skinuti sa App Storea.

Kao bitnu stavku cijelog razvoja moramo spomenuti razvojne okvire s kojima ćemo se često susretati a to su Cocoa i Cocoa Touch. Cocoa je razvojni okvir za izradu korisničkih sučelja za aplikacije i programe na operativnom sistemu OS X, dok je Cocoa Touch UI razvojni okvir za razvoj aplikacija na iOS-u (za iPhone, iPod Touch i iPad). Cocoa Touch je na Mac OS X Cocoa API i primarno je pisana u Objective-Cu. Cocoa i Cocoa Touch slijede MVC model razvoja softvera. Neke od glavnih mogućnosti koje nam pruža Cocoa Touch je animacija, multitasking i prepoznavanje pokreta.

Objasnimo ukratko razliku između iOS i OS X operativnog sustava. OS X je operativni sistem koji se koristi na Appleovim desktop računalima i prijenosnim računalima kao što su iMac, Mac mini, MacBook Pro, MacBook Air itd. iOS je operativni sistem koji se koristi na Apple-ovim prijenosnim uređajima kao što su iPod touch, iPad, iPhone.

Za kraj ćemo spomenuti App Store koji je centralna platforma za distribuciju i prodaju mobilnih aplikacija za iOS koji je razvijen od strane Applea. Na jednu stranu App Store omogućava razvijačima aplikacija upload svojih aplikacija te prodaju istih krajnjim korisnicima. Vlasnik aplikacije koji je uploadao aplikaciju na App Store može pružiti besplatan download svoje aplikacije ili je može prodavati po određenoj cijeni krajnjim korisnicima.

Na drugu stranu App Store omogućava krajnjim korisnicima pretragu objavljenih aplikacija te im pruža mogućnost kupovine ili besplatnog downloada. U slučaju kupovine, transakcija se odvija posredstvom Apple-a gdje se prodavaču uzima 30% provizije. App Store je otvoren 10.06.2008. kao update iTunes-a. Početkom 2012. na App Store-u je bilo preko 1 100 000 aplikacija. Kako bi ravijači aplikacija mogli objavljivati svoje aplikacije na App Store-u moraju biti članovi Apple Developer Programa koji nudi niz alata za pomoć pri prodaji aplikacija kao što je npr. App Analytics koji služi za prikaz svih statističkih podataka vezanih uz samu aplikaciju. Godišnja naknada za iOS Developer Program iznosi 99$, dok godišnja naknada za iOS Developer Enterprise Program iznosi 299$, dok je iOS Developer University Program besplatan.

1.2. Xcode - Korisničko sučelje

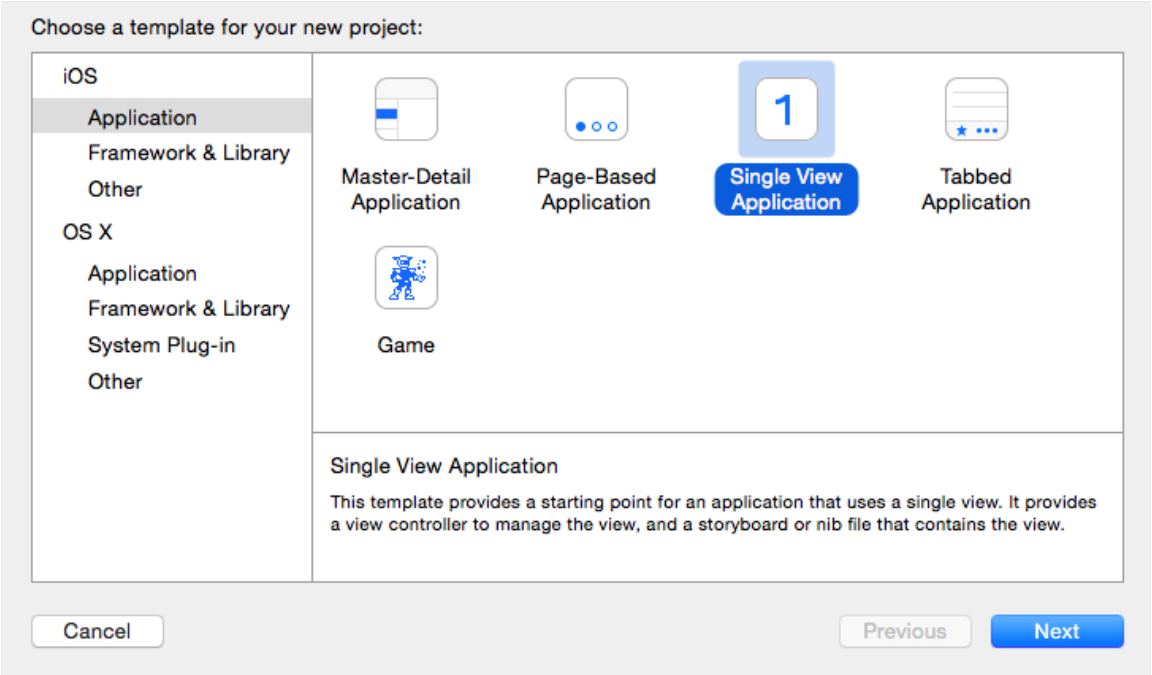
Kao što smo rekli u prethodnom poglavlju, Xcode je program koji nam služi kao razvojna okolina za iPhone aplikacije. Kako je Xcode nezaobilazna stavka cijelog razvoja, u ovom poglavlju prezentirat ćemo korisničko sučelje i sve njegove glavne funkcionalnosti.

Xcode možete skinuti unutar App Storea. U trenutku pisanja ovog diplomskog rada najnovija verzija Xcodea je bila 6.1.1. i unutar nje su rađeni svi projekti iz ovog diplomskog rada.

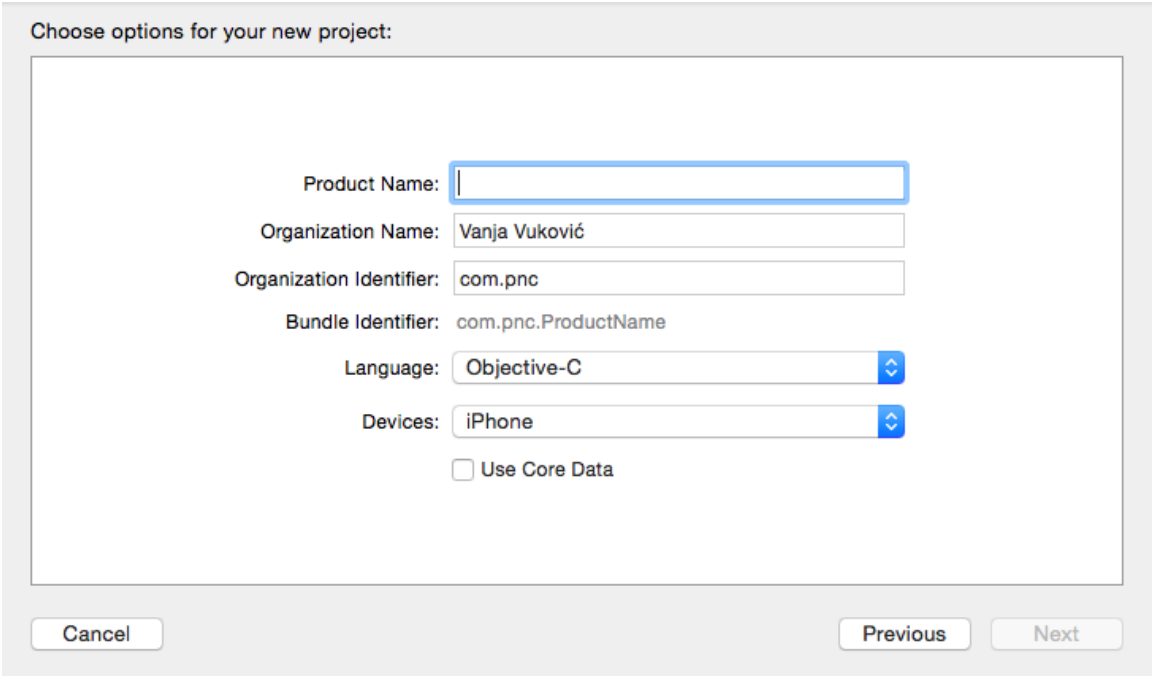
Prilikom pokretanja Xcodea otvara nam se prozor za brzo otvaranje postojećeg projekta ili kreiranje novog.

Na slici prozora nude nam se tri opcije. Prvu opciju opisat ćemo u sljedećem poglavlju koja je jedna od velikih novosti koja je došla sa Xcodeovom verzijom 6. Ako želimo kreirati novi projekt otvara nam se Xcode s prozorom koji nam nudi predloške (*eng. template*) za našu aplikaciju. Treća opcija nam nudi otvaranje već postojećeg projekta koje možemo pronaći u desnom panelu prozora.

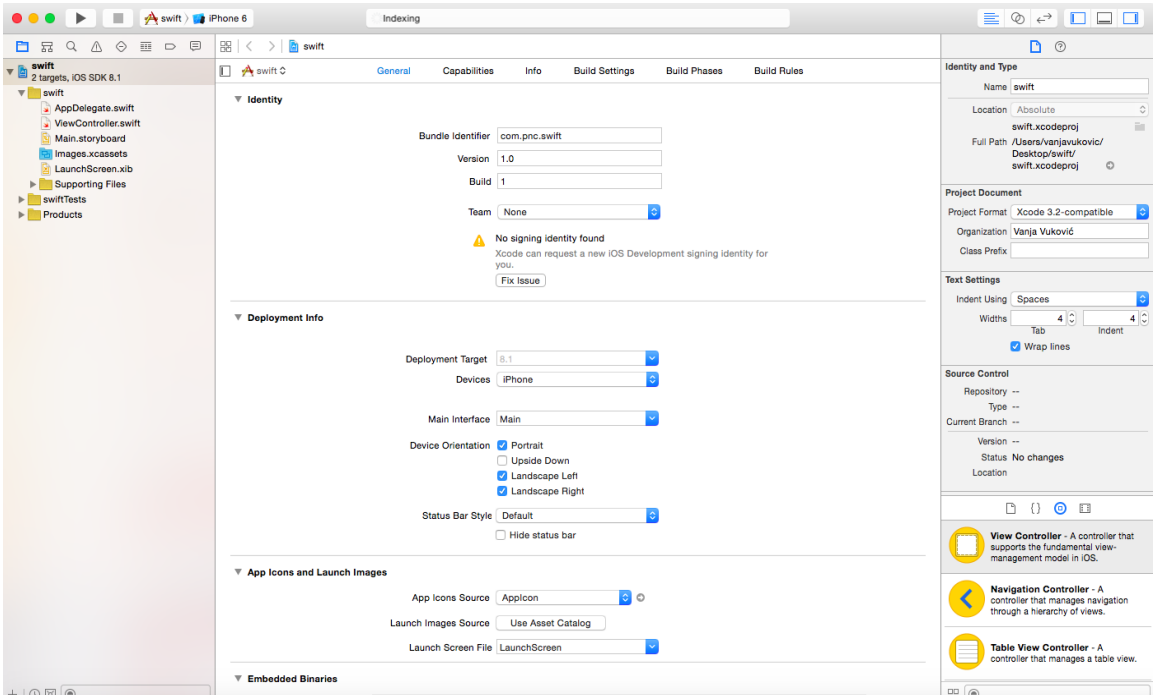
Na sljedećoj slici vidimo prozor koji nam se otvara nakon što odaberemo kreiranje novog projekta. U lijevom panelu imamo listu projekata kakve možemo kreirati. Oni su podijeljeni u dvije grupe, ovisno o operativnom sustavu za koji razvijamo projekt. U ovom trenu nas zanima tab pod imenom Aplication.



U tabu iOS aplikacija nudi nam se pet predložaka aplikacija. Prva četiri predloška aplikacija razlikuju se po korisničkom sučelju. Naravno, svako korisničko sučelje kasnije možemo u potpunosti mijenjati. Predlošci nam daju početan kostur korisničkog sučelja aplikacije. Isti takvo korisničko sučelje možemo sami napraviti no puno nam je jednostavnije urediti već postojeće. Peti predložak je namijenjen razvoju igara, posebice 3D igara, no ovim predloškom se nećemo detaljnije baviti u ovom diplomskom radu.



Nakon odabira predloška i kreiranja projekta otvara nam se korisničko sučelje Xcodea. Kao što možemo vidjeti na slici, korisničko sučelje se sastoji od 4 glavne komponente (na slici označeno brojevima od 1-4.) koje možemo prikazivati ili sakrivati.

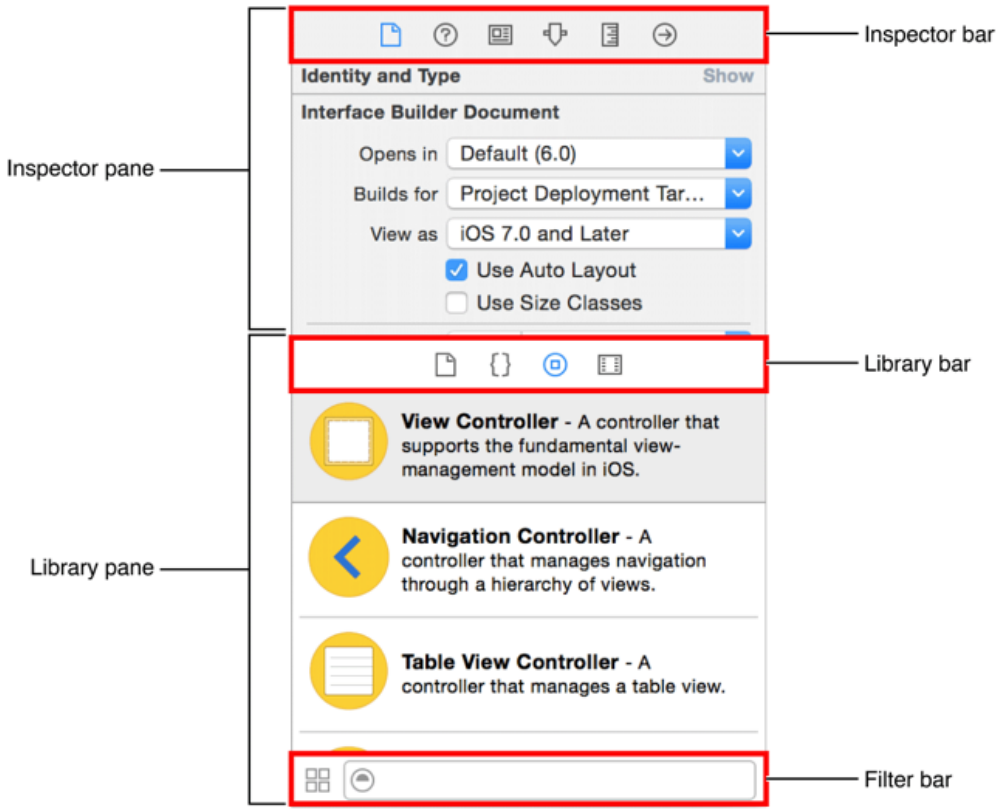


Pod brojem 1 nalazi se Main Window Toolbar koji je poprilično jednostavan. Unutar Toolbara možemo pokretati ili zaustavljati aplikaciju klikom na Run gumb, mijenjati uređaj na kojem ćemo simulirati rad aplikacije te mijenjati izgled samog korisničkog sučelja programa Xcode (skrivati ili prikazivati Navigator, Debug area i Utilities).

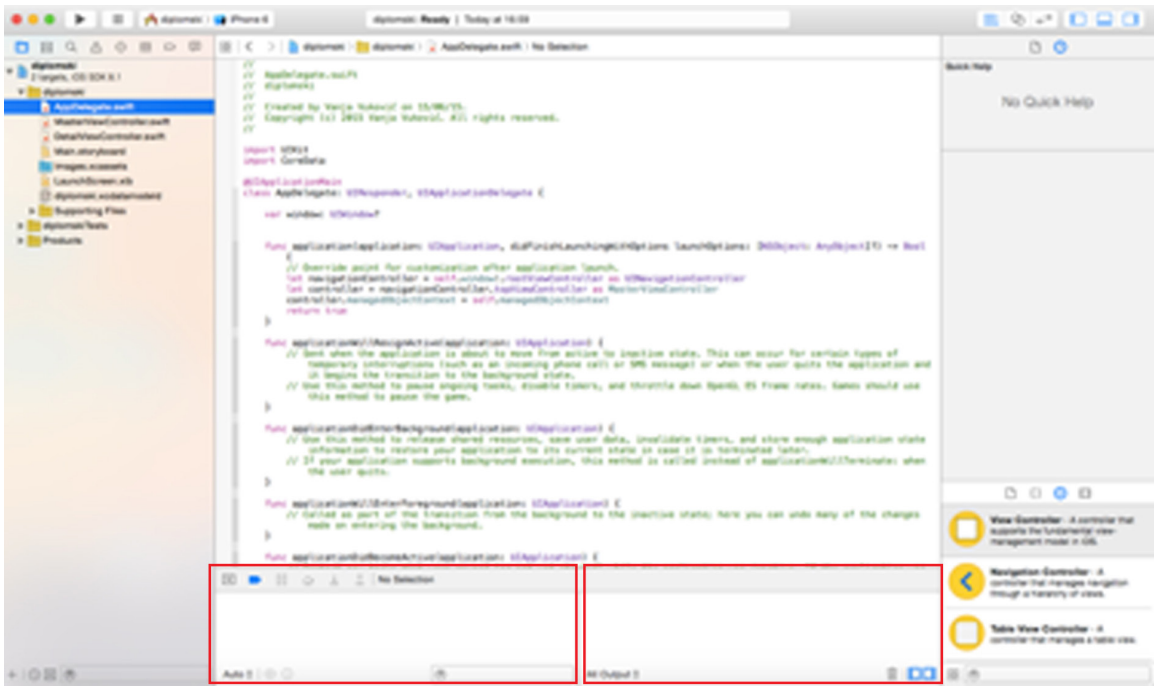
Pod brojem 2 nalazi se Navigator Area. Unutar Navigator Area možemo pronaći sve datoteke koje se nalaze unutar našeg projekta. Detaljnije o vrstama datoteka i njihovom rasporedu unutar projekta pričat ćemo detaljnije u sljedećim poglavljima diplomskog rada. Unutar Navigatora imamo 8 različitih pogeda koje prikazujemo odabirom jedne od ikona u tabovima na vrhu Navigatora a to su redom: Project Navigator, Symbol Navigator, Find Navigator, Issue Navigator, Test Navigator, Debug Navigator, Breakpoint Navigator, Report Navigator.

Pod brojem 3 nalazi se Editor area koji prikazuje sadržaj datoteke koja je odabrana u Navigatoru. Unutar glavnog pogleda se nalazi i editor unutar kojeg uređujemo sadržaj odabranoe datoteke.

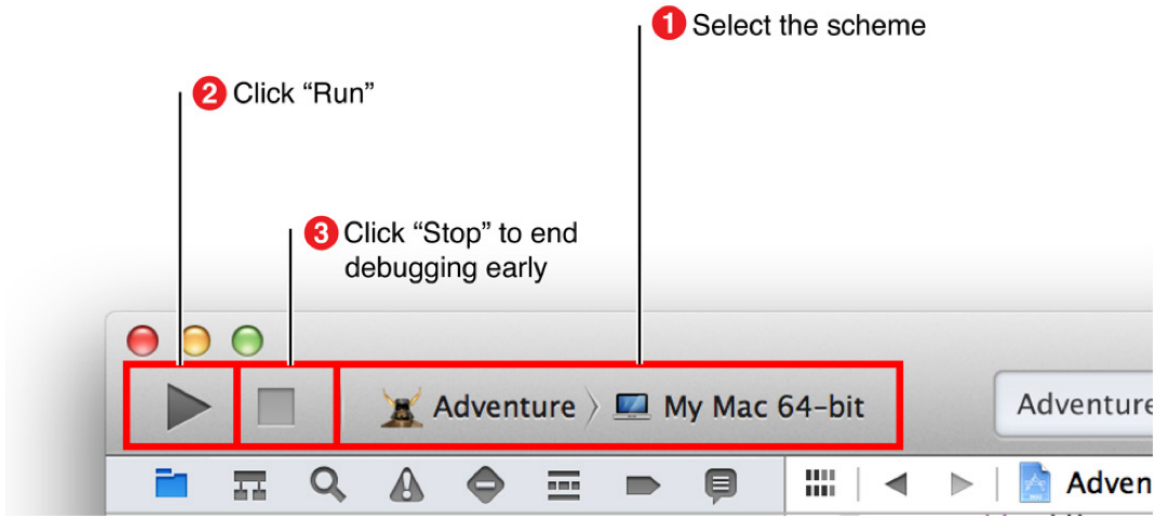
Pod brojem 4 nalazi se Utilities Area koji se sastoji od dva dijela: Inspector pane i Library pane. Unutar Inspector panea možemo pronaći inspectore a unutar Library panea možemo pronaći resource library koje možemo koristiti unutar svog projekta. Objekte iz biblioteke *(eng. librarya)* koji su nam potrebni kreiramo *drag and dropom* na Editor Area.



Po defaultnim postavkama, Debug area je skriven te se prikazuje klikom na View -> Debug area -> Show debug area koji će nam služiti kod debugiranja aplikacija.



Kako bi pokrenuli i debugirali aplikaciju možemo se koristiti “brzim” opcijama u toolbaru. Pogledajmo sliku:



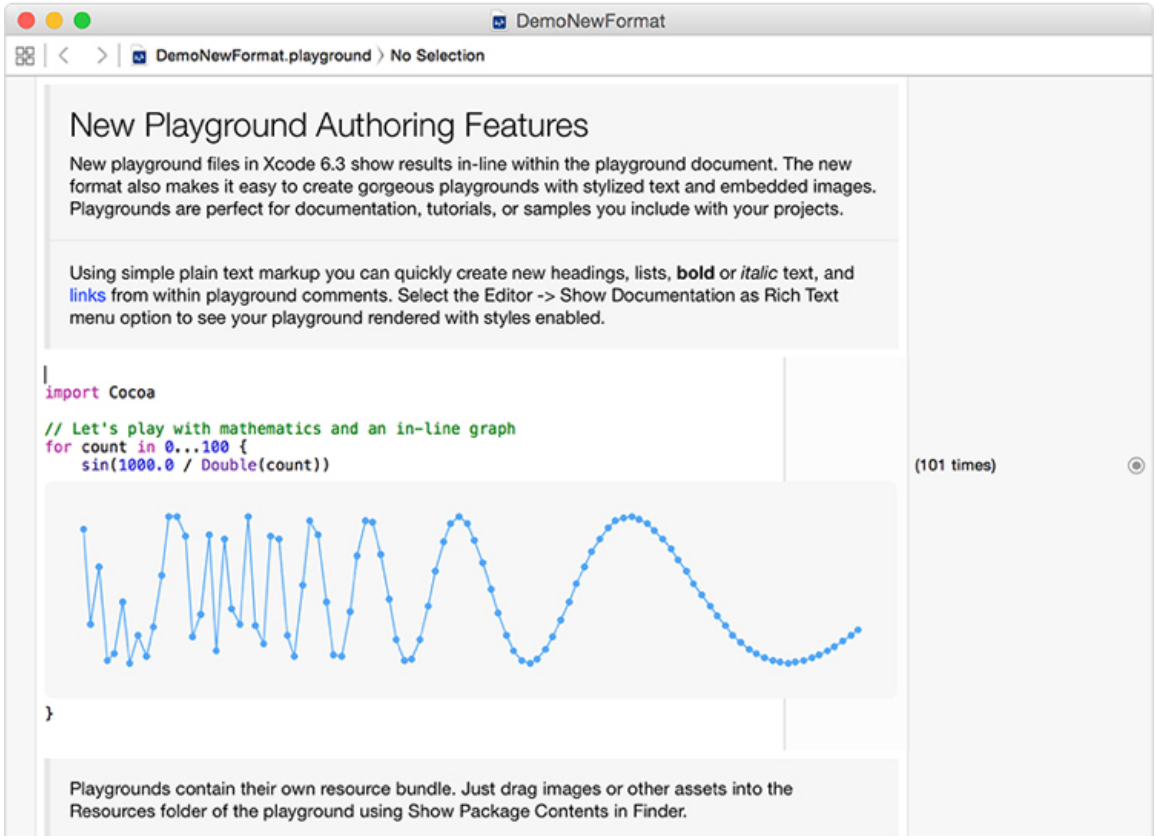
Klikom na (1) odabiremo uređaj na kojem ćemo simulirati rad aplikacije. Prije samog uređaja možemo vidjeti ime našeg projekta kojeg ćemo pokrenuti. Klikom na Run button pokrenuti ćemo aplikaciju na odabranom uređaju. Kada želimo zatvoriti prozor koji nam se otvorio klikom na Run button unutar kojeg je pokrenuta aplikacija trebamo samo kliknuti na gumb pod (3) - Stop button. Ova opcija nam omogućava testiranje aplikacije na različitim uređajima bez da ih fizički posjedujemo. Kako Apple izdaje nove uređaje, simulatori za njih su dostupni u updateima Xcodea.

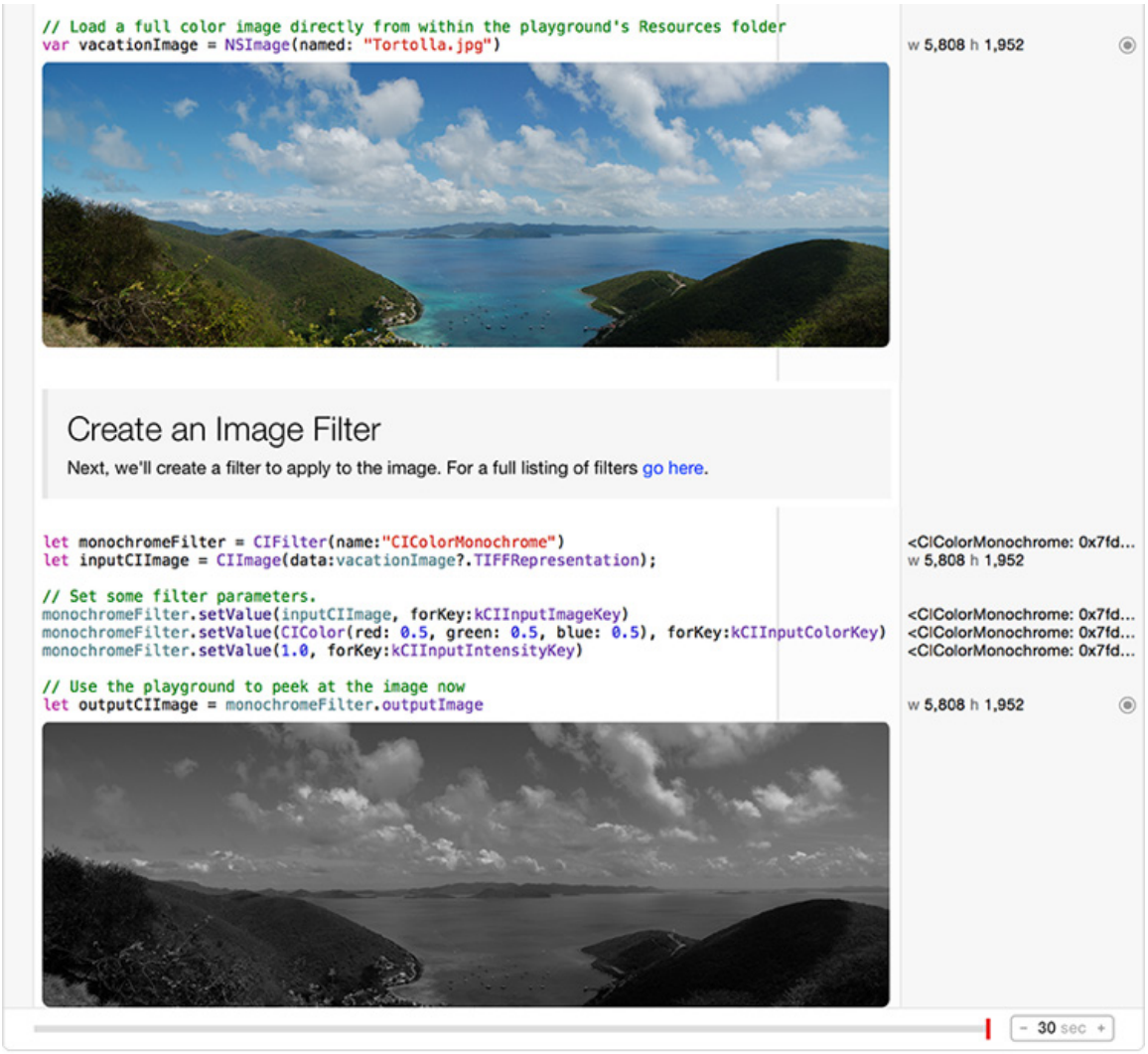
Prozor koji nam se otvorio pokretanjem aplikacije možemo vidjeti na sljedećoj slici. Prozor naravno mijenja izgled ovisno o odabranom uređaju.

1.3. Playground

Playground je novi alat koji je došao s verzijom Xcodea 6. Playground nam olakšava pisanje aplikacija u Swiftu. Kada pišemo kod u Swiftu unutar playgrounda, odmah nam se prikazuje rezultat izvršavanja tog koda. Ukoliko naš kod sadrži petlje, možemo pregledati vrijednosti varijabli kako su se mijenjale u svakom koraku petlje. Pregled varijabli unutar grafa, provjeravanje svakog koraka iscrtavajući pogled ili pregledavanje animirane SpriteKit scene samo su neke od mogućnosti koje nam nudi playground. Ovakav način pisanja koda smanjuje mogućnost grešaka u kodu te nam olakšava samo testiranje. Nakon što napišemo kod koji zadovoljava naše potrebe i utvrdimo da ne sadrži greške jednostavno kopiramo napisani kod u postojeći projekt. Ukoliko pišemo kod direktno unutar samog projekta, moramo pokrenuti aplikaciju kako bi pregledali rezultat napisanog koda što nam može uzeti podosta vremena s obzirom na to da kreiranje simulatora zahtjeva određeno vrijeme.

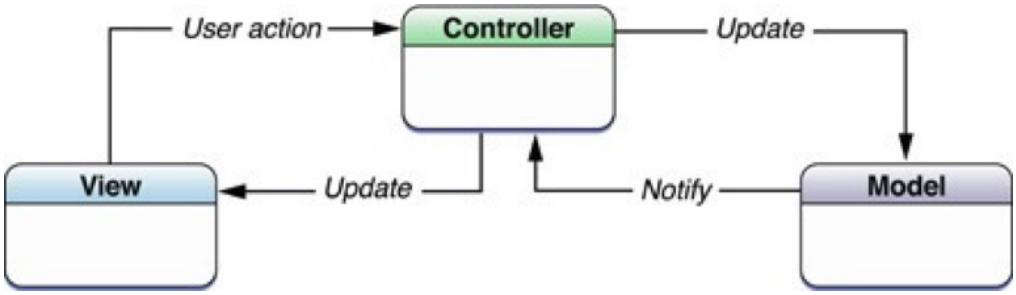
Opišimo neke od konkretnih situacija koje nam olakšava playground. Zamislimo da razvijamo novi algoritam koji ćemo koristiti unutar naše aplikacije. Kada bi algoritam pisali direktno unutar naše aplikacije postoji mogućnost da greška u samoj aplikaciji proizvodi grešku u kodu te nam je grešku kao takvu teže otkriti. Ne znamo da li je problem u aplikaciji ili u algoritmu. Na prethodno opisani način razvijemo algoritam unutar playgrounda, testiramo ga te zatim kod kopiramo unutar aplikacije. Unutar playgorunda možemo isprobavati nove API-je ili novu Swift sintaksu.





1.4. MVC model

U ovom poglavlju pričat ćemo o arhitekturi iPhone aplikacija. Model na kojem se zasniva arhitektura iPhone aplikacija je Model-view-controller (u daljnjem tekstu koristit ćemo skraćenicu MVC). Glavna ideja koju predstavlja MVC se zasniva na podjeli aplikacije u tri dijela od kojih je svaki zadužen za jedan dio prezentacije informacije korisniku. Kao što i samo ime modela govori, komponente od kojih se aplikacija treba sastojati su: Model, View i Controller. Točnije govoreći, MVC objektima dodjeljuje ulogu jednog od tri prethodno nabrojane komponente. MVC model osim što propisuje kakve uloge objekti poprimaju unutar aplikacije, MVC definira i način komunikacije između njih. Prednost ovako dizajnirane aplikacije je jasno razdvojen kod, kod prikaza sadržaja korisniku i kod koji nam služi za dohvaćanje i baratanje podacima. Također, tako razdvojen kod nam omogućava lakši pronalazak grešaka.



Opišimo svaki od dijelova MVC-a. Glavna komponenta MVC-a je model. Model opisuje podatke specifične za aplikaciju i definira logiku i način manipulacije tih podataka. Model opisuje ponašanje vezano uz određenu problemsku domenu. Iz tog razloga određeni model može biti ponovno upotrebljen u sličnoj problemskoj domeni. Model može imati jednu ili više poveznica s drugim modelima. Podaci koji opisuju trenutno stanje aplikacije a nalaze se u modelu (unatoč tome što su spremljeni u bazu podataka) obično moraju ostati u modelu nakon što su učitani u aplikaciju. Model ne bi trebao biti direktno spojen s Viewom, odnosno ne bi trebali komunicirati direktno već posredstvom Controllera. Radnje korisnika unutar Viewa koje mijenjaju ili kreiraju nove podatke, komuniciraju s modelom posredništvom Controllera. Ista stvar vrijedi i u suprotnoj situaciji kada model želi promijeniti prikaz podataka u Viewu. Model prvo obavijesti Controller koji zatim mijenja prikaz podataka u View.

View objekt je objekt koji unutar aplikacije prikazuje podatke korisnicima. View objekt reagira na korisničke akcije. Glavni zadatak View objekta je prikaz podataka korisniku (podataka iz model objekta putem controller objekta). Drugi zadatak View objekta je uređivanje podataka od strane korisnika. Kada korisnik unosi podatke, *drag-and-dropa* objekte unutar aplikacije i sl., on to sve radi unutar View objekta. View objekt tada sve korisnikove radnje (odnosno izmjenu podataka) prenosi Controlleru. U praksi često ćemo koristiti slične tipove Viewa (npr. Master View).

Promjena podataka unutar View objekta se odvija putem controllera kojem je model proslijedio podatke. Kada korisnik unutar View objekta promijeni podatke, View prosljeđuje podatak controlleru koji ga prosljeđuje modelu.

Controller se nalazi "između" modela i viewa. On je "posrednik", tj. glavna uloga controllera je interpretacija korisničkih akcija koje su napravljene unutar Viewa i njihovo prosljeđivanje modelu. Kada se desi promjena podataka unutar modela, controller prosljeđuje nove podatke View objektu koji ih onda prikazuje. Controller može povezivati jedan ili više modela s jednim ili više view objekata.

Unutar jedne aplikacije možemo imati jedan Controller koji će komunicirati sa svim Viewovima i Modelima. No, unutar jedne aplikacije možemo imati i više MVC-a koji međusobno komuniciraju i uzajamno djeluju.

2. Poglavlje

Razvoj korisničkog sučelja

U ovom poglavlju ćemo pričati o samom razvoju aplikacije. U prvom dijelu ćemo govoriti o arhitekturi aplikacije baziranoj na prethodno opisanom MVC modelu, gdje se koji dio fizički nalazi unutar aplikacije, kako ih kreirati i kako koristiti predloške za određene objekte iz MVC-a. Bavit ćemo se razvojem grafičkog sučelja koje u principu ne uključuje pisanje koda u Swiftu. Skoro cijeli posao za nas odradi Xcode no zbog velikog opsega mogućnosti ukratko ćemo objasniti na što trebam obratiti pozornost.

2.1. Uvod

Jedna od prvih faza razvoja iphone aplikacije je dizajn korisničkog sučelja (*eng. user interface*).

Xcode nam pruža dvije opcije kreiranja elemenata i njihovih stilskih pravila. Prva opcija je ona standarda, pisanjem koda i definiranjem stilskih pravila u samom kodu. Druga opcija je kreiranje korisničkog sučelja aplikacije kroz korisničko sučelje Xcodea bez pisanja i jedne linije koda. Elemente kao i druga stilska pravila unosimo u forme koje same kreiraju kod potreban za implementaciju sučelja aplikacije. U sljedećim poglavljima razmatrat ćemo drugu opciju kreiranja korisničkog sučelja.

Definirati položaj i međusobni odnos elemenata unutar korisničkog sučelja možemo unutar samog koda. Pisanje suhoparnog koda u verziji Xcodea 6 je zamijenjena opcijom Auto layout.

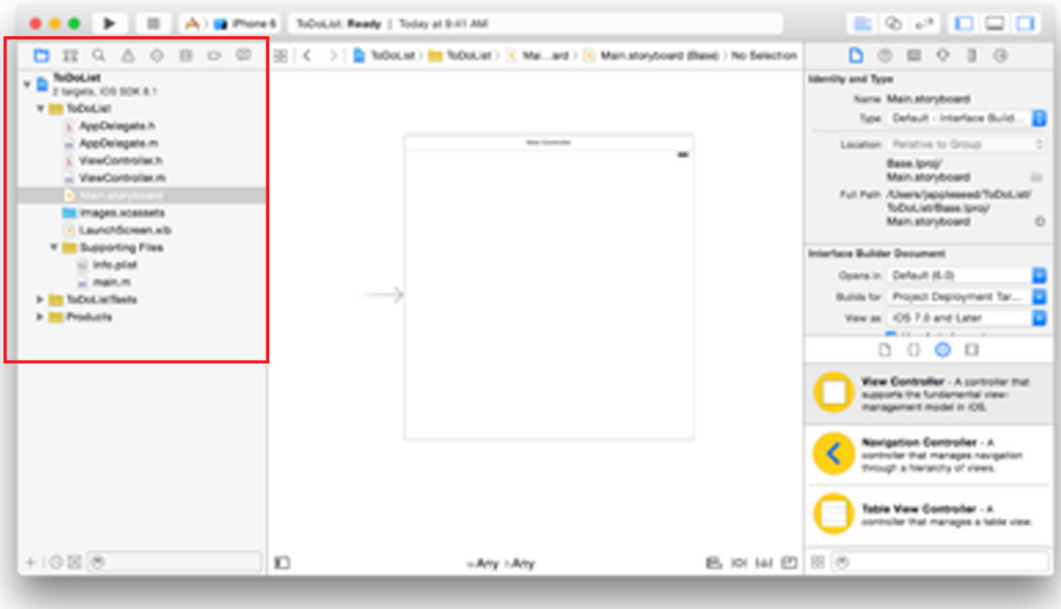
Auto layout nam pruža mogućnost da kroz korisničko sučelje Xcodea možemo postaviti niz parametara koji definiraju položaj elemenata unutar korisničkog sučelja kao što je razmak između određenih elemenata, margine i mnoge druge stavke dizajna.

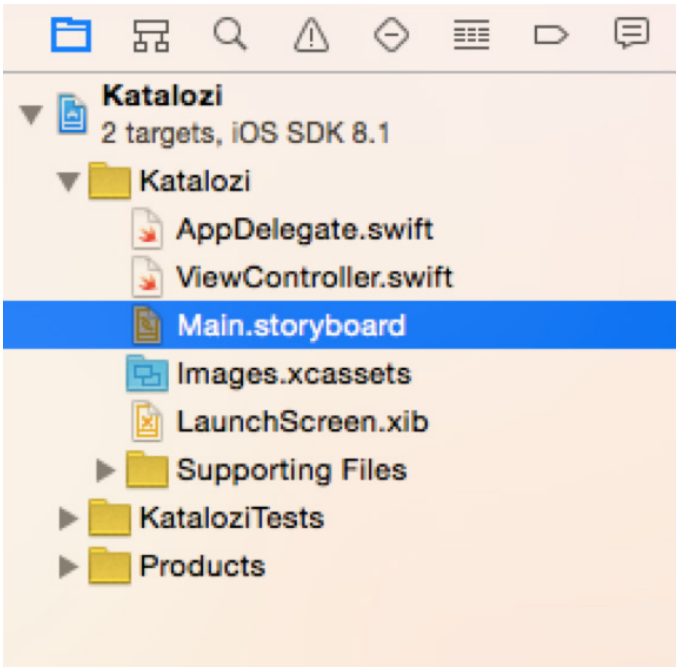
Kod dizajniranja korisničkog sučelja trebamo obratiti pozornost na par glavnih faktora koji će utjecati na dizajn samog sučelja. Trebamo razmisliti koja je glavna svrha aplikacije, tko su naši željeni korisnici te koji dijelovi aplikacije će biti najkorišteniji od strane korisnika. Glavni cilj aplikacije može biti npr. prodaja proizvoda (aplikacija ebay), pregledavanje slika drugih korisnika (Instagram) ili čitanje novinskih članaka (Dnevnik). Svaki nabrojani cilj aplikacije određuje korisničko sučelje na svoj način. Ukoliko aplikacija kao cilj ima prodaju proizvoda tada će korisničko sučelje aplikacije biti orijentirano na sam opis

proizvoda kao što su slike i opis proizvoda te košaricu i mogućnost kupovine te će tako ograničiti sučelje od korištenja elemenata koji nisu nužni za ostvarivanje samog cilja aplikacije. Analogno vrijedi i za ostale aplikacije. Ciljani korisnici također utječu na sam razvoj korisničkog sučelja. Ukoliko su ciljani korisnici starije životne dobi, imamo pretpostavku da se naši korisnici teže služe mobilnim aplikacijama te bi tada korisničko sučelje kao takvo trebalo biti jednostavno, sa malo elemenata koji ne smiju biti predetaljni. Prije dizajniranja korisničkog sučelja trebamo razmisliti koji će dijelovi aplikacije biti više korišteni od ostalih od strane korisnika. Dijelove koji će biti korišteniji od ostalih treba staviti u prvi plan i maksimalno pojednostaviti. Kao primjer možemo uzeti login sučelje te impressum. Pretpostavka je da će više korisnika koristiti login sučelje nego čitati impressum. Tada login sučelje treba staviti na vrh, odnosno na početan pogled aplikacije dok impressum može stajati na dnu ili se čak ni ne mora nalaziti na početnom pogledu aplikacije.

2.2. Storyboard

Vratimo se na tehnički dio aplikacije. Za početak spomenimo jedan od nezaobilaznih dijelova Xcodea koji nam koriste za izradu korisničkog sučelja aplikacije a to je storyboard. Pogledajmo datoteke koje su kreirane od strane Xcodea nakon što smo kreirali novi projekt. Sve datoteke koje se nalaze unutar projekta vidimo u Project navigatoru.





Storyboard je vizualna prezentacija korisničkog sučelja naše aplikacije unutar koje vidimo sve zaslone (*eng. screen*), sadržaj pojedinog zaslona, prijelaze između njih. Klikom na datoteku Main.storyboard dolazimo do dijela aplikacije unutar kojeg možemo dizajnirati i implementirati korisničko sučelje unutar grafičkog okruženja. Pomoću *drag and dropa* te unošenjem parametara kroz grafičko sučelje dizajniramo korisničko sučelje aplikacije. U svakom trenutku vidimo krajnji rezultat, odnosno trenutni izgled našeg korisničkog sučelja. Odmah vidimo da li smo postigli željeni rezultat te što nam ne izgleda kako smo zamislili. Unutar storyboarda definiramo i poveznice među samim zaslonima.

Storyboard se otvara unutar Editor area. Pozadinu Storyboarda zovemo Canvas unutar kojeg smještamo elemente korisničkog sučelja. Kao što možemo vidjeti s prethodne slike, storyboard sadrži jednu scenu (*eng. scene*), tj. koja ne sadrži ni jedan element. Strelica s lijeve strane koja pokazuje na početnu scenu označava scenu koja će se prva prikazati korisniku nakon otvaranja aplikacije. Tu strelicu nazivamo storyboard entry point. U ovom trenu scena sadrži jedan pogled. O pogledima ćemo detaljno govoriti kasnije.

Možemo primijetiti da zasloni unutar Stroyboarda ne odgovaraju dimenziji ni jednog od iPhonea. Storyboard je zapravo generalizirana prezentacija korisničkog sučelja na bilo kojem Appleovom uređaju. Storyboard koristimo za izradu prilagodljivog (*eng. adaptive*) korisničkog sučelja koje ne ovisi ni o jednim dimenzijama bilo kojeg uređaja već se automatski njemu prilagođava. Prednost ove vrste korisničkog sučelja je ta što razvijač aplikacija ne radi posebno korisničko sučelje za svaki od uređaja već radi jedno korisničko sučelje neovisno o broju različitih uređaja. Kasnije ćemo govoriti o Auto layout opciji koja nam uvelike pomaže kod dizajniranja prilagodljivih korisničkih sučelja. Ukoliko se

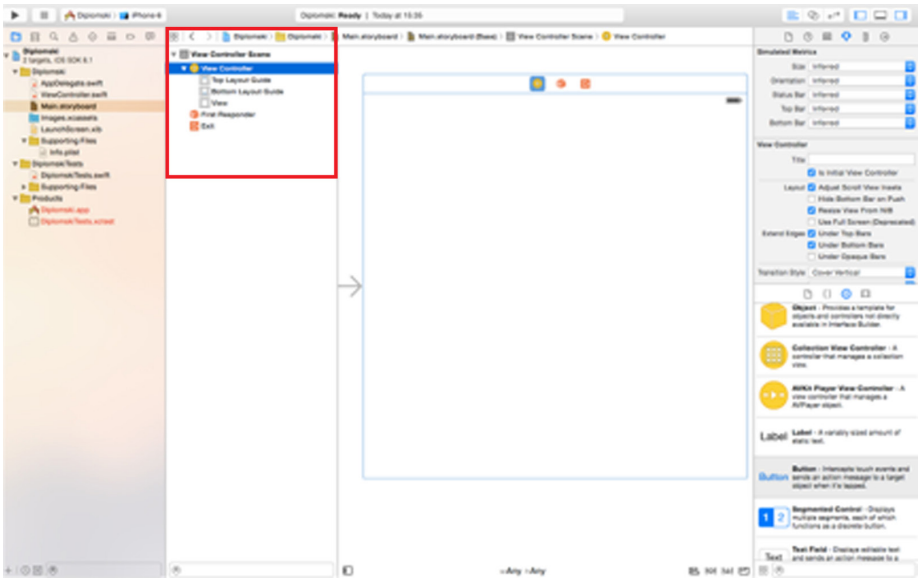
radi o kompleksnom sučelju tada postoji opcija dizajniranja sučelja za svaku od potrebnih dimenzija uređaja.

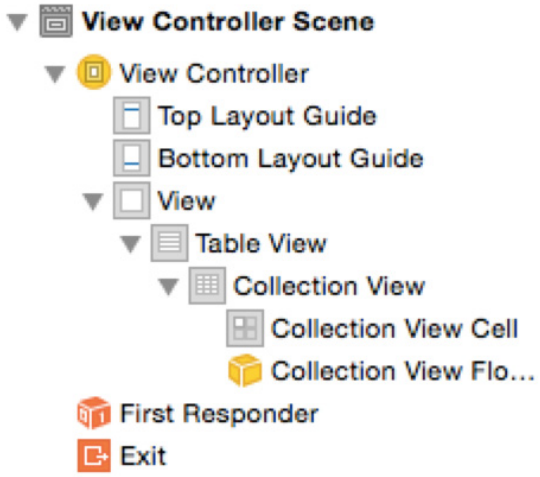
2.3. Pogled

Opišimo ukratko glavni element koji koristimo u izradi korisničkih sučelja a to je pogled. Pogled je zapravo objekt iz biblioteke objekata. Pogled prikazuje sadržaj korisnicima. pogled može sadržavati druge objekte. Pogled možemo zamisliti kao blokove pomoću kojih grupiramo i prikazujemo ostale objekte unutar korisničkog sučelja aplikacije. Pogled može sadržavati i druge poglede. Iz tog razloga postoji hijerarhija pogleda koja definira odnos pojedinih vrsta pogleda s obzirom na ostale. Pogled koji sadrži druge poglede nazivamo super-pogled (*eng. superview*), dok poglede koji su sadržani unutar nekog drugog pogleda nazivamo pod-pogled (*eng. subview*). Pojedini pogled može sadržavati više pogleda, odnosno imati više pod-pogleda, dok može imati samo jedan super-pogled.

Na vrhu hijerarhije pogleda nalazi se objekt prozor (*eng. window*) koji je prezentiran instancom klase UIWindow. Prozor je objekt koji sam ne prikazuje sadržaj korisničkog sučelja. U prozor dodajemo objekte tipa pogled unutar kojih onda dodajemo sadržaj za koji želimo da se prikazuje unutar korisničkog sučelja.

Kako bi prikazali određeni sadržaj prvo moramo dodati pogled u prozorovu hijerarhiju pogleda. To možemo napraviti pisanjem koda. Ukoliko se služimo storyboardom, taj dio posla je za nas odrađen automatski (pod pretpostavkom da smo kod kreiranja cijelog projekta odabrali neki od predefeiniranih predložaka aplikacije. Popis objekata i njihov poredak u hijerarhiji vidimo iz sljedećeg panela:

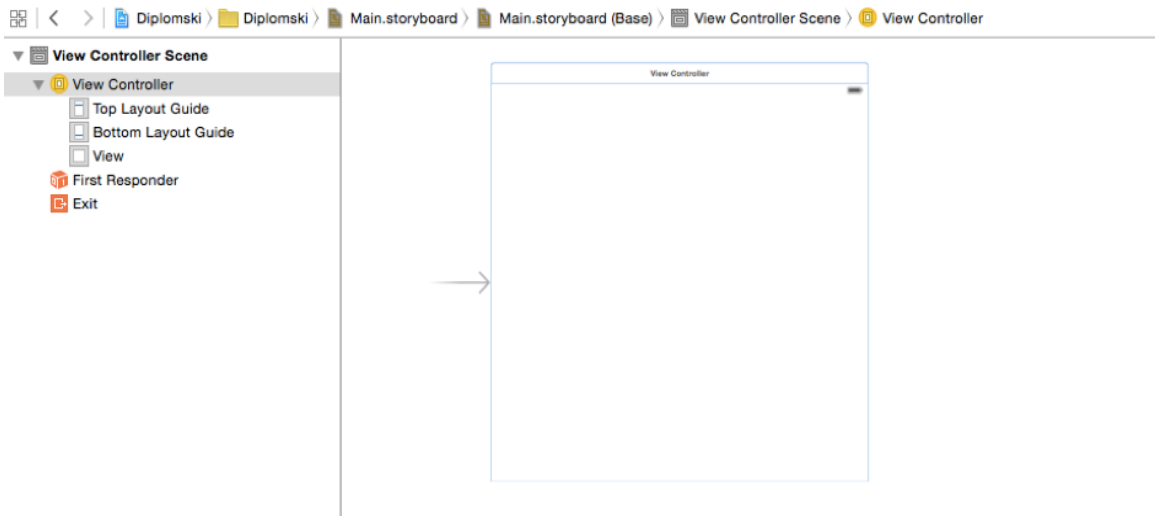




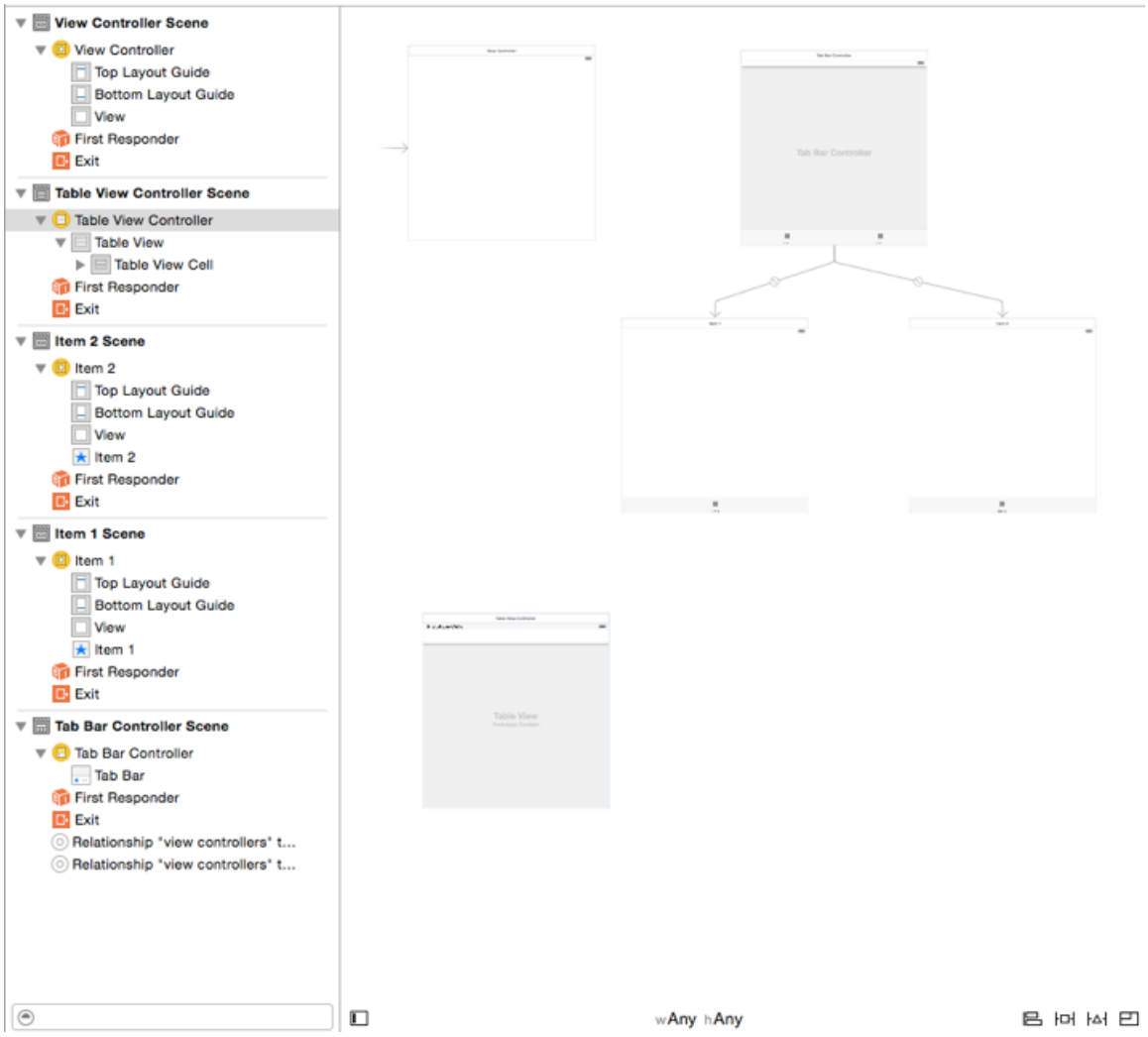
Kao što možemo vidjeti iz popis svih objekata iz hijerarhije, objekt pogled se nalazi unutar hijerarhije. Iz sljedeće slike možemo grafički vidjeti odnos super-pogleda i pod-pogleda. Odnosno situaciju kada pogled sadrži druge poglede.

2.4. Učitavanje grafičkog sučelja

Sada ćemo objasniti kako što se zapravo dešava u pozadini s korisničkim sučeljem pri pokretanju aplikacije, odnosno kako se ono učitava i prikazuje. Kada korisnik pokrene aplikaciju, učitava se storyboard, kreiraju se instance View controllera klasa koje se koriste u aplikaciji, učitava se sadržaj sadržajnog View controllera za svaki View controller i za kraj se dodaju pogledi u svaki od pripadajućih View Controllera zajedno sa svim pripadajućim pogledima. Za sada se



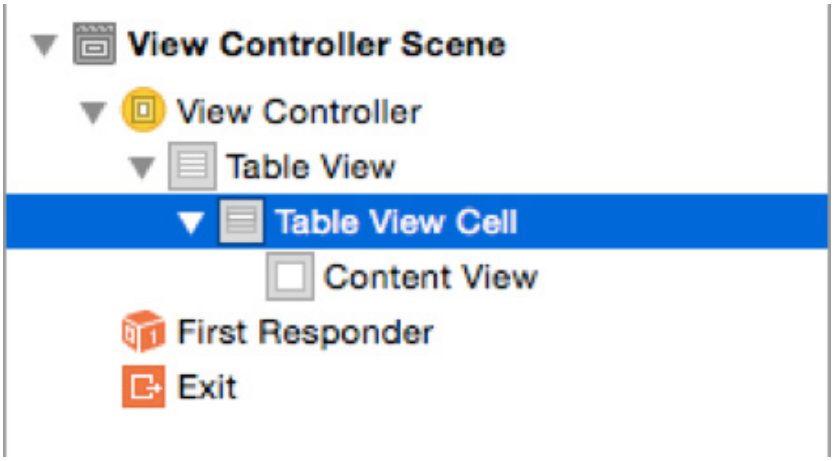
nismo detaljno pozabavili View controllerima već smo ih samo spomenuli i opisali glavnu zadaću u dijelu s MVC paradigmom. Primijetimo da aplikacija može imati više View controllera ovisno o broju pogleda koji se nalaze unutar aplikacije. Ukoliko kod kreiranja projekta kao predložak odaberemo Single View Application, tada će naša aplikacija imati samo jedan View controller. Na sljedećoj slici vidimo situaciju kada aplikacija ima više View controllera.



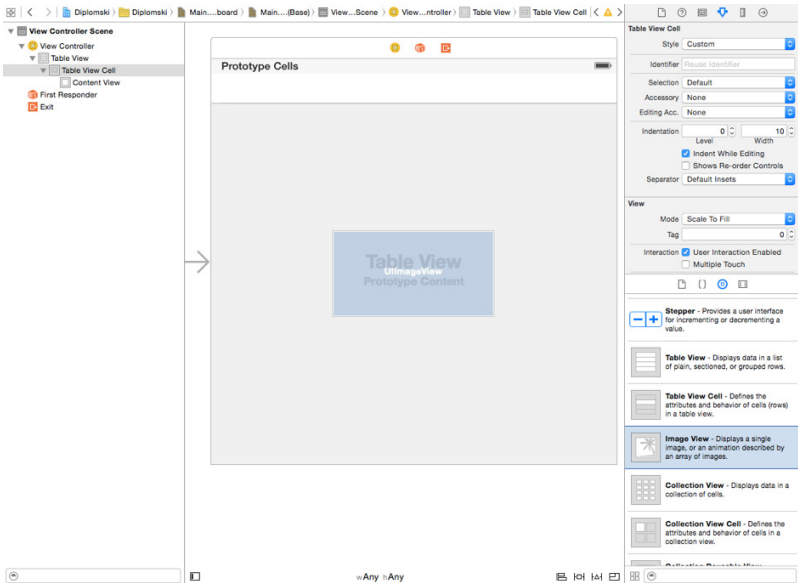
2.5. Dodavanje elemenata korisničkog sučelja

Ukoliko postoji potreba za dodavanjem dodatnih pogleda tada u donjem desnom kutu programa možemo pronaći listu svih predefiniranih objekata koje možemo koristiti. Među njima se nalaze i pogledi. Jednostavnim odabirom objekta i *drag-and-dropom* na Storyboard kreiramo novi objekt unutar našeg korisničkog sučelja. Analogno vrijedi i za elemente koji nisu tipa pogled.

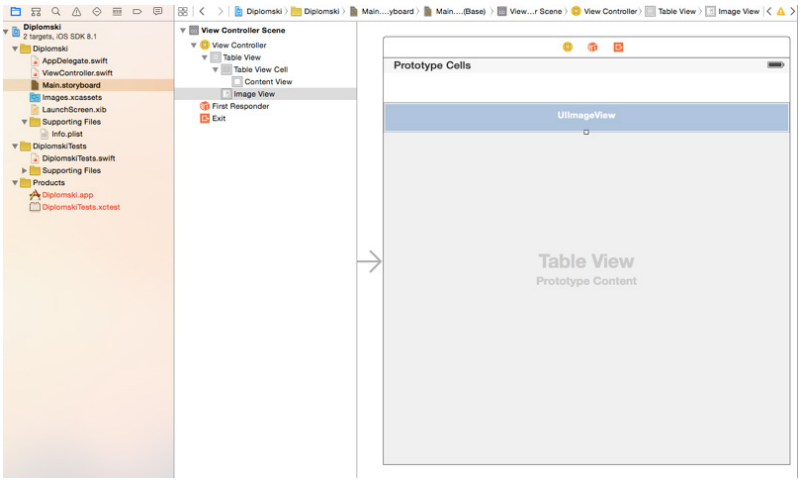
Kod dodavanja novih objekata trebamo obratiti pozornost na hijerarhiju pogleda, odnosno gdje ćemo unutar hijerarhije pogleda dodatni novi objekt. Kod pregleda svih objekata koji se nalaze unutar našeg prozora prvo odabiremo objekt za koji želimo da bude roditelj novog objekta i zatim *drag-and-dropamo* željeni objekt.



S gornje slike vidimo da imamo više pogleda unutar kojih možemo dodati novi objekt. U našem slučaju dodavat ćemo novi pogled za prikaz slike. Prvo smo označili Table view cell unutar kojeg želimo dodati Image view objekt. Sada u biblioteci objekata tražimo Image view i *drag-and-dropamo* ga na Storyboard.



S prethodne slike vidimo u donjem desnom kutu biblioteku objekata i Image view koji je označen. Image view *drag-and-dropamo* na Storyboard, gdje smo prethodno označili Table cell view koji će biti super-pogled za Image view. Nakon što smo ovo napravili vidimo promjenu u hijerarhiji pogleda, te promjenu izgleda samog storyboarda koji nam odmah grafički prikazuje novu izmjenu korisničkom sučelju aplikacije.



Analogni postupak vrijedi i za ostale objekte kao što je Button, Date Picker i sl.

U sljedećem poglavlju nabrojat ćemo neke od objekata iz biblioteke objekata te vidjeti koji nam sve objekti stoje na raspolaganju za izradu našeg korisničkog sučelja.

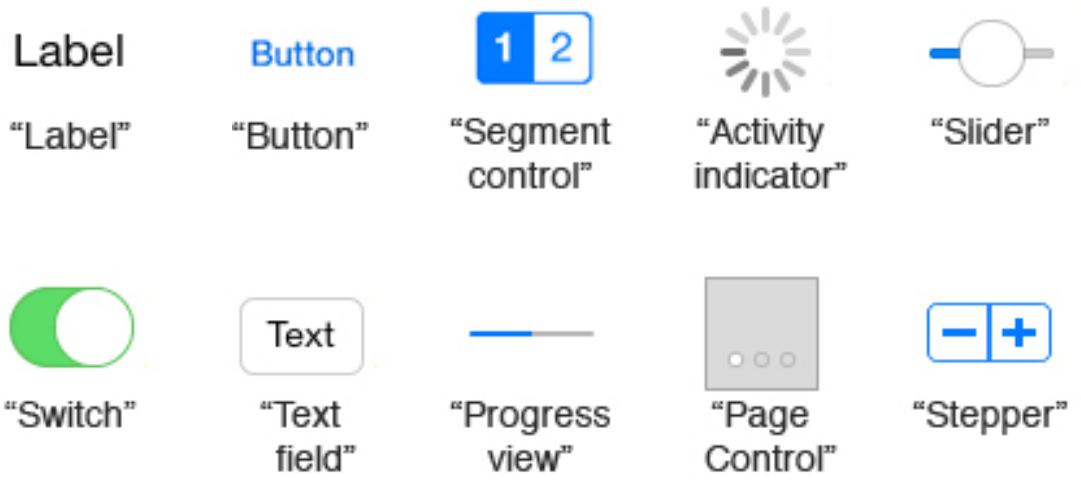
2.6. Biblioteka objekata (*eng. object library*)

Za početak ćemo pregledati koje sve vrste pogleda imamo na raspolaganju. Prije dizajniranja korisničkog sučelja aplikacije bitno je odabrati pogled koji će najviše odgovarati potrebama naše aplikacije. Apple je u svojoj dokumentaciji podijelio poglede ovisno o njihovoj glavnoj namjeni. Na primjer, nećemo koristiti isti pogled za prikazivanje slika ili za prikazivanje navigacije. UIKit razvojni okvir sadrži mnoge vrste pogleda za prikaz i grupiranje podataka. UIKit View objekt je instanca UIVIEW klase ili neke od njezinih podklasa. Pogledajmo kategorizaciju pogleda iz Appleove dokumentacije:

Kategorija	Namjena	Primjena
 Content	Prikaz određenog elementa kao naprimjer slika ili tekst	Pregled slika, oznaka
 Collections	Prikaz kolekcija ili grupa	Pregled kolekcija, tablica
 Controls	Izvođenje radnje ili prikaz informacije	Gumb, slider, prekidač
 Bars	Upravljanje ili izvođenje radnje	Postavke, kratice oznaka, navigacija
Input	Primanje teksta unosom korisnika	Pretraživanje, pregled teksta
 Containers	Primjena containera za druge preglede	Pregled, scroll pregled
Modal	Prekidanje normalnog rada aplikacije kako bi se dopustilo korisniku za izvede određenu radnju	Pregled radnji, pregled upozorenja

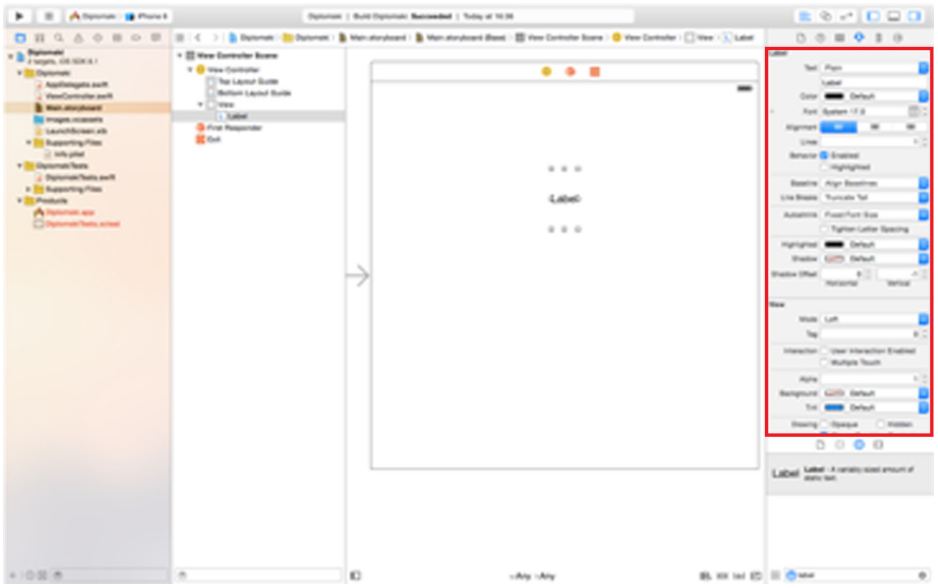
UIKit razvojni okvir sadrži standardne poglede za prezentaciju sadržaja koje možemo koristiti u izradi korisničkog sučelja. Kod naprednijih aplikacija može se pojaviti potreba za posebnim (*eng. custom*) pogleda koji ne postoji u razvojnom okviru. Tada možemo sami definirati vlastitu klasu pogleda za naše potrebe. Ta nova klasa koju ćemo definirat će zapravo biti podklasa UIVIEW klase.

Neki od elemenata koji nam stoje na raspolaganju za izradu korisničkog sučelja iz biblioteke objekata.

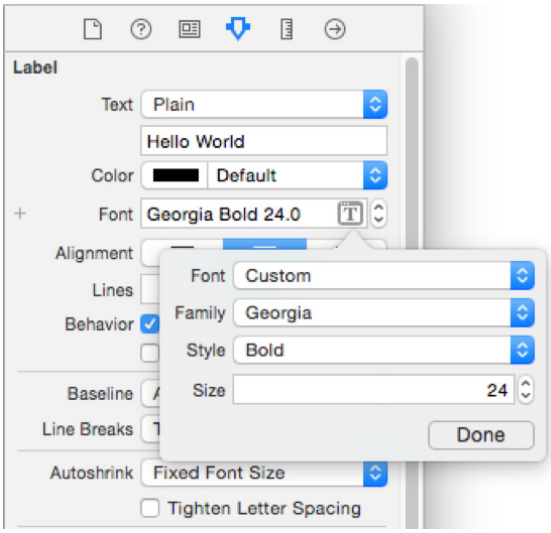


2.7. Atributi

Nakon što dodamo elemente i postavimo na željenu poziciju preostaje nam definirati atribute elemenata. Kako bi definirali atribute elemenata moramo kliknuti na željeni element čije atribute želimo urediti. Nakon toga nam se na desnoj strani Xcodea otvara Atribute inspector.

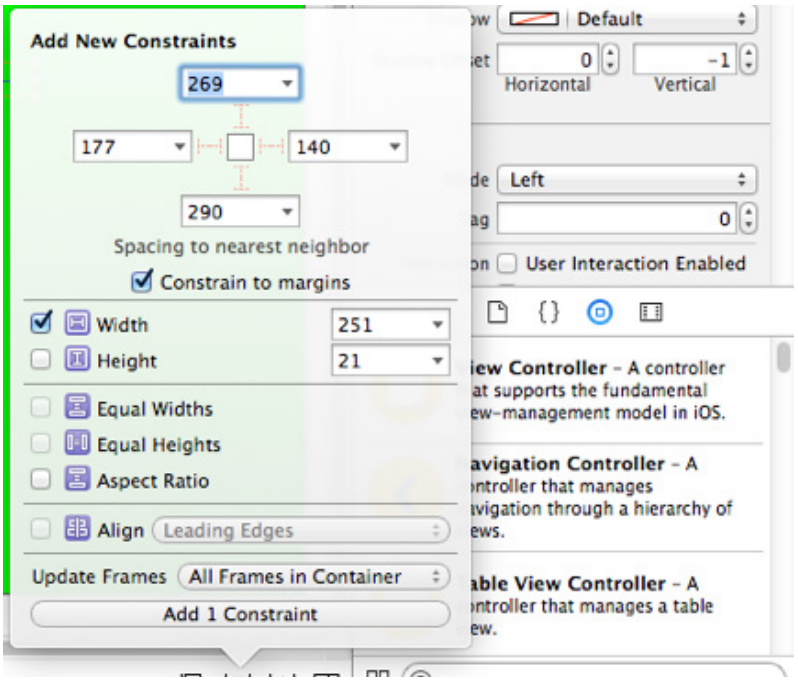


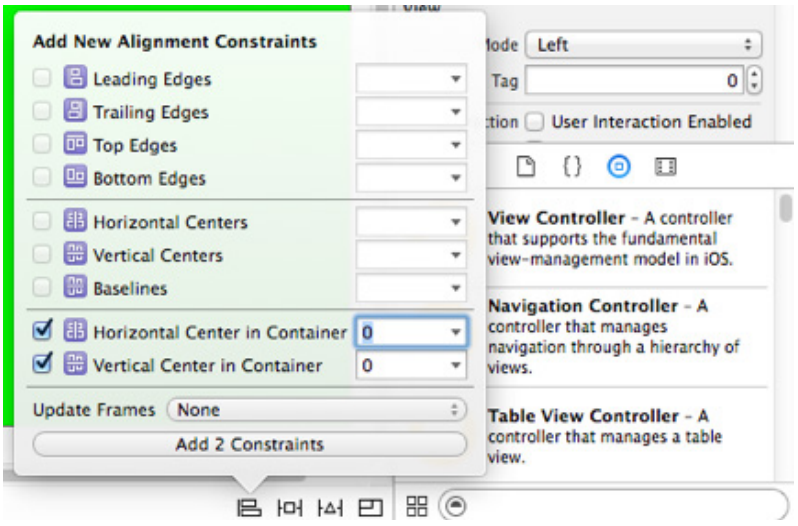
Unutar Atribute inspectora imamo popis atributa koje možemo definirati za odabrani element. Atributi se mogu razlikovati ovisno o vrsti elemenata. Uzmimo za primjer objekt Label. Na slici vidimo dio atributa koje možemo definirati za objekt Label.



2.8. Autolayout

Nakon što dodamo objekte i definiramo njihove atribute preostaje nam definirati njihov položaj u prozoru i njihov međusobni položaj. Trebamo uzeti u obzir da se određene aplikacije moraju moći pregledavati u portretnom i pejzažnom položaju uređaja (iPhone ili iPad). Trebamo definirati udaljenosti od rubova uređaja odnosno margine i padding kako se na određenim uređajima ili položajima uređaja ne bi poremetio izgled korisničkog sučelja. Sa Xcodeom 6 dolazi i nova opcija pod imenom Auto layout koja nam uvelike olakšava posao s definiranjem layouta elemenata.





Na gornjoj slici možemo vidjeti jednu od formi Auto layout alata koji je dio Xcodea. Za svaki od elemenata u korisničkom sučelju možemo postaviti parametre iz gornje forme. Parametri definiraju položaj elementa u kontejneru u kojem se nalazi. Element može biti centriran, položen u desno ili lijevo. Također imamo mogućnost definiranja odmaka od ruba ekrana u pikselima. Možemo postaviti parametre koji će određivati međusobnu udaljenost dva elementa. Kombiniranjem svih parametara trebamo definirati položaje svih elemenata tako da korisničko sučelje bude funkcionalno neovisno o položaju uređaja.

3. Poglavlje

Swift

3.1. Uvod

U ovom poglavlju bavit ćemo se Swiftom kao programskim jezikom koji ćemo koristiti u back-end dijelu izrade iPhone aplikacija. Velika novost koju bi trebali spomenuti vezanu za Swift 2 je sljedeća: na The Apple Worldwide Developers Conference (WWDC) objavljeno je da će Swift postati open source pred kraj 2015. godine. Uz Swiftov kod, Apple će objaviti i kompajler i standardne biblioteke (*eng. libraries*) pod OSI licencom.

Svi primjeri iz sljedećih poglavlja pisat ćemo unutar Playgrounda u Xcodeu 6 radi preglednije analize koda i rezultata. U poglavlju 4. razmotrit ćemo aplikaciju koja je rezultat primijenjenog znanja Swifta.

3.2. Hello world

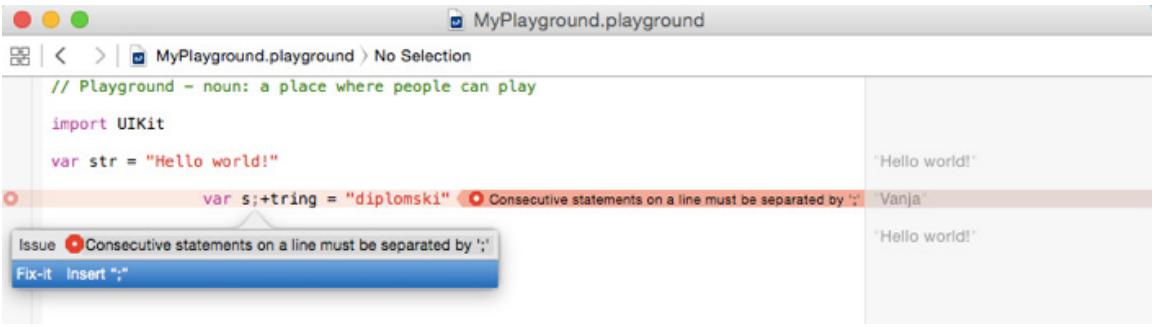
Za početak napisat ćemo jednostavni program Hello world u Swiftu kako bi se upoznali za sintaksom i izgledom Swifta.

Program Hello world je poprilično jednostavan i kao takvog ga nećemo posebno objašnjavati. Ovaj primjer ćemo iskoristiti kako bi istaknuli prednosti Playgorunda kojim ćemo se koristiti u pisanju Swift koda. Nakon kreiranja projekta u Xcodeu koji je početak izrade naše aplikacije, Xcode za nas kreira niz datoteka s kodom. Kod potreban za izradu iPhone aplikacije može biti poprilično dug i kompliciran. Kao takav nam otežava pisanje dodatnog koda i njegovo testiranje. U tu svrhu nam služi alat Playgorund unutar Xcodea. U Playgorundu pišemo Swift kod, kojeg Playground izvršava. Na desnoj strani imamo panel koji zapravo ima ulogu break pointova, tj. prikazuje nam vrijednosti varijabli ili ispis kroz cijeli program. U primjeru Hello world na početku smo importali biblioteku UIKit, te nam Playgorund ne ispisuje nikakvu vrijednost za ovu liniju koda. Nakon toga smo deklarirali varijablu pod imenom "str" i dodijelili joj kao vrijednost znakovni niz "Hello world!" te ju zatim ispisali pomoću naredbe println. U obje linije Playground ispisuje kao "Hello world!".

Prvi put kao vrijednost varijable a drugi put kao ispis naredbe.

Ovaj način pisanja Swift koda unutar Playgorunda nam očito jako pomaže kod testiranja koda, jer na jednostavan način možemo testirati dijelove cijele aplikacije.

Ukoliko napravimo grešku u kodu, Playground nam odmah javlja u kojoj liniji je nastala greška te nam daje i opis same greške.



3.3. Osnovna sintaksa

Swift program se sastoji od mnoštva tokena. Token može biti ključna riječ (*eng. keyword*), identifikator (*eng. identifier*), konstanta, znakovni niz (*eng. string*) ili simbol.

Komentari koji nam služe za objašnjenje koda i napomene, te se ne izvršavaju pokretanjem programa označavamo sa // ako se naš komentar sastoji od jedne linije. // stavljamo na početak komentara. Ukoliko se naš komentar sastoji od više linija, kako ne bi svaku liniju posebno označavali sa //, početak komentara označavamo sa /* a kraj sa */.

```
// My first program in Swift
```

```
/* My first program in Swift is Hello, World!
/* Where as second program is Hello, Swift! */
```

Iz iskustva pretpostavljamo da svaku liniju koda završavamo s točkom zarez (;). Na početku smo naglasili da je jedna od prednosti Swifta jednostavnost i čitljivost koda, pa tako Swift ne zahtijeva ; na kraju svake naredbe. Ukoliko razvijač aplikacija želi stavljati ; na kraju svake naredbe, kompajler neće javiti grešku. Ako ne koristimo ; na kraju svake naredbe, pretpostavka je da svaku naredbu pišemo u svojoj liniji. Dok ako pišemo više naredbi u istoj liniji tada moramo različite naredbe odvajati sa ;.

```
import Cocoa
/* My first program in Swift */
var myString = "Hello, World!"; println(myString)
```
Try it

Identifikatori koje koristimo za imenovanje varijabli, funkcija ili bilo kojeg drugog objekta definiranog od strane korisnika) u Swiftu moraju počinjati sa slovima a-z ili A-Z ili sa _. Nakon početnog znaka identifikator može imati proizvoljan broj prethodno navedenih znakova. Swift ne dozvoljava upotrebu posebnih znakova kao što je @, $, % i sl. Swift je case sensitive, odnosno razlikuje velika i mala slova. Npr. varijabla "Diplomski" nije ista sto i varijabla "diplomski", što možemo vidjeti iz sljedećeg primjera:

Navedimo nekoliko primjera dobro konstruiranih identifikatora:

Kako bi koristili rezerviranu riječ kao identifikator, moramo staviti jednostruki navodnik (`) na početku i na kraju rezervirane riječi. Npr. class je rezervirana riječ i kao takva se ne može koristiti kao identifikator, dok `class` možemo koristiti kao identifikator.

3.4. Ključne riječi (*eng. keywords*)

Ključne riječi su rezervirane riječi od strane Swifta. Ključne riječi ne možemo koristiti kao identifikatore osim ako ih ne definiramo na prethodno opisani način. Ovo su Swiftove ključne riječi:

Keyword-ovi korišteni u deklaracijama

class	deinit	enum	extension
func	import	init	internal
let	operator	private	protocol
public	static	struct	subscript
typealias	var		

Keyword-ovi korišteni u izjavama

break	case	continue	default
do	else	fallthrough	for
if	in	return	switch
where	while		

Keyword-ovi korišteni u izrazima i tipovima

as	dynamicType	false	is
nil	self	Self	super
true	_COLUMN_	_FILE_	_FUNCTION_
LINE			

3.5. Varijable

Varijable u Swiftu deklariramo pomoću ključne riječi var, imena varijable te vrijednosti varijable.

```
var variableName = <initial value>
```

```
import Cocoa

var varA = 42
println(varA)
```

Kao što možemo primijetiti kod varijabli ne trebamo definirati tip varijable. No, ukoliko želimo definirati i tip varijable tada to radimo na sljedeći način:

```
var variableName:<data type> = <optional initial value>
```

```
import Cocoa

var varA = 42
println(varA)

var varB:Float

varB = 3.14159
println(varB)
```

Vrijednosti varijabli ispisujemo pomoću funkcije println. Vrijednosti varijable možemo ispisati unutar znakovnog niza tako što ispred varijable stavimo \ (a na kraju varijable stavimo). Jasnije možemo vidjeti iz sljedećih primjera.

```
import Cocoa

var varA = "Godzilla"
var varB = 1000.00

println("Value of \(varA) is more than \(varB) millions")
```

Rezultat gornjeg programa je sljedeći:

```
Value of Godzilla is more than 1000.0 millions
```

3.6. Konstante

Konstante u Swiftu deklariramo pomoću ključne riječi let.

```
let constantName = <initial value>
```

```
import Cocoa

let constA = 42
println(constA)
```

Try it

Analogno varijablama, uz konstantne možemo definirati i tip konstante na sljedeći način:

```
var constantName:<data type> = <optional initial value>
```

```
import Cocoa

let constA = 42
println(constA)

let constB:Float = 3.14159

println(constB)
```

Try it

Imenovanje i ispisivanje konstanti funkcionira identično kao i imenovanje i ispisivanje varijabli.

3.7. Naredbe za grananje (*eng. if statements*)

Sintaksa:

```
import Cocoa

var varA:Int = 10;

/* Check the boolean condition using if statement */
if varA < 20 {
   /* If condition is true then print the following */
   println("varA is less than 20");
}
println("Value of variable varA is \(varA)");
```

Try it

3.8. Petlje

Jedna od najbitnijh petlji, kada radimo sa složenim strukturama je for-in petlja. Pomoću for-in petlje iteriramo kroz kolekciju objekata, niz (*eng. array*) ili niz znakova u znakovnom nizu.

Sintaksa for-in petlje izgleda ovako:

```
for index in var {
   statement(s)
}
```

Jedan konkretan primjer programskog koda koji sadrži for-in petlju i rezultat izvršavanja tog koda bi izgledao ovako:

```
import Cocoa

var someInts:[Int] = [10, 20, 30]

for index in someInts {
   println( "Value of  index is \(index)")
}
```

Sintaksa i primjer for petlje izgleda ovako:

```
for init; condition; increment{
   statement(s)
}
```

```
import Cocoa

var someInts:[Int] = [10, 20, 30]

for var index = 0; index < 3; ++index {
   println( "Value of someInts[\(index)] is \(someInts[index])")
}
```

```
Value of someInts[0] is 10
Value of someInts[1] is 20
Value of someInts[2] is 30
```

Sintaksa i primjer while petlje izgleda ovako:

```
while condition
{
   statement(s)
}
```

```
import Cocoa

var index = 10

while index < 20
{
   println( "Value of index is \(index)")
   index = index + 1
}
```


```
Value of index is 10
Value of index is 11
Value of index is 12
Value of index is 13
Value of index is 14
Value of index is 15
Value of index is 16
Value of index is 17
Value of index is 18
Value of index is 19
```

Sintaksa i primjer **do-while** petlje izgleda ovako:

```
do
{
   statement(s);
}while( condition );
```

```
import Cocoa

var index = 10

do{
   println( "Value of index is \(index)")
   index = index + 1
}while index < 20
```

```
Value of index is 10
Value of index is 11
Value of index is 12
Value of index is 13
Value of index is 14
Value of index is 15
Value of index is 16
Value of index is 17
Value of index is 18
Value of index is 19
```

Navesti ćemo još tri kontrolne naredbe za petlje (*eng. loop control statement*) i njihove primjere, koje u Swiftu mogu promijeniti standardno izvršavanje petlji.

Kontrola izjava	Opis
continue statement	Zaustavlja trenutni rad petlje, te nastavlja rad petlje na sljedećoj iteraciji
break statement	U potpunosti zaustavlja rad petlje, te program nastavlja izvršavati sljedeću liniju koda.
fallthrough statement	Simulira ponašanje C-ovske switch petlje.

Continue naredba

```
import Cocoa

var index = 10

do{
   index = index + 1

   if( index == 15 ){
      continue
   }
   println( "Value of index is \(index)")
}while index < 20
```

```
Value of index is 11
Value of index is 12
Value of index is 13
Value of index is 14
Value of index is 16
Value of index is 17
Value of index is 18
Value of index is 19
Value of index is 20
```

Primijetimo da u gornjem kodu nije ispisan indeks 15, jer naredba za grananje u slučaju kada je index jednak 15, izvršava naredbu continue i zanemaruje ostatak koda i počinje petlju ispočetka.

Break naredba

```
import Cocoa

var index = 10

do{
   index = index + 1

   if( index == 15 ){
      break
   }
   println( "Value of index is \(index)")
}while index < 20
```

Try it

```
Value of index is 11
Value of index is 12
Value of index is 13
Value of index is 14
```

Falltrough naredba

Switch petlja u Swiftu završava svoje izvršavanje čim dođe do prvog slučaja (*eng. case*) koji zadovoljava a ne provjerava ostatak uvjeta u switch petlji kao što je slučaj u C-u i C++. Falltrough naredba oponaša switch naredbu iz C-a i C++. U C-u i C++ trebamo koristiti break ukoliko želimo zaustaviti izvršavanje petlje nakon što je petlja našla prvi uvjet koji

zadovoljava.

```
switch(expression){
   case constant-expression  :
      statement(s);
      break; /* optional */
   case constant-expression  :
      statement(s);
      break; /* optional */

   /* you can have any number of case statements */
   default : /* Optional */
      statement(s);
}
```

U sljedećem primjeru vidimo generičku sintaksu switch naredbu u Swiftu:

```
switch expression {
   case expression1  :
      statement(s)
      fallthrough /* optional */
   case expression2, expression3  :
      statement(s)
      fallthrough /* optional */

   default : /* Optional */
      statement(s);
}
```

Slijedi primjer switch naredbe u Swiftu bez korištenja fallthrough naredbe:

```
import Cocoa

var index = 10

switch index {
   case 100  :
      println( "Value of index is 100")
   case 10,15  :
      println( "Value of index is either 10 or 15")
   case 5  :
      println( "Value of index is 5")
   default :
      println( "default case")
}
```

```
Value of index is either 10 or 15
```

Sada ćemo dati primjer switch naredbe u Swiftu sa korištenjem fallthrough naredbe:

```
import Cocoa

var index = 10

switch index {
   case 100  :
      println( "Value of index is 100")
      fallthrough
   case 10,15  :
      println( "Value of index is either 10 or 15")
      fallthrough
   case 5  :
      println( "Value of index is 5")
   default :
      println( "default case")
}
```

```
Value of index is either 10 or 15
Value of index is 5
```

3.9. Znakovni niz (*eng.string*)

U ovom poglavlju ćemo navesti glavne funkcije koje koristimo u radu sa znakovnim nizovima i primjere za neke do njih, dok ćemo ostale samo nabrojati. Nećemo ih detaljno objašnjavati jer pretpostavljamo da je čitaoc upoznat s osnovama rada sa znakovnim nizovima.

Kreiranje znakovnih nizova

```
import Cocoa

// String creation using String literal
var stringA = "Hello, Swift!"
println( stringA )

// String creation using String instance
var stringB = String("Hello, Swift!")
println( stringB )
```

```
Hello, Swift!
Hello, Swift!
```

Prazni znakovni niz

```
import Cocoa

// Empty string creation using String literal
var stringA = ""

if stringA.isEmpty {
   println( "stringA is empty" )
} else {
   println( "stringA is not empty" )
}

// Empty string creation using String instance
let stringB = String()

if stringB.isEmpty {
   println( "stringB is empty" )
} else {
   println( "stringB is not empty" )
}
```

```
stringA is empty
stringB is empty
```

Konkatenacija znakovnih nizova

```
import Cocoa

let constA = "Hello,"
let constB = "World!"

var stringA = constA + constB

println( stringA )
```

```
Hello,World!
```

Duljina znakovnih nizova

```
import Cocoa

var varA   = "Hello, Swift!"

println( "\(varA), length is \(count(varA))" )
```

Try it

```
Hello, Swift!, length is 13
```

Ostale funkcije za znakovne nizove

S.No	Funkcije/Operatori i namjena
1	**isEmpty** Bool vrijednost koja determinira dali je niz prazan
2	**hasPrefix(prefix: String)** Funkcija koja provjerava postoje li pruženi parametar kao prefix niza ili ne
3	**hasSuffix(suffix: String)** Funkcija koja provjerava postoje li pruženi parametar kao prefix niza ili ne
4	**toInt()** Funkcija koja pretvara numeričku vrijednost niza u broj
5	**count()** Globalna funkcija koja zbraja broj znakova u nizu
6	**utf8** Svojstvo za vračanje niza i prikazivanje u UTF-8

7	**utf16** Svojstvo za vračanje niza i prikazivanje u UTF-16
8	**unicodeScalars** Svojstvo za vračanje niza u Unicode Scalar
9	**+** Operator za povezivanje dva niza u obliku lanca, ili niza u slovo ili dva slova
10	**+=** Operator za dodavanje niza ili slova u posojeći niz
11	**==** Operator za determiniranje jednakosti dvaju niza
12	**<** Operator za izvođenje leksikografske usporedbe za determiniranje vrijednosti dvaju niza
13	**==** Operator za determinaciju jednakosti dvaju niza

3.10. Enumeracije i strukture

U Swiftu enumeracije deklariramo pomoću ključne riječi enum. Enumeracije su deifinirane unutar klase a pridružene vrijednosti kao varijable članice te klase.

```
enum enumname {
   // enumeration values are described here
}
```

Navedimo jedan primjer koda koji sadrži enumeracije i rezultat pokretanja tog koda:

```
enum names{
   case Swift
   case Closures
}
var lang = names.Closures
lang = .Closures
switch lang
{
   case .Swift:
   println("Welcome to Swift")
   case .Closures:
   println("Welcome to Closures")
   default:
   println("Introduction")
}
```

```
Welcome to Closures
```

Kod struktura postoji razlika između struktura u Swiftu i struktura u Objective-C i C-u. Strukture ne trebaju implementacijske datoteke niti razvojne okvire. Strukture nam omogućavaju da kreiramo jedanu datoteku i proširimo njen razvojni okvir automatski na ostale blokove.

Sintaksa definiranja strukture:

```
Structures are defined with a 'Struct' Keyword.
struct nameStruct {
   Definition 1
   Definition 2
    ---
   Definition N
}
```

3.11. Klase (eng. class)

3.11.1. Sintaksa

Swift je objektno orijentirani programski jezik. Zato ćemo se detaljno pozabaviti klasama, odnosno objektima. Pretpostavljamo da je čitatelj upoznat sa filozofijom objektno orijentiranih jezika te klase i ostale pojmove kao takve nećemo posebno objašnjavati već ćemo se baviti sintaksnim dijelom i posebnostima Swifta kao objektno orijentiranog jezika.

Sintaksna definicija klase u Swiftu izgleda ovako:

```
Class classname {
   Definition 1
   Definition 2
    ---
   Definition N
}
```

Primjer definiranja jedne klase izgleda ovako:

```
class student{
   var studname: String
   var mark: Int
   var mark2: Int
}
```

dok kreiranje instance klase izgleda ovako:

```
let studrecord = student()
```

Navedimo jedan konkretan primjer klase. Za trenutak zanemarimo definiranje varijabli članica (*eng. properties*) i definiranje metoda (*eng. methods*).

```
class MarksStruct {
   var mark: Int
   init(mark: Int) {
      self.mark = mark
   }
}

class studentMarks {
   var mark = 300
}
let marks = studentMarks()
println("Mark is \(marks.mark)")
```

Nakon pokretanja gornjeg koda u Playgroundu dobivamo sljedeći rezultat:

```
Mark is 300
```

3.11.2. Varijable članice

Varijablama članicama pristupamo na standardni način kao i u većini programskih jezika. Ukoliko želimo pozvati varijablu članicu tada pozivamo instance i nakon nje samu varijablu članicu, dok je instanca odvojena sa '.' od varijable članice.

```
class studentMarks {
   var mark1 = 300
   var mark2 = 400
   var mark3 = 900
}
let marks = studentMarks()
println("Mark1 is \(marks.mark1)")
println("Mark2 is \(marks.mark2)")
println("Mark3 is \(marks.mark3)")
```

Rezultat gornjeg koda je sljedeći:

```
Mark1 is 300
Mark2 is 400
Mark3 is 900
```

Apple u svojoj dokumentaciji dijeli varijable članice u dvije grupe:

1. Stored Properties
2. Computed Properties

3.11.2.1. Stored properties

Stored properties je vrsta varijabli članica koja sprema vrijednost varijable ili konstante te je dio neke instance. Stored property može biti variable stored properties ili constant stored properties. Ukoliko se radi o variable stored properties tada ju deklariramo pomoću ključne riječi var a ukoliko se radi o constant stored properties tada ju deklariramo pomoću ključne riječi let. Vrijednosti samih varijabli članica možemo postaviti unutar same definicije klase ili tijekom inicijalizacije.

```
1  struct FixedLengthRange {
2      var firstValue: Int
3      let length: Int
4  }
5  var rangeOfThreeItems = FixedLengthRange(firstValue: 0, length: 3)
6  // the range represents integer values 0, 1, and 2
7  rangeOfThreeItems.firstValue = 6
8  // the range now represents integer values 6, 7, and 8
```

U navedenom primjeru varijable članice firstValue i length su stored properties i njihove vrijednosti ne postavljamo u samoj definiciji klase već prilikom same inicijalizacije. Kod inicijalizacije vrijednost varijable članice firstValue postavljamo na 0, a naknadno mijenjamo joj vrijednost na 6. Primijetimo da ne bi naknadno mogli promijeniti vrijednost varijable članice length jer je definirana kao konstanta.

U slučaju kada samu strukturu definiramo kao konstantu, tada ne možemo mijenjati vrijednost ni jedne varijable članice bez obzira da li je varijabla ili konstanta.

```
1  let rangeOfFourItems = FixedLengthRange(firstValue: 0, length: 4)
2  // this range represents integer values 0, 1, 2, and 3
3  rangeOfFourItems.firstValue = 6
4  // this will report an error, even though firstValue is a variable property
```

U navedenom primjeru smo instancu rangeOfFourItems deklarirali kao konstantu. Kada smo pokušali promijeniti vrijednost varijable članice koja je varijabla, dobili smo grešku.

3.11.2.2. Lazy stored properties

Lazy stored property je podvrsta Stored propertya čija inicijalna vrijednost nije postavljena sve do trenutka kada lazy stored property nije korišten po prvi put. Takvu vrstu varijable definiramo tako što ispred njene deklaracije pišemo ključnu riječ lazy.

Ovakav tip variable koristimo u situacijama kada nam inicijalna vrijednost varijable članice nije poznata na početku ili ovisi o drugim varijablama čije vrijednosti nisu poznate prije završetka inicijalizacije instance. Druga situacija u kojoj nam lazy stored properties olakšavaju posao je sljedeća: neka se vrijednost varijable članice računa na vrlo kompleksan i vremenski složen način, a samu varijablu ne koristimo često unutar instance. Tada je pametno ne računati vrijednost varijable prilikom inicijalizacije instance jer će nam uzeti puno vremena a nismo sigurno hoćemo li uopće koristiti.

```
1  class DataImporter {
2      /*
3      DataImporter is a class to import data from an external file.
4      The class is assumed to take a non-trivial amount of time to initialize.
5      */
6      var fileName = "data.txt"
7      // the DataImporter class would provide data importing functionality here
8  }
9
10 class DataManager {
11     lazy var importer = DataImporter()
12     var data = [String]()
13     // the DataManager class would provide data management functionality here
14 }
15
16 let manager = DataManager()
17 manager.data.append("Some data")
18 manager.data.append("Some more data")
19 // the DataImporter instance for the importer property has not yet been created
```

Iz danog koda vidimo da je klasa DataImporter zadužena za import podataka iz vanjske datoteke. Klasa DataManager sadrži jednu Lazy stored varijablu i jednu Stored varijablu. Prilikom inicijalizacije varijable manager koja je tipa DataManager, inicijalizirana je varijabla data ali ne i varijabla importer. Pretpostavka je da instanca importer možda prilikom svog korištenja neće trebati podatke iz vanjske datoteke, a importanje može biti dugotrajno i memorijski zahtjevno. U gornje navedenom kodu, vrijednost varijable importer nije još

postavljena jer sama varijabla nije ni korištena.
Tek kada ju pozovemo, bit će kreirana.

```
println(manager.importer.fileName)
// the DataImporter instance for the importer property has now been created
// prints "data.txt"
```

3.11.2.3. Computed properties

Druga vrsta varijabli članica su Computed properties. Za razliku od Stored properties, Computed properties ne spremaju vrijednost varijable već ju računaju iz danih parametara. Za ovu vrstu varijabli članica moramo definirati funkcije getter i setter kako bi im dodijelili odnosno promijenili vrijednost koju čuvaju.

Pogledajmo sljedeći primjer:

```
struct Point {
    var x = 0.0, y = 0.0
}
struct Size {
    var width = 0.0, height = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set(newCenter) {
            origin.x = newCenter.x - (size.width / 2)
            origin.y = newCenter.y - (size.height / 2)
        }
    }
}
var square = Rect(origin: Point(x: 0.0, y: 0.0),
    size: Size(width: 10.0, height: 10.0))
let initialSquareCenter = square.center
square.center = Point(x: 15.0, y: 15.0)
println("square.origin is now at (\(square.origin.x), \(square.origin.y))")
// prints "square.origin is now at (10.0, 10.0)"
```

Strukture Point i Size su nam pomoćne strukture koje koristimo u funkcijama get i set za varijablu članicu center unutar strukture Rect.

U danom primjeru smo deklarirali varijablu square koja je tipa Rect, te inicijalizirali vrijednosti varijabla članica origin i size. Prilikom inicijalizacije, varijabla članica center je izračunata pomoću prve dvije varijable članice. Nismo trebali posebno definirati vrijednost varijable članice center kod inicijalizacije. Varijabla članica center je u ovom primjeru Computed property. Naknadno smo mijenjali vrijednost varijable center. Postavili smo joj vrijednost na Point(x:15.0, y: 15.0). Primijetimo da je center varijabla članica tipa Point. U slučaju mijenjanja vrijednosti Computed property nakon same inicijalizacije, poziva se funkcija set. Nova vrijednost se sprema unutar Computed varijable te se poziva funkcija set koja u ovom slučaju mijenja vrijednost preostalih varijabli.

3.11.3. Metode

Sintaksa metoda je identična sintaksi funkcija. Spomenimo samo razliku između Objective-C i Swifta što se tiče metoda. Objective-C dozvoljava definiranje metoda jedino unutar klasa dok u Swiftu klase, strukture i enumeracije mogu imati pripadne metode.

Sintaksa metoda izgleda ovako:

```
func funcname(Parameters) -> returntype
{
    Statement1
    Statement2
       ---
    Statement N
    return parameters
}
```

Navedimo i primjer konkretne metode i rezulatat pokretanja koda:

```
class calculations {
    let a: Int
    let b: Int
    let res: Int

    init(a: Int, b: Int) {
        self.a = a
        self.b = b
        res = a + b
    }

    func tot(c: Int) -> Int {
        return res - c
    }

    func result() {
        println("Result is: \(tot(20))")
        println("Result is: \(tot(50))")
    }
}
let pri = calculations(a: 600, b: 300)
pri.result()
```


```
Result is: 880
Result is: 850
```

3.11.4. Nasljeđivanje

Nasljeđivanje klasa nećemo posebno objašnjavati već ćemo pokazati primjer podklase u Swiftu te primjer overridea metode.

```
class StudDetails
{
    var mark1: Int;
    var mark2: Int;

    init(stm1:Int, results stm2:Int)
    {
        mark1 = stm1;
        mark2 = stm2;
    }

    func print()
    {
        println("Mark1:\(mark1), Mark2:\(mark2)")
    }
}

class display : StudDetails
{
    init()
    {
        super.init(stm1: 93, results: 89)
    }
}

let marksobtained = display()
marksobtained.print()
```

Override funkcija je opcija kreiranja metode za podklasu pod istim imenom kao za superklasu (*eng. superclass*). Kako bi definirali takvu funkciju, ispred definicije same funkcije stavljamo ključnu riječ override. Ukoliko u podklasi želimo pristupati metodama i varijablama članicama iz superklasu, tada umjesto imena instance stavljamo ključnu riječ super.

```
class cricket {
    func print() {
        println("Welcome to Swift Super Class")
    }
}

class tennis: cricket  {
    override func print() {
        println("Welcome to Swift Sub Class")
    }
}

let cricinstance = cricket()
cricinstance.print()

let tennisinstance = tennis()
tennisinstance.print()
```

Try it

Nakon pokretanja gornjeg koda dobivamo sljedeći rezultat

```
Welcome to Swift Super Class
Welcome to Swift Sub Class
```

4. Poglavlje

Primjer aplikacije

Stečeno znanje smo primjenili u izradi iPhone aplikacije pod nazivom “Telefonski imenik”. U aplikaciji korisnik ima mogućnost dodavanja novih kontakata, editiranje i pregled postojećih. Kreiranje posebnih grupa kontakta u koje može grupirati kontakte. Korisnik ima mogućnost postavljanje kontakta kao favorita.

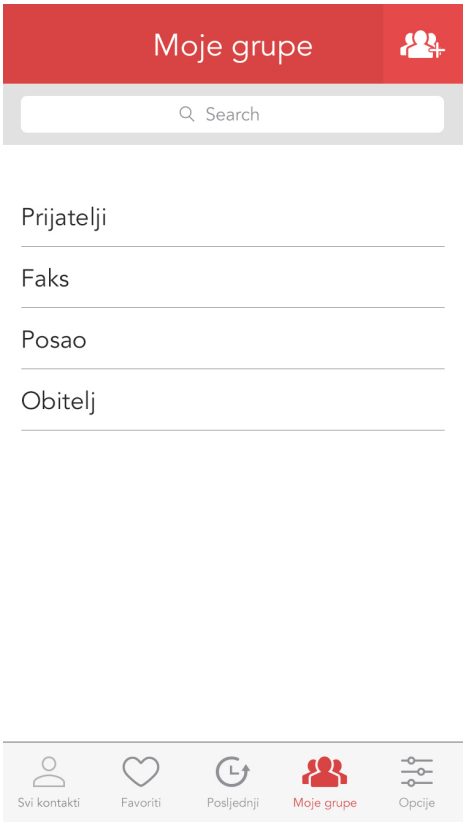
U izradi aplikacije za smo primijenili prethodno stečena znanja za kreiranje i implementaciju korisničkog sučelja te znanje Swifta za izradu back-end dijela aplikacije. Također se u aplikaciji susrećemo sa nekim dijelovima koje nismo detaljno objasnili u radu. Te dijelove ćemo spomenuti i objasniti u izlaganju ove aplikacije.

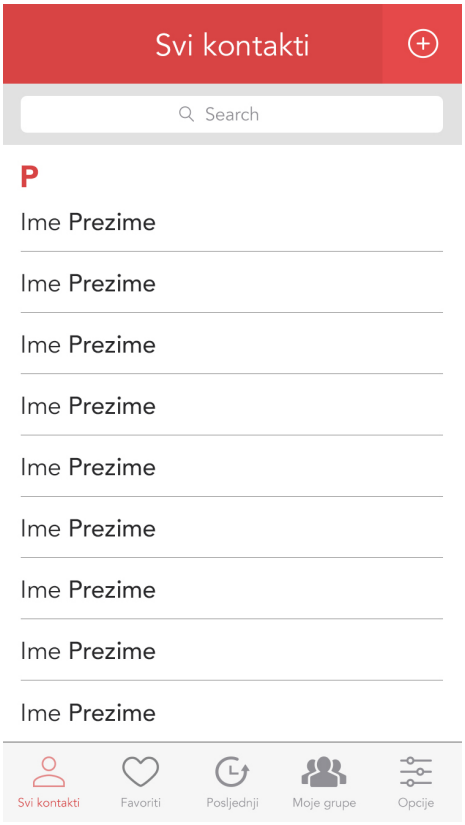
Za idejno rješenje korisničkog sučelja koristili smo Adobe Photoshop. Nakon što smo nizom izmjena došli do zadovoljavajućeg rješenja, pomoću Xcodea i opcije Autolayout implementirali smo samo sučelje. U izradi sučelja koristili smo i Storyboard. Po završetku implementacije korisničkog sučelja krenuli smo na izradu back-end dijela. Back-end dio aplikacije smo radili u Swiftu. Prilikom pisanja Swift koda puno smo koristili Playground koji nam je olaksao testiranje i otkrivanje grešaka.

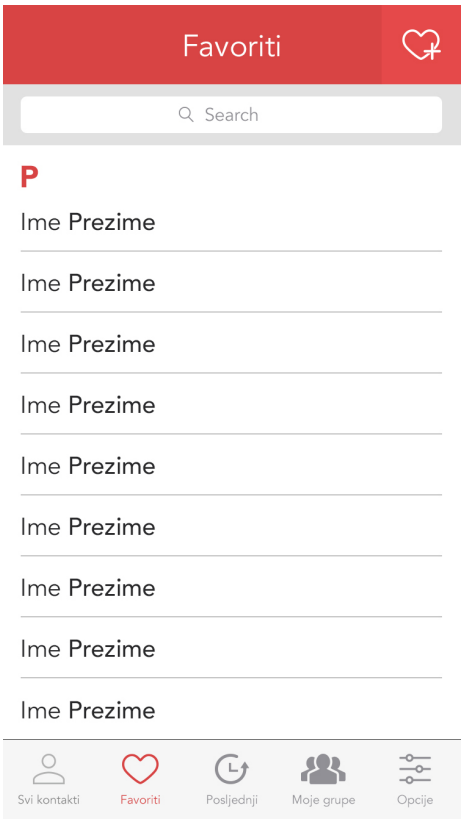
Objasnimo redom dio po dio korisničkog sučelja i njegove funkcionalnosti, elemente koje smo koristili u izradi te način kako smo implementirali back-end dio aplikacije.

Na sljedećoj slici možemo vidjeti LaunchScreen aplikacije, odnosno zaslon koji se prikazuje korisniku dok se aplikacija učitava.

Krenimo na zaslone Svi kontakti, Favoriti, Moje grupe. Nabrojeni zasloni imaju različite funkcionalnosti ali korisnička sučelja su im implementirana na sličan način. Kao pogled smo koristili Table view, unutar kojeg smo stilizirali poglede Table view cell. Navedenim pogledima smo smanjili širinu i povećali padding. Font koji smo koristili u cijeloj aplikaciji je Avenir. Kako bi spremali podatke unesene od strane korisnika kreirali smo bazu podataka sa par tablica. Kreirali smo tablicu pod imenom Favoriti u koju smo spremali id kontakata koje je korisnik dodao kao favorite. Za potrebe zaslona Moje grupe smo kreirali tablicu u koju bi spremali id kontakta i id grupe kojoj je dodijeljen od strane korisnika aplikacije.

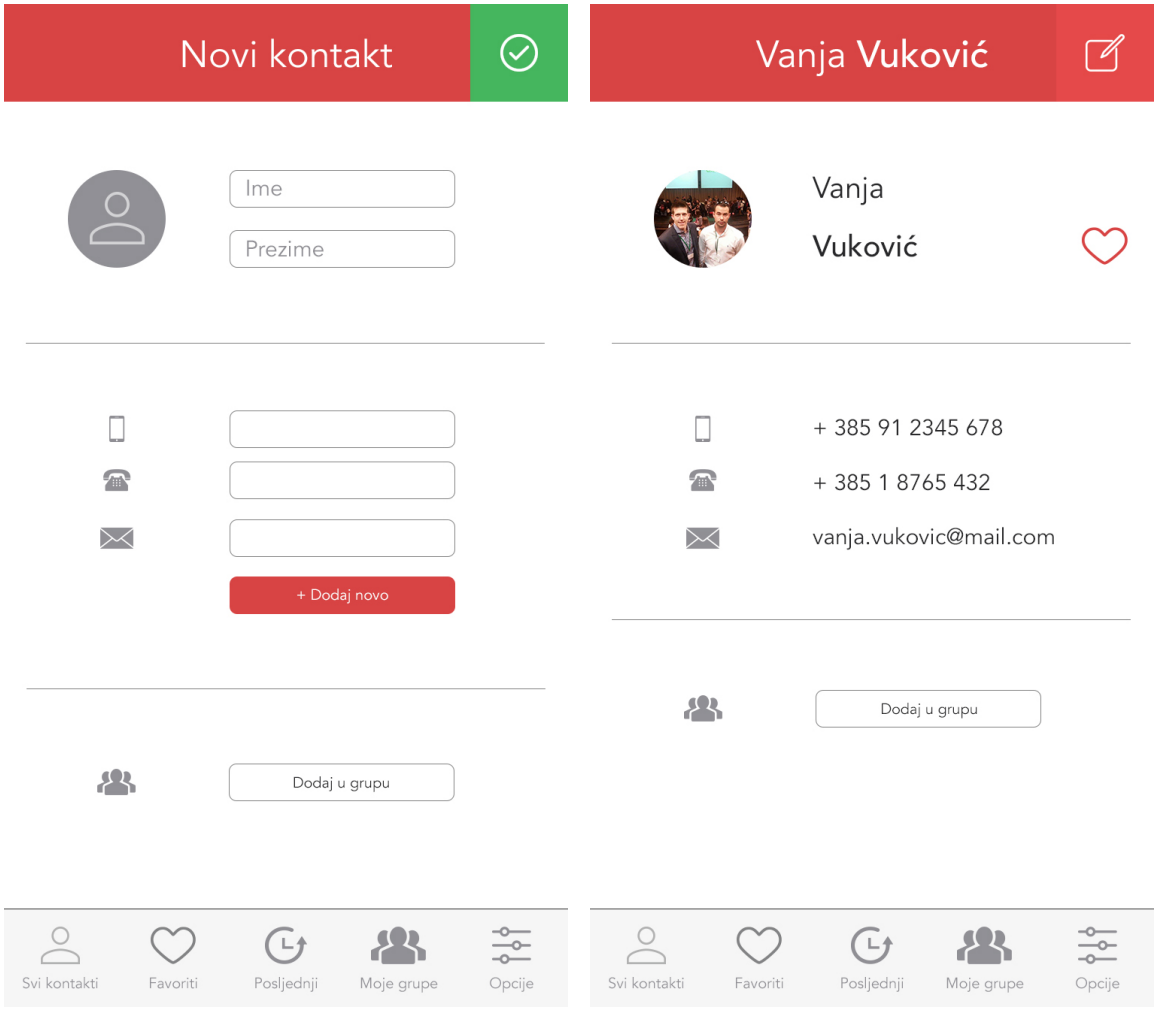

Moje grupe
Search
Prijatelji
Faks
Posao
Obitelj
Svi kontakti
Favoriti
Posljednji
Moje grupe
Opcije


Svi kontakti
Search
P
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Svi kontakti
Favoriti
Posljednji
Moje grupe
Opcije


Favoriti
Search
P
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Ime Prezime
Svi kontakti
Favoriti
Posljednji
Moje grupe
Opcije

Na dnu svakog zaslona smo stavili objekt tab bar koji je zadužen za navigaciju. Nakon što smo dodali Tab bar u Storyboard, morali smo dodati još unutar Tab bara tri objekta tipa Tab Bar Item jer Tab Bar po defaultu sadrži dva objekta Tab Bar Item. Naknadno dodane Tab Bar Iteme smo stilizirali i povezali sa zaslonima.

Sučelje za dodavanje novog kontakta je riješeno pomoću niza objekata tipa Text Field pomoću kojih korisnik unosi podatke kontakta. Nakon dodavanja kontakta, podaci se spremaju u tablicu pod nazivom Korisnici. Odabirom kontakta u pregledu svih kontakata korisniku se prikazuje zaslon sa podacima pojedinog kontakta.



5. Poglavlje

ZAKLJUČAK

Vidjeli smo da na raspolaganju imamo više programskih jezika i alata pomoću kojih možemo izrađivati iPhone aplikacije. Također, vidjeli smo da se izrada jedne aplikacije sastoji od više međusobno neovisnih dijelova kao što je priprema i izrada korisničkog sučelja aplikacije te back-end dijela aplikacije. Pošto je izrada iPhone aplikacija široko područje u diplomskom radu, nismo uspjeli dotaknuti složenijih dijelova već smo se ograničili na dijelove i novitete koji su specifični za ovu vrstu softvera kao što je npr. Storyboard i Playground. Uz specifične dijelove, u diplomskom radu smo uključili neophodna poglavlja vezana uz objektno orijentirano programiranje. Kako se tržište mobilnih aplikacija konstantno mijenja i napreduje, potrebno je isto tako konstantno pratiti i učiti promjene koje dolaze. Kako broj korisnika mobilnih aplikacija raste, pretpostavljamo da tržište može samo napredovati. Hardverski napredak povlačit će i automatski softverski napredak tehnologija što ostavlja veliki prostor softverskim tvrtkama za preuzimanje svog dijela tržišta, a razvijačma aplikacija nove izazove i tehnologije kojima će se morati prilagoditi kako bi ostali u toku. Primjenu tehnologija, programskih jezika i alata opisanih u ovom diplomskom radu možemo primijeniti u izradi mobilnih aplikacija i desktop aplikacija za operativni sistem OS X. Kako se poslovanje seli na *cloud* tehnologije, a broj korisnika pametnih telefona povećava, zaključujemo da mobilne aplikacije postaju nezaobilazni dio svakodnevnog života, odnosno nezaobilazno znanje većine programera.

Literatura

[1] J. Conway, A. Hillegass, C. Keur, iOS Programming - The Big Nerd Ranch Guide, 4th Edition, Big Nerd Ranch Guides, 2014.

[2] C. Hockenberry: iPhone App Development - The Missing Manual, O'Reilly Media, 2010. M.A.

[3] Lassoff, T. Stachowitz: Swift Fundamentals - The Language of iOS Development, LearnToProgram Inc, 2014.

[4] Swift, dostupno na https://developer.apple.com/swift/ (lipanj 2015.)

[5] Xcode, dostupno na https://en.wikipedia.org/wiki/Xcode (lipanj 2015.)

[6] Swift tutorial, dostupno na http://www.tutorialspoint.com/swift/ (kolovoz 2015.)

[7] Swift tutorial, dostupno na http://codewithchris.com/course/how-to-make-an-app-with-no-programming-experience/ (srpanj 2015.)

Sažetak

U ovom radu bavimo se izradom iPhone aplikacija pomoću programskog jezika Swift. Obradili smo sva područja koja su nam potrebna za izradu aplikacije, od izrade korisničkog sučelja, do samog objektno orijentiranog jezika Swift. Naučeno smo primijenili u izradi jedne konkretne aplikacije koju smo na kraju funkcionalno opisali te dali izgled njenog dizajna. U nekim dijelovima dajemo usporedbu Swifta s njegovom alternativom: programskim jezikom Objective-C. Također, govorimo o MVC modelu izrade aplikacija, potrebnim alatima i njihovim prednostima.

Summary

In this paper, we are creating iPhone application using programming language Swift. We processed all the areas that we need to create applications, from making user interface, to the object-oriented language Swift. We apply lessons learned to the particular application which we have implemented. We have functionally described application at the end. In some parts we give comparison of Swift with its alternative: programming language Objective-C . Also, we are talking about the MVC model, required tools and their benefits.

Životopis

Rođen sam u Zagrebu 1989. godine. U Zagrebu sam pohađao Osnovnu školu Retkovec. Za vrijeme osnovnoškolskog obrazovanja sudjelovao sam na natjecanjima iz matematike. Po završetku osnovne škole upisao sam XV. gimnaziju, nakon koje upisujem Ekonomski fakultet u Zagrebu. Nakon jedne godine studija na Ekonomskog fakultetu, napustio sam studij ekonomije te upisao PMF - Matematički odsjek. Na četvrtoj godini studija s partnerom Filipom Žicom sam osnovao softversku tvrtku Pix and Codes koja se bavi web tehnologijama i mobilnim aplikacijama a danas broji 5 zaposlenih i mnogobrojne uspješno završene projekte.