Project1 - Storm viewer

100 days of Swift - Paul Hudson

Nikola Krstevski (prevod na srpski jezik) - June 4, 2019

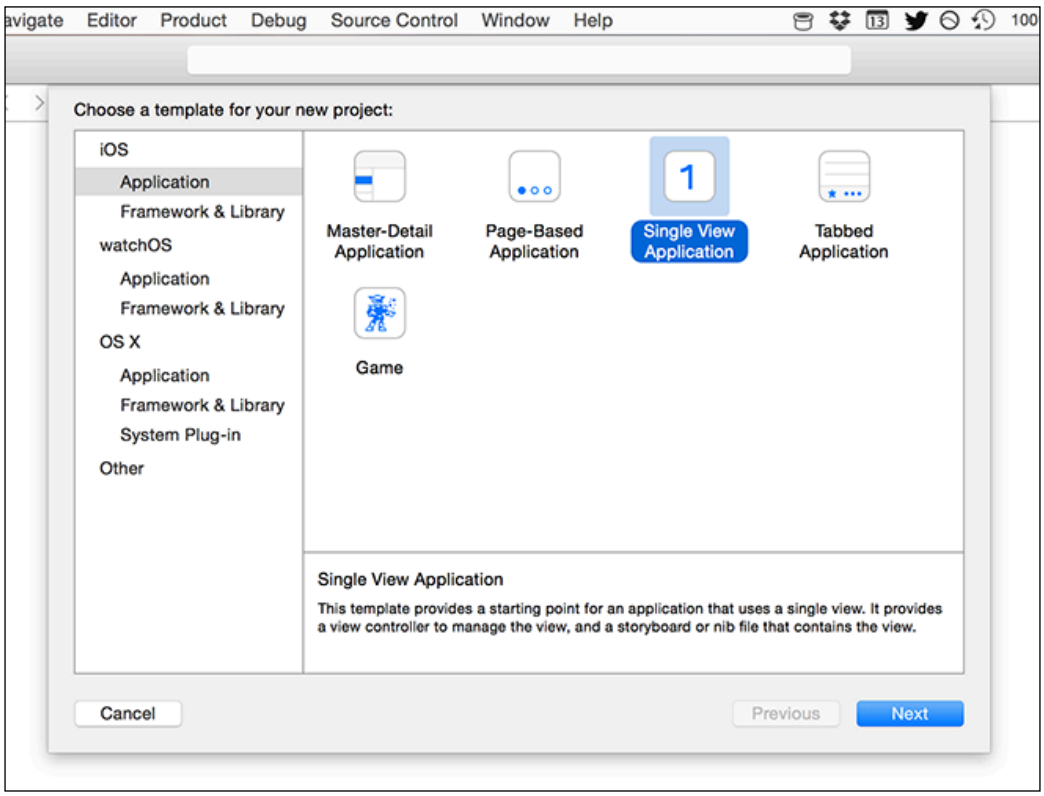
Table of Contents

Setting up

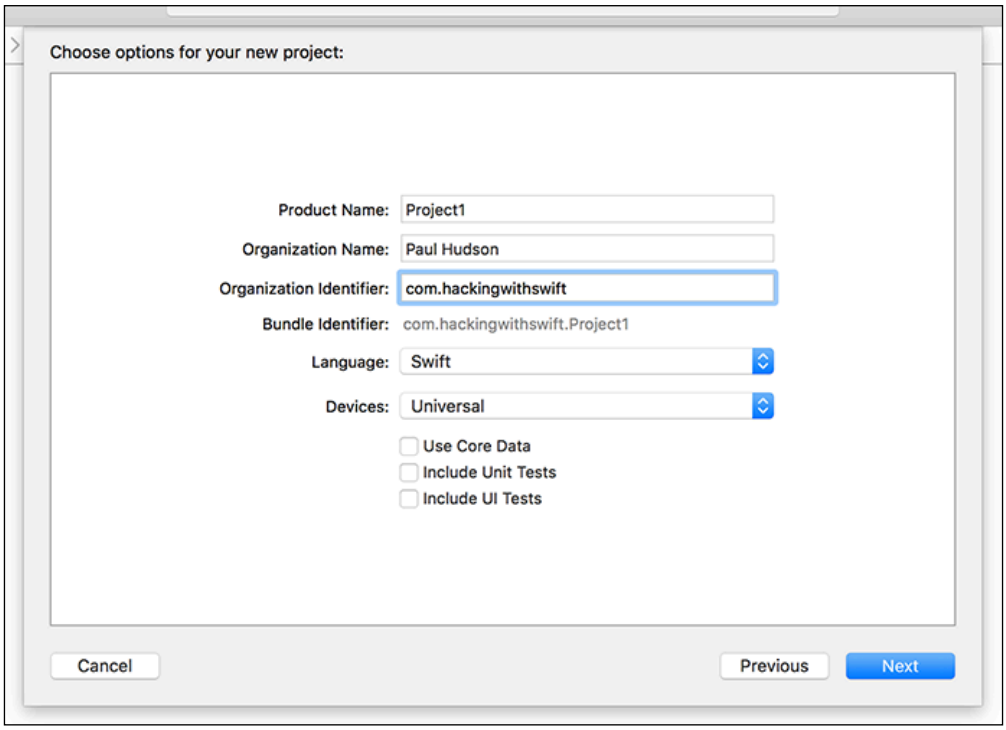
(Podesavanja za rad)

U ovom projektu napravicete aplikaciju koja omogucava korisniku da se krece kroz listu slika i izabere jednu od njih za gledanje. Ona je namerno jednostavna zato sto postoji mnogo drugih stvari koje morate usput nauciti, prema tome, budite strpljivi jer ce ovo malo da potraje!

Pokrenite Xcode I izaberite "Create a new Xcode project" sa ekrana dobrodoslice. Izaberite Single View App iz liste I kliknite na Next. Kao Product name unesite Project1 I postarajte se da ste izabrali Swift kao jezik.



Jedno od polja koje je obavezno popuniti je "Organization Identifier", koji je unikalni identifikator i koji se obicno formira od domena vaseg licnog sajta napisanog unazad. Na primer, ja bih stavio com.hackingwithswift kada bih pravio aplikaciju za app store. Morate napisati nesto validno unutra ako mislite da objavite vasu aplikaciju, a ako ne zelite, mozete jednostavno da napisete com.example.



Vazna napomena: neki od sablona(templates) Xcoda, imaju polja za potvrdu(checkboxes) “Use Core Data”, "Include Unit Tests" i "Include UI Tests”. Molim vas postarajte se da su ova polja odčekirana za ovaj projekat i sve ostale projekte u ovoj seriji. Postoji samo jedan projekat gde ovo nece biti slucaj, a to cu vam veoma jasno naglasiti kada za to dodje vreme!

Sada kliknite ponovo na Next i bicete pitani gde zelite da sacuvate vas projekat - u redu je da to bude na vasem radnom stolu. Kada ovo uradite, bice Vam predstavljen primer projekta koji je Xcode napravio za vas. Prvo sto treba da uradimo je da se postaramo da ste sve podesili pravilno a to znaci da pokretanje programa takvim kakv je trenutno.

Kada pokrenete projekat, moci cete da birate koji ce uredjaj vas simulator predstavljati ili mozete da izaberete pravi telefon ako ga imate prikljucenog na kompjuter. Ove opcije se nalaze pod imenom Product > Destination menu i trebalo bi da vidite iPad, iPhone 8 itd. Takodje postoji precica do ovog menija: na gornjem levom delu prozora Xcoda-a se nalazi PLAY i STOP dugme i odmah pravo do njih treba da pise Project1 i onda Device name(ime uredjaja). Mozete da pritisnete na to ime uredjaja i da izaberete neki drugi uredjaj. Za sada molim vas izaberite iPhone XR i pritisnite PLAY trougaono dugme u gornjem levom uglu. Ovo ce kompajlirati (sastaviti) vas kod (sto je u stvari proces pretvaranja koda u instrukcije koje iPhone moze da razume), a zatim pokrenuti simulator i odmah zatim i samu aplikaciju. Kao sto cete videti kada zapocnete interakciju sa aplikacijom, ona pokazuje samo veliki beli ekran i ne radi apsolutno nista drugo osim toga - bar ne jos uvek!

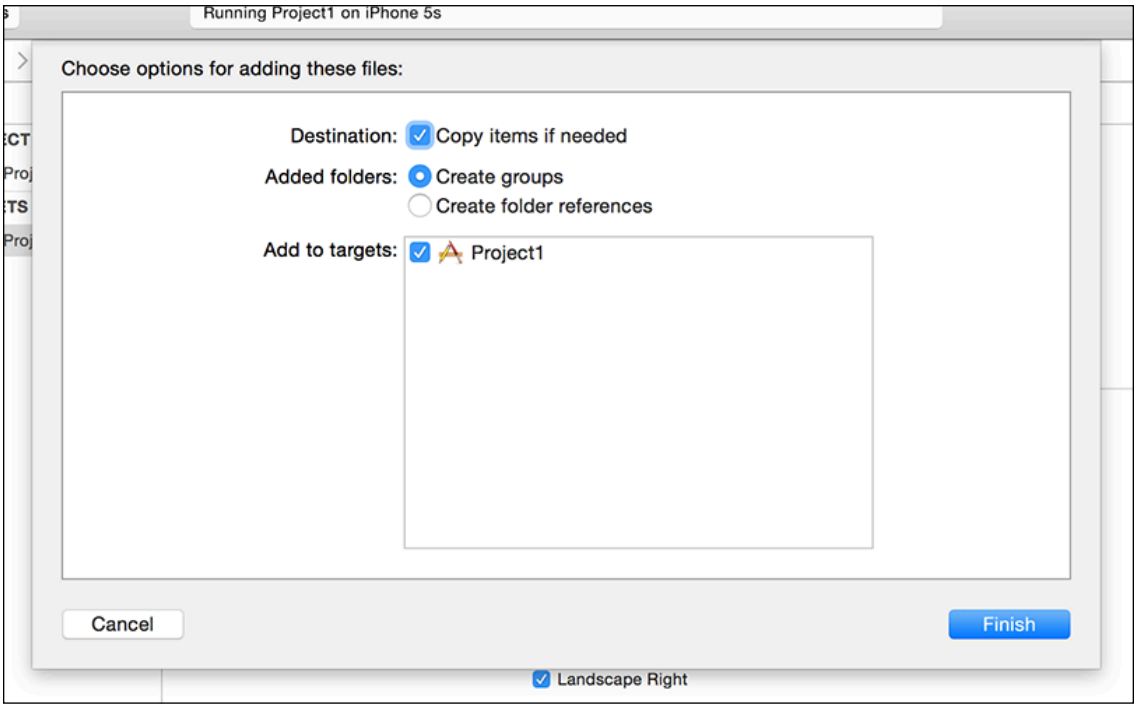
Tokom ucenja cete veoma cesto zaustavljati projekte, tako da je potrebno da znate sledecih nekoliko stvari:

- Mozete da pokrecete vas projekat pritiskom na **Cmd + R**. Ovo je ekvivalent pritisnutom PLAY dugmetu
- Mozete da zaustavite tekuci projekat pritiskom na **Cmd + .** kada je Xcode izabran.
- Ako ste napravili promene u tekucem projektu, samo pritisnite **Cmd + R** ponovo. Xcode ce vas obavestiti da zaustavite trenutnu probu pre nego sto zapocnete novu. Pobrinite se da oznacite polje "Do not show this message again" da izbegnete buduca takva upozorenja.

Ovaj projekat je zasnovan na tome da korisnici biraju slike koje ce gledati, pa cete morati da skinete nekoliko slika za ovaj projekat sa GitHub-a (https://github.com/twostraws/HackingWithSwift), i pogledajte u datoteci "project-files". Videcete jos jednu datoteku unutra koja se zove "Content". Hocu da prevucete tu datoteku pravo u vas Xcode projekat, odmah ispod reda gde pise "Info.plist".

Savet: ako niste sigurni sta da skinete, koristite ovaj link https://github.com/twostraws/HackingWithSwift/archive/master.zip - to je zip-ovan fajl sa svim mojim projektima.

Pojavice se prozor koji ce vas pitati kako zelite da dodate fajlove: postarajte se da oznacite "Copy items if needed" i "Create groups". **Veoma vazno**: **nemojte da izaberete "Create folder reference" jer u tom slucaju vas projekat nece raditi.**



Pritisnite Finish i videcete kako se u Xcode-u pojavljuje zuta datoteka pod imenom Content. Ako vidite plavu datoteku umesto zute , to znaci da niste izabrali "Create groups" zbog cega cete imati probleme sa pracenjem ovog kursa!

Listing images with FileManager

(Listing slika sa FileManager)

Slike koje sam vam dao dolaze od National Oceanic and Atmospheric Administration (NOAA), koja je Americka vladina organizacija i shodno tome proizvodi sadrzaj koji je za javnu upotrebu i koji mozemo slobodno koristiti. Jednom kada su kopirani u vas projekat Xcode ce ih automatski ugraditi u vasu zavrsenu aplikaciju da biste im mogli pristupiti.

Iza kulisa, iOS aplikacija je u stvari direktorijum koji sadrzi mnogo fajlova: sam binarni (koji je kompajlovana verzija vaseg koda, spremna za rad), sve medijske sadrzaje koje vasa aplikacija koristi, bilo koje vizuelne layout-e koje imate plus niz drugih stvari kao sto su metadata i security entitlements(bezbednosna prava).

Ovi direktorijumi koji pripadaju aplikaciji se zovu BUNDLES, i imaju ekstenziju .app. Zato sto su nasi medijski fajlovi slobodni unutar fascikle, mozemo da trazimo da nam sistem kaze koji su sve fajlovi unutra i da onda izvucemo te koje hocemo. Mozda ste primetili da imena svih slika pocinju sa "nssl" (sto je skracenica za National Severe Storms Laboratory), sto cini nas zadatak jednostavnim: izlistati sve fajlove koji se nalaze u direktorijumu nase aplikacije i izvuci sve one koji pocinju sa "nssl"

Za sada cemo ucitati listu i odštampati je u Xcode-ov ugradjeni log viewer, ali uskoro cemo ih naterati da se pojave u nasoj aplikaciji.

Prema tome, prvi korak: otvorite ViewController.swift. O View Controleru je najbolje da mislite kao o jednom ekranu koji sadrzi informacije, a za nas je to trenutno samo jedan veliki prazak ekran. ViewController.swift je odgovoran za pokazivanje tog praznog ekrana i trenutno ne sadrzi mnogo koda. Trebalo bi da vidite nesto ovako:

```
import UIKit


class ViewController: UIViewController {

    override func viewDidLoad() {

        super.viewDidLoad()

        // Do any additional setup after loading the view,
typically from a nib.

    }

}
```

Tu se nalaze 4 interesantne stvari o kojim zelim da razgovaram, pre nego sto krenemo dalje.

1. Fajl pocinje sa `import UIKit`, sto znaci "ova datoteka ce se pozivati na alate za iOS korisnicki interfejs" (engl. this file will reference the iOS user interface toolkit").

2. Red `class ViewController: UIViewController` znaci: "hocu da napravim novi ekran sa podacima koji se zove ViewController, a koji je baziran na UIViewController - u". Kada vidis tip podataka koji pocinje sa "UI" to znaci da on dolazi iz "UIKit". `UIViewController` je Apple-ov podrazumevani tip ekrana koji je prazan i beo sve dok ga mi ne promenimo.

3. Red `override func viewDidLoad()` zapocinje metodu. Kao sto znate, rec override je potrebna jer ona znaci "zelimo da promenimo Apple-ovo podrazumevano ponasanje od UIViewController-a. viewDidLoad ( ) je pozvan od strane UIKit kada se ekran ucita i spreman je za prilagodjavanje nasim potrebama(eng. customization)

4. Metod `viewDidLoad ( )` sadrzi jedan red koda koji glasi super.viewDidLoad() i jednu liniju sa komentarima (to je red koji pocinje sa //). Ovo super znaci "kazi Apple-ovom UIViewController-u da izvrsi prvo svoj kod pre nego sto ja izvrsim svoj" i kao sto cete videti koristi se bas dosta.

Vraćaćemo se na ovaj kod bas mnogo u buducim projektima pa se ne brinite ako je za sada sve pomalo maglovito.

Nemate brojeve redova? Dok čitate kod, korisno je da imate omogucene brojeve redova tako da mnogo lakse mozete da ukazete na odredjen deo koda. Ako u vasem Xcode-u ovo nije automatski omogucena opcija, predlazem vam da je odmah omogucite: Idite na meni Xcode i izaberite Preferences, a onda izaberite Text Editing tab i postarajte se da je čekirana opcija Line numbers.

Kao sto sam rekao ranije, viewDidLoad ( ) se poziva kada se ekran ucitao i kada je spreman za vasu kastomizaciju. Mi cemo da stavimo jos malo koda unutar te metode da bismo ucitali NSSL slike. Dodajte ovo ispod linije gde pise super.viewDidLoad ( ).

```
let fm = FileManager.default

let path = Bundle.main.resourcePath!

let items = try! fm.contentsOfDirectory(atPath: path)


for item in items {

    if item.hasPrefix("nssl") {
```

```
            // this is a picture to load!
        }
    }
```

Napomena: Neki napredni iskusni Swift developer-i ce procitati ovaj kod, videce try! I pocece da mi pisu ljuta pisma. Ako sami pozelite to da uradite, molim vas prvo nastavite da citate.

To je velika gomila koda, od koje je vecina stvari nova. Prodjimo korak po korak kroz ono sto ova gomila koda radi:

- Red **let fm = FileManager.default** deklarise konstantu koja se zove fm i dodeljuje joj vrednost vracenu od FileManager.default. Ovo je tip podataka koji nam dozvoljava da radimo sa sistemskim fajlovima (file system) i u nasem slucaju cemo ga koristiti da trazimo datoteke.

- Red **let path = bundle.mail.resourcePath!** deklarise konstantu koja se zove path I koja je postavljena na putanju paketa nase aplilikacije (na putanju app's bundle). Zapamtite, bundle je direktorijum koji sadrzi nas kompajlirani program i sve nase podatke. Prema tome, ova linija govori "kazi mi gde da nadjem sve one slike koje sam dodao u mojoj aplikaciji".

- Red **let items = try! fm.contentsOfDirectory(at path: path)** deklarise trecu konstantu koja se zove items I koja je postavljena na sadrzaj direktorijuma koji se nalazi na putu - path. Na kom putu? Pa na onom putu koji smo napisali jedan red ranije. Kao sto vidite dugacka imena koja Apple daje svojim metodama stvarno cine njihov kod samo-opisnim. Konstanta ce biti "array of strings" koja sadrzi imena fajlova.

- Red **for item in items {** zapocinje petlju(loop) koja ce se izvrsiti jednom za svaki element koji nadjemo u paketu nase aplikacije(app's bundle). Zapamtite: red ima otvorenu vitičastu zagradu koja oznacava pocetak novog bloka koda i zatvorenu vitičastu zagradu 4 reda nize. Sve unutar tih zagrada ce biti izvrseno svaki put kada petlja bude prolazila kroz kod.

- Red **if item.hasPrefix("nssl") {** je prva linija unutar nase petlje. Do ovog trenutka vec imamo prvi filename spreman za rad i koji ce se zvati item. Da bismo odlucili da li je to file koji je nama potreban ili ne, koristicemo metodu hasPrefix ( ): ova metoda uzima jedan parametar(prefix koji trazi) i vraca nazad ili true ili false. Ono "if" na pocetku reda znaci da je ovo uslovna izjava(conditional statement): ako ("if") red ima prefix "nssl", onda… sledi jos jedna otvorena zagrada koja oznacava novi blok koda. Ovaj put kod ce biti izvrsen samo ako hasPrefix ( ) vrati nazad true.

- Na samom kraju - red // this is a picture to load! je komentar - ako stignemo dovde , item sadrzi ime slike koju cemo ucitati iz naseg paketa aplikacija (bundle) tako da nam je potrebno da ga negde uskladistimo.

U ovom slucaju savrseno je u redu da koristimo **Bundle.main.resourcePath!** i **try!** Zato sto ako se ovaj kod ne izvrsi, to znaci da nasa aplikacija ne moze da čita sopstvene podatke što ukazuje na to da nesto ozbiljno nije u redu. Neki Swift developeri pokusavaju da napišu kod da bi izbegli ove katastrofalne greske za vreme izvrsavanja koda (at runtime), ali veoma često, na žalost, oni samo maskiraju pravi problem koji iza toga lezi.

Trenutno, nas kod ucitava listu fajlova koji se nalaze unutar naseg paketa aplikacije(apps bundle), zatim petljom prolazi kroz njih trazeci one cije ime pocinje na "nssl". Medjutim, nas kod zapravo nista ne radi sa tim fajlovima(slikama), koje je pronasao, tako da je nas sledeci korak da napravimo array od svih tih "nssl" slika, da bismo mogli da se pozovemo na njih kasnije, a ne svaki put da ponovo prolazimo kroz nas direktorijum resursa iznova i iznova.

Tri konstante koje smo upravo stvorili (fm, path i items) zive unutar metode viewDidLoad () i bice unisteni cim se ovaj metod zavrsi. Mi zapravo zelimo da nadjemo nacin na koji cemo da prikacimo(privezemo) nase podatke celom ViewController-u tako da oni postoje sve vreme dok postoji i nas ekran. Ovo je zapravo savrsen primer kada treba koristiti osobine (property) - mozemo da damo nasoj klasi ViewController koliko god zelimo ovakvih osobina i onda da ih citamo i pisemo koliko god cesto je to potrebno dok ekran postoji.

Da bismo napravili property potrebno je da ga deklarisemo izvan metode (nikako unutar). Do sada smo deklarisali konstante koristeci naredbu let ali ovaj array ce se menjati unutar nase petlje tako da ga moramo deklarisati kao promenjivu. Isto tako, moramo reci Swiftu kakav tip podata ce on da drzi - u nasem slucaju to je array of strings gde ce svaki clan da bude ime nase slike "nssl".

Dodajte ovaj red koda odmah pre viewDidLoad ( ):

```
var pictures = [String]()
```

Ako ste pravilno postavili taj red, vas kod bi trebalo da izgleda ovako:

```
class ViewController: UIViewController {

    var pictures = [String]()

    override func viewDidLoad() {

        super.viewDidLoad()

        let fm = FileManager.default
```

Ovaj array slika ce biti stvoren istovremeno kada se stvori i ekran ViewController i postojace sve vreme dokle god postoji i sam ekran. On ce trenutno biti prazan zato sto ga jos uvek nismo napunili ni sa čim, ali ako nista drugo bar je tu, spreman da ga popunimo.

Sta mi zaista zelimo je da u array slika dodamo sve fajlove koji ispunjavaju uslov nase petlje. Da bismo to uradili moramo da zamenimo nas postojeci komentar (//this is a picture to load!) sa kodom koji ce da doda svaku sliku u pictures array.

Na svu srecu Swift-ovi array imaju ugradjenu metodu koja se zove append koju mozemo da iskoristimo i dodamo bilo koje clanove koje zelimo. Sada zamenite komentar //this is a picture to load! sa ovim:

```
pictures.append(item)
```

To je sve! Uznemirujuce je sto posle toliko posla nasa aplikacija izgleda da ne radi nista kada pritisnete play - videcete isti beli ekran koji ste videli i ranije. Da li smo uspeli, ili smo tiho potunuli?

Da bismo saznali unesite ovaj kod na kraju viewDidLoad () odmah pre zatvorene zagrade:

```
print(pictures)
```

To govori Swiftu da odstampa sadrzaj pictures - a na Xcode- ovu konzolu za otklanjanje gresaka (debugg console). Kada pokrenete program sada trebalo bi da vidite kako se ovaj tekst pojavljuje u dnu vaseg ekrana : “["nssl0033.jpg", "nssl0034.jpg", "nssl0041.jpg", "nssl0042.jpg", "nssl0043.jpg", "nssl0045.jpg", "nssl0046.jpg", "nssl0049.jpg", "nssl0051.jpg", “nssl0091.jpg”]”

Napomena: iOS voli da štampa gomilu neinteresantnih poruka u konzolu za otklanjanje gresaka. Ne brinite ako unutra vidite gomilu drugog teksta koji ne prepoznajete - samo pomerajte ekran dok ne vidite gornji tekst i ako ga vidite onda ste spremni da nastavite dalje.

Designing our interface

(Dizajniranje naseg interfejsa)

Nasa aplikacija ucitava slike oluje korektno ali ne radi nista zanimljivo sa njima. Štampanje na konzolu za otklanjanje gresaka je velika pomoc u otklanjanju gresaka ali vam obecavam da nije dovoljno da biste napravi sjajnu aplikaciju koja se dobro prodaje.

Da bismo ovo popravili, nas sledeci korak je da stvorimo korisnicki interfejs koji izlistava slike tako da kosrisnici mogu da izaberu jednu od njih. UIKit - iOS korisnicki okvir interfejsa (engl. Inteface framework) - ima veoma mnogo ugradjenih alata za korisnike koje mozemo koristiti da napravimo mocne aplikacije koje izgledaju i rade onako kako to korisnici ocekuju.

Za ovu aplikaciju nasa glavna komponenta korisnicnog interfejsa se zove UITableViewController. On je baziran na osnovi UIViewController-a - Apple-ov najsonovniji tip ekrana - koji pored svoje jednostavnosti ipak daje mogucnost pokazivanja redova podataka koji mogu biti skrolovani (scrolled) i izabrani. UITableViewController mozete videti u mnogim aplikacijama kao sto su Settings, Mail, Notes, Health, i mnogim drugim - on je mocan kontroler, fleksibilan i ekstremno brz tako da nije ni čudo zasto se nalazi u toliko mnogo aplikacija.

Nas postojeci ViewController ekran je baziran na UIViewController-u Ali mi zelimo da umesto njega, on bude baziran na UITableViewController-u. Ovo ne zahteva mnogo posla ali cete upoznati novi deo Xcoda koji se naziva Interface Builder (skraceno IB).

Preci cemo na Interface builder za nekoliko trenutaka ali prvo moramo da unesemo jednu malu izmenu u ViewController.swift. Nadjite ovaj red:

```
class ViewController: UIViewController {
```

To je red koji kaze “napravi novi ekran koji se zove ViewController I neka on bude baziran na Apple-om UIViewController ekranu”. Zelim da ga promenita na sledece:

```
class ViewController: UITableViewController {
```

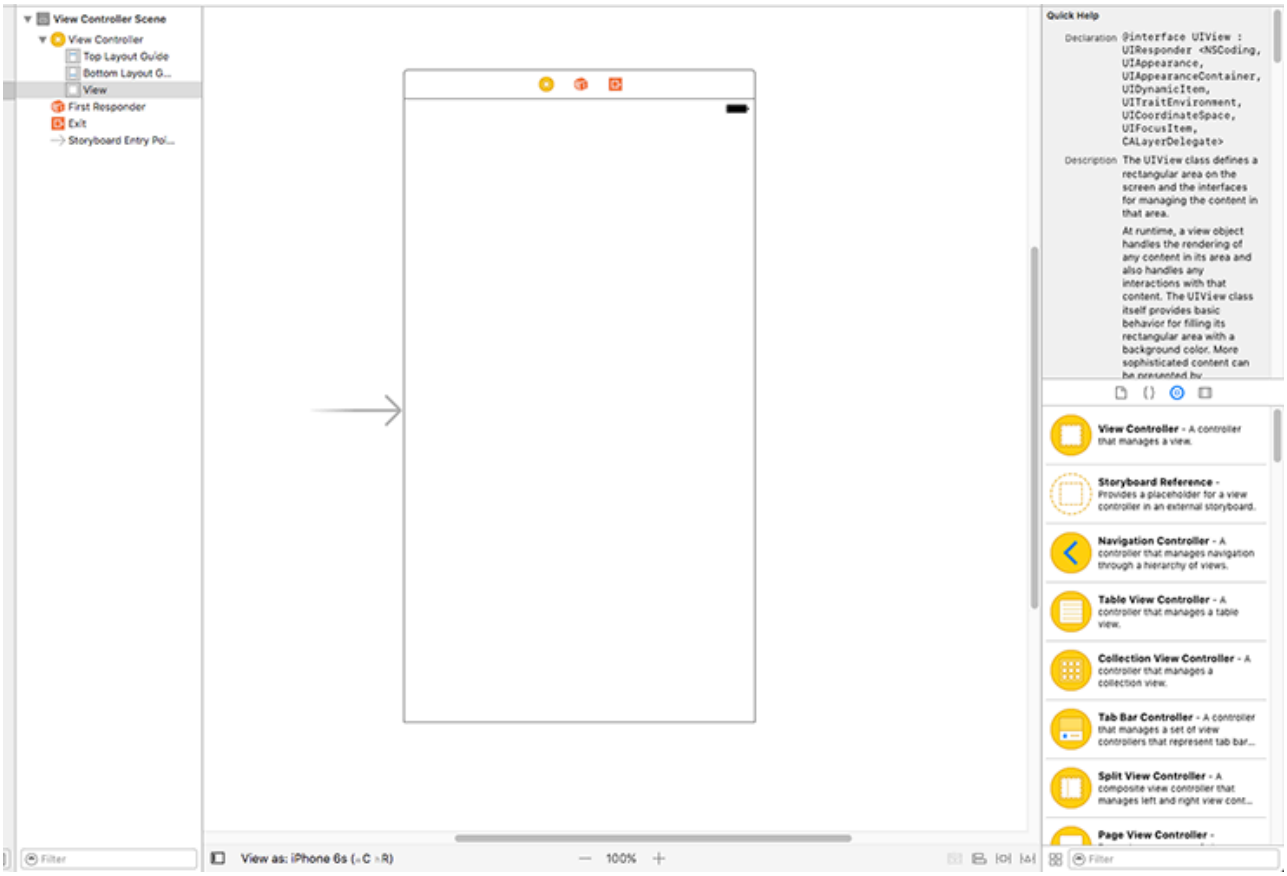
To je samo mala ali veoma vazna razlika: ona znaci da ViewController sada nasledjuje svoju funkcionalnost od UITableViewController umesto od UIViewController-a, koja nam pruza veoma veliku kolicinu dodatnih funkcionalnosti kao sto cete videti za koji trenutak.

Iza kulisa, UITableViewController se I dalje gradi na vrhu UIViewController-a. Ovo se zove hijerarhija klasa i to je uobicajen nacin za brzu izgradnju funkcionalnosti.

Promenili smo kod za ViewController tako da se on sada gradi na UITableViewController-u ali isto tako moramo i paralelno da promenimo korisnicki interfejs da bi se oni slagali. Korisnicki interfejs moze u potpunosti biti napisan u kodu ako vi to zelite i mnogi developeri rade upravo to. Medjutim, veoma češće korisnicki interfejs se stvara koriscenjem grafickog editora koji se zove Intefrace Builder. Mi moramo da "kazemo" IB -u da je ViewController zapravo table view controller, da bi se promene poklapale sa onima koje smo naravili u nasem kodu.

Sve do ovoga trenutka, mi smo sve radili unutar ViewController.swift ali sada hocu da iskoristite project navigator (panel sa leve strane) i da tamo izaberete fajl Main.storyboard. Storyboard-i sadrze korisnicki interfejs za vasu aplikaciju i omogucavaju vam da vizualizujete ili deo ili celu aplikaciju na jednom ekranu.

Kada izaberete Main.storyboard, prebacicete se u vizuelni editor Interface Builder-a I trebalo bi da vidite nesto kao sto je na donjoj slici:



Taj veliki beo prostor je ono sto proizvodi veliki beli prostor kada se aplikacija pokrene. Ako ubacite nove komponente na taj prostor, one ce biti vidljive kada se aplikacija pokrene. Medjutim, mi ne zelimo to da uradimo - u stvari mi uopste ne zelimo taj veliki beo prostor, tako da cemo da ga izbrisemo.

Najbolji nacin za gledanje, selekciju, edit-ovanje i brisanja items-a u Interface Builder-u je da se koristi prozor document outline ali postoji dobra sansa da je on sakriven od vas tako

da cemo prvo njega da pokazemo. Idite na Editor menu i izaberite Show Document Outline - verovatno je treca opcija od vrha. Ako umesto toga vidite Hide Document Outline, to znaci da je document outline vec vidljiv.

Document Outline vam pokazuje sve komponente na svim ekranima vaseg storyboarda. Trebalo bi da vidite "View Controller Scene", molim vas da je selektujete i pritisnete Backspace na vasoj tastaturi da biste je uklonili.

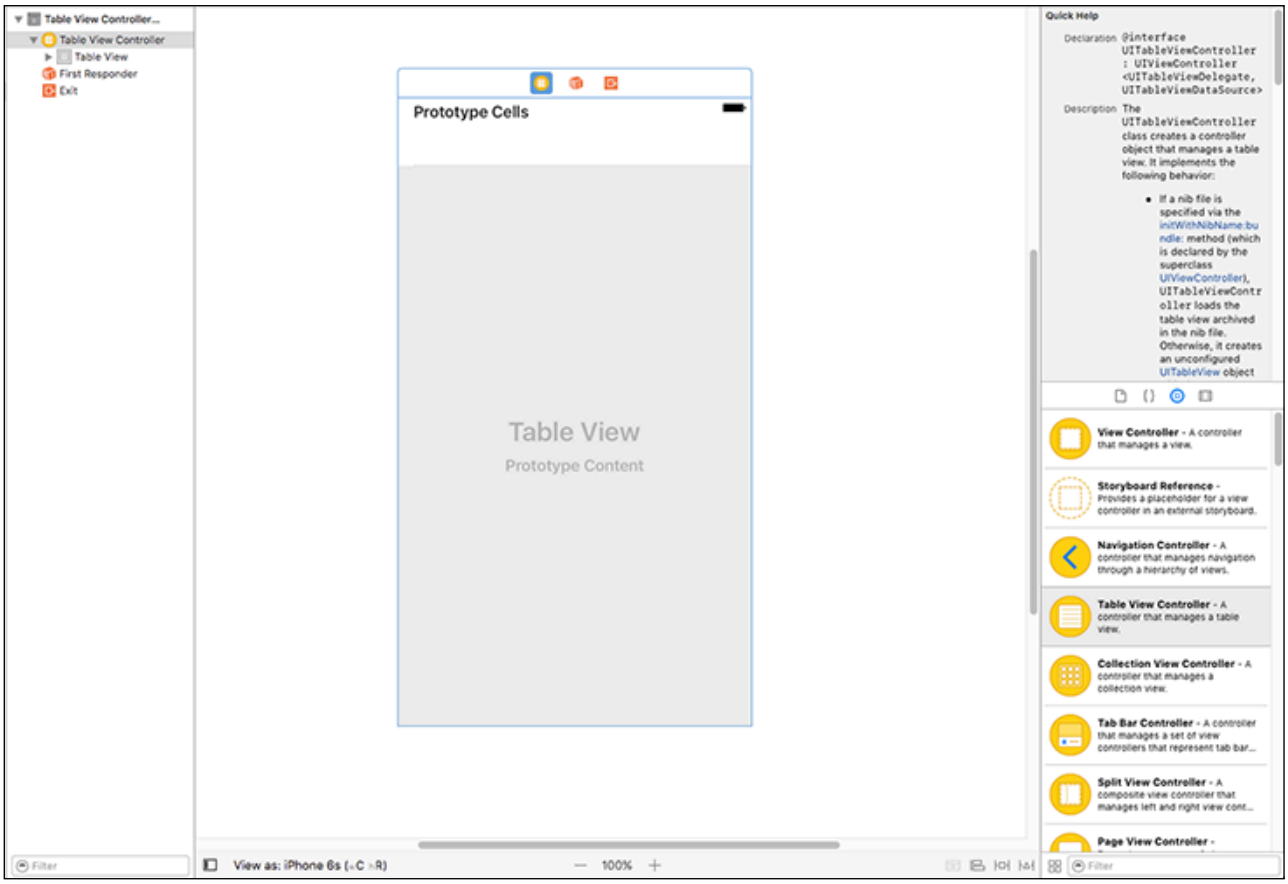
Umesto dosadnog i starog UIViewController-a mi hocemo fensi novi UITableViewController da se poklopi sa promenama koje smo napravili u nasem kodu. Da bismo ga stvorili, pritisnite Cmd+Shift+L da biste otvorili biblioteku sa objektima (object library). Ako ne volite precice preko tastature, umesto toga mozete da odete na View menu i da odaberete Libraries >Show library.

Biblioteka sa objektima je sada otvorena preko prozora Xcoda-a i sadrzi selekciju grafickih komponenti koje mozete izabrati, prevući i preuredti prema sopstvenim zeljama. Ona sadrzi veoma mnogo komponenata tako da je bolje da unesete par pocetnih slova u "Objects" da biste suzili izbor.

Savet: Ako zelite da bilioteka sa objektima bude otvorena posle toga kad nesto izvucete iz nje, koristite Alt+Cmd+Shift+L posle cega ce ona biti pokretni prozor(movable) kome cete moci da menjate velicinu(resizable).

Trenutno, komponenta koju mi zelimo se zove Table View Controller. Ako ukucate "table" unutar Filter box-a videcete Table View Controller, Table View, i Table View Cell. Svi oni su medjusobno razliciti prema tome postarajte se da izaberete Table View Controller - on ima zutu pozadinu na svojoj ikonici.

Pritisnite na komponentu Table View Controller, prevucite je napolje na veliki beo prostor koji se postoji na mestu gde je bio prethodni view controller. Kada otpustite dugme posle prevlacenja na platno storyboarda, on ce se pretvoriti u ekran koji izgleda kao na slici nize:



Finishing touches for the user interface
(Zavrsno glancanje korisnickog interfejsa)

Pre nego sto zavrsimo ovde moramo da napravimo nekoliko manjih izmena.

Prvo moramo da kazemo Xcodu da da je ovaj storyboard table view controller isti onaj koji imamo napisanog u kodu unutar ViewController.swift. Da bismo to uradili pritisnite Alt+Cmd+3 da aktivirate Identity Inspector (ili idite na View > Utilities > Show Identity Inspector) I onda pogledajte na samom vrhu kutiju koja se zove "Class". Tamo ce svetlo sivim tekstom biti napisan "UITableViewController" ali kada kliknete na njegovu desnu stranu treba da viditi padajuci meni koji sadrzi "ViewController" - molim vas da ga odmah izaberete.

Drugo, moramo da kazemo Xcode - u da je ovaj novi table view controller je upravo to sto treba biti pokazano prvo kada se aplikacija pokrene. Da biste to uradili pritisnite Alt+Cmd+4 da biste ukljucili Atribute inspector (ili idite na View > Utilities > Show Atributes Inspector). Odmah zatim potrazite polje za potvrdu koje se zove "Is Inital View Controller" i uverite se da je čekirano.

Trece, hocu da iskoristite document outline da biste pogledali unutar novog table view controller-a. Unutra biste trebali da vidite "Table View" koji dalje sadrzi "Cell". Table view

cell je odgovorna za pokazivanje jednog reda podataka u tabeli i mi upravo i hocemo da pokazemo pojedinacno ime svake slike u po jednoj celiji.

Molim vas izaberite "Cell", a onda u Atributes inspector unesite tekst "Picture" unutar polja za tekst oznacenog kao Identifier. Dok se nalazite tu promenite Style option na koji se nalazi na vrhu Atributes inspectora - trebalo bi da je po default-u podesen na Custom, a vi ga promenite na Basic.

Na samom kraju, cemo ceo table view controller da postavimo u nesto drugo. To je nesto sto ne moramo da konfigurisemo ili da brinemo o njemu; veoma uobicajen element korisnickog interfejsa i mislim da cete ga prepoznati odmah. Naziva se navigacioni kontroler koga mozete videti u akciji u aplikacijama kao sto su Settings i Mail - on obezbedjuje tanku sivu traku na vrhu ekrana i odgovoran je za klizecu animaciju sa desna na levo koja se pojavljuje kada klizite(sliding) izmedju ekrana na iOS.

Da bismo postavili nas table view controller na navigacioni kontroler, sve sto treba da uradite je da idete na Editor menu i da izaberete Embed in > Navigation Controller. Interface Builder ce pomeriti vas postojeci view controller na desno I dodati navigacioni kontroler oko njega - trebalo bi da vidite simuliranu sivu traku vise vaseg table view. Interface Builder ce takodje pomeriti "Is Initial view Controller" property na navigacioni kontroler.

U ovom trenutku ste vec uradili dovoljno da biste pogledali rezultate vaseg rada: pritisnite Play dugme ili Cmd + R ako zelite da se osecate malo napredno. Kada se vas kod pokrene, sada ce te videti obicno belo polje zamenjeno sa velikim praznim table view. Ako kliknete na nju i vučete vaseg miša unaokolo, vidcete da on klizi i odskace onako kako biste i ocekivali, iako unutra jos uvek nema nikakvih podataka. Takodje bi trebalo da vidite sivu navigacionu traku na vrhu; to ce biti vazno malo kasnije.

Showing lots of rows
(Prikazivanje mnogo redova)

Sledeci korak je da nateramo table view da pokaze neke podatke. Konkretno, zelimo da on pokaze listu "nssl" slika, jednu za svaki red. Apple-ov tip podataka - UITableViewController obezbedjuje podrazumevano ponasanje za mnogo stvari ali u ovom slucaju je podrazumevano da postoji 0 redova.

Nas ViewController ekran je izgradjen na bazi UITableViewController-a I moze da premosti podrazumevano (default) ponasanje Apple-og table view I da obezbedi kastomizacuju tamo gde je ona potrebna. Vi samo treba da premostite delove koje zelite; sve podrazumevane vrednosti su osetljive.

Da biste naterali table da pokaze redove, moramo da premostimo dva ponasanja: koliko redova ce da bude pokazano i sta ce svaki red da sadrzi. Ovo se radi pisanjem dve

specijalno imenovane metode ali ako ste novi u swiftu one mogu izgledati malo cudno za vas. Da bih bio siguran da svi mogu da prate, obajsnicu ovo detaljno - ipak je ovo vas prvi projekat!

Pocnimo sa metodom koja podesava koliko ce redova da se pojavi u table. Dodajte ovaj kod odmah posle kraja viewDidLoad ( ) - ako pocnete da kucate "numberof" mozete da iskoristite autokomplektaciju Xcoda koja ce da uradi veliki deo posla za vas:

```
override func tableView(_ tableView:
UITableView,numberOfRowsInSection section: Int) -> Int {
    return pictures.count
}
```

Napomena: to mora da bude napisano POSLE KRAJA viewDidLoad (), sto znaci posle zatvorenih viticastih zagrada.

Ovaj metod sadrzi rec "table view" cak tri puta, sto je u pocetku veoma zbunjujuce, takoda cemo objasniti sta to zapravo znaci:

- Rec **override** znaci da je ovaj metod vec definisan ranije ali mi zelimo da redefinisemo njegovo postojece podesavanje sa novim ponasanjem. Ako ga ne biste premostili onda bi se prethodno definisani metod izvrsio i na ovoj trenutnoj stadiji bi rekao da zapravo i nema redova.

- Rec **func** zapocinje novu funkciju ili novi metod; swift koristi istu rec za oba. Tehnicki receno, metod je zapravo funkcija koja se pojavljuje unutar klase, kao i nas ViewController, u suprotnom nema nikakve razlike.

- Sledece je ime metoda: **tableView**. To ne zvuci veoma korisno ali nacin na koji Apple definise metode je da osigura da se informacija koja se u njih unese - parametri - nazove korisno, a u ovom slucaju prva stvar koja se unese unutra je table view koji je pokrenuo kod. Table view, kao sto ste verovatno vec i sami zakljucili, je klizajuca stvar koja ce da sadrzi imena svih nasih slika, i istovremeno komponenta koja je srž u iOS-u.

- Sledeca stvar je tableView: UITableView, koji je table view koji je pokrenuo kod. Ali ovo sadrzi dva dela informacije odjednom: tableView je ime koje mozemo da koristimo da ukazemo na table view unutar metode, i UITableView je tip podataka - deo(bit)koji opisuje sta je to.

- Najvazniji deo metode dolazi sledeci: **numberOfRowsInSection section: Int**. Ovo opisuje sta metod zapravo radi. Mi znamo da to sto on radi ukljucuje table vew zato sto je to ime metode ali deo koji je zapravo akcija je numberOfRowsInSection: ovaj deo koda ce biti aktiviran kada iOS bude zeleo da sazna koliko ima redova u table view. Deo **section** je tu zato sto table views mogu biti razdeljeni na sekcije, na nacin kao sto aplikacija Contacts deli imena po prvom slovu. Mi imamo samo jednu sekciju

tako da mozemo ignorisati ovaj broj. Deo Int znaci “ovo ce biti intidzer” sto znaci ceo broj kao sto je 3, 30, 3568 itd.

- Na samom kraju **-> Int** znaci “ovaj metod mora da vrati intidzer”, sto ce zapravo biti broj redova koji ce postojati u table.

Postoji jos jedna stvar koju sam izostavio, a to sam uradio sa razlogom jer je to pomalo zbunjujece na ovom stadijumu vase swift karijere. Da li ste primetili _ unutra? Nadam se da se secate da to znaci da prvi parametar nije prosledjen unutra koristeci ime kada se pozove eksterno - ovo je ostatak od Objective-C, gde je ime prvog parametra obicno bilo ugradjeno u ime metode.

U ovom slucaju, metod se zove tableView () zato sto je prvo parametar table view sa kojim mi radimo. Ne bi imalo mnogo smisla da se pise tableView(tableView: someTableView), tako da koriscenjem donje crte znaci da biste umesto toga vi napisali tableView(someTableView).

Necu da se pretvaram da je lako razumeti kako metode u swiftu izgledaju i rade, ali najbolja stvar koju mozete da uradite trenutno je da se ne brinete mnogo ako ne razumete sve bas najbolje zasto sto ce posle nekoliko sati kodiranja sve doci na svoje.

Najmanje sto treba zapamtiti je da se ove metode pozivaju ukazujuci njihovo ime (tableView) i bilo koji imenovan parametar. Parametri bez imena su referencirani kao donje linije _. Prema tome, da biste mu dali njegovo puno ime, metod koji ste upravo napisali se poziva ovako **tableView(_ :numberOfRowsInSection:)** - glomazno, znam, upravo zbog cega gomila ljudi govori samo o delu koji je zapravo vazan na primer, “u metodi **numberOfRowsInSection**”.

Napisali smo samo jedan red koda u metodi koji je glasio return. picures.count. To znaci “posalji nam nazad broj slika u nasem nizu(array)” tako da mi zapravo trazimo da bude onoliko redova koliko ima slika u array.

Dequeuing cells
(Uklanjanje ćelija)

To je prvi od dve metode koji treba da napisemo da bismo zavrsili ovu fazu pravljenja aplikacije. Drugi je da specifiramo kako svaki red treba da izgleda, a to prati sličnu konvenciju davanja imena kao u prethodnoj metodi. Dodajte ovaj kod sada:

```
override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
"Picture", for: indexPath)
    cell.textLabel?.text = pictures[indexPath.row]
    return cell
```

```
}
```

Ponovo cemo razbiti prethodan kod na delove da biste videli kako sve to funkcionise.

Prvo, **override func tableView(_ tableView: UITableView** je identican kao i prethodni metod: ime metode je tableView ( ) I ona ce da primi table view kao svoj prvo parametar. Donja crta _ znaci da ona ne mora da ima eksterno ime, zato sto je ono isto kao i ime metode.

Drugo, **cellForRowAt indexPath: IndexPath** je vazan deo imena metode. Metod se zove cellForRowAt i bice pozvan kada vi treba da obezbedite red. Red koji treba da se pokaze je specifiran u parametru: indexPath koji je tipa IndexPath. IndexPath je tip podataka koji sadrzi i i broj sekcije i broj reda. Mi imamo samo jednu sekciju tako da taj deo mozemo ignorisati i koristiti samo broj reda.

Trece **-> UITableViewCell** znaci da ovaj metod mora da vrati table view cell. Ako se secate mi smo vec stvorili jednu unutar Interface builder-a i dali smo mu identifier " Picture" koji cemo sada da koristimo.

Ovo je mesto gde dolazi do izrazaja iOS magija: ako pogledate na aplikaciju Settings, videcete da u bilo koje vreme, na ekran moze da stane samo oko 12 redova, sto zavisi od veličine vašeg telefona.

Da bi sacuvao vreme CPU i RAM, iOS stvara samo onoliko redova koliko mu je potrebno za rad. Kada se jedan od redova pomeri za granicu ekrana, iOS ce ga uzeti i postaviti u red za ponovno koriscenje, spremnog da bude recikliran u novi red koji dolazi odozdole. Ovo znaci da mozete da pregledate stotine redova u sekundi, dok se iOS ponasa lenjo, i da izbegava da stvara nove table view ćelije - on jednostavno reciklira postojece.

Ova funkcionalnost je implementirana pravo u iOS i upravo je to sto nas kod radi na ovoj liniji:

```
let cell = tableView.dequeueReusableCell(withIdentifier:
"Picture", for: indexPath)
```

To stvara novu konstantu koja se zove cell, uklanjanjem reciklirane celije sa table. Moramo da mu damo identifier tipa celije koji moramo da recikliramo, tako da cemo da unesemo isto ime koje smo dali Inteface Builder-u: "Picture". Takodje ubacujemo i index path koji je bio trazen; ovo interno koristi table view.

To ce nam vratiti table view cell sa kojom mozemo raditi da prikazemo informaciju. Mozete da dizajnirate svoj sopstveni table view cell ako zelite(vise o tome mnogo kasnije!). ali mi koristimo ugradjen Basic stil koji ima oznaku teksta (text label). Tu nastupa red broj dva: on daje oznaci teksta ćelije(text label of the cell) isti tekst kao sto je naziv slike u nasem nizu(array). Evo ga kod:

```
cell.textLabel?.text = pictures[indexPath.row]
```

Čelija ima property koje se zove textlabel, ali ona je optional: moze postojati textLabel ili ne mora postojati - ako biste na primer, dizajnirali svoj sopstveni. Radije nego da pisete kod i proveravate da li text label postoji ili ne, swift nam dozvoljava da koristimo znak pitanja - textLabel? - sto znaci "uradi ovo, ako text label stvarno postoji tamo negde, ili u suprotnom ne radi nista.

Mi zelimo da podesimo da oznaka teksta(label text) bude korektno ime iz naseg niza slika(pictures array) i to je upravo ono sto ovaj kod radi. indexPath.row ce da sadrzi broj reda koji se od nas trazi da unesemo, tako da cemo da koristimo to da procitamo odgovarajucu sliku iz pictures, i postavimo je u oznaku slika celije(cell's text label).

Poslednja linija u metodi je return cell. Setite se ova metoda ocekuje da se vrati table view cell, tako da moramo da posaljemo nazad taj koji smo stvorili - to radi return cell.

Sa ove dve zapravo male metode na mestu, mozete da pokrenete vas kod ponovo i da vidite kako izgleda. Ako sve bude dobro, trebalo bi da vidite 10 view cells, i svaku sa drugacijim imenom slike unutra. Ako kliknete na bilo koju od njih ona ce posiveti, ali nista drugo se nece desiti. Ajde to da popravimo sada…

Building a detail screen

(Izgradnja detaljnog ekrana)

U ovom trenutku u nasoj aplikaciji imamo listu slika koje mozemo da izaberemo, ali iako mozemo da kliknemo na njih nista se zapravo ne desava. Nas sledeci cilj je da dizajniramo novi ekran koji ce se otvoriti kada korisnik klikne na bilo koji red. Nateracemo taj novi ekran da pokaze izabranu sliku preko celog ekrana, klizeci automatski u stranu kada kliknete na sliku.

Ovaj zadatak se moze podeliti na dva manja zadatka. Prvo, moramo da napisemo novi kod koji ce da bude "domacin" ovom novom detaljnom ekranu. Drugo, moramo nacrtati korisnicki interfejs za ovaj ekran unutar Inteface Builder-a.

Pocnimo sa laksim delom: kreiranje novog koda koji ce da drzi detaljni ekran. U meniju idite na File menu i izaberite New > File, i otvorice se prozor pun opcija. Sa te liste izaberite iOS > Cocoa Touch Class i onda kliknite na Next.

Od vas ce se traziti da imenujete novi ekran i takodje da kazete iOS-u na cemu cete ga graditi. Molim vas unesite "Detail View Controller" za ime i "UIViewController" za "SubclassOf". Postarajte se da "Also create XIB file" ostane deselektovano, a zatim kliknite na Next i Create da biste dodali novi fajl.

Prvi deo posla je zavrsen - imamo novi fajl koji ce da sadrzi kod za detaljni ekran.

Drugi zadatak trazi malo vise razmisljanja. Vratite se na Main.storyboard, gde cete videti nasa dva postojeca view controller-a: navigacioni kontroler sa leve strane i table view controller sa desne strane. Sada cemo da dodamo novi view controller - novi ekran - koji ce biti nas detaljni ekran.

Prvo, otvorite biblioteku sa objektima i nadjite tamo "view controller". Prevucite ga na prazan prostor udesno od vaseg postojeceg view controller-a. Mozete ga postaviti bilo gde ali je lepo drzati vase ekrane uredjene tako da prate logicni raspored sa leva na desno.

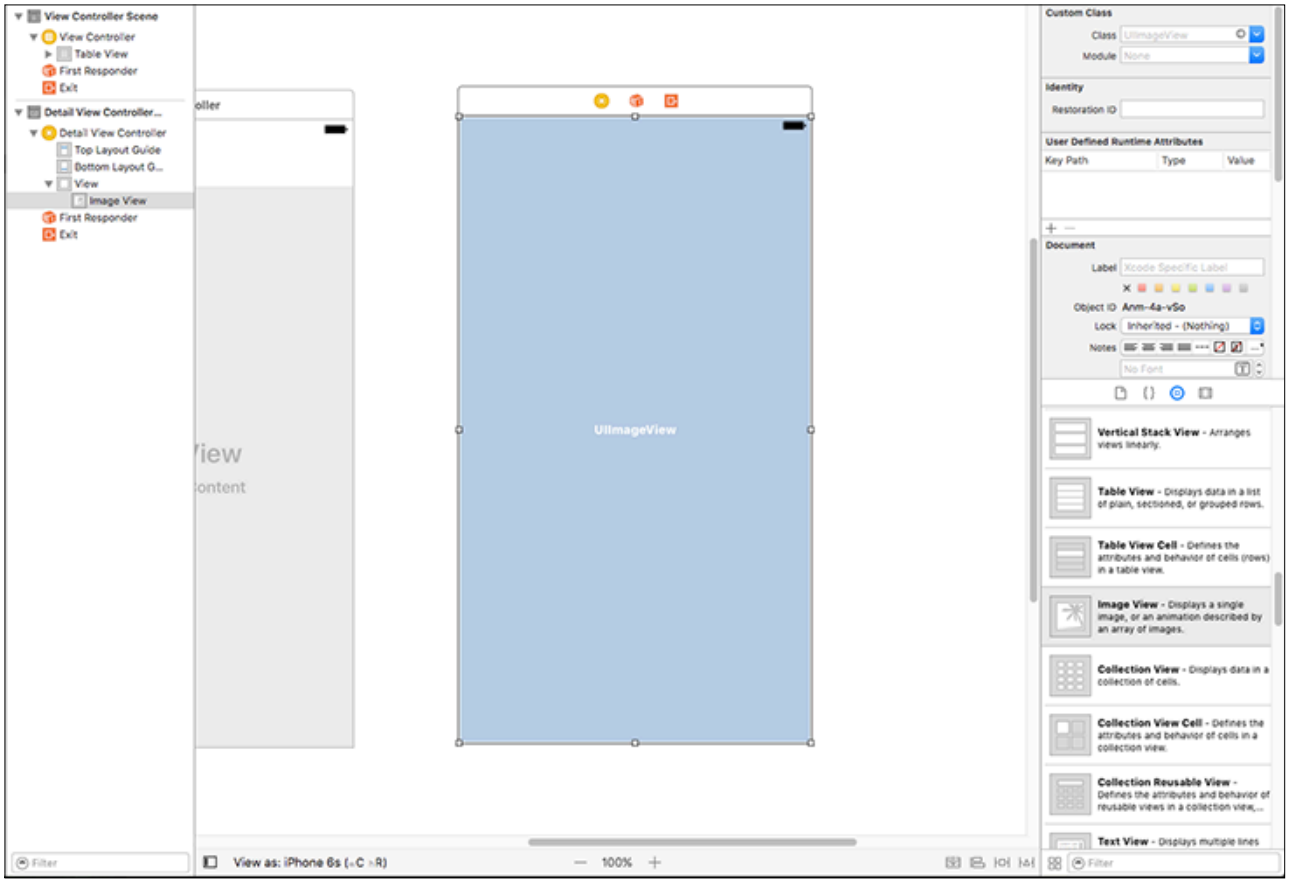
Sada kada pogledate unutar kontura dokumenta(document outline), videcete da se pojavio drugi "View Controller scene": jedan je za tabelarni prikaz (table view), a drugi je za detaljni prikaz (detail view). Ako niste sigurni koji je koji, samo kliknite na na taj novi ekran - unutar tog velikog belog praznog prostora koji je upravo stvoren - i on bi trebalo da izabere odgovarajucu scenu unutar kontura dokumenta.

Kada smo prethodno stvorili nasu table view cell, dali smo joj identifikator tako da je mozemo ucitati u kod. Moramo da uradimo istu stvar za ovaj novi ekran. Kada ste ga izabrali pre nekoliko trenutaka, on je postao oznacen u kontruru dokumenta(document outline). Iznad toga se nalazi "View Controller" sa zutom ikonicom pored - kliknite na to da biste izabrali ceo view kontroller.

Da biste ovom kontroleru dali ime idite na identity inspector tako sto cete stisnuti Cmd+Alt+3 ili koristeci meni. Sada unesite "Detail" tamo gde je napisano "Storyboard ID". To je sve: sada kada pisemo kod mozemo se pozvati na ovaj kontroler koristeci njegov ID -> "Detail". Kad ste vec tu, molim vas kliknite na strelicu pored Class box i izaberite "DetailViewController" tako da se nas korisnicki interfejs poveze sa novim kodom koji smo ranije napravili.

Sada ide interesantan deo: mi zelimo da ovaj ekran pokaze sliku izabranu korisnikom, veliku i lepu tako da moramo da koristimo novu komponentu korisnockog interfejsa koja se zove "UIImageView". Kao sto mozete zakljuciti iz imena, ona je deo UIKit-a(otuda "UI") i odgovorna je za prikaz slika - savrseno!

Pogledajte u biblioteku objekata i nadjite Image View; verovatno ce vam biti lakse ako budete koristili polje za pretragu ponovo. Kliknite i prevucite image view iz biblioteke na detail view controller i zatim otpustite. Sada rasirite njegove ivice tako da popune ceo view controller.



Ovaj image view trenutno nema nikakav sadrzaj i je ispunjen bledo plavom pozadinom i recima UIImageView. Trenutno mu necemo dodeljivati nikakav sadrzaj, mada je to nesto

sto cemo uraditi kada se program pokrene. Za sada moramo da kazemo image view kako da prilagodi svoju velicinu nasem ekranu, bilo da je to iPhone ili iPad.

Ovo moze da izgleda pomalo čudno zato sto smo ga upravo postavili da popuni view controller i ima istu velicinu kao i view controller, tako da bi to trebalo da bude to, zar ne? Pa ne bas sasvim! Razmislite malo o tome: postoji mnogo iOS uredjaja na kojima vasa aplikacija moze da radi i svi oni su razlicitih velicina. Prema tome, kako ce image view da da se pokaze kada ga pokazuju na iPhone 6 plus ili cak na iPad-u?

iOS ima brilijantan odgovor na ovo koji na mnogo nacina radi kao magija da uradi to što je vama potrebno. Odgovor je Auto Layout: on vam omogucava da definisete pravila po kojima ce vasi views biti pokazani, i automatski se stara da se ta pravila postuju.

Ali - I ovo je veliko ALI! - Auto Layout ima dva svoja pravila, od kojih oba morate ispostovati upravo vi:

- Vasa layout pravila moraju biti potpuna. To znaci da ne mozete da obezbedite samo X poziciju za nesto, a morate da obezbedite i Y poziciju. Ako je proslo neko vreme od toga kada ste bili u skoli X pozicija je pozicija levo od ekrana, a Y je pozicija od vrha ekrana.

- Vasa layout pravila ne smeju da protivrece jedna drugom. To jest, ne mozete da napisete da je view 10 tacaka udaljen od leve ivice, 10 tacaka od desne i 1000 tacaka sirok. Ekran iPhone-a 5 je sirok samo 320 tacaka, tako da ce vas layout sa tim parametrima biri matematicki nemoguć. Auto Layout ce pokusati da se izvuce iz ovih problema kršeći pravila sve dok ne nadje resenje ali zavrsni rezultat nikad nece biti to sto ste vi zeleli.

Auto Layout pravila bolje poznati pod nazivom "Constraints" mozete stvoriti u potpunosti unutar Interface Buildera i on ce vas upozoriti ako ne pratite ona dva pravila. Čak ce vam i pomoci da ispravite i neke greske tako sto ce sugerisati pravilan nacin da se nesto uradi. Napomena: sugerisan nacin da za popravku moze biti ispravan ali i ne mora - koračajte pazljivo!

Mi cemo sada stvoriti cetiri ogranicenja(constraints): po jedno za vrh, dno, levo i desno od image view tako da se on rasiri i popuni view controller ne vezano za njegovu velicinu. Postoji mnogo nacina za dodavanje Auto Layout ogranicenja. Ali trenutno najlaksi nacin je da izaberete image view i onda odete na Editor meni > Resolve Auto Layout Issuses > Resset to suggested Constraints.

Videcete da se ova opcija nalazi dva puta na listi zato sto su to dve suptilno razlicite opcije ali u ovom trenutku je svejedno koju cete izabrati. Ako volite prečice za tastaturu pritisnite Shift+Alt+Cmd+= da biste postigli istu stvar.

Vizualno vas layout ce izgledati priblizno identicno kada dodate ova ogranicenja, ali postoje dve sustinske razlike. Prvo, postoji tanka plava linija koja okruzuje UIImageView

na detaljnom kontroleru prikaza (detail view controller), sto je zaprvo nacin na koji Interface Builder pokazuje da image view ima korektno definisan Auto Layout.

Druga stvar, u document outline panelu videcete novi unos za "Constraints" ispod image view. Sva cetiri ogranicenja koja su dodata su skrivena unutar te Constraints item i mozete ih rasiriti i pogledati ako ste radoznali.

Sa dodatim ogranicenjima, postoji samo jos jedna stvar koju ovde moramo uraditi pre nego sto zavrsimo sa Interface Builder-om, a to je da povezemo nas novi image view sa kodom. Vidite, imati image view unutar layout-a jednostavno nije dovoljno - ako zapravo zelimo da koristimo image view unutar koda, moramo da napravimo osobinu (property) za njega koja je privezana za layout.

Ova osobina je kao pictures array koji smo prethodno napravili, ali poseduje malo "interesantniju" sintaksu swifta koju treba da pokrijemo. Jos lukavije, ona je stvorena koristeci stvarno bizaran deo korisnicnog interfejsa dizajniranog da postavi vas mozak u petlju ako ste ranije koristili neke druge graficke IDEs.

Hajde da pocnemo, a ja cu vam usput sve objasniti. Xcode ima specijalan display layout koji se zove Assistant editor koji deli Xcode na dva dela: deo koji ste imali ranije na vrhu i slican pogled pri dnu. U ovom slucaju ce nam pokazati interface builder na vrhu i mesto za pisanje koda za detail view controller pri dnu.

Xcode odlucuje koji ce kod da pokaze na osnovu toga koja je stavka izabrana u Interface Builder-u, tako da se obavezno postarajte da je image view i dalje izabran i u meniju izaberite View > Assistant editor > Show Asisstant editor. Ili ako vise volite precice sa tastature, stisnite Alt+Cmd+Return.

Xcode moze da prikaze Assistant editor kao dva vertikalna prozora pre nego kao dva horizontalna. Nalazim da je sa horizontalnim prozorima lakse raditi - tj jedan iznad drugoga. Mozete da se prebacujete izmedju njih tako sto cete ici na View > Assistant Editor i izabrati ili Assistant Editors On Right ili Assistant Editors on Bottom.

Bez obzira koju od ove dve opcije izaberete, trebalo bi da vidite detail view controller u Interface Builder-u u jednom prozoru i source code za DetailViewController.swift u drugom prozoru.

Xcode zna da treba da učita DetailViewController.swift zato sto ste vi promenili klasu za ovaj ekran da bude "DetailViewController" malo pre nego sto ste promenili njegov Storyboard ID.

A sada o bizarnom delu korisnickog interfejsa. Zelim da uradite sledece:

1. Postarajte se da je izabran image view
2. Držite taster Ctrl na vasoj tastaturi.

3. Pritisnite dugme na vasem misu na image view I drzite ga pritisnutim - ne pustajte ga.

4. Dok istovremeno drzite pritisnute Ctrl i dugme na vasem mišu, prevucite iz image view na vaš kod - na drugi prozor u Assistant editoru.

5. Dok pomerate vas kursor trebalo bi da vidite plavu liniju koja će se rastezati od prozora gde je image view sve do vaseg koda. Rastegnite tu linuju tako da ona pokazuje izmedju **class DetailViewController: UIViewController {** i **override func viewDidLoad() {**.

6. Kada se budete nalazili izmedju njih trebalo bi da se pojavi horozontalna plava linija zajedno sa tooltip(objasnjenjem) koje kaze Insert Outlet Or Outlet Connection. Kada to budete videli mozete otpustiti i Ctrl i dugme na vasem misu(nije bitno koje cete otpustiti prvo).

Ako ste pratili ove korake trebalo bi da se pojavi balon koji ima pet polja Connection, Object, Name, Type, i Storage.

Podrazumevane(default) opcije bi trebalo da budu "Outlet" za connection, "Detail View Controller" za object, "UIImage View" za type i "Strong" za storage. Ako vidite "Weak" za storage, molim vas promenite ga na "Strong"- Xcode ce zapamtiti to podesavanje za ubuduce.

Ne dirajte ni jedno od tih podesavanja osim Name - napisite unutra "imageView". Kada zavrsite sa tim kliknite na Connect dugme i Xcode ce ubaciti red sa kodom u DetailViewController.swift. Trebalo bi da vidite sledece:

```
class DetailViewController: UIViewController {
    @IBOutlet var imageView: UIImageView!

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

Levo od novog koda u praznini pored broja reda, se nalazi sivi krug sa linijom oko njega. Ako prevedete kursor preko njega videcete da ce image view da zasvetli - taj mali krug je nacin na koji vam Xcode govori da su redovi koda povezani sa image view u vasem storyboard-u.

Dakle mi smo Ctrl-prevlacili iz Inteface Builder-a u nas Swift file, i Xcode nam je napisao red koda kao rezultat tog prevlacenja. Neki delovi tog koda su novi, pa hajde da objasnimo sve deo po deo:

- **@IBOutlet**: Ovaj atribut se koristi da kaze Xcod-u da postoji veza izmedju ove linije koda i Inteface Builder-a
- var: ovim deklarisemo novu promenljivu (variable) ili variable property
- **imageView**: ovo je bilo ime osobine dodeljene UIImageView. Primetite kako su koriscena velika slova: varijable i konstante treba da pocinju malim slovima, a zatim upotrebite veliko slovo na pocetku svake naredne reci.Na primer myAwesomeVariable. Ovo se ponekad zove "camel case"
- **UIImageView!** : Ovo deklarise da je property tipa UIImageView I ponovo vidimo simbol za implicitly unwrapped optional : **!** . Ovo znaci da UIImageView moze da bude tamo ili ne mora da bude, ali smo sigurni da ce definitivno biti tamo kada pozelimo da ga koristimo.
- Ako ste se mucili da razumete "implicitly unwrapped optionals" - ne brinite, oni su komplikovani!), ovaj kod vam moze to uciniti malo jasnijim. Vidite, kada se stvori detail view controller, njegov view jos uvek nije ucitan - to je samo neki kod koji radi na procesoru(CPU).

Kada su osnovne stvari uradjene (na primer, dodeljivanje dovoljno memorije da drzi sve to), iOS ide unapred i ucitava layout iz storyboard-a, onda povezuje sve outlet-e iz storyboarda sa kodom.

Dakle, kada je detail controller prvi put napravljen, UIImageView ne postoji zasto sto jos uvek nije stvoren - ali mi svejedno moramo da imamo slobodnog mesta u memoriji za njega. U ovom trenutku property je nil ili samo neka prazna memorija. Ali kada se ucita view i kada se povezu outlet-i, UIImageView ce da pokaze na pravi UIImageView, a ne na nil, da bismo poceli da ga koristimo.

Ukratko, UIImageView pocinje svoj zivot kao nil, onda dobija vrednost pre nego sto ga koristimo, tako da smo sigurni da on nikada nece biti nil do trenutka kada zelimo da ga iskoristimo - skolski primer implicitly unwrapped optionals. Ako jos uvek ne razumete implicitly unwrapped optionals to je sasvim u redu - samo nastavite i vremenom ce postati jasni.

To je nas detaljni ekran (detail screen complete - zavrsili smo sa interface builderom za sada i mozemo se vratiti na kod. Ovo takodje znaci da smo zavrsili sa assistent editorom tako da se mozemo vratiti na fullscreen editor tako sto cemo ici na View > Standard Editor > Show Standard Editor.

Loading Images with UIImage

(Učitavanje slika sa UIImage)

U danom momentu imamo nas originalan table view controller koji je pun slika koje mozemo odabrati, plus a detail view controller koji se nalazi na storyboard. Sledeci cilj je da pokazemo detail screen kada pritisnemo bilo koji table row (red).

Da bismo ovo omogucili moramo da dodamo jos jedan specijalno nazvan metod u klasu ViewController. Ovaj metod se zove **tableView(_ didSelectRowAt:)**. On uzima vrednost IndexPath kao sto je to radio i cellForRowAt koji nam je govorio sa kojim redom trenutno radimo. Ovaj put je potrebno uraditi malo vise posla:

1. Moramo da naprvimo osobinu(property) u DetailViewConrtoller koja ce da drzi ime slike koju cemo ucitati.

2. Primenicemo metod **didSelectRowAt** koji ce da ucita DetailVireController sa storyboard.

3. Na samom kraju popunicemo viewDidLoad ( ) unutar DetailViewController tako da on ucita sliku u sebe na osnovu imena koje smo postavili ranije.

Resimo svako od njih po redu, polazeci od prvog: pravimo osobinu (property) u DetailViewController koji ce da drzi ime slike koju cemo da ucitamo.

Ova osobina ce da bude String - ime slike koju cemo ucitati - ali ona mora da bude *optional* String zato sto kada prvo stvorimo viewController ona(osobina) nece postojati. Podesicemo je odmah, ali ona svoje postojanje pocinje prazna.

Prema tome, dodajte ovu osobinu(property) u DetailViewController odmah ispod postojece linije @IBOutlet:

```
var selectedImage: String?
```

Resili smo prvi zadatak I odmah prelazimo na sledeci: implementirati didSelectRowAt tako da on ucita DetailViewController iz storyboard-a.

Kada smo stvorili detail view controller u storyboard-u dali smo mu ID “Detail” sto nam omogucava da da ga ucitamo iz storyboard-a koristeci metod koji se zove **instantiateViewController(withIdentifier:)** Svaki view controller ima property -> storyboard koja je ili ta sama storyboard iz koje je ona ucitana ili nil. U slucaju ViewController-a to ce biti Main.storyboard koje je ta ista storyboard koja sadrzi detail view controller , tako da cemo je ucitati odatle.

Ovaj zadatak mozemo podeliti na tri manja zadatka od kojih su dva potpuno nova:

1. Ucitati detail view controller layout iz storyboard-a.
2. Podesiti njegovu selectedImage property da bude tacna stavka iz niza slika (pictures array)
3. Pokazati novi view controller.

Prvi zadatak se radi pozivajuci instantiateViewController, ali on ima **dve** male nijanse. **Prvo** pozivamo osobinu storyboarda koju dobijamo iz Apple-ovog tipa UIVIewController, ali ona je *optional* zato sto Swift ne zna da smo dosli iz storyboarda. Prema tome, moramo da koristimo **? (optional chaining)** kao kada smo podesavali text label na nasu celiju(cell) : "probaj da uradis ovo ali ne radi nista ako postoji problem".

Drugo, iako ce nam instantiateViewController() poslati nazad DetailViewController, ako sve bude radilo pravilno, Swift misli da ce nam on vratiti UIViewController jer Swift ne moze da vidi unutar storyboarda-a i da zna sta je sta.

Ovo deluje zbunjujuce ako ste novi u programiranju, tako da cu to objasniti malo drugacije koristeci analogiju. Recimo da zelite da odete sa nekim na sastanak veceras i trazite od mene da vam obezbedim karte za neki dogadjaj. Ja odlazim i kupujem vam karte koje vam onda dajem u koverti. Ja sam ispunio moj deo dogovora: vi ste trazili karte a ja sam ih obezbedio. Ali kakve su to karte - da li su to karte za neki sportski dogadjaj? Da li su to karte za operu? Karte za voz? Jedini nacin na koji vi mozete saznati kakve su karte u pitanju je da otvorite tu kovertu i pogledate.

Swift ima isti problem: instantiateViewController ima povratnu vrednost(return type) UIViewController, tako da sto se tice Swifta, bilo koji view controller koji je napravljen pomocu njega je u stvari UIViewController. Ovo nama pravi problem jer mi zelimo da podesimo osobinu (property) koju smo napravili u DetailViewController-u. Resenje: moramo da kazemo Swiftu da ono sto ima ima nije ono sto misli da ima.

Tehnicki izraz za ovo je "typecasting": trazimo od Swifta da tretira jednu vrednost kao da je drugog tipa. Swift ima nekoliko nacina na koji se ovo moze uraditi, ali mi cemo da koristimo najbezbedniji nacin, to zapravo znaci sledece: "molim te pokusaj ovo da tretiras kao DetailViewController, ali ako ne uspes, onda ne radi nista i produzi dalje".

Jednom kada imamo gotov detail view controller na nasim rukama, mozemo da podesimo da njegova selectedImage osobina bude jednaka sa pictures[indexPath.row] isto kao ssmo uradili i sa cellForRowAt - to je laksi deo.

Treci mini korak je da napravimo da novi ekran pokaze sam sebe. Vec ste videli da view controller-i imaju opcionu storyboard osobinu koja ili sadrzi storyboard iz koje su ucitani

ili sadrzi nil. Oni isto imaju opcionu osobinu navigationController koja sadrzi navigacioni kontroler u kome se oni nalaze - ako on postoji ili nil u suprotnom slucaju.

Za nas je ovo savrseno zato sto su navigacioni kontroleri odgovorni za pokazivanje ekrana. Naravno, oni daju tu lepu sivu traku na vrhu koju vidite u mnogim aplikacijama, ali su takodje odgovorni za odrzavanje velike gomile ekrana kroz koje se korisnici krecu.

Po default-u oni sadrze prvi view controller koji smo za njih napravlili u storyboardu ali kada se stvaraju novi prozori mozete ih gurnuti na navigation controller gomilu, da bi oni glatko klizili bas kao sto je to slucaj kada udjete u Settings. Sto vise ekrana gurate na tu gomilu, oni jednostavno nastavljaju da klizaju dalje. Kada se korisnici vrate nazad za jedan ekran - tj. kada kliknu na dugme Back ili prevlacenjem sa leva na desno - navigacioni kontroler ce automatski da unisti stari view controller i da oslobodi njegovu memoriju.

Ova tri mini koraka kompletiraju novu metodu tako da je vreme za pisanje koda. Molim vas, dodajte ovu metodu u ViewController.swift - ja sam dodao komentare zbog lakseg razumevanja.

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {

    // 1: try loading the "Detail" view controller and typecasting it to be
DetailViewController

    if let vc = storyboard?.instantiateViewController(withIdentifier: "Detail")
as? DetailViewController {

        // 2: success! Set its selectedImage property

        vc.selectedImage = pictures[indexPath.row]



        // 3: now push it onto the navigation controller

        navigationController?.pushViewController(vc, animated: true)

    }

}
```

Pogledajmo malo blize na momenat red gde je if let. Postoje tri njegova dela koja mogu da budu neispravna: storyboard property moze da bude nil (u cijem ce slucaju ? - optional chainig zaustaviti da se ostatak reda izvrsi), pozivanje instatiateViewController ( ) moglo bi da propadne ako smo trazili “Fzzzz” ili neki drugi invalidni ID sa storyboard-a, i deo sa

typecast-om - as? - isto moze da propadne zasto sto mozemo nazad da dobijemo view controller drugog tipa.

Prema tome, tri stvari u toj jednoj liniji imaju potencijal da propadnu. Ako ste pratili sva moja uputstva one nece propasti ali bitno je uociti da one imaju potencijal da propadnu. Tu dolazi do izrazaja snaga naredbe if let: ako bilo koja od tih stvari vrati nil (tj. ako propadne) onda se kod unutar naredbe if let nece izvrsiti). Ovo garantaju da je program u bezbednom stanju pre izvrsenja bilo kakve akcije.

Postoji samo jos jedna mala stvar koju moramo uraditi pre nego sto pogledamo rezultate: moramo da nateramo sliku da se ucita na image view u DetailViewController.

Ovaj novi kod ce povuci sa sobom novi tip podataka(data type), koji se zove UIImage. On nema View u svom imenu tako da to nije nesto sto mozete da pogledate - to nije nesto sto je zapravo vidno korisnicima. Umesto toga, UIImage je tip podataka koji cete koristiti da ucitate slike kao sto su PNG ili JPEGs.

Kada napravite UIImage, postoji parametar - named koji vam omogucava da navedete ime slike koja ce se ucitati. UIImage onda trazi ime te datoteke u app's bundle i ucitava ga. Ubacivanjem osobine(By passing in the `selectedImage` property here) selectedImage, koja je poslata od ViewController-a, ucitaće se slika koju je korisnik izabrao.

Medjutim, selectedImage ne moze biti korisceno direktno. Ako se secate, stvorili smo ga ovako:

```
var selectedImage: String?
```

Taj znak pitanja ? znaci da on moze ima vrednost ili ne mora(nil), a Swift vam ne dozvoljava da koristite ove "mozda" bez da ih prethodno proverite. Ovo je jos jedna prilika za koriscenje if let: mozemo da proverimo da li selectedImage ima vrednost i ako je tako da je izvucemo za koriscenje, a u suprotnom - ne radi nista.

Dodaj ovaj kod u viewDidLoad( ) unutar DetailViewController, posle pozivanja super.viewDidLoad( ):

```
if let imageToLoad = selectedImage {
    imageView.image  = UIImage(named: imageToLoad)
}
```

Prvi red je komanda koja "unwraps the optional" u selectedImage. Ako je usled nekog razloga selectedImage jednak nil(sto on u teoriji nikad ne sme da bude) onda linija imageView.image nece nikada biti izvrsena. Ako ima vrednost, ta vrednost ce biti postavljena u imageToLoad, onda prosledjena UIImage-u i ucitana.

Ok, to je to: pritisnitte Cmd + R da pokrenete aplikaciju i probajte je! Trebalo bi da mozete da izaberete bilo koju od slika, da ih gledate kao slajd i prikazete ih na punom ekranu.

Primeticete da smo dobili Back dugme na navigacionoj traci koje nam omogucava da se vratimo nazad u ViewController. Ako pazljivo stisnete i vucete., videcete da mozete takodje da stvorite gest prevlacenja(swipe gesture) - kliknite na samu levu ivicu ekrana I prevucite je na desno, kao sto biste uradili sa vasim palcem na telefonu.

Final tweaks: hidesBarsOnTap and large titles

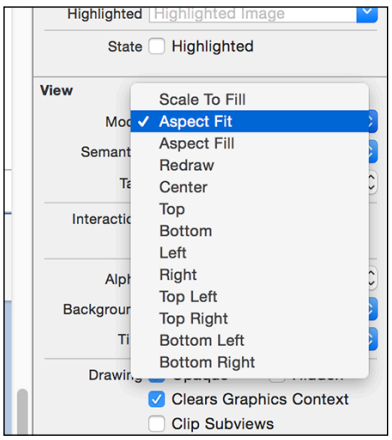
(Konačna podešavanja: hidesBarsOnTap(skrivanje traka na dodir) i veliki naslovi)

U ovom trenutku imate aplikaciju koja radi: mozete da pritisnete Cmd+R da je pokrenete, mozete prelistati slike u tabeli i kliknuti na bilo koju od njih da biste je pregledali. Ali pre nego sto zavrsimo ovaj projekat, postoji nekoliko manjih izmena koje cemo da napravimo da bi konačni rezultat izgledao pristojnije.

Prvo, mozda ste primetili da se sve slike rastežu da bi se uklopile u ekran. Ovo nije slučajno - to je podrazumevano podešavanje od UIimageView.

Potrebno je samo nekoliko klikova misem da bi se ovo ispravilo: izaberite Mail.storyboard, izaberite image view u detail view controller-u I zatom izaberite attributes inspector. Ovo je u desnom delu prozora, blizu vrha i on je četvrti od šest inspectora.

Ako ne volite da ga lovite i trazite na ovaj nacin, samo pritisnite Cmd+Alt+4 da biste ga otvorili. Rastezanje slike po ekranu uzrokuje view mode, koje je dugme padajuceg menija koje je po default-u namesteno ili na "Aspect Fit" ili "Aspect Fill" zavisno od verzije Xcoda koju imate. Probajte da promenite na "Aspect Fill" da biste videli kako se slika povećava i popunjava raspoloživom prostoru na ekranu.



Ako ste se pitali, Aspect Fit postavi sliku tako da je sve vidljivo. Postoji takodje Scale To Fill koji postavi sliku tako da nema slobodnog prostora ni sa jedne strane razvlačeći sliku i po X i po Y osi. Ako koristite Aspect Fill slika ce efektivno visiti izvan oblasti ekrana, tako da se trebate osigurati da ste izabrali Clip To Bounds da biste izbegli prepunjavanje prikaza(image overspilling).

Druga promena koju cemo napraviti je da dozvolimo korisnicima da vide sliku preko celog ekrana tj bez navigacije na ekranu. Postoji veoma lak nacin da se ovo uradi, a to je osobina (property od UINavigationController-a koji se zove hidesBarsOnTap. Kada je ova osobina podesena na true, korisnik moze da klikne bilo gde na ekran trenutnog view controllera da bi sakrio navigation bar, a zatim da klikne ponovo da bi ga ponovo pokazao.

Upozorenje: morate veoma pazljivo da ga podesite kada radite sa iPhon-ima. Ako bi oni bili namesteni sve vreme, to bi napravilo haos kada bi korisnik pokusao da izabere nesto. Prema tome, moramo ga omogućiti kada radimo sa detail view controller-om, a onda da ga onemogucimo kada ga krijemo.

Vec ste upoznali metodu viewDidLoad ( ), koja se poziva kada je učitan view controller layout. Postoji jos nekoliko drugih koje se mogu pozvati kada view treba da se pokaze, kada je pokazan i kada je uklonjen. Oni su, respektivno: viewWillApear ( ), viewDidApear ( ), viewWillDisappear ( ), viewDidDisappear ( ). Mi cemo da koristimo viewWillAppear ( ) i viewWillDisappear ( ), da bismo izmenili property hidesBarsOnTap tako da je ona nameštena na true samo kada je pokazan detail view controller.

Otvorite DetailViewController.swift, a onda dodajte ove dve metode direktno na kraju metode viewDidLoad ( ) :

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    navigationController?.hidesBarsOnTap = true
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    navigationController?.hidesBarsOnTap = false
}
```

Postoje neke vazne stvari ovde koje moramo napomenuti:

- Za svaku od ovih metoda mi koristimo override zato sto one vec imaju podesene default vrednosti u UIViewController-u a mi trazimo da se iskoriste naše. Ne brinite se ako niste sigurni kada da trazite override, a kada ne, zato sto kada ga ne iskoristite, a trebali ste, Xcode ce vam reci.
- Obe metode imaju jedam parametar: bez obzira da li je akcija animirana ili ne. U ovom trenutku nas i nije briga, tako da mi to mozemo ignorisati.

- Obe metode ponovo koriste prefiks super: super.viewWillAppear() i super.viewWillDisappear ( ). Ovo znaci: “reci mom roditeljskom tipu podataka da su pozvane ove metode”. U ovom slučaju, to znaci da da ce metod biti prosledjen na UIViewController, koji moze da obavi sopstvenu obradu.

- Mi ponovo koristimo navigationController property, koja ce da radi lepo zato sto smo gurnuti na navigation controller stack iz ViewController. Prostupamo osobini koristeci ❓, tako da ako se kojim slucajem ne nalazimo unutar navigation controller-a, linije hideBarsOnTap nece uraditi nista.

Ako sada budete pokrenuli aplikaciju videcete da mozete da kliknete da biste videli sliku u punoj velicini, i ona vise nece biti raširena. Dok gledate sliku videcete da mozete da kliknete na nju da bi navigational bar nestao i da kliknete ponovo na sliku da bi se on pojavio ponovo.

Treca izmena je mala ali važna. Ako pogledate druge aplikacije koje koriste table view i navigational controllers da pokazu ekrane(najbolji primer za ovo je Settings app), možete da primetite sive strelice na krajevima table view cells. Ovo se naziva disclosure indicator i to je suptilan način na koji se korisniku nagoveštava da ako klikne taj red može dobiti još informacija.

Potrebno je samo nekoliko klikova u Interface Builde-u da biste dobili ove strelice. Otvorite Main.storyboard onda kliknite na table view cell - to je ona na kojoj je napisano “Title” direktno ispod “Prototype Cells”. Table View sadrži ćeliju, ćelija sadrži content view(prikaz sadržaja), a content view sadrži label (oznaku) koja se zove “Title” tako da je veoma lako da pogrešite i izaberete pogrešnu stvar. Na kraju, možda će vam biti lakse da iskoristite document outline da biste izabrali pravu stvar - vi želite da izaberete stvar označenu kao “Picture” koje je reuse identifier koji smo prilozili na nas table view cell.

Kada je to izabrano, trebalo bi da mozete da odete na attributes inspector I da vidite “Style: Basic”, “Identifier: Picture”, itd. Takodje ćete videte “Accessory: None” - molim vas promenite to na “Disclosure Indicator”, koji će uzokovati pojavljivanje sive strelice.

Četvrta izmena je mala ali važna: postavicemo neki tekst u sivu traku(gray bar) na vrhu. Vec ste videli da view controllers imaju storyboard i navigationalController properties koje dobijamo od UIViewController-a. Oni takodje imaju i title property koje navigacioni kontroler automatski čita: ako obezbedite ovaj title on ce biti pokazan na sivoj traci navigacionog bara na vrhu.

U ViewController, dodajte ovaj kod viewDidLoad() posle poziva super.viewDidLoad():

```
title = "Storm Viewer"
```

Ovaj title se automatski koristi za "Back" dugme, tako da korisnici znaju gde se zapravo vracaju.

U DetailViewController možemo dodati nesto ovako viewDidLoad-u () :

```
title = "View Picture"
```

To bi funkcionisalo ali umesto toga ćemo da koristimo dinamički tekst: mi cemo da pokažemo ime izabrane slike umesto toga.

Dodajte ovo viewDidLoad-u () u DetailViewController-u:

```
title = selectedImage
```

Mi ne moramo da razmotamo (unwrap) selectedImage ovde zasto sto i selectedImage i title su optional strings - mi dodeljujemo jedan optional string drugom. title je optional zato sto je po defaultu nil: view controllers nemaju title, prema tome ne pokazuju nikakav tekst u navigation bar-u.

Large titles
(Veliki naslovi)

Ovo je u potpunosti opcionalna promena, ali zeleo sam da vam je predstavim lepo i malo ranije tako da mozete da probate to sami.

Jedan od Apple-ovih dizajn vodiča je da se koriste veliki naslovi - tekst koji se pojavljuje na sivoj traci na vrhu aplikacija. Podrazumevan stil je mali tekst, sto je ono sto smo mi do sada imali, ali sa nekoliko linija koda mozemo da usvojimo novi dizajn.

Prvo dodajte ovo u viewDidLoad( ) u viewController.swift:

```
navigationController?.navigationBar.prefersLargeTitles = true
```

To obezbedjuje velike naslove u nasoj aplikaciji, i odmah cete primetiti razliku: "Storm Viewer" je postalo mnogo veće, i svi naslovi slika u detail view controller-u su takodje veliki. Primeticete da naslovi vise nisu staticki - ako povucete na dole nezno videcete da se naslov rasteze lagano, a kada krenete da otpustate on se polako vraca u prvobitno stanje.

Apple preporucuje koriscenje velikih naslova samo kada to ima smisla, a to obicno znaci samo na prvom ekranu vase aplikacije. Kao sto ste videli podrazumevano ponasanje (kada je omoguceno) je da imate velike naslove svugde, ali to je zato sto svaki novi view

kontroler koji je gurnut na gomilu navigacionih kontrolera, nasledjuje stil njegovo prethodnika.

U pvoj aplikaciji mi zelimo da se “Storm Viewer” pojavi veliko, ali da detail screen izgleda normalno. Da bismo to omogucili moramo da dodamo red koda u viewDidLoad ( ) u DetailViewController.swift:

```
navigationItem.largeTitleDisplayMode = .never
```

To je sve sto je potrebno - veliki naslovi bi trebalo da se ponasaju ispravno.