Aantekeningen Udemy cursus ‘The complete iOS8 and Swift course’

**Section 5 – Navigation, Storage and Live Content**

**LET OP: SECTION 5 IS HERZIEN, TEKST MOET NOG AANGEPAST WORDEN**

**Les 39: Navigation bars en toolbars (en 40 voor code files)**

In deze les wordt ingegaan op het toevoegen van een *navigation bar* aan een app. Creëer een nieuw project en sleep vanuit de Attributes Inspector een navigation bar naar het storyboard. De navigation bar heeft een titel die handmatig aangepast kan worden.

Aan de navigation bar kunnen *bar button items* toegevoegd worden. Voeg er een toe rechts op de navigation bar. Standaard krijgt dit de naam ‘item’, maar er zijn een aantal voorgedefinieerde buttons beschikbaar, deze kunnen in de Attributes Inspector onder Identifier benaderd worden.

Onderaan het scherm kan een *toolbar* toegevoegd worden, dit is vergelijkbaar met de *navigation bar*. De *toolbar* heeft standaard links een *bar button item* beschikbaar. Voeg je nog een *bar button item* toe dan komt deze direct rechts van de linker *bar button item* te staan. Dit gebeurt ook wanneer nog meer *bar button items* worden toegevoegd. Mbv. *flexible space bar button items* kunnen de *bar button items* gelijkmatig verdeeld worden over de *toolbar*, door deze tussen de *bar button items* te plaatsen.

De buttons doen nog niets, in de volgende les worden acties aan de buttons gekoppeld.

De code staat in 5.1.

**Les 41: App: Stopwatch (en 42 voor code files)**

In deze les wordt een timer app gebouwd. In *viewcontroller.swift* definieer een timer object in *viewDidLoad*:

var timer = NSTimer.scheduledTimerWithInterval(1, target:self,

selector: Selector(“result”), userInfo: nil, repeats: true)

Dit definieert een timer die elke seconde afloopt en dan de functie ‘result’ uitvoert. De target is ‘self’, dat is de onderhavige viewcontroller waarin de timer is gedefinieerd. Met de parameter userInfo kan info worden doorgespeeld naar de selector (result in dit geval), daar maken we hier geen gebruik van.

De result functie gebruiken we om de timer te updaten, voor nu drukken we bij wijze van test een string ‘test’ af:

func result() {

println(“test”)

}

Selecteer de viewcontroller het storyboard en stel de grootte in de Attributes Inspector in op ‘iPhone 4-inch’. Zet een *navigation bar* bovenaan op de viewcontroller, net onder de batterij indicator (zodat deze zichtbaar blijft) en zet de naam op Stopwatch. Voeg een *bar button item* toe links op de *navigation bar* en noem deze Reset.

Voeg een *toolbar* onderaan op de viewcontroller toe en voeg *bar button items* toe voor Play (links) en Pause (rechts). Voeg hiertussen een *flexible space bar button item* toe.

Voeg voor het tonen van de tijd een *label* toe midden op het scherm, maak deze lekker groot en geef het de waarde ‘0’ (nul). Centreer het label op het scherm mbv. *Auto Layout* . Kijk mbv. *preview* hoe het er tot nu toe uitziet. Het label is zowel in portrait als landscape mode netjes gecentreerd, maar de navigation en toolbar niet.

Selecteer de navigation controller in het storyboard en voeg *nearest neighbour constraints* toe: 0 voor links en rechts en 8 voor top.

Doe hetzelfde voor de toolbar. Voeg *nearest neighbour constraints* met waarde 0 toe voor links, rechts en bottom.

Voeg nu een *action* toe voor de Play button. Rechtsklik op de play button en sleep naar de code in view controller.swift, noem de functie ‘play’. Zet de code voor de timer definitie in de play functie. De timer variabele moet globaal worden gedefinieerd omdat we die ook buiten de ‘play’ functie nodig hebben. Definieer timer als volgt:

var timer = NSTimer()

Creëer een *outlet* voor het label en noem dit ‘time’. Definieer een counter variabele en zet deze initieel op 0. In de functie ‘result’ pas nu de counter en het label aan:

counter++

time.text = String(counter)

De typecast is nodig omdat counter een integer is en het textveld van time een string.

Creëer een action voor de pause button en noem deze ‘pause’. Voor het stilzetten van de functie bestaat een invalidate functie:

timer.invalidate()

Verder hoeft er niets te gebeuren.

Voeg voor de Reset knop een action ‘reset’ toe. Bij een reset moet de counter variabele gereset worden:

timer.invalidate()

counter = 0

time.text = “0”

De app is nu volledig functioneel. Een nuttige aanvulling is het tonen van de uren, minuten en seconden, etc. etc.

De code is te vinden in 5.2.

**Les 43: Table views (en 44 voor code files)**

In deze les wordt een begin gemaakt met de omgang met *table views*. Creëer een nieuw project en zet de grootte op ‘iPhone 4 inch’.

Een *table view* kan in principe vanuit de *Attribute Inspector* op het storyboard op de zichtbare *viewcontroller* gezet worden. Echter is de standaard viewcontroller beperkt in de omgang met *table views*. Daarvoor is het veel handiger om een *table viewcontroller* te gebruiken, die is ontworpen voor de omgang met *table views*.

Voor deze les verwijderen we de standaard viewcontroller die zichtbaar is in het storyboard. Het storyboard is nu leeg. Zet nu een *table viewcontroller* op het storyboard. Hierin is al een *table view* opgenomen. In de *Attribute Inspector* staat het *content type* van de table view default op *dynamic prototypes*, dit houdt in dat de tabel dynamisch wordt opgebouwd en beheerd. Dit komt later in de lessen aan bod. Voor nu maken we gebruik van *Static Cells*. Er verschijnen nu 3 cellen in de table view.

Selecteer de *table view* in het storyboard en check de opties in de *Attribute Inspector*. De table kan worden ingedeeld in *secties*, de *Style* kan worden aangepast, etc.

Klik nu op een cel en vervolgens op de *Content view*. In de *Attribute Inspector* verschijnen nu diverse opties, bijv. *Mode* waarmee de positie van de content in de cel kan worden geregeld en diverse lay-out opties zoals *alpha* (ondoorzichtigheid: 1=ondoorzichtig, 0=volledig transparant), tekstkleur, achtergrondkleur etc.

Klik nu op de cel zelf en zet de *Style* van *Custom* naar *Basic*. Dit geeft de mogelijkheid om content in de cel in te voeren. Evenzo voor de styles *Right Detail*, *Left Detail* en *Subtitle*.

Vul in de drie cellen activiteiten in: Buy bread, Go swimming en Send emails.

De code is te vinden in 5.3.

**Les 45: Populating a table (en les 46 voor de code files)**

**DEZE LES IS KOMPLEET HERZIEN EN HEET NU APP: TIMES TABLES**

In deze les wordt ingegaan op het beheren van een tabel via code. Verwijder de table viewcontroller van de vorige les en sleep een standaard *viewcontroller* naar het storyboard. Zet er vervolgens een *table view* op. De reden dat het zo gebeurt is dat we niet met statische content gaan werken, waar juist een table viewcontroller voor nodig is, maar volgens mij had er net zo goed een table viewcontroller op het storyboard gezet kunnen worden… In de *Attributes Inspector* is in het *Content* veld voor de table view in ieder geval *Dynamic Prototypes* ingevuld.

Voor de *table view* moeten nu de *datasource* en de *delegate* opgezet worden. De *datasource* geeft aan waar de table view zijn data vandaan haalt en de *delegate* geeft aan wie de table view beheert. In ons geval is de zojuist gecreëerde *viewcontroller* zowel de *datasource* als de *delegate*. Selecteer de table view in het storyboard, rechtsklik en sleep vanuit het *Table View* context menu zowel de *DataSource* als de *Delegate* naar het gele symbool net boven de view controller. Nu kunnen we vanuit *viewcontroller.swift* met de table view werken.

Vul de tableview delegate in achter de class definition:

class ViewController: UIViewController, *UITableViewDelegate*

Om informatie te krijgen over het UITableViewDelegate protocol beweeg je de muis naar het woord in de code en druk de Cmd toets in, klik vervolgens en er verschijnt een rij met methoden die tot het protocol behoren. Twee hebben we nodig:

* numberOfRowsInSection: aantal rijen per sectie
* cellForRowAtIndexPath: toont de inhoud van een specifieke cel

We maken een tabel met 4 rijen (en 1 sectie):

func tableView(tableView:UITableView, numberOfRowsInSection section:Int)

🡪 Int {

return 4

}

Functie cellForRowAtIndexPath vraagt de gegevens op voor table row <indexPath>. Deze vullen we in waarna Cocoa zorgt voor de weergave ervan op het scherm:

func tableView( tableView:UITableView,

cellForRowAtIndexPath indexPath: NSIndexPath)

🡪 UITableViewCell {

let cell = UITableViewCell( style: UITableViewCellStyle.Default,

reuseIdentifer: “cell” )

cell.textLabel?.text = “Table cell content”

return cell

}

Deze invulling heeft tot gevolg dat elke table cell dezelfde inhoud heeft, nl. “Table cell content”. Run de app en er verschijnt een tabel met 4 ingevulde rijen.

We creëren nu een globale class array met 4 elementen die elk in een table cell terecht komen:

let myArray = [“Frank”, “Anita”, “Marcella”, “Sophia”]

En verander het statement in cellForRowAtIndexPath dat het text field van de cell initialiseert:

Cell.textLabel?.text = myArray[indexPath.row]

Run de app en constateer dat de tabel nu met de namen uit myArray worden gevuld.

Zie 5.4 voor de code.

**Les 47: Persistent Storage (en les 48 voor de code files)**

Het permanent opslaan van ingevoerde informatie heet in iOS termen persistent storage. Hiervoor bestaan diverse mogelijkheden, de meeste zijn tamelijk complex en gebruiken Core Data, in deze les wordt een eenvoudige manier behandeld die Core Data niet gebruikt. ‘Use Core Data’ is een optie die tijdens het creëren van een nieuw project kan worden aangevinkt. We maken voor deze les een nieuw project aan en vinken deze optie niet aan.

We gebruiken de module ‘user defaults’, waar de gegevens in dictionaries als name-value pairs worden opgeslagen:

NSUserDefaults.standardUserDefaults().setObject(“Frank”, forKey:”yourName”)

NSUserDefaults.standardUserDefaults().synchronize

Hier wordt achtereenvolgens de key ‘yourName’ van de waarde ‘Frank’ voorzien waarna deze wordt gesynchroniseerd in de user defaults.

Het ophalen van gegevens uit de user defaults gaat als volgt;

NSUserDefaults.standardUserDefaults().objectForKey(“yourName”)

Dit haalt de waarde van het object ‘yourName’ op. Let op dat er een variabele nodig is om deze waarde in op te slaan, het statement hierboven geeft alleen een returnwaarde terug.

Run de app, het zal blijken dat de variabele+waarde worden opgeslagen wanneer de return waarde van objectForKey wordt geprint.

Haal nu bij wijze van test de eerste twee regels weg, zodat alleen de variabele ‘yourName’ wordt opgehaald. De waarde ‘Frank’ wordt nog steeds teruggegeven, waaruit blijkt dat persistent storage werkt!

Zie 5.5 voor de code.

**Les 49: Segues (en les 50 voor code files)**

Voor de nog te bouwen to-do list app hebben we meerdere schermen nodig en dus meerdere view controllers. De overgang naar een andere viewcontroller is onderwerp van deze les.

Creëer een nieuwe applicatie, maar selecteer deze keer een Tabbed Application (ipv single view app). Open nu het storyboard, daar zijn nu 3 controllers te zien: 2 *view controllers* en 1 *tab bar controller*. De tab bar controller heeft 2 knoppen, om naar één van beide view controllers te stappen. De *segues* (spreek uit ‘seg-ways’) zijn de pijlen tussen de view controllers die de overgang aangeven.

Zet bij wijze van test in de *viewDidLoad* functie van beide viewcontrollers een onderscheidend *println* statement, resp. ‘First view loaded’ en ‘Second view loaded’. Run nu de app en klik op de knoppen. Beide println statements worden slechts één keer uitgevoerd en dus niet elke keer dat een schermovergang plaatsvindt, daarvoor hebben we de *viewDidAppear* functie die elke keer wordt aangeroepen dat een view op het scherm verschijnt.

De code staat in 5.6.

**Les 51: Controlling the Keyboard (en 52 voor code files)**

In deze les staat de omgang met het keyboard centraal. Dit gebeurt adhv. een (input) textfield en een label dat de in het textfield ingetikte tekst toont wanneer op een button wordt gedrukt. Creëer een nieuw project hiervoor en voeg een label, button en textfield toe aan het storyboard. Zet het label redelijk onderaan het scherm (zodat het toetsenbord eroverheen valt wanneer het opkomt).

Creëer outlets voor het textfield en het label, creëer een action ‘buttonPressed’ voor de button. In buttonPressed wordt de in het textfield ingevoerde tekst gekopieerd naar de labeltekst:

label.text = textField.text

Run de app, voer wat tekst in in het textfield en druk op de button. Omdat het toetsenbord niet verdwijnt als op de button wordt gedrukt, is niet te zien dat de ingevoerde tekst in het label verschijnt. Het keyboard verschijnt wel automatisch, maar gaat niet automatisch weg. Dat moet gecodeerd worden.

Hiervoor is als basis het *UITextFieldDelegate* protocol nodig, vul dit in bij de class definition:

class ViewController: UIViewController, UITextFieldDelegate {

*UITextFieldDelegate* bevat een aantal functies waarmee het keyboard gecontroleerd kan worden. Cmd-klik op het *UITextFieldDelegate* woord en de lijst van methoden verschijnt. De functie die we nodig hebben is *textFieldShouldReturn*, deze wordt aangeroepen als de Return key wordt ingedrukt. De code is als volgt:

func textfieldShouldReturn(textField: UITextField!) -> Bool {

textField.resignFirstResponder()

return true

}

Om dit te laten werken moet de *delegate* van *textField* op de view controller ingesteld worden. Doe dit door te rechtsklikken op *textField*, de lijn naar het gele symbool boven in het scherm te trekken en vervolgens in het verschenen context menu ‘*Delegate’* te selecteren. Run de app en constateer dat het keyboard verdwijnt wanneer op de return knop wordt gedrukt. Constateer ook dat de labeltekst pas wordt aangepast als op de button wordt gedrukt.

Het is gebruikelijk dat het keyboard ook verdwijnt als buiten het invoerveld (textField) wordt gedrukt, dat is nu echter niet het geval. Daarvoor is een andere functie nodig: touchesBegan. Deze functie wordt overridden:

override func touchesBegan(touches:NSSet!, withevent event: UIEvent!) {

self.view.endEditing = true

}

Nu verdwijnt het keyboard ook als buiten het invoerveld wordt getapt.

NB. Dit wordt ook bereikt met de *resignFirstResponder* functie (uit eigen ervaring).

De code staat in 5.7.

**Les 53: App: To Do List (en 54 voor code files)**

In deze les wordt een to do list app gebouwd. Deze bestaat uit twee schermen: het eerste scherm is een tabel dat de todo items weergeeft, in het tweede scherm kan een nieuw item toegevoegd worden.

Creëer een nieuw project, een *Tabbed Application*. Net als in de vorige les 49 bestaat het storyboard uit 3 controllers. De *First viewcontroller* gebruiken we voor de table view, de tweede om de details van een item in te voeren. Pas de naam onderaan het scherm van de eerste viewcontroller aan (in de toolbar) in ‘To Do List’ en van de tweede viewcontroller in ‘Add An Item’. Zet de grootte van beide viewcontrollers op ‘*iPhone 4 inch*’ en haal de voorbedrukte tekst van de schermen af. Voeg een *table view* toe aan de eerste viewcontroller, hou het bovenste gedeelte van het scherm met de battery indicator vrij.

Run de app. In portrait mode ziet de tabel er goed uit, in landscape mode is hij rechts afgekapt. Zet daarvoor in *Auto Layout* de linker en rechter *spacing to nearest neighbour* op 0.

Nu wordt het tweede scherm ingericht. Voeg een label toe ‘Add our new to do item’. Lijn het uit in Auto Layout. Voeg een textfield toe, maak het zo breed mogelijk. Voeg een button ‘Add Item’ toe. Kies zelf hoe de control elementen uitgelijnd worden. Voeg een outlet ‘toDoItemTextField’ toe voor het textfield in de code van de tweede viewcontroller, voeg een action ‘addItemButtonPressed’ toe voor de button.

Creëer een string array toDoItems die initieel leeg is:

var toDoItems:[String] = []

Wanneer op de button wordt gedrukt wordt het ingevoerde item aan de array toegevoegd, Voeg de volgende code toe aan addItemButtonPressed:

toDoItems.append(toDoItem.text)

Voeg bij wijze van test

println(toDoItems)

Hier aan toe.

Nu gaan we de *toDoItems* array zichtbaar maken in de table view. Stel de delegate en datasource van de table view in op de tweede viewcontroller (gele icon boven aan scherm). Voeg de protocollen *UITableViewDelegate* en *UITableViewDataSource* toe aan de class definitie.

NB. De laatste is niet nodig… blijkt uit les 43.

De definitie van toDoItems verplaatsen we naar de code van de eerste viewcontroller, omdat deze daar het eerst wordt gebruikt. Door de definitie van toDoItems buiten de class definitie te plaatsen, is toDoItems ook beschikbaar in andere code files.

Voeg de tableView functies numberOfRowsInSection en cellForRowAtIndexPath toe aan de code van de eerste view controller, deze worden als volgt:

func tableView(tableView: UITableView, numberOfRowsInSection section: Int)

-> Int {

return toDoItems.count

}

func tableView( tableView: UITableView,

cellForRowAtIndexPath indexPath: NSIndexPath!)

-> UITableViewCell {

var cell = UITableViewCell( style: UITableViewCellStyle.Default

reuseIdentifier: “Cell”)

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

return cell

}

Er zijn nu nog een aantal problemen: (1) het keyboard verdwijnt niet wanneer een item wordt toegevoegd en (2) wanneer een item is toegevoegd dan is dat niet zichtbaar in de tabel, deze moet nl. worden refreshed.

Voor het verdwijnen van het keyboard moet:

* (1) het *UITextFieldDelegate* *protocol* opgenomen worden in de class definitie van de tweede viewvontroller
* (2) op het storyboard moet de *delegate* van het *textfield* ingesteld worden op de tweede viewcontroller: rechtsklik op textfield en sleep naar gele viewcontroller icon (NB Dit kan natuurlijk net zo goed in de code).
* (3) de functie *textFieldShouldReturn* geimplementeerd worden:

func textFieldShouldReturn(textField: UITextField!) -> Bool {

toDoItemTextField.resignFirstResponder()

return true

}

* (4) de functie *touchesBegan* *overriden* worden:

override func touchesBegan(touches: NSSet!,

withEvent event: UIEvent!) {

self.view.endEditing(true) // resignFirstResponder werkt ook!

}

Voor het verversen van de tabel is een referentie naar de tabel nodig. Creëer hiervoor in de eerste viewcontroller code zelf een *outlet*:

@IBOutlet var tasksTable: UITableView

De link met de tabel in het storyboard moet dan nog worden gemaakt. Rechtsklik op de *table view*, er verschijnt een context menu. Klik op het cirkeltje rechts van ‘*New Referencing Outlet*’ en sleep naar het gele viewcontroller icon. Er verschijnt nu een lijstje, waar ook *tasksTable* in staat. Selecteer deze en klaar is Kees.

NB Dit kan natuurlijk ook door in het storyboard te rechtsklikken op de tabel en naar de code te slepen.

De functie *viewWillAppear* is nodig om de tabel te herladen. Voeg deze toe:

override func viewWillAppear(animated: Bool) {

tasksTable.reloadData()

}

Run de app, het werkt nu beter. Het zou nog wat mooier zijn als het keyboard verdwijnt als op de *Add Item button* wordt gedrukt. Voeg daartoe het statement

self.view.endEditing(true)

toe aan functie *addItemButtonPressed*.

De app werkt nu prima, op het permanent opslaan van de tabel na. Dit doen we mbv. *NSUserDefaults* waarmee we de array *toDoItems* opslaan. In de video treedt echter een ernstige fout op wanneer het statement

NSUserDefaults.standardUserDefaults.setObject(toDoItems, “toDoItems”)

wordt ingegeven, zo ernstig dat de editor niet meer verder wil. De uitleg hiervan is dat *NSUserDefaults* alleen met *immutable* objecten kan omgaan, terwijl toDoItems een *mutable* object is. De oplossing is om een constante array aan te maken die op toDoItems wordt geïnitialiseerd:

let fixedToDoItems = toDoItems

NSUserDefaults.standardUserDefaults.setObject(fixedToDoItems, “toDoItems”)

NSUSerDefaults.standardUserDefaults.synchronize()

NB. In de nieuwste versie van XCode 6.1.1 speelt dit probleem niet meer.

Het ophalen van de opgeslagen items moet aan het begin van het programma gebeuren. Dit wordt in de les in viewWillAppear van de eerste viewcontroller gedaan, maar volgens mij moet het in viewDidLoad gebeuren, anders gebeurt het telkens wanneer het scherm getoond wordt wat niet fout maar ook niet nodig is. Anyway, dit is de code die aan viewWillAppear wordt toegevoegd:

if (var storedToDoItems: AnyObject =

NSUserDefaults.standardUserDefaults.objectForKey(“toDoItems”) {

toDoItems = []

for var i=0; i<storedToDoItems.count; i++ {

toDoItems.append(storedToDoItems[i] as NSString)

}

}

Deze code vraagt wat uitleg. De array *toDoItems* kan niet simpel in één statement uit de *user defaults* gehaald worden, omdat functie *objectForKey* een type *AnyObject* teruggeeft wat niet gelijk is aan het type van toDoItems (= array van strings). Daarom gebruiken we de variabele *storedToDoItems* om de opgeslagen gegevens terug te halen en deze worden vervolgens element na element overgeheveld naar *toDoItems* in de *for-loop*.

Het *if statement* checkt of *storedToDoItems* niet leeg (nil) is, zo ja dan kan *toDoItems* worden gevuld. Hiervoor wordt *toDoItems* wordt eerst leeggemaakt.

Omdat *storedToDoItems* van het type *AnyObject* is, moet elk element naar een *NSString* worden gecast om het in *toDoItems* op te kunnen slaan, *toDoItems* is nl. een array van strings.

Het enige belangrijke wat nu nog ontbreekt is de mogelijkheid een tabel entry te verwijderen. Hiervoor is de functie commitEditingStyle nodig, onderdeel van de UITableViewdelegate:

func tableView( tableView: UITableView!,

commitEditingStyle editingStyle: UITableViewCellEditingStyle,

forRowAtIndexPath indexPath: NSIndexPath) {

if (editingStyle == UITableViewCellEditingStyle.Delete) {

toDoItems.removeAtIndex(indexPath.row)

let fixedToDoItems = toDoItems

NSUserDefaults.standardUserDefaults.setObject(

fixedToDoItems, “toDoItems”)

NSUSerDefaults.standardUserDefaults.synchronize()

tasksTable.reloadData()

}

}

Hier wordt – als daadwerkelijk een item wordt verwijderd (editingStyle check) – het item met index indexPath.row uit de tabel verwijderd, wordt vervolgens de array gesynchroniseerd in de permanente opslag en wordt de tabel opnieuw geladen.

Run de app en test ermee. Een item verwijderen kan door het item naar links te swipen.

De code files staan in 5.8.

**Les 55: Accessing Web Content (en 56 voor code files)**

Creëer een nieuw project, single view application. Voor web content is een object van het type *NSURL* nodig. Met de class *NSURLSession* kunnen we een *data task* uitvoeren (naast *download* en *upload tasks*).

Voeg de volgende code toe in viewDidLoad:

var url = NSURL(string:”www.stackoverflow.com”)

let task = NSURLSession.sharedSession().dataTaskWithURL(url!){

(data, response, error) in

println(data)

}

task.resume()

De functie *dataTaskWithURL* heeft feitelijk twee parameters: de *url* en een *completion handler*, een code blok. De syntax van dit stuk code zou normaliter moeten zijn:

dataTaskWithURL(url, {(data, response, error) in

println(data)

})

Maar het code blok mag dus ook apart uitgeschreven worden, een kwestie van smaak.

Wanneer de app gerund wordt de internet pagina opgehaald en weergegeven als een lange lijst hexadecimale codes. Om het zichtbaar te maken als HTML code moet het als zodanig worden opgevraagd. Vervang de println(data) met:

println(NSString(data: data, encoding: NSUTF8StringEncoding))

De code staat in 5.9.

**Les 57: Working With Strings (en 58 voor code files)**

In de volgende les 56 wordt een weather app gebouwd. Deze haalt data van een website en presenteert data in een leesbaar formaat. Hiervoor is het nodig om de ontvangen data – een string – te manipuleren, wat het onderwerp is van deze les.

Concatenating strings:

var str1 = “Frank”

var str2 = “ de Groot”

var str12 = str1+str2 // resultaat: “Frank de Groot”

Loop over string karakters:

for c in str12 {

println(c)

}

Het default string type in Swift is *String*. Een erfenis van Objective-C is het type *NSString*, dat ook in Swift gebruikt kan worden en nodig is voor een groot aantal methoden in Cocoa. De definitie en initialisatie van een NSString gaat als volgt:

var newString = “Test string” as NSString

Het loopen door de string karakters kan echter niet met een NSString. NSString heeft echter de nodige methoden – zoals het opvragen van substrings – die alleen op NSStrings werken en niet op Strings. Voorbeeld:

var subString = newString.*substringToIndex*(6)

Dit geeft de eerste 6 karakters van newString.

Om nu hetzelfde te bereiken met een String variabele, moet deze naar een NSString worden gecast:

var subString = (*str12 as NSString*).substringToIndex(6)

Uiteraard is er ook een functie substringFromIndex:

var subString = (str12 as NSString).*substringFromIndex*(5)

En de meest flexibele variant waarin een range wordt opgegeven:

var subString = (str12 as NSString).*substringWithRange(*

*NSRange(location:5, length: 4)*)

Deze vraagt een parameter van het type NSRange.

Het testen of een string voorkomt in een NSString variabele:

newString.*containsString*(“Groot”) // resultaat: true

Het opsplitsen van een string in de aparte woorden gaat als volgt:

newString.*componentsSeparatedByString*(“ “)

Dit levert een array van strings op, waarbij de spatie als splitsingskarakter wordt gebruikt.

Het omzetten van alle karakters naar hoofd- cq. Kleine letters:

newString.*uppercaseString*

newString.*lowercaseString*

De code staat in 5.10.

**Les 59: App: What’s The Weather (en 60 voor code files)**

In deze les wordt een app gebouwd waarmee het weer in een op te geven stad kan worden opgevraagd. Creëer hiervoor een nieuw project, een single view application.

De app haalt de HTML content van de website [*www.weather-forecast.com*](http://www.weather-forecast.com) en haalt hier specifieke weer informatie uit. Dit is overigens in het algemeen niet de beste manier om informatie op te halen, beter is om een voorhanden api te gebruiken, zoals bijv. bij *openweathermap.org* (waarvan mijn api key **fe93226b4d243c083c9cc9e6d089971a** is).

De app wordt mooi gemaakt met een plaatje dat van *unsplash.com* afkomstig is. Exporteer het plaatje naar png format en kopieer het plaatje vanuit de Finder naar de project directory.

Stel de viewcontroller in op ‘iPhone 4 inch’ en creëer een image view fullscreen in het storyboard. Stel de view mode in de Attributes Inspector in zoals gewenst (bijv. Center), zodanig dat het plaatje er het best als achtergrond uitziet.

Creëer een label op het scherm ‘*Enter a city to find out the weather’*. Voeg een textfield toe, maak dit schermbreed. Voeg een button ‘*What’s the weather’* toe. Voeg nog een label toe waarin de weer informatie wordt weergegeven, maak dit voldoende groot en geeft het bijv. 4 regels ruimte.

Maak vervolgens outlets en actions aan:

* een outlet voor het textfield genaamd ‘*cityTextField’*
* een action voor de button genaamd ‘*getWeatherButtonPressed’*
* een outlet voor het weerlabel genaamd ‘*weatherInfoLabel’*

De reden dat voor *weatherforecast.com* is gekozen is dat de URL’s er tamelijk eenvoudig uitzien. De stad is nl. een onderdeel van de URL die mbv. stringsubstitutie tamelijk eenvoudig geconstrueerd kan worden. Voeg deze in *getWeatherButtonPressed* toe:

var city = cityTextField.text

var urlString = “http://www.weather-forecast.com/locations/” +

city + “/forecasts/latest”

Let op dat er spaties voor en na de ‘+’ operators moeten staan, anders klaagt XCode.

De stadsnaam in de url mag geen spaties bevatten (zoals “San Francisco”), die moeten er nog uit voordat de url opgevraagd kan worden:

var city = cityTextField.text.stringByReplacingOccurrencesOfString(“ “,

withString: “”)

Nu kan de content opgehaald worden. Voeg het volgende toe in *getWeatherButtonPressed*:

var url = NSURL(string: urlString)

let task = NSURLSession.sharedSession().dataTaskWithURL(url,

{(data, response, error) in

// Druk de ontvangen data bij wijze van test af

println(NSString(data: data,

encoding:NSUTF8StringEncoding))

)}

task.resume()

Nu moet de data die we nodig hebben nog uit de ontvangen data string gehaald worden. Dit is het deel dat staat achter “3 Day Weather Forecast Summary:”. Toon de webpagina als HTML en het blijkt direct achter *“<span class=”phrase”>*” te staan, en wel de eerste keer dat deze tekst in de ontvangen data voorkomt. De truc is nu de ontvangen string op te splitsen, met deze substring *(“<span class=”phrase”>”)* als separator. Vervang de *println* in *getWeatherButtonPressed* door:

var urlContent = NSString(data: data, encoding: NSUTF8StringEncoding)

var contentArray = (urlContent! as NSString).componentsSeparatedByString(

“<span class=\”phrase\”>”)

Let op dat de double quotes onderdeel van de string zijn, door een “\” ervoor te zetten wordne ze in de string separator meegenomen ipv. dat de compiler ze beoordeelt als het einde van de string (waarna een rits syntax errors volgt).

Het deel dat we nodig hebben staat op positie 1 in de array, toon dit bij wijze van test:

println(contentArray[1])

Run de app, de benodigde content wordt getoond. Het blijkt dat we het deel voor de eerste *“</span>*“ nodig hebben. Dat halen we er op dezelfde manier uit:

var newContentArray = contentArray[1].componentsSeparatedByString(

“</span>”)

Het deel dat we nodig hebben staat in newContentArray[0]:

self.weatherInfoLabel.text = newContentArray[0] as String //Let op NSString!!

Het keyboard staat nu nog in de weg, voeg nog toe:

self.view.endEditing(true)

De app laat nu de weersvoorspelling zien, echter wordt de temperatuur nog niet netjes weergegeven. Het graden symbool verschijnt als HTML code *“&deg;”* in het label. In Objective-C zijn methoden om het graden symbool te tonen, in Swift (nog) niet. Daarom vervangen we dit als volgt:

self.weatherInfoLabel.text = newContentArray[0].

stringByreplacingOccurrencesOfString(“&deg;”, withString: “º”) as String

Het graden symbool kan dmv *Alt-0* ingetikt worden.

De app is nu in staat om het weer voor bestaande steden op te vragen. Wordt echter een niet bestaande stad ingevoerd dan crashed de app, omdat in dat geval *contentArray[1]* nil is. In de video wordt hiervoor de functie *bridgeToObjectiveC* gebruikt, maar het kan eenvoudiger door te testen of de *containsString* functie een resultaat <> nil teruggeeft, daarvoor moet een uitroepteken achter *urlContent* gezet worden:

if (urlContent! As NSString).containsString(“<span class=\”phrase\”>”) {

// extraheer weer data en toon deze in het label

} else {

self.weatherInfoLabel.text = “Unknown city”

}

In voorkomende gevallen kan het lang duren voordat de weerdata is geladen en duurt het tot wel 30 sec. voordat de weer info wordt getoond, hoewel de loginfo direct verschijnt. Dit komt doordat Swift een zg. asynchrone taal is en dus meerdere zaken tegelijk kan uitvoeren, waardoor het updaten van het label lang kan duren.

Dit kan ondervangen worden door de main thread stop te zetten waarna direct het volgende statement in het parameter blok wordt uitgevoerd:

dispatch\_async(dispatch\_get\_main\_queue(){

// uit te voeren statements

)}

Zet dit statement direct voor de initialisatie van *self.weatherInfoLabel*, in beide branches van de if-else.

De app is hiermee af. De code is te vinden in 5.11.

**Section 6 – Media: Images, Maps & Music**

**Les 61: Working with images (en 62 voor code files)**

In deze les wordt ingegaan op het tonen en manipuleren van plaatjes (images). De gepresenteerde plaatjes worden van [*www.heathersanimations.com*](http://www.heathersanimations.com) gehaald. Deze les maakt gebruik van een geanimeerd plaatje (zoals het ruimtemannetje) die bestaat uit een aantal frames. De frames worden afzonderlijk als .png opgeslagen. Vervolgens worden de .png plaatjes allemaal vanuit de Finder met drag en drop naar de projectdirectory in XCode gesleept. Vink ‘Copy items if needed’ aan zodat de plaatjes daadwerkelijk in de project directory staan.

Creëer een image view in het storyboard, maak het kleiner dan de beschikbare ruimte. Vul in de Attributes Inspector de naam van het eerste frame (bijv. “frame1.png”). Creëer een outlet voor de imageview en noem dit bijv. “alien”.

Voeg een button toe. Als hier op wordt gedrukt wordt het plaatje aangepast. Creëer een action “buttonPressed” in de viewcontroller voor de button. Hier wordt het volgende image frame getoond, was het laatste getoond dan wordt het eerste frame weer getoond, etc. De code is als volgt:

counter++ // een klasse variabele die initieel 1 is

if (counter >5) {

counter = 1

}

alien.image = UIImage(named: “alien\(counter).png”)

De code staat in 6.1.

**Les 63: Animations (en 64 voor code files)**

De alien app van de vorige les wordt uitgebreid met animaties. Hiervoor worden een aantal nieuwe functies geïntroduceerd.

De functie *viewDidLayoutSubviews* wordt aangeroepen juist voordat de view op het scherm wordt getoond.

De functie *viewDidAppear* wordt aangeroepen als de view op het scherm is verschenen.

Bij de eerste animatie wordt de alien initieel onzichtbaar links van het scherm gezet en daarna daarvandaan naar het middelpunt van het scherm bewogen. In *viewDidLayoutSubviews*:

alien.center = CPointMake(alien.center.x – 400, alien.center.y)

De animatie volgt in *viewDidAppear*:

UIView.animateWithDuration(1, animations: {

self.alien.center = CGPointMake( self.alien.center.x + 400, self.alien.center.y)

})

Let op het gebruik van self, dit is nodig omdat de code in een closure zit.

De volgende animatie doet een ‘fade in’ van de alien. Hiervoor gebruiken we de alpha waarde van de imageview. Een waarde van 0 maakt de alien onzichtbaar, 1 is volledig ontransparant.

In *viewDidLayoutSubviews*:

alien.alpha = 0

In *viewDidAppear*:

UIView.animateWithDuration(1, animations: {

self.alien.alpha = 1

})

De volgende animatie laat de alien vanuit het niets groeien:

In *viewDidLayoutSubviews*:

alien.frame = CGRectMake(100, 20, 0, 0)

Dit definieert een rechthoek met linksboven coördinaten (100, 20) en hoogte en breedte 0. In *viewDidAppear*:

UIView.animateWithDuration(1, animations: {

self.alien.frame = CGRectMake(100, 20, 100, 200)

})

De ‘stop frame’ animation laat de frames achtereenvolgens zien. Hiervoor wordt een timerfunctie gebruikt (zie ook de stopwatch app), deze wordt in *viewDidLoad* gedefinieerd:

timer = NSTimer.scheduledTimerWithTimeInterval(0.1, target: self,

Selector: Selector(“result”), userInfo: nil, repeats: true)

Functie result:

func result() {

counter++

if (counter > 5) { counter = 1 }

alien.image = UIImage(named:”frame\(counter).png”)

}

Het resultaat hiervan is dat de alien eerst vanuit het niets op het scherm verschijnt waarna elke 0,1 seconde een alien frame wordt getoond.

De code staat in 6.2.

**Les 65: Game: Tic Tac Toe (en 66 voor code files)**

In deze les wordt de tot nu toe opgedane kennis gebruikt om een Tic Tac Toe app te maken. Creëer een nieuw project *TicTacToe* met een single view. Haal plaatjes voor het bord, het kruis en het rondje van het internet en kopieer deze naar de projectdirectory.

Open het storyboard en zet er een imageview op, ter grootte van vrijwel het gehele board, alleen de top met de batterij indicator blijft vrij. Vul in de *Attribute Inspector* de naam van het bord plaatje in (“board.png” bijv.). Pas de hoogte van het image view aan zodat het vierkant is (320x320 pixels), bevestig deze waarde in Auto-Layout en centreer het image op het scherm.

NB. Zet de schermgrootte op iPhone 4 inch.

Het spelbord staat in de portrait mode nu mooi gecentreerd, maar in de landscape mode lijkt het nog nergens op. Gebruik hiervoor de preview mode van het storyboard en zet dit in landscape mode. In Auto-Layout, zet de breedte en hoogte van de imageview vast op 320 en centreer de imageview horizontaal en vertikaal.

Voeg vervolgens een button item toe, verwijder de button tekst en stel de image ervan in op het rondje. Stel de hoogte en breedte hiervan op dezelfde waarde, zodanig dat het in een hokje past (75 pixels in mijn geval), bevestig deze constraints daarna in Auto-Layout.

Stel de positie van het rondje in ten opzichte van de imageview van het speelbord. Doe dit door rechts te klikken op het rondje, dit vasthouden en een lijntje naar links te trekken, totdat de imageview blauw wordt en een Horizontal Spacing menu verschijnt. Selecteer daar Top uit. Doe hetzelfde maar dan het lijntje naar beneden trekken. Selecteer dan Left uit het Vertical Spacing menu.

Kopieer de button vervolgens één voor één naar de overige bordposities en positioneer elke button precies in het midden van zijn bordpositie, op dezelfde manier als hierboven beschreven.

Run de app om te kijken of alles daadwerkelijk goed staat.

Creëer voor de button linksboven een outlet button0 in *viewcontroller.swift*. Creëer tevens een action *buttonPressed* voor deze buttons en voeg daar de volgende code aan toe:

@IBAction func buttonPressed(sender: AnyObject) {

var image = UIImage(named: "tictactoe\_kruisje")

sender.setImage(image, forState: UIControlState.Normal)

}

Creëer voor de andere buttons hetzelfde action item. Run de code bij wijze van test, wordt nu op een willekeurige button gedrukt dan verandert deze in een kruisje. Als dit werkt haal dan van alle buttons de images in de *Attribute Inspector* weg zodat een leeg spelbord overblijft.

Nu volgt de programmalogica. Als op een knop wordt gedrukt moet afwisselend een rondje en een kruisje verschijnen, te beginnen met een rondje. Hiervoor wordt een integer variabele *goNumber* gedefinieerd die initieel op 1 wordt gezet. Is goNumber oneven, dan moet een rondje verschijnen en anders een kruisje. In de *buttonPressed* functie gebruiken we hiervoor de *modulo* functie (%) . Deze komt er als volgt uit te zien:

@IBAction func buttonPressed(sender: AnyObject) {

var image:UIImage

if (goNumber++ % 2 == 1) {

// oneven: rondje laten zien

image = UIImage(named: "tictactoe\_rondje")!

} else {

// even beurt: kruisje laten zien

image = UIImage(named: "tictactoe\_kruisje")!

}

sender.setImage(image, forState: UIControlState.Normal)

}

Run de app ter controle. Beurtelings verschijnt een rondje en een kruisje op het bord wanneer op een veld wordt gedrukt. De app heeft nog twee problemen: (1) er kan meerdere keren op een veld worden gedrukt, als er al wat staat dan verandert dit en (2) een winnaar met drie rondjes of kruisjes op een rij wordt niet herkend.

Om dit te verhelpen gebruiken we een array *gameState* die voor elk veld de status bijhoudt: 0 = leeg, 1 = rondje, 2 = kruisje:

var gameState = [0, 0, 0, 0, 0, 0, 0, 0, 0]

Om in *buttonPressed* de button te herkennen gebruiken we de *tag* ervan. Dit is een nummer dat aan een button kan worden gekoppeld. Vul deze in voor de buttons, start met 0 voor de button linksboven, de button rechts ervan krijgt tagnummer 1, etc. en de button rechtsonder krijgt tagnummer 8. In *buttonpressed* gebruiken we *sender.tag* om het tagnummer te achterhalen. De code wordt nu als volgt:

@IBAction func buttonPressed(sender: AnyObject) {

if (gameState[sender.tag] == 0) {

var image:UIImage

if (goNumber++ % 2 == 1) {

// oneven: rondje laten zien

image = UIImage(named: "tictactoe\_rondje")!

gameState[sender.tag] = 1

} else {

// even beurt: kruisje laten zien

image = UIImage(named: "tictactoe\_kruisje")!

gameState[sender.tag] = 2

}

sender.setImage(image, forState: UIControlState.Normal)

}

}

Run de app en constateer dat het eerste probleem is opgelost. Om te zien dat er een winnaar creëren we een constante 2-dimensionale array waarin alle winnende combinaties zijn opgenomen:

let winningCombs =

[[0,1,2], [3,4,5], [6,7,8], // horizontale winnende combinaties

[0,3,6], [1,4,7], [2,5,8], // vertikale winnende combinaties

[0,4,8], [2,4.6]]

In *buttonPressed* controleren we of er een winnende combinatie van rondjes of kruisjes is:

for combination in winningCombs {

if (gameState[winningCombs[0] != 0) { // mag niet leeg zijn

if (gameState[winningCombs[0]]==gameState[winningCombs[1]]

&&

gameState[winningCombs[1]]==gameState[winningCombs[2]]) {

winner = gameState[winningCombs[0]]

}

}

}

De app is nu in staat een winnaar te bepalen. Dit moet nog netjes worden gemeld (bijv. dmv. de springende alien uit de vorige les 60), het spel moet worden gestopt en er moet worden gevraagd of opnieuw wordt gespeeld.

Voor de winnaar melding gebruiken we een label dat we over het speelbord heenleggen. Neem een donkere achtergrond (bijv. blauw) zodat het speelbord eronder niet te zien is en gebruik lichte letters (bijv. wit). Align het label tov. de imageview zodat het er in horizontale toestand ook goed uitziet, zet ook de breedte en hoogte vast. Creëer een outlet *winnerlabel* voor het label in *viewController.swift*. Het label moet initieel verborgen worden, mag pas getoond worden als er een winnaar is. In *viewDidLoad* zetten we het label links van het scherm waardoor het onzichtbaar is:

winnerLabel.center = CGPointMake( winnerLabel.center.x-400,

winnerLabel.center.y)

Het *winnerLabel* zetten we met een animatie op het scherm. Voeg in *buttonPressed* de volgende code toe:

if (winner != 0) {

// Laat winnerLabel met een animatie zien

println("winnaar")

if (winner == 1) {

winnerLabel.text = "Plusjes hebben gewonnen"

} else {

winnerLabel.text = "Kruisjes hebben gewonnen"

}

UIView.animateWithDuration(1, animations: {

self.winnerLabel.center = CGPointMake( self.winnerLabel.center.x+400,

self.winnerLabel.center.y)

self.winnerLabel.alpha = 1

})

}

Deze code werkt op zich goed, echter zodra het eerste rondje wordt gespeeld, verschijnt het label (zonder tekst) over het speelbord heen. Kennelijk iets wat in XCode veranderd is. De oplossing hiervoor is om de alpha waarde van het label initieel (in *viewDidAppear)* op 0 te zetten, en in *buttonPressed* op 1 als winner!=0 is.

Er zijn nog enkele issues. Wanneer er een winnaar is, kan er nog worden doorgespeeld. Dit is niet te bedoeling en wordt eenvoudig voorkomen door in *buttonPressed* het eerste if-statement aan te passen naar:

if (gameState[sender.tag] != 0) && (winner == 0) {

Verder is het nog niet mogelijk een nieuw spel te starten. Hiervoor creëren we een button ‘Play again’ over het speelbord. Geef het een onderscheidende achtergrondkleur, bijv. groen. Creëer een outlet *playAgainButton* en een action *playAgainButtonPressed*. In *viewDidAppear* maken we de button onzichtbaar door de alpha op 0 te zetten:

playAgainButton.alpha = 0

Wanneer een winnaar is gevonden zetten we hem in de animatie weer op 1:

self.playAgainButton.alpha = 1

In *playAgainButtonPressed* moeten de game variabelen gereset worden, de *winnerLabel* en *playAgainButton* onzichtbaar gemaakt worden en moeten alle buttons weer ‘leeg’ gemaakt worden:

goNumber = 0

winner = 0

gameStat = [0, 0, 0, 0, 0, 0, 0, 0, 0]

winnerLabel.alpha = 0

winnerLabel.center = CGPointMake( winnerLabel.center.x-400,

winnerLabel.center.y)

playAgainButton.alpha = 0

var button: UIButton

for var i:Int in 0..8 {

button = view.viewWithTag(i) as UIButton

button.setImage(nil, forState: UIControlState.Normal)

}

LET OP: Dit werkt alleen als alle views een uniek tagnummer hebben. Dit betekent dat de views in het storyboard die nog tagnummer 0 hebben, een ander tagnummer (> 10) moeten krijgen.

Er zijn natuurlijk allerlei verbeteringen mogelijk: animatie voor de rondjes en kruisjes, score bijhouden, een lijn trekken van de winnende combinatie, etc.

De code staat in 6.3.

**Les 67 Using maps (en 68 voor code files)**

MapKit framework toevoegen in de Build phases van het project. Import MapKit toevoegen aan code. MapView in het storyboard plaatsen en runnen.

Inzoomen op map in simulator mbv Alt-key en muis.

Om wat met de mapview te kunnen doen moet de delegate ervan vastgelegd worden, door in het storyboard rechts te klikken op de mapview en in het pop-up menu de delegate optie te selecteren en deze naar de view controller te trekken (geel symbooltje).

Tevens de MapViewDelegate toevoegen aan de ‘class ViewController’ die de mapview gaat bewerken.

In deze les wordt een app gemaakt die een mapview laat zien met een zelfgekozen locatie als middelpunt. De locatie bestaat uit een latitude en longitude, die met Google Maps kunnen worden opgezocht.

De map laat een region zien, deze wordt gedefinieerd door de locatie en de ‘span’. De span bestaat uit een ‘latitude delta’ en ‘longitude delta’ en is een maat voor het te tonen gebied waarop wordt ingezoomd, uitgedrukt in kaarteenheden (lengte cq. breedtegraden). Een standaard maat voor beide is 0,01 (afh. van gewenste zoomniveau).

Code is te vinden in les 6.4 code files.

**Les 69 Adding annotions to maps (en 70 voor code files)** dmv longpress gesture

Annotatie variabele creëren dmv

var annotation = MKPointAnnotation()

Vergeet de () niet! Definieert de annotation als instantie van klasse MKPointAnnotation.

Annotaties hebben attributen zoals de coordinaten (lat+long), title en subtitle. Voeg annotatie toe aan de mapview dmv

mymap.addAnnotation(annotation)

Definieer een longpress gesture dmv

var lpGesture = UILongPressGestureRecognizer(target:self, action:”action:”)

Geef aan (in sec) hoe lang minimaal gedrukt moet worden om van een long press gesture te spreken:

lpGesture.minimumPressDuration = 2.0

Target is hier de view controller die de gesture moet afvangen., “action:” is de naam van de functie die wordt uitgevoerd als de longpress gedetecteerd wordt. Let op de dubbele punt, deze geeft aan dat functie “action” een parameter heeft:

func action(gestureRecognizer:UIGestureRecognizer)

Was de dubbele punt weggelaten dan had de compiler gezocht naar een functie “action” zonder parameters.

Voeg de gesture recognizer toe aan de mapview:

myMap.addGestureRecognizer(lpGesture)

De functie action moet vaststellen waar op het beeldscherm de longpress heeft plaatsgevonden. Dit kan dmv

var lpScreenPoint = gestureRecognizer.locationInView(self.myMap)

Let op dat omdat myMap buiten functie “action” als klasse variabele is gedefinieerd, deze moet worden geïdentificeerd met

self.myMap

De beeldschermcoördinaten moeten omgezet worden naar map coördinaten:

var lpMapPoint:CLLocationCoordinate2D

= myMap.convertPoint(lpScreenPoint, toCoordinateFromView:self.myMap)

Vervolgens kan de annotatie gedefinieerd worden als hierboven en aan de mapview myMapworden toegevoegd.

Code is weer te vinden in les 6.4 code files.

**Les 71 Determining a user’s location (en 72 voor de code files)**

CoreLocation framework toevoegen in de Build phases van het project. Import CoreLocation toevoegen aan code.

Om locaties te kunnen opvragen moet de app daarvoor toestemming worden gegeven door de gebruiker. Hiervoor moet in info.plist de volgende property worden toegevoegd:

*NSLocationWhenInUseUsageDescription*

Bij eerste keer draaien van de app zal deze toestemming vragen om de locatie van de gebruiker op te vragen. De string in het Value veld van deze property zal in een pop-up verschijnen (dit is in feite de reden voor het opvragen van de locatie).

Een andere property voor als de app niet actief is (in de background loopt) is

*NSLocationAlwaysUsageDescription*

Ook hier weer in het Value veld van deze property de string invullen die wordt getoond in de pop-up. Zonder deze properties (vooral de eerste) kan de app geen gebruik maken van location services!

Om locatiemanager functies te gebruiken in de view controller moet deze voldoen aan het CLLocationManagerDelegate protocol. De verplichte functies in dit protocol moeten door de viewcontroller geïmplementeerd worden.

Nu kan de location manager variabele gedefinieerd worden:

var locMgr = CLLocationManager()

De viewcontroller is de delegate van de location manager:

locMgr.delegate = self

Vervolgens de nauwkeurigheid van de positiebepaling door locMgr instellen:

locMgr.desiredAccuracy = kCLLocationAccuracyBest (of een andere waarde)

Dan de toestemming voor het bepalen van de gebruikerslocatie opvragen:

locMgr.requestWhenInUseAuthorization()

Dit is afhankelijk van de opgegeven property in info.plist. Als alternatief kan

locMgr.requestAlwaysAuthorization()

worden opgegeven.

Nu kan het updaten van de locatie gestart worden:

locMgr.startUpdatingLocation

Voor het verwerken van de ontvangen locaties is een functie van het CLLocationManagerDelegate protocol nodig:

func locationManager(manager:CLLocationManager,

didUpdateLocations locations:AnyObject[])

De parameter *locations* is een array, het kan soms voorkomen dat meerdere locaties worden teruggegeven. Rob Percival stelt dat de eerste locatie uit *locations* gebruikt moet worden:

var userLoc:CLLocation = locations[0] as CLLocation

Let op dat *locations* van het type AnyObject is en naar een CLLocation moet worden gecast.

Mbv de code uit les 64 kan de huidige locatie in de mapview getoond worden.

Een andere functie is nodig om fouten bij het opvragen van locaties te ondervangen:

func locationManager(manager:CLLocationManager,

didFailWithError error:NSError)

Deze functie wordt ook aangeroepen als de gebruiker geen toestemming geeft aan de app om zijn locatie op te vragen.

Als de app op de simulator draait dan kan in menu Debug – Location gekozen worden welke locatie (statisch of dynamisch) gesimuleerd wordt.

NB. Locatie None geeft de didFailWithError error eenmalig (en niet continu). Zolang de locatie None blijft wordt nl. geen nieuwe locatie doorgegeven.

**Les 73 WhereAmI app (en 74 voor de code files)**

In deze les wordt een app gemaakt die op één scherm in tekst (dus geen kaart) de actuele locatie (lat+long), snelheid, koers, hoogte en het dichtstbijzijnde adres laat zien.

NB. De heading zoals in eerste instantie genoemd wordt niet geïmplementeerd omdat hiervoor een andere functie (didUpdateHeading) nodig is.

Zie les 68 voor de basis: CoreLocation framework, app properties en location manager definiëren en initialiseren.

Vervolgens labels toevoegen op het storyboard voor de diverse te tonen gegevens. In de viewcontroller class code outlets creëren voor de output labels.

Mbv het embed in-string command de ingelezen gegevens kopiëren naar de labelteksten:

latitudeLabel.text = “(\currentLocation.coordinate.latitude)”

longitudeLabel.text = “(\currentLocation.coordinate.longitude)”

courseLabel.text = “(\currentLocation.course)”

speedLabel.text = “(\currentLocation.speed)”

altitudeLabel.text = “(\currentLocation.altitude)”

Het weergeven van het dichtstbijzijnde adres is wat meer werk:

CLGeocoder().reverseGeocodeLocation(currentLocation, completionHandler:

{(placemarks, error) in

if (error) { println(“Error: \(error)”) }

else {

let p = CLPlacemark(placemark: placemarks?[0]

as CLPlacemark

}

}

NB placemarks bevat de informatie waaruit de adresgegevens geëxtraheerd kunnen worden. Dit is een array met elementen van het type CLPlacemark. Net als bij het ontvangen van een locatie-update gebruiken we de eerste placemark op index 0.

Variabele p bevat een aantal elementen die het dichtstbijzijnde adres specificeren:

* subThoroughFare: huisnummer (indien bekend, kan ook nil zijn)
* thoroughFare: straatnaam
* locality: stad
* subLocality: stadsgebied
* administrativeArea: staat/provincie

Hiernaast voor de hand liggende elementen zoals postalCode, country etc.

Zie voor code de files in project 6.7.

**Les 75 Advanced segues (en 76 voor code files)**

In deze les wordt wat dieper ingegaan op segues: scherm overgangen. Er wordt een app gemaakt die bestaat uit een table view (de initiële view). Wanneer op een table view element wordt geklikt dan verschijnt het bijbehorende detail info scherm. Vanuit hier kan weer terug worden gegaan naar de table view.

Naast de beheersing van scherm overgangen wordt ingegaan op het overhandigen van data naar een ander scherm, in casu het rijnummer van het tabel element waar op is geklikt.

Creëer een nieuw project en sleep een table view controller in het storyboard, onder de eerste view controller. Er staan nu dus 2 view controllers op het storyboard. Stel de table view controller voor deze test app in op static cells (via Attribute Inspector – Content) (dynamische content volgt in de volgende les). Stel de 3 rijen van de table view in op Basic style (via Attribute Inspector – Style), waarna de titel in elk van de cellen kan worden ingevoerd (rij 0 t/m 2).

Creëer een button op de eerste view controller. De bedoeling hiervan is dat wanneer hier op wordt gedrukt, de table view verschijnt. Rechtsklik hiervoor op de button en sleep deze naar de table view controller. Er verschijnt nu een menu waarin het type overgang kan worden gekozen. Ze werken allemaal, ieder heeft een specifieke animatie.

XCode heeft geen viewcontroller geassocieerd met de table view controller, die hebben we nodig om vanuit code de table view te beheren. Creëer hiervoor een nieuwe Cocoa Touch class (File – New – File). Noem deze naar wens, het moet een subclass zijn van UITableViewController. In het project window verschijnt nu de TabeViewController.swift file.

Nu moet de link tussen de table viewcontroller in het storyboard en de nieuwe class gelegd worden. Selecteer in het storyboard de table viewcontroller, selecteer rechts op het scherm de Identity Inspector en selecteer bij Custom class de class in die zojuist gecreëerd is. Hiermee is de link gelegd: de nieuw class controleert de table view controller in het storyboard.

*NB. In de video wordt met een XCode 6 Beta gewerkt. Daar is het nodig om een functie init(coder aDecoder:NSCoder) toe te voegen, anders volgt een exception. Bij mijn versie van XCode 6.1.1 is dit niet nodig.*

Wordt de huidige code gerund dan verschijnt een lege tabel op het scherm nadat op de button is gedrukt. Dit is omdat de functies die de table view definiëren nog niet volledig zijn geïmplementeerd. Het aantal secties en aantal rows per section staan default op 0. Zet deze op resp. 1 en 3. Wordt de app gerund dan verschijnen de table row titels op het scherm.

Nu moet nog de weg terug geïmplementeerd worden. Klikken/tappen op een tabel element laat de initiële view weer zien. Hiervoor moeten elk van de 3 tabel elementen gekoppeld worden aan de initiële view controller, zoals eerder andersom is gedaan.

Tot slot moet nog het rijnummer van het tabel element waarop is geklikt terug worden gegeven aan de initiële view controller. Hiervoor definiëren we een globale variabele SelectedRow in *viewcontroller.swift*, dus buiten de class definition van ViewController. Deze variabele is hierbij globaal toegankelijk vanuit andere swift code files, zonder dat een import oid zoals in Objective-C nodig is.

De meest voor de hand liggende manier om het geselecteerde rijnummer op te vragen is de functie *tableview-****will****SelectRowAtIndexPath* te gebruiken in de table viewcontroller swift file. Deze wordt getriggerd nadat op een tabel element is geklikt en vóórdat de segue plaatsvindt.

NB er is ook een functie *tableview-****did****SelectRowAtIndexPath*, maar die gaat hier niet werken omdat die pas wordt geactiveerd nadat de segue heeft plaatsgevonden. Ik heb dit getest en het effect is dat het vorige geselecteerde rijnummer wordt afgedrukt in de viewDidLoad functie van de initiële viewcontroller.

De code is te vinden in 6.8.

**Les 77 Memorable places (en 78 voor code files)**

In deze les wordt een app ontwikkeld die bestaat uit 2 schermen: 1 mapview waarop de gebruiker na een longpress een locatie kan opslaan, de opgeslagen locaties worden getoond in het tweede scherm, een table view die initieel leeg is. Er kan heen en weer worden gesprongen tussen de twee schermen.

**User interface**

Start met de WhereAmI app van les 70 (code in project 6.7). Laat voor nu alle labels en outlets op het storyboard staan. Dit scherm zal worden gebruikt als de map view.

Voeg een navigation bar toe bovenaan het scherm en geef het de titel My Map oid. Voeg een bar button item rechts en noem deze ‘Find me’ oid (geeft de huidige locatie centraal weer op het scherm) en links een bar button item Back (terug naar scherm met opgeslagen locaties). Voeg vervolgens de map view toe over de bestaande labels en outlets heen.

Dan de table viewcontroller toevoegen aan het storyboard. Hier willen we ook een navigation bar toevoegen, maar dat gaat niet op de standaard manier via het Attributes Inspector menu in een table viewcontroller. Het kan in een tabelelement maar dat is zinloos. Hiervoor heeft XCode een andere oplossing. Zorg dat de table viewcontroller geselecteerd is en ga in XCode naar het Editor menu, kies *Embed In – Navigation Controller*. Er verschijnt nu een aparte navigation viewcontroller in het storyboard met een link naar de table viewcontroller. De navigation viewcontroller kan veel, maar voor nu is het nut ervan dat een navigation bar verschijnt in de table viewcontroller.

De navigation bar is niet goed zichtbaar maar is er wel. Check dat de optie *Shows Navigation Bar* is aangevinkt. Vink ook de optie *Is Initial Viewcontroller* aan, zodat de tabel als eerste op het scherm verschijnt bij opstarten van de app.

Pas de titel aan (bijv. My Favorites oid) en voeg een bar button item toe (selecteer de *Add* functie om een locatie toe te voegen aan de tabel).

De table viewcontroller bestaat alleen op het storyboard, er is nog geen code waarmee de tableview beheerd kan worden. Voeg hiertoe een nieuwe class ‘TableViewController’ (of een naam naar wens toe) via *File-New-File…* en selecteer *Cocoa Touch Class*. Stel de custom class van de table viewcontroller in het storyboard in op de zojuist aangemaakte class (*Custom Class* in de *Identity Inspector*).

Om controle in de code te hebben over de tabel cellen moet een identifier in het cel prototype worden ingevoerd. Selecteer hiertoe in het storyboard de *Table view cell*, selecteer de *Attribute Inspector* en vul een naam in het *Identifier* veld, bijv. ‘Cell’.

**Data model: places**

De ‘longpress’ locaties worden opgeslagen in de variabele *places*. Dit is een array van dictionaries. Van elke locatie wordt de naam, de latitude en de longitude opgeslagen:

var places = [Dictionary<String, String>()]

De labels en values van de dictionary zijn beide strings.

**NB**. Dit creëert een array van dictionaries met een leeg element, wat leidt tot een tabel met een leeg eerste element. Dit is ongewenst. Beter is om places op éé n van de volgende manieren te definiëren:

var places = [Dictionary<String,String>]() // haakjes buitenboord

var places:[Dictionary<String,String>] = []

Het toevoegen van een locatie aan *places* gaat dan als volgt:

places.append([“name”: “Taj Mahal”, “lat”:”27.175277”, “long”:”78.042128”])

Het aantal secties van de tabel is 1:

override func numberOfSectionsInTableView(tableView: UITableView) -> Int {

return 1

}

Het aantal elementen in de tabel is gelijk aan de size van *places*:

override func tableView( tableView: UITableView,

numberOfRowsInSection section: Int) -> Int {

return places.count

}

Het weergeven van een tabel element moet de naam van de locatie tonen:

override func tableView( tableView:UITableView,

cellForRowAtIndexPath indexPath: NSIndexPath)

-> UITableViewCell {

let cell = tableView.dequeReusableCellWithIdentifier(“Cell”,

forIndexPath: indexPath) as UITableViewCell

cell.textLabel.text = places[indexPath.row][“name”]

return cell

}

Let op dat de cell identifier exact dezelfde naam heeft als zojuist in het prototype cell is ingevoerd.

**User interface: tonen kaart wanneer een locatie in de tabel wordt getapt**

Creëer een segue van de table viewcontroller naar de ‘map’ viewcontroller. Geef deze segue een identifier, klik hiervoor op de segue (de pijl) en vul een naam in de Attribute Inspector, bijv. “findPlace”. Deze is nodig in de begeleidende code wanneer een rij in de tabel wordt geselecteerd:

override func tableView( tableView: UITableView,

didSelectRowAtIndexPath indexPath:NSIndexPath) {

performSegueWithIdentifier("findPlace", sender: indexPath)

}

De kaart mapview verschijnt nu in beeld wanneer op een tabelelement wordt geklikt. De navigation bar die we eerder aan de map viewcontroller hadden toegevoegd is echter niet zichtbaar. De navigation bar van onze navigation viewcontroller wordt getoond. Om dit te voorkomen moet deze juist voor de segue (performSegueWithIdentifer) worden uitgezet:

self.navigationController?.navigationBarHidden = true

De kaart die wordt getoond heeft nog niet de gekozen locatie als middelpunt. Hiertoe creëren we een globale variabele *activePlaceIndex*, de index in *places* van de in de tabel geselecteerde locatie.

Omdat de map viewcontroller ook toegang moet hebben tot de *places* array, moet die ook globaal gedefinieerd worden (dus vóór de class definitie).

Om de mapview te beheren moet een outlet ervan worden gecreëerd in de *viewcontroller.swift*. Om de locatie te tonen moeten de latitude en longitude *strings* uit de dictionary worden geconverteerd naar *doubles*. Hiervoor moet de *String* eerst naar een *NSString* geconverteerd worden waarop de *doubleValue* methode kan worden toegepast:

let latitude = NSString(places[activePlaceIndex][“lat”]).doubleValue

let longitude = NSString(places[activePlaceIndex][“long”]).doubleValue

Hiermee kan de region worden ingesteld die in de mapview wordt getoond (zie ook les 64):

var latitude:CLLocationDegrees = lat

var longitude:CLLocationDegrees = long

var latDelta:CLLocationDegrees = 0.01 // latitude zoomlevel

var longDelta:CLLocationDegrees = 0.01 // longitude zoomlevel

var span:MKCoordinateSpan =

MKCoordinateSpanMake(latDelta, longDelta)

var location:CLLocationCoordinate2D =

CLLocationCoordinate2DMake(latitude, longitude)

var region:MKCoordinateRegion =

MKCoordinateRegionMake(location, span)

myMapView.setRegion(region, animated:true)

Tappen op de Find Me button centreert de kaart op de actuele locatie. Creëer hiertoe een *action* voor de button in *viewcontroller.swift* en voeg de volgende code toe:

manager.requestWhenInUseAuthorization()

manager.startUpdatingLocation()

Het verzoek om toegang tot de actuele locatie hoeft pas te worden gedaan als op de FindMe button wordt gedrukt (hoewel wellicht beter om bij opstart te doen…). Het idee is dat de actuele locatie enkel wordt getoond wanneer op de FindMe button wordt gedrukt en dan eenmalig wordt opgevraagd, zodat de batterij wordt gespaard. Locaties worden doorgegeven door de methode *locationManager-didUpdateLocations*. Hierin wordt de doorgekregen locatie in de mapview getoond en de locatiemanager wordt gestopt:

func locationManager( manager:CLLocationManager,

didUpdateLocations locations:[AnyObject]) {

var userLocation:CLLocation = locations[0] as CLLocation

var latitude:CLLocationDegrees = userLocation.coordinate.latitude

var longitude:CLLocationDegrees = userLocation.coordinate.longitude

var latDelta:CLLocationDegrees = 0.01 // latitude zoomlevel

var longDelta:CLLocationDegrees = 0.01 // longitude zoomlevel

var span:MKCoordinateSpan =

MKCoordinateSpanMake(latDelta, longDelta)

var location:CLLocationCoordinate2D =

CLLocationCoordinate2DMake(latitude, longitude)

var region:MKCoordinateRegion =

MKCoordinateRegionMake(location, span)

myMapView.setRegion(region, animated:true)

manager.stopUpdatingLocation()

}

NB Om de huidige locatie met een blauwe punt oid. te laten zien moet een attribuut van de mapview aangezet worden:

myMapView.showsUserLocation = true

**User interface: implementatie van de Back knop in de mapview**

Creëer een segue van de Back knop naar de navigation controller (niet de table controller). Wordt de app nu gerund dan blijkt de Back knop prima te werken, echter is de navigation bar boven de table view verdwenen, vanwege de eerdere wijziging in de code van *tableview-didSelectRowAtIndexPath*.

Dit kan worden ondervangen door in de (map) *viewcontroller.swift* de methode *prepareForSegue* te overriden. Alleen als de betreffende segue wordt uitgevoerd als gevolg van de Back knop moet de navigation bar worden aangezet. De segue moet daartoe een naam gegeven worden in het storyboard. Klik op de segue en selecteer de Attributes Inspector en vul de naam in het *Identifier* veld in (bijv. “backToMyPlaces”). De code is als volgt:

override func prepareForSegue(segue:UIStoryboardSegue!, sender: AnyObject!){

if (segue.identifier == “backToMyPlaces”) {

self.navigationController.NavigationBarHidden = false

}

}

Run de app en de navigation bar blijft op het scherm wanneer in de map view op de Back button wordt gedrukt. Het valt op dat in de navigation bar van de table view een “< Back” button is toegevoegd. Wordt hier op gedrukt dan verschijnt de map view zonder navigation bar.

Een betere – en eenvoudiger – oplossing is om ipv. obv. segues en de *tableview-didSelectRowAtIndexPath* methode de navigation bar te verbergen c.q. weer te geven wanneer een view wordt verlaten. Dit kan dmv. *viewWillDisappear*:

override func viewWillDisappear(animated: Bool){

// aanzetten in map viewcontroller, uitzetten in table viewcontroller code

self.navigationController.navigationBarHidden = true/false

}

**User interface: implementatie van de Add (+) knop in de table view**

De bedoeling van deze knop is een locatie toe te voegen aan de My Places lijst. In de les gebeurt dit op een wat vreemde manier. Het enige wat de knop namelijk doet is de map view tonen met de huidige locatie centraal. Hiervoor wordt een segue (genaamd “addLocation”) gecreëerd van de Add (+) knop naar de map viewcontroller. In de code van de map viewcontroller wordt een *prepareForSegue override* ingevoegd die checkt of het de “addLocation” segue betreft.

Omdat de viewcontroller moet weten op welke locatie moet worden ingezoomd, wordt de activeLocationIndex in prepareForSegue op -1 gezet, teneinde onderscheid te maken met de selectie van een bestaande locatie in de table view. Wanneer de *viewDidLoad* voor de map viewcontroller wordt uitgevoerd, checkt deze of de waarde -1 is. Zo ja, dan wordt dezelfde code als in de *findMe* methode uitgevoerd:

locationManager.requestWhenInUseAuthorization()

locationManager.startUpdatingLocation()

Is *activeLocationIndex* ongelijk aan -1 dan wordt ingezoomd op de locatie in *places* met de betreffende index.

Het argument om *activeLocationIndex* te misbruiken is om geen extra globale variabele te gebruiken, maar echt chique is het gebruik van een ongeldige indexwaarde niet.

**User interface: toevoegen annotations aan geselecteerde locatie uit My Places tabel**

Wanneer een locatie wordt geselecteerd uit de My Places tabel wordt deze centraal op de mapview getoond en wordt er een annotatie aan toegevoegd, met coördinaten en titel van de geselecteerde locatie. Voeg in *viewDidLoad* van *ViewController.swift* de volgende code toe aan het blok waar *activePlaceIndex > -1*:

var annotation = MKPointAnnotation()

annotation.coordinate = location // coordinaten van de geselecteerde locatie

annotation.title = places[activePlaceIndex][“name”]

myMapview.addAnnotation(annotation)

**User interface: toevoegen locatie na long press**

Zie ook les 66. Definieer de longpress gesture recognizer in *viewDidLoad* van *ViewController.swift*:

var longPressGR = UILongPressGestureRecognizer(

target: self action:”addLocation:”) // Dubbele punt!!

longPressGR.minimumPressDuration = 2.0

myMapview.addGestureRecognizer(longPressGR)

Methode *addLocation* voegt de huidige locatie toe aan *places*. Voor de title van de annotatie wordt het dichtstbijzijnde adres gebruikt (via reverse geocoding, zie les 70). Dit kan natuurlijk ook via een invulscherm ingegeven worden (oefening). De code wordt dan:

func addLocation(gestureRecognizer:UIGestureRecognizer){

var touchPoint = gestureRecognizer.locationInView(self.myMapview)

var newCoordinate = myMapview.convertPoint(touchPoint,

toCoordinateFromView: self.myMapview)

// loc is nodig om juiste type voor reverseGeocodeLocation te krijgen

var loc: CLLocation( latitude: newCoordinate.latitude,

longitude: newCoordinate.longitude)

CLGeocoder().reverseGeocodeLocation(loc,

completionHandler: {(placemarks, error) in

{

if (error) { println(“Geocoding error \(error)”)}

else {

let p = CLPlacemark(placemarks:

placemarks?[0] as CLPlacemark)

var subThoroughFare: String

var thoroughFare: String

if (p.subThoroughfare) {

subThoroughFare = p.subThoroughFare

} else {

subThoroughFare = “”

}

if (p.thoroughfare) {

thoroughFare = p.thoroughFare

} else {

thoroughFare = “”

}

var annotation = MKPointAnnotation()

annotation.coordinate = newCoordinate

var title = “\(subThoroughFare) \( thoroughFare)”

if (title == “”) {

title = “Unknown location added at \(date)”

}

// let op: *self* is nodig in een closure

self.myMapview.addAnnotation(annotation)

// voeg de locatie tot slot toe aan *places*

places.append([“name”]:title,

[“lat”]: newCoordinate.latitude

[“long”]:newCoordinate.longitude)

}

}

}

Wordt de app gedraaid dan werkt deze op zich goed, echter wordt een nieuwe locatie 2 keer toegevoegd aan de MyPlaces tabel. Dit komt omdat de *addLocation* methode 2 keer wordt aangeroepen: zodra de minimum tijd voor de long press is verstreken en wanneer de long press wordt beëindigd (= vinger los van scherm). Dit kan worden ondervangen door te checken dat de state van de gesture recognizer op ‘began’ staat. Voeg de volgende code toe bovenaan *addLocation*:

if (gestureRecognizer.state == UIGestureRecognizerState.Began) {

// zet de code binnen dit blok

}

Tot zover deze app. Wat nog ontbreekt is permanente opslag, dit kan mbv *NSUserDefaults* worden gerealiseerd (zie les…). Ook de horizontale weergave is niet in orde.

De code is te vinden in 6.9.

**Les 79 Working with audio (en 80 voor code files)**

Start met een audio file (bijv. “bach.mp3”) in de project directory te zetten. Sleep hiertoe een audio file vanuit de Finder naar de project directory en selecteer ‘Copy items if needed’ zodat het ook daadwerkelijk in de project directory terecht komt.

Voeg vervolgens vanuit de *Build phases* de library *AVFoundation.framework* toe en voeg het statement

import AVFoundation

toe in *viewcontroller.swift*.

Definieer in *viewcontroller.swift* een (voor nu leeg) audioplayer object:

var audioPlayer: AVAudioPlayer = AVAudioPlayer()

In *viewDidLoad*, creëer een string die aangeeft waar de audiofile staat:

var fileLocation = NSString(string: NSBundle.mainBundle().pathForResource(

“bach”, ofType:”mp3”)!)

Let op dat de filenaam en type als aparte parameters worden doorgegeven. Vergeet het uitroepteken niet!

Initialiseer nu de audioplayer:

var error:NSError? = nil

audioPlayer = AVAudioPlayer( contentsOfURL: NSURL(string: fileLocation),

error: &error)

Let op dat de error parameter van het type NSErrorPointer? Is (een optional). Daarom wordt het adres van error doorgegeven (&error) en wordt error een NSError optional gedefinieerd.

De muziek wordt afgespeeld door de audioplayer te starten:

audioPlayer.start()

Voeg in het storyboard 2 buttons toe: “stop” en “volume” en creëer actions in *viewcontroller.swift*. De player wordt als volgt gestopt:

audioPlayer.stop()

Het volume is een getal tussen 0 en 1. Zet het op 0 wanneer op de volumeknop wordt gedrukt:

audioPlayer.volume = 0

De code files staan in 6.10

**Les 81 App: Back to Bach (en 82 voor code files)**

In deze les wordt de muziek player app van de vorige les uitgebreid met wat controls voor start/stop, pauze en volume.

Voeg in het storyboard een navigation controller toe bovenaan het scherm (zet grootte op 4 inch voor iPhone 5) en een toolbar onderaan het scherm.

Voeg een bar button item toe aan de navigation bar: een Play button (ingebouwd in XCode).

Voeg bar button items toe aan de toolbar: een Pause button (ook ingebouwd) en een Stop button, gebruik daarvoor niet de ingebouwde X maar noem deze ‘Stop’. Voeg tussen deze buttons een ‘flexible space bar button item’ toe.

Voeg voor de volumeregeling een slider toe aan het storyboard.

Om het af te maken wordt een Bach plaatje toegevoegd. Creëer een imageview in het storyboard die een deel van het scherm beslaat en vul in de Attribute Inspector bij Image de naam van het plaatje in (= “bach.jpg”).

Voeg actions aan de code toe voor alle toegevoegde buttons. Voeg voor de slider ook een outlet *volumeSlider* toe. De code voor de buttons is eenvoudig, resp.

audioPlayer.play()

audioPlayer.stop()

audioPlayer.pause()

De slider min en max waarde staan in de Attribute Inspector default op 0 en 1, wat overeenkomt met de min en max volumewaarden voor de audioplayer. Aanpassen van het volume is als volgt:

audioPlayer.volume = volumeSlider.value

De app werkt nu. Echter de Stop button werkt meer als een pauze knop, de muziek wordt niet naar het startpunt teruggezet.. Voeg in de action functie voor de stop button het volgende toe:

audioPlayer.currentTime = 0

De code staat in 6.11.

Het afspelen van meerdere files kan vrij eenvoudig worden gerealiseerd door een table viewcontroller toe te voegen met daarin een tableview met per element een nummer. Afhankelijk van het geselecteerde item wordt de *fileLocation* aangepast. Dit komt in een volgende les aan de orde.

**Les 83: Swipes and Shakes (en 84 voor code files)**

In deze les wordt een app gebouwd die swipe en shake bewegingen registreert en hiervan melding doet op de console. That’s it, geen storyboard nodig hiervoor.

De code voor een right-swipe gesture recognizer:

var swipeRight = UISwipeGestureRecognizer(target: self, action: “swiped:”)

swipeRight.direction = UISwipeGestureRecognizerDirection.Right

self.view.addGestureRecognizer(swipeRight)

Action is de functie die wordt uitgevoerd wanneer de right swipe wordt herkend. Vergeet de “:” niet aan het einde van de naam (*swiped* heeft parameters).

De code voor *swiped*:

func swiped(gesture: UIGestureRecognizer) {

if (let swipeGesture=gesture as? UISwipeGestureRecognizer) {

switch swipeGesture.direction {

case UISwipeGestureRecognizerDirection.Right:

println(“Right swipe”)

default:

break

}

}

}

De code voor andere swipe richtingen (Up, Down, Left) ligt voor de hand. Let op het vraagteken in het if statement, dit is nodig voor het geval de gesture geen swipe is.

Swipes zijn gerelateerd aan een viewcontroller. Shakes zijn dat niet. Wanneer een shake wordt gedetecteerd dan wordt de functie motionEnded aangeroepen. Deze kan overridden worden met een eigen invulling:

override func motionEnded( motion: UIEventSubType,

withEvent event: UIEvent!){

if event.subtype == UIEventSubType.MotionShake {

println(“Shake detected”)

}

}

Een shake kan in de simulator worden gesimuleerd via menu Hardware – Shake Gesture.

Code is in 6.12.

**Les 85: Sound shaker (en 86 voor code files)**

In deze les wordt een eenvoudige app gebouwd waarin gedownloade muziek via een shake kan worden afgespeeld. In de les worden sounds van de site *soundbible.com* gehaald.

Download een aantal sounds, zet deze in een folder en sleep de folder vanuit de Finder naar de XCode projectdirectory. Vink ‘copy items if needed’ zodat de sounds naar de projectdirectory worden gekopieerd.

Voeg in Build Phases de AVFoundation.framework toe aan het project en importeer AVFoundation in *viewcontroller.swift*.

Voor het creëren van de audioplayer zie les 76.

De namen van de soundfiles worden in een array gezet:

var soundFiles = [“bean”, “belch”, “giggle”, “pew”, “pig”] //en nog meer…

Via een randomizer wordt na een shake één van de soundfiles afgespeeld. De random index wordt als volgt verkregen:

var soundIndex = Int(arc4random\_uniform(UInt32(soundFiles.count)))

Let op dat de parameter van *arc4random*() van het type *UInt32* is. Een array index is echter van het type Int, vandaar de cast naar Int om het geheel heen.

De code is te vinden in 6.13.

**Section 7 – Core Data**

**Les 87: Introducing Core Data (en 88 voor code files)**

Core Data is in essentie een manier om appdata op te slaan. Tot nu toe hebben we hiervoor *NSUserDefaults* gebruikt, maar dat is door de benodigde type conversies tamelijk omslachtig in de omgang en niet geschikt als de app grote hoeveelheden data wil opslaan.

Creëer een nieuwe applicatie en vink ditmaal de ‘*Use Core Data’* optie aan. In de *AppDelegate*.*swift* file verschijnen nu de nodige functies die we niet eerder gezien hebben, zoals *persistentStoreCoordinator*, *managedObjectContext*, *saveContext*, etc.

In de projectdirectory staat ook een file *<projectnaam>.xcdatamodeld*, hier staat de app database feitelijk in, de structuur ervan is ook zichtbaar.

In *AppDelegate.swift* wordt de *CoreData* geimporteerd. In de *ViewController.swift* file niet, omdat niet elke viewcontroller dat nodig heeft. Voor deze demo wel, voeg deze toe aan ViewController.swift:

import CoreData

Voordat er gecodeerd wordt voegen we data toe aan de CoreData database. Selecteer hiertoe de file *<projectnaam>.xcdatamodeld* in de projectdirectory en druk op *Add Entity* onderaan het scherm. Voeg nu een Attribute “username” met type “String” toe. Doe hetzelfde voor “password”, ook met type “String”. Geef de tabel waar nu Entity bij staat de naam Users. Dit is onze CoreData datastructure.

Voeg nu de volgende code toe aan viewDidLoad:

var appDel: AppDelegate = UIApplication.sharedApplication().delegate

as AppDelegate

var context:NSManagedObjectContext = appDel.managedObjectContext!

We gaan nu een gebruiker aan de Core Data database toevoegen:

var newUser = NSEntityDescription.insertNewObjectForEntityForname(

“Users”, inManagedObjectContext: context) as NSManagedObject

newUser.setValue(“Frank”, forKey: “username”)

newUser.setValue(“1234”, forKey: “password”)

context.save(nil) // nil: error parameter gebruiken we niet

De gebruiker wordt nu opgeslagen in de CoreData dabase, maar er wordt nog niets uit gehaald. Dat gaat dmv. een fetch request als volgt:

var request = NSFetchRequest(entityName: “Users”) // de tabelnaam

request.returnsObjectsAsFaults = false

var results = context.ExecuteFetchRequest(request, error: nil)

println(results)

Run de app, in de video worden meerdere entries met dezelfde naam en password getoond, het betreft echter verschillende objecten. Dit is normaal gedrag.

De opgehaalde informatie heeft nog geen bruikbaar formaat. Dit kan verkregen worden door in een *for-loop* door de *results* variabele te lopen:

if (results?.count > 0) {

for result: AnyObject in results! {

println(result)

}

} else {

println(“No results”)

}

Het if-statement is een goede gewoonte om eerst te testen of er überhaupt iets in de CoreData database staat.

Als de username of password benaderd moet worden dan kan dit ook, door de veldnamen rechtstreeks te gebruiken:

println(result.username)

println(result.password)

De code is te vinden in 7.1.

**Les 89: Working With Core Data (en 90 voor code files)**

Deze les gaat in op bijzonderheden met de CoreData app van de vorige les 84 en andere acties mbt. Core Data zoals het verwijderen en opzoeken van data.

De methode om een keyvalue op te vragen, zoals

result.password

levert een *optional* resultaat op, en dit wordt ook als zodanig weergegeven als dit resultaat met een *println* wordt afgedrukt, er verschijnt *<Optional(“pass”)>* in de output console. Om dit te voorkomen en alleen de string af te drukken kan de keyvalue met een temp variabele worden opgevraagd:

let pass = result.password!

Het uitroepteken zorgt dat de waarde geforceerd wordt overgedragen aan de variabele pass. Het risico is dat als het password leeg is, dit een nil resultaat oplevert waardoor de app crashed. Dit kan worden afgevangen met een if-statement:

if let pass = result.password! {

println(pass)

}

Wanneer wordt getracht de username op te vragen geeft XCode een error dat het veld username onbekend is bij het object result, hoewel het wel degelijk bestaat. Dit kan op een nette en betere manier opgelost worden door de valueForKey functie te gebruiken:

var username = result.valueForKey(“username”) as? String

De ‘*as String’* frase is nodig omdat *valueForKey* een resultaat van het type *AnyObject* teruggeeft en we juist een *String* willen hebben. Dit is ook beter dan de variabele ‘*username’* van het type *AnyObject* te definiëren.

Het vraagteken is nodig omdat XCode anders klaagt (= syntax eror) dat een ‘*forced downcast in conditional binding produces non-optional type String*’. We moeten daarom een optional downcast toepassen.

**Verwijderen van core data items**

Het verwijderen van een Core Data element gaat als volgt:

context.deleteObject(result as NSManagedObject)

context.save(nil) // NB Dit hoeft niet direct, maar niet vergeten!

waarbij result gebruikt werd in de loop (zie vorige les 84).

**Opzoeken van core data items**

Dit gaat mbv. een predicaat, d.i. een zoekcriterium:

request.predicate = NSPredicate( format:”username = %@”,

“Frank” )

var results = context.executeFetchRequest(request, error: nil)

Dit levert alle records met username “Frank” op in variabele results. De “%@” is een placeholder voor de string(s) die als parameter meegegeven worden, een overblijfsel uit Objective-C. In het voorbeeld is dit alleen “Frank”, maar er kunnen meerdere parameters meegegeven worden.

**Updaten van core data items**

Als voorbeeld het bijwerken van het password van user “Frank”:

result.setValue(“abcdefg”, forKey: “password”)

Hierbij is result de variabele die gebruikt wordt in de loop.

De code staat in 7.2.

**Les 91: Downloading Images From The Web (en 92 voor code files)**

Deze les gaat over permanente opslag, maar dan niet dmv. Core Data. Er wordt een foto van het internet geladen, deze wordt opgeslagen , vanuit de opslag geladen en afgebeeld op het scherm. De gebruikte technieken zijn nodig voor de RSS app die later zal worden ontwikkeld.

Creëer een nieuw *Single View Application* project, vink Core Data niet aan, het zit niet in de weg maar voor deze app is het niet nodig. Zoek een plaatje op het internet, rechtsklik en selecteer *Save URL*. De url hebben we nodig in de volgende code in *viewDidLoad*:

let url = NSURL(string:”http://…….jpg”)

let urlRequest = NSURLRequest(URL: url)

De foto wordt via een *asynchrone data request* geladen. Omdat het asynchroon is, is het van belang dat zeker gesteld wordt dat het plaatje daadwerkelijk geladen is als het op het scherm wordt gezet. Dit gebeurt dan ook in de *completion handler* van de *sendAsynchronousRequest*, die pas wordt aangeroepen als de asynchrone request volledig is uitgevoerd (al dan niet foutloos):

NSURLConnection.sendAsynchronousRequest(

urlRequest,

queue: NSOperationQueue.mainQueue(), // de standaard queue

completionHandler: { (data, response, error) in

if (error) {

println(“Error \(error)”)

} else {

let image = UIImage(data: data)

self.myImage = image // bij wijze van test

// code om foto op te slaan volgt nog

}

})

Let op dat *self* nodig is om de *myImage* outlet te benaderen omdat die code in een closure zit. Creëer een image view op het storyboard met outlet ‘*myImage’*.

Run de app om na te gaan dat de foto wordt geladen en afgebeeld. We vervolgen met het opslaan van de foto. Maak commentaar van statement *self.myImage = image* en voeg de volgende code toe:

var documentsDirectory:String? // is initieel nil, maar waarom ? nodig??

var paths:[AnyObject] =

NSSearchPathForDirectoriesInDomains(

NSSearchPathDirectory.DocumentDirectory, //= app directory

NSSearchPathDomainMask.userDomainMask, //= default waarde

true) // Bool expand tilde y/n

if (paths.count > 0){

documentsDirectory=paths[0] as? String

var savePath = documentsDirectory! + “<naam foto>.jpg”

NSFileManager.defaultManager().createFileAtPath(

savePath, contents: data, attributes: nil)

self.myImage.image = UIImage(named: savePath)

} else {

println(“No document directory returned for app”)

}

Functie *NSSearchPathForDirectoriesInDomains* geeft een array van directories terug waarin de app zijn data mag opslaan. Dit is er normaliter één, dus we gebruiken de eerste als we tenminste iets terug krijgen. Vervolgens wordt hieruit de file met naam *savePath* gecreëeerd met als inhoud de geladen foto (= *data parameter*). Tot slot wordt de foto op het scherm gezet, maar nu door deze te laden vanuit de file (*named: savePath*).

Om te testen dat de foto daadwerkelijk is opgeslagen, halen we het deel dat de foto van internet laat weg en runnen vervolgens de code.

Code is te vinden in 7.3.

**Les 93: Processing JSON data (en 94 voor code files)**

JSON staat voor *JavaScript Object Notation* (zie [www.w3schools.com](http://www.w3schools.com)) en is een veelgebruikte structuur om data op te slaan. Het wordt veel gebruikt en beschouwd als de eenvoudiger XML. Het is de de facto standaard voor data uitwisseling tussen apps en webservices. We hebben het nodig voor de nog te bouwen RSS app.

Creëer een Single View Application project zonder Core Data. Zoek een site op die Json data produceert, zoals [www.telize.com/geoip](http://www.telize.com/geoip) of de *OpenWeatherMap* api (zie de site voor details).

Voeg de volgende code toe aan *viewDidLoad*:

let urlPath = “http://www.telize.com/geoip”

let url = NSURL(string: urlPath)

let session = NSURLSession.sharedSession()

let task = session.dataTaskWithURL(url, completionHandler: {

(data, response, error) -> Void in *// NB “-> Void” mag ook weg*

if (error) {

println(“Error \(error)”)

} else {

println(data) *// bij wijze van test*

}

})

task.resume() *// haal data op*

Run de app en constateer dat de data wordt geladen maar ongeformatteerd wordt weergegeven en niet als Json data. Vervang *println(data)* door:

let jsonResult = NSJSONSerialization.JSONObjectWithData(

data,

options: NSJSONReadingOptions.MutableContainers,

error: nil) *// NB error normaal niet negeren, opties niet nodig*

as NSDictionary

println(jsonResult)

Aan de hand van het JSON format weten we dat de binnengehaalde data bestaat uit *key-value pairs: een dictionary*, vandaar de *typecast*. Functie *JSONObjectWithData* heeft nog een optie en bijbehorende error parameter, maar die gebruiken we hier niet.

Een individueel element uit de dictionary kan eenvoudig worden opgevraagd:

println(jsonResult[“country\_code”])

, uiteraard mits *country\_code* een dictionary element is.

De code staat in 7.4.

**Les 95: Webviews (en 96 voor de code files)**

In deze les wordt ingegaan hoe een *webview* aan een applicatie wordt toegevoegd en er web content in kan worden gezet. Rob Percival laat nog een bijzonder type applicatie zien dat kan worden gecreëerd vanuit XCode, de *PhoneGap* based applicatie (staat er bij mij niet bij). Hiermee kan een applicatie in HTML en javascript worden ontwikkeld en in een fullscreen webview worden gerund. Voor meer info zie *phonegap.com*.

In deze les laten we een website zien in een webview en ruwe HTML content. Creëer een nieuwe applicatie en maak een web view aan het storyboard. Creëer een outlet van de web view genaamd *myWebview*. Voeg de volgende code toe aan *viewDidLoad*:

var url = NSURL(string: “htp://www.google.com”)

var request = NSURLRequest(URL: url)

myWebview.loadRequest(request)

Rund de app en zie hoe een fullscreen versie van de webpagina wordt getoond. Navigeren naar bijv. naar de vorige pagina kan niet, meer dan een full screen pagina is het niet.

Vervolgens wordt getoond hoe je standaard HTML kan laden in een webview. Pas de code als volgt aan:

var html = “<html><head></head><body><h1>

<Hello World!></h1></body></html>”

webview.loadHTMLString(html, baseURL: nil)

Hiermee wordt een blanco pagina met de tekst “Hello World!” geladen.

De code staat in 7.5.

**Les 97: App: Blog Reader (en 98 voor code files)**

In deze les wordt een blogger reader app ontwikkeld. Hiervoor creëren we een *master-detail application* voor een *universal device* – dus zowel iPhone als iPad (vink *Core Data* aan). Het mooie hiervan is dat er een user interface beschikbaar wordt gesteld die zowel voor iPhone als iPad geschikt is maar anders werkt. Dit wordt duidelijk wanneer de template die zojuist gecreëerd is wordt gerund. Op de iPhone verschijnt een master scherm, een tabel waarop elementen kunnen worden toegevoegd. Wordt op een (niet leeg) tabel element geklikt dan verschijnt het detail info scherm.

Wordt de template op de iPad gerund, dan verschijnen de master en detail scherm op één scherm, met in het detail scherm de info van het geselecteerde tabel element.

Rob Percival gebruikt voor deze app de *blogger api* van Google, zie op *developers.google.com*. Hiervoor is een *api key* nodig (net als bij openweathermap.org). Ga naar de pagina op *developers.google.com* over de *Blogger api* en zoek de tekst mbt *‘Acquiring and using an API key’*. Klik op de *Google Developers Console* link en klik op *Create Project* op de volgende pagina. Voer in het volgende scherm de projectnaam ‘*Blogger*’ in en wacht vervolgens tot deze gecreëerd is. Vervolgens verschijnt het *project dashboard*. Open links op het scherm ‘*APIs & auth*’ en klik op *APIs*. Zoek het zojuist gecreëerde Blogger project in de lijst die nu verschijnt. Aciveer de API door op de *OFF knop* te drukken waardoor deze *ON* gaat (in groene achtergrond). Er mogen 10.000 requests per dag gedaan worden, meer dan genoeg voor onze app.

Klik vervolgens in het menu links op het scherm op *Credentials*. Creëer een nieuwe key, selecteer *Browser key* en laat het tekstvak leeg zodat de api overal vandaan toegankelijk is (dus niet van een specifieke site). Klik op Ok waarna de API key in het scherm getoond wordt. Kopieer deze naar het clipboard.

Zoek vervolgens de tekst ‘*Receiving a blog by its URL’* op. Kopieer de link in de browser url invoerveld, en zet er *‘&key=<key>*’ achter waarbij voor <key> de API key ingevuld moet worden. Vervolgens verschijnt de Json data op het scherm.

Waneer de link zonder api key wordt ingevoerd verschijnt een foutmelding in Json format dat het aantal requests overschreden is.

De documentatie op developer.com geeft alle info die nodig is om met de Blogger api om te gaan, zie bijv. ‘*Retrieving posts from a blog’*.

We starten de app met het creëren van het data model. Selecteer de <projectnaam>.*xcdatamodeld* file en voeg een entity toe (*Add Entity* knop), noem deze *BlogItems*. Voeg hier String 4 attributen aan toe: *title*, *publishedDate*, *content* en *author*.

Wat betreft de lay-out willen we de *Edit* en *+* buttons in de navigation bar boven de tabel verwijderen en moeten de titels van de schermen (nu Master en Detail) aangepast worden. Verder ziet het er goed uit.

De buttons worden in de code van *MasterViewController.swift* toegevoegd. Haal deze code weg. Pas in het storyboard de titels van de Master en Detail schermen aan, noem deze resp. *Blog Items* en *Blog Content*. In het *Blog Content* scherm gaan we later een *webview* toevoegen.

We starten met het toevoegen van blog items aan de table view. Pas in *MasterViewController.swift* de volgende code aan:

- in *numberOfSectionsInTableView*:

return 1

- in *numberOfRowsPerSection*:

return 3 // bij wijze van test, dit verandert nog

- in *cellForRowAtIndexPath*:

// verwijder het self.configureCell statement

cell.textLabel.text = “Blog Item” // bij wijze van test, later dynamisch

Run de app. Er verschijnt een tabel met 3 blog items. Het klikken op één van de items geeft een runtime error, omdat we de betreffende code weggehaald hebben. Dat wordt als eerste gefixed.

De overgang van master naar detail view gebeurt in *prepareForSegue*. De eerste twee statements zijn in orde, maar de *fetchedResults* functie mislukt omdat de objecten niet meer bestaan (die hebben wij kennelijk weggehaald). Verwijder dit statement.

Verwijder ook het *controller.detailItem = object* statement.

De laatste twee statements laten we staan, deze creëren nl. de Back button voor ons en die hebben we nodig.

Kijk vervolgens in de *DetailViewController.swift* naar de *viewDidLoad*. Hierin wordt een functie *configureView* aangeroepen. Deze stelt de labeltekst in op de timestamp. Dit halen we weg, we willen immers geen timestamp maar een blog item laten zien. Laat alleen de volgende code in configureView staan:

if let label = self.detailDescriptionLabel {

label.text = “Item tapped”

}

Run de app opnieuw en selecteer een blog item, de app crashed niet meer en in het detailscherm verschijnt ‘Item tapped’, er gebeurt dus iets als op een blog item wordt geklikt.

Nu gaan we de Json content downloaden. Dit doen we in de viewDidLoad van MasterViewController.swift. Zet de code uit les 90 hier in:

let urlPath = “http://www.telize.com/geoip”

let url = NSURL(string: urlPath)

let session = NSURLSession.sharedSession()

let task = session.dataTaskWithURL(url, completionHandler: {

(data, response, error) -> Void in *// NB “-> Void” mag ook weg*

if (error) {

println(“Error \(error)”)

} else {

let jsonResult = NSJSONSerialization.JSONObjectWithData(

data,

options: NSJSONReadingOptions.MutableContainers,

error: nil) *// NB error normaal niet negeren*

as NSDictionary

println(jsonResult)

}

})

task.resume() *// haal data op*

Run de app. Op het scherm gebeurt niets maar de Json content wordt opgehaald, dus dat deel werkt. De gebruikte URL klopt natuurlijk niet, dat passen we bij deze aan. We gebruiken de officiële Google blogspot (op *google.blogspot.co.uk*). Zoek op *developers.google.com* naar informatie over het binnenhalen van de posts van een blog. Een blog heeft een id die in de URL moet worden opgenomen, tevens moet daar de eigen api key in opgenomen worden:

https://www.googleapis.com/blogger/v3/blogs/<blog-id>/

posts?key=<YOUR-API-KEY>

NB Dit is één regel. Hierin is *<blog-id>* de id van een specifieke blog (zoals 2399953 cq 10861780 die in de les wordt gebruikt) en *<YOUR-API-KEY>* de api key die eerder is opgevraagd voor dit project. Vervang de *urlPath* in de code door de geconstrueerde url en run de code om te testen. In de console verschijnt de geformatteerde Json data.

De volgende stap is de ontvangen Json data geschikt te maken voor display in de tabel. Er komt een enorme berg Json data binnen. Nadere analyse van de Json data leert dat de post is opgedeeld in een aantal items. Vervang de *println(jsonResult)* door

println(jsonResult[“items”]

Run de app opnieuw, de afzonderlijke blog items worden nu in de console getoond. Het statement

println(jsonResult[“items”][0])

toont het eerste item. Het item zelf is ook een *dictionary* blijkt uit nadere beschouwing. Het bevat o.a. de *author*, die we willen tonen in de tabel. Het element *author* is zelf ook weer een *dictionary*. We beginnen met het extraheren van de *author* van het eerste blog item. Vervang de *println(jsonResult[“items”][0])* door:

var item = jsonResult[“items”][0] as NSDictionary

var authorDictionary = item[“author”] as NSDictionary

var author:NSString = authorDictionary[“displayName”] as NSString

println(author)

De typecasts zijn nodig om met het juiste type variabelen te werken en niet met het type *AnyObject* dat standaard wordt teruggegeven. Andere data die we uit de post gebruiken:

var content:String = item[“content”] as NSString

var title:String = item[“title”] as NSString

var publishedDate = item[“published”] as NSString

// plus bij wijze van test een println van deze items

De content wordt als HTML doorgegeven, daarvoor hebben we een *webview* nodig om het netjes weer te geven (volgt later).

Nu de afzonderlijke elementen van een post correct worden binnengehaald, kunnen we verder met het weergeven van alle ontvangen posts. Creëer hiervoor een *array van dictionaries*:

var items:[[String:String]()]

De array items wordt in een *for-loop* gevuld met de ontvangen data:

var item:AnyObject

var authorDictionary: anyObject

for var i=0; i<jsonResult[“items”].count; i++ {

items.append([String:String]()]

item = jsonResult[“items”][i] as NSDictionary

items[i][“content”] = item[“content”] as? NSString

items[i][“title”] = item[“title”] as? NSString

items[i][“publishedDate”] = item[“published”] as? NSString

authorDictionary = item[“author”] as NSDictionary

items[i][“author”] = authorDictionary[“displayName”]

}

println(items) // bij wijze van test

De variabelen *item* en *authorDictionary* worden als *anyObject* gedefinieerd, in de assignment volgt een typecast zodat het type klopt.

Het statement *items.append([String:String]()]* creëert een lege *dictionary*, dit moet voor iedere items index gedaan worden, omdat elk array element op zichzelf een *dictionary* is.

Het *vraagteken* bij de *‘as’* niet vergeten.

NB Gooi de eerdere code waarin de afzonderlijke elementen worden gevuld weg.

In principe hebben we nu alle data om weer te geven in de tabel. Maar we doen dit vanuit *Core Data*. Daarom gaan we eerst Core Data vullen met de ontvangen data. Omdat alle benodigde data in één array items staat, had dit eenvoudiger met *NSUserDefaults* geïmplementeerd kunnen worden. Omdat Core Data meer mogelijkheden heeft, zoals het opzoeken van data, en bij wijze van oefening wordt in deze les Core Data gebruikt.

Voeg in *viewDidLoad* de volgende code toe:

var appDel:AppDelegate = UIApplication.sharedApplication().delegate

as AppDelegate

var context:NSManagedObjectContext = appDel.managedObjectContext!

var newBlogItem = NSEntityDescription.insertNewObjectForEntityForName(

“BlogItems” // de naam van de tabel in Core Data

inManagedObjectContext: context)

as NSManagedObject

// Voeg bij wijze van test de volgende info toe:

newBlogItem.setValue(“Frank”, forKey:”author”)

newBlogItem.setValue(“New blog item”, forKey:”title”)

newBlogItem.setValue(“Blogs are great”, forKey:”content”)

newBlogItem.setValue(“2015-01-16”, forKey:”publishedDate”)

context.save(nil)

// En haal deze wijze van test op uit Core Data:

var request = NSFetchrequest(entityName: “BlogItems”)

request.returnsObjectsAsFaults = false

var result = context.executeFetchRequest(request, error:nil)

println(results)

Run de app en zie dat het in Core Data opgeslagen test blog item correct wordt getoond in de console. We kunnen nu verder met het opslaan in Core Data van de ontvangen posts. Definieer daartoe *newBlogItem* als globale variabele, juist buiten de *for-loop*:

var newBlogItem: NSManagedObject

Kopieer de zojuist ingevoerde code naar het einde van de *for-loop*:

newBlogItem = NSEntityDescription.insertNewObjectForEntityForName(

“BlogItems” // de naam van de tabel in Core Data

inManagedObjectContext: context)

as NSManagedObject

// Voeg bij wijze van test de volgende info toe:

newBlogItem.setValue(items[i][“author”], forKey:”author”)

newBlogItem.setValue(items[i][“context”], forKey:”title”)

newBlogItem.setValue(items[i][“content”], forKey:”content”)

newBlogItem.setValue(items[i][“publishedDate”], forKey:”publishedDate”)

context.save(nil)

Zet de volgende code direct na de *for-loop* om na te gaan dat de opslag in Core Data gelukt is:

// En haal deze wijze van test op uit Core Data:

var request = NSFetchrequest(entityName: “BlogItems”)

request.returnsObjectsAsFaults = false

var result = context.executeFetchRequest(request, error:nil)

println(results)

Run de app. Alle ontvangen posts worden in de console getoond. Merk op dat de context niet volledig wordt getoond. Dit komt doordat deze als onderdeel van de items variabele is opgeslagen en items is een object object variabele. Wanneer objecten de console worden gelogd, worden deze afgekapt. Ga er van uit dat de volledige content is opgeslagen in Core Data.

Omdat de app al een paar keer gedraaid heeft met Core Data, zijn er dubbele posts opgeslagen. Er is nl. geen check geweest of een ontvangen post al in de Core Data was opgeslagen. Ook wordt alles opgehaald, ook oude posts die mogelijk niet meer interessant zijn. En dan staan er nog een aantal test posts in die voor deze cursus zijn aangemaakt. Allemaal zaken die feitelijk nog ingebouwd zouden moeten worden, eventueel via een dialoog. Voor deze les wordt het simpel gehouden: de data wordt bij elke run gereset. Voeg hiertoe direct na het *else-statement* de volgende code toe:

var request = NSFetchRequest(entityName: “BlogItems”)

request.returnsObjectsAsFaults = false

var result = context.executeFetchRequest(request, error:nil)

for result in results {

context.deleteObject(result as NSManagedObject)

context.save(nil) // geen error check, moet eigenlijk wel!

}

Omdat de *request* en *result* variabele hier al gedefinieerd worden, moet de *‘var’* definitie onderaan de *for-loop* nog weggehaald worden. Run de app en check dat alle overbodige posts verdwenen zijn.

De volgende stap is de ontvangen items in de *table view* te tonen. Omdat we gebruik maken van het *master-detail application* template is een groot deel van het werk al gedaan. We kunnen nl. de variabele *fetchedResultsController* gebruiken, zie hiervoor *MasterViewController.swift*. Pas hier het volgende aan:

* vervang in de *entity* assignment de itemnaam door “*BlogItems*”
* vervang in de *sortDescriptor* assignment de key door “*datePublished*”

In de functie *configureCell*, vervang in de *cell.textLabel.text* assignment de key door “*title*”.

Run de app, de titels worden nu in de tabel getoond, echter passen ze vrijwel allemaal niet in de beschikbare ruimte. Pas daartoe in het *main storyboard* de lettergrootte van de titel in het *prototype cell* van de *Master Viewcontroller* aan, zet deze bijv. 14. Run de app, er wordt nu meer tekst getoond.

Om het nog wat mooier te maken, passen we de *style* van de *cell* aan. Zet deze van *Basic* over naar *Subtitle*. Dit biedt de mogelijkheid om extra informatie toe te voegen, bijv. de author en datum. Voeg aan *configureCell* de volgende code toe:

cell.detailTextLabel.text = object.valueForKey(“author”).description + “ at “

+ object.valueForKey(“datePublished”).description

Haal het loggen naar de console weg (*println(items)* statement) en run de app. Om het nog wat mooier te maken kan het in de Json data meegeleverde image worden meegegeven. Elk tabelelement heeft hiervoor een image attribuut, zie het main storyboard. Deze kan natuurlijk ook in de code geïnitialiseerd worden.

De app nadert zijn voltooiing. De detail view moet nog ingevuld worden met de content van het in de *table view* geselecteerde item. Omdat de content die in de Json data is opgenomen HTML code is, gebruiken we daarvoor een *webview*. Creëer een *webview* in de *Detail viewcontroller* en maak het full screen. Zet in *Auto Layout* alle *spacing to nearest neighbour constraints* op 0. Creëer een *outlet* voor de *webview* in *DetailViewController.swift* en noem deze ‘*contentWebview’*.

Het is de bedoeling dat de content van het actieve item (= het geselecteerde item) uit de tabel wordt getoond. De code in de *Master viewcontroller* biedt hiervoor de nodige aanknopingspunten. Creëer als eerste in *MasterViewController.swift* een globale variabele *activeItem*. Juist voordat naar de *Detail viewcontroller* wordt gesprongen, wordt de functie *prepareForSegue* aangeroepen. Hierin moet activeItem worden geïnitialiseerd, dit kan op dezelfde manier waarop in *configureCell* de *textLabel* en *detailTextLabel* velden zijn geïnitialiseerd. Voeg in *prepareForSegue* direct na de definitie van de *indexPath* variabele de volgende code toe:

let object = self.fetchedResultController.objectAtIndexPath(indexPath)

as ManagedObject

activeItem = object.valueForkey(“content”).description

Verwijder de code uit functie *configureView* en zet de volgende code ervoor in de plaats:

webview.loadHTMLString(activeItem, basURL: nil)

De app is hiermee klaar. Diverse verbeteringen zijn nog mogelijk, zoals het toevoegen van een foto aan elk tabel item. Zelf zou ik een extra niveau willen toevoegen waarin uit diverse blogs gekozen kan worden. De HTML opmaak kan nog worden aangepast ,zoals font en fontgrootte. Auteur en datum kunnen nog in labels in de detail view getoond worden. Etc. etc.

**Section 8 – Instagram Clone**

**Les 99: Introduction to Parse.com**

In deze sectie wordt de kernfunctionaliteit van *Instagram* nagebouwd. Het komt er in essentie op neer dat je je eigen foto’s kunt delen met anderen, dat je foto’s van bekenden kunt opzoeken en het soort foto’s kunt opzoeken die je leuk vindt. Onze Instagram kloon app laat je een foto in de app importeren en delen met de mensen die jou volgen. Nieuwe technieken in deze sectie zijn (1) de toegang tot de camera en het toegang krijgen tot opgeslagen foto’s van anderen en (2) het opslaan van data op het internet, zodat dit gedeeld kan worden met andere apparaten. Voor het laatste gebruiken we het platform *Parse.com* (recent overgenomen door FaceBook). Parse.com is een zg. backend die het o.a. mogelijk maakt om gratis (!) on-line data op te slaan zodat geen eigen servers nodig zijn, push notifications uit te voeren en basic computation. En dit alles zonder dat ingewikkelde code nodig is.

Alternatieven voor Parse.com zijn o.a. Apple’s CloudKit en FireBase (recent opgekocht door Google).

Parse.com biedt drie producten: (1) Parse Core voor de opslag en manipulatie van data in de cloud, (2) Parse Push waarmee op eenvoudige wijze Push notifications aan een app kunnen worden toegevoegd en (3) Parse Analytics waarmee kan worden bekeken hoeveel en wanneer gebruik wordt gemaakt van een app en hoe de app gebruikt wordt.

Parse heeft (nog?) een genereus gratis programma: 20 GB file en database opslag, tot 30 requests per seconde, 2 TB data transfer per maand en 1 background job.

Om zelf de app na te bouwen moet een account bij Parse.com worden aangemaakt. Doe dit, vul Instagram voor de naam van de applicatie in en wanneer gereed kies dan voor Data – Mobile – iOS – Objective-C. Swift wort ook ondersteund maar staat niet expliciet genoemd, vandaar dat Objective-C moet worden gekozen. Kies vervolgens voor New Project.

In de volgende les wordt de installatie van de SDK behandeld.

**Les 100: Parse Hello World (en 101 voor code files)**

In deze les wordt de Parse library aan een nieuw project gekoppeld en wordt data op de Parse.com backend opgeslagen.

Het blanco project dat van Parse.com kan worden gedownload is helaas in Objective-C geschreven. We zullen daarom zelf ons blanco project in Swift gaan schrijven in deze les. Daarvoor moeten we de SDK downloaden. Klik op *Downloads* onderaan de pagina (in de les nog bovenaan) en vervolgens op *iOS SDK*, bij mij op 20-jan-2015 is het versienummer daarvan inmiddels 1.6.3. Unzip vervolgens de gedownloade file.

Creëer een nieuwe *Single View* application, noem het Instagram en vink Core Data af, dit gebruiken we niet in deze les. Select iPhone als device. Save het project in een folder naar keuze.

Het eerste wat nodig is is de integratie van de Parse sdk in ons project. Sleep hiertoe vanuit de Finder de files *Bolts.framework* en *Parse.framework* naar het Instagram project in XCode. Vink *Copy items if needed* aan. Ga vervolgens in XCode naar *Build Phases* en open *Link Binary With Libraries*. De beide frameworks staan hier al tussen, maar er zijn er nog veel meer nodig. Ga hiervoor weer naar de *Parse.com* website en selecteer *Quickstart* vanuit het menu links op het scherm (in de les staat dit nog bovenaan het scherm). Selecteer vervolgens *Data* – *Mobile* – *iOS* – *Objective*-C – *Existing Project*. Scroll naar beneden en er verschijnt een flinke lijst met libraries die allemaal in het project moeten worden meegenomen. Deze lijst is inmiddels groter dan die uit de les. Het toevoegen van een library gebeurt door in de Build Phases pagina op het + symbool te klikken. Er verschijnt een lijst, waaruit één van de benodigde libraries kan worden geselecteerd. Doe dit voor alle benodigde libraries uit de lijst van Parse.com.

Scroll verder op de Parse.com webpagina. Er verschijnen instructies om code toe te voegen aan *AppDelegate.m*. Dit is echter allemaal Objective-C code dus hebben we een workaround nodig. Het eerste statement:

#import <Parse/Parse.h>

kan niet in Swift code opgenomen worden. Voor de import van de Parse.h file moeten we een zg. *bridge header* creëren. Creëer in XCode een nieuwe file (menu File – New – File…). Selecteer de *Objective-C* file uit het window dat verschijnt en noem het bijv. BridgeHeader. Laat File Type op Empty File staan. Antwoordt vervolgens Yes op de vraag of je een *Objective-C bridging header* wil configureren. Er zijn nu 2 files aan het Instagram project toegevoegd: een *BridgeHeader.m* file en een *Instagram-Bridging-Header.h* file. De *BridgeHeader.m* file hebben we niet nodig (hij is leeg – Empty File) en kan verwijderd worden.

Zet het import statement

#import <Parse/Parse.h>

in de *Instagram-Bridging-Header.h* file. De file bevat verder geen code. De Parse library is nu beschikbaar voor onze te bouwen app.

In het vervolg wordt een stukje code geschreven om data op de Parse.com backend op te slaan. Voeg in *viewDidLoad* in *ViewController.swift* de volgende code toe:

Parse.setApplicationId( <your Instagram application id>,

clientKey: <your client id>)

Merk op dat de *auto-predict* ook op de Parse library objecten werkt.

De <*your Instagram application id*> en <*your client id*> zijn eerder op Parse.com aangemaakt, zie hiervoor de openstaande webpagina (net onder de #import <Parse/Parse.h>. Elke op Parse.com geregistreerde applicatie krijgt een eigen application id. De client id is gekoppeld aan het Parse.com account.

Data wordt op Parse.com als klasse objecten opgeslagen. Een klasse heeft diverse attributen. Voor deze les slaan we de high scores op van personen. Zo ontstaat een klasse ‘highScores’ met attributen ‘score’ en ‘userName’. Voeg de volgende code toe in viewDidLoad:

var highScore = PFObject(className: “highScores”)

highScore.setObject(“Frank”, forKey:”name”)

highScore.setObject(100, forKey:”score”)

score.saveInBackground() // NB is nu score.save() geworden

Op deze manier kunnen allerlei typen objecten aan klasse highScore toegevoegd worden, dus ook bijv. arrays, dictionaries en samengestelde objecten.

Run de app. Er gebeurt niets zichtbaars in de app zelf. Ga nu naar Parse.com en open het dashboard, dit is te vinden door de muis over je profielnaam te bewegen, rechtsboven op het scherm (dit is veranderd tov. de les) en op *Instagram* (= applicatienaam) onder *Dashboard* te klikken.

Hoewel de applicatie werkt, is er geen check gedaan of het opslaan succesvol is verlopen. Dit is wel nodig, want het werkt alleen bij een werkende internet verbinding. De check kan in de *saveInBackgroundWithBlock* functie worden uitgevoerd door de block parameter ervan te gebruiken. Vervang de *saveInBackground* aanroep door:

score.saveInBackgroundWithBlock { (success: Bool, error: NSError) -> Void in

if (success == true) {

println(“Score created with ID \(score.objectId)”)

} else {

println(“Error storing score: \(error)”)

}

}

Run de app opnieuw. De score wordt opnieuw opgeslagen, een melding verschijnt in de console. In het dashboard op Parse.com zijn nu 2 identieke scores met verschillende object id’s te zien. Dit is even zoeken, maar het staat er echt!

Zet nu de internetverbinding uit en run de app opnieuw. De foutmelding verschijnt nu in het console.

De code is te vinden in 8.2.

**Les 102: Retrieving and updating data with Parse (en 103 voor code files)**

In deze les wordt ingegaan op het ophalen en aanpassen van op parse.com opgeslagen data (objecten). Dit gebeurt op basis van de code van de vorige les 100.

Zet de code voor initialisatie en opslag van het highScore object in commentaar. Het ophalen van een object gaat dmv. Een query:

var query = PFQuery(className: “highScore”)

query.getObjectInBackgroundWithId(<object id>, {

(highScore: PFObject!, error:NSError!) -> Void in

if (error == nil) {

// toon alle highScore data

println(“Opgehaalde highscore data: \(highScore)”)

// toon alleen de naam van de opgehaalde highScore data

println(“Name: \(highScore.objectForkey(“name”))”)

// alternatief:

println(“Name: \(highScore[“name”])”)

} else {

println(“Fout bij ophalen highscore data: \(error)”)

}

})

Het <object id> is in dit geval een vaste ingevoerd id van een eerder opgeslagen object, het kan op de Parse.com site worden opgezocht.

Het aanpassen van data gaat als volgt. Vervang de println() statements na de test if (error == nil) door:

highScore[“name”] = “Franky”

highscore[“score”] = 300

highScore.save() // NB saveInBackground() is obsolete, niet meer gebruiken

In de les maakt Rob een fout door een verkeerde veldnaam (zeg “newscore”) voor de score te gebruiken. Het effect is dat dit veld wordt toegevoegd aan de klasse highScore, en dat het record dat we updaten een waarde in dit veld krijgt. Andere bestaande highScore records krijgen het veld er wel bij maar het is dan leeg.

De code staat in 8.3.

Op Parse.com is veel documentatie te vinden met voorbeeldcode, zowel in Objective-C als Swift.

**Les 104: Getting images from the photo library and camera**

In deze les gebruiken we de code van de voorgaande lessen, deze wordt uitgebouwd tot de volledige Instagram kloon die later wordt ontwikkeld. Zet de code van de voorgaande lessen in commentaar, deze gebruiken we later weer.

Voeg in ViewController.swift de volgende protocollen toe:

class ViewController: UIViewController, UINavigationControllerDelegate,

UIImagePickerControllerDelegate

De navigation controller delegate hebben we nodig tbv. het beheer van de camera en de foto bibliotheek en de image picker controller delegate om een foto te selecteren.

Creëer een button ‘*Pick image’* en een image view op het main storyboard. Creëer een action ‘*pickImageButtonPressed’* voor de button en een outlet ‘*pickedImage’* voor de image view.

Nu volgt een stuk code waarmee een image uit de foto bibliotheek geselecteerd kan worden:

var imagePicker = UIImagePickerController()

imagePicker.delegate = self

imagePicker.sourceType = UIImagePickerControllerSourceType.PhotoLibrary

// NB alternatief is camera

image.allowsEditing = false // Hebben we hier niet nodig

self.presentViewController(image, animated:true, completion: nil

// completion block ook niet nodig hier

Run de app. Wanneer op de button wordt gedrukt verschijnt een image picker waarmee een foto uit de foto bilbiotheek geselecteerd kan worden. Meer niet.

De geselecteerde foto moet nog getoond worden in de *image view*. Dit wordt in de les in een aantal stappen gerealiseerd. Ten eerste wordt de *imagePickerController* functie *didFinishPickingImage* geimplementeerd:

func imagePickerController(

picker: UIImagePickerController!,

didFinishPickingImage image:UIImage,

editingInfo: [NSObject : AnyObject]!) {

println(“Image selected”) // bij wijze van test

}

Wanneer een foto wordt geselecteerd verschijnt daarvan een melding in de console. Wat opvalt is dat er een leeg vlak op het scherm verschijn. Dit is een afdruk van de picker viewcontroller en wordt veroorzaakt doordat we zelf de methode *didFinishPickingImage* geïmplementeerd hebben. De picker viewcontroller moeten we daarom zelf sluiten. Voeg de volgende code toe aan *didFinishPickingImage*:

self.dismissViewControllerAnimated(true, completion: nil)

Dit haalt de view controller die ‘*modally’* (schermvullend) werd gepresenteerd door de ontvanger (= *self*) van het scherm’. In dit geval dus de image picker controller die modaal werd gepresenteerd door de ‘main’ view controller.

Het tonen van de foto in de *image view* volgt hier direct na:

pickedImage.image = image

Run de app, wanneer een foto wordt geselecteerd wordt deze in de image view op het scherm getoond.

Het maken van een foto met de camera – in plaats van een foto uit de foto bilbiotheek selecteren – gaat in principe eenvoudig door het *source type* van de *imagePicker* in te stellen op camera:

imagePicker.sourceType = UIImagePickerControllerSourceType.Camera

Op een device met een camera zal dit werken. Wordt de app in de simulator gedraaid dan crashed deze echter: er is geen camera. Om dit te voorkomen moet getest worden of de camera beschikbaar is. Dit volgt nog later.

De code is te vinden in 8.4.

**Les 106: Spinners and alerts**

*Spinners* en *alerts* zijn user interface elementen die in de nog te bouwen Instagram app gebruikt zullen worden. De spinner wordt getoond als data van Parse.com wordt gehaald. Deze les gebruikt de code van de voorgaande les.

Voeg een aantal buttons aan het storyboard toe:

* ‘*Pause app’* en action ‘*pauseApp’* voor het starten van een spinner en het disablen van de app, zodat er niet gerommeld kan worden in de app terwijl gewacht wordt,
* ‘*Restore app’* en action ‘*restoreApp’* voor het stoppen van een spinner en het weer enablen van de app,
* ‘*Create Alert’* en action ‘*createAlert’* voor het tonen van een alert melding.

Creëer binnen de view controller een globale variabele voor de activity indicator:

var activityIndicator: UIActivityIndicatorView = UIActivityIndicatorView()

Deze hebben we zowel in *pauseApp* als *restoreApp* nodig. Voeg voor het creëren en opstarten van de spinner de vogende code aan *pauseApp* toe:

activityIndicator = UIActivityIndicatorView(

frame: CGRectMake(0, 0, 50, 50))

activityIndicator.center = self.view.center

activityIndicator.hidesWhenStopped = true

activityIndicator.activityIndicatorViewStyle = UIActivityIndicatorViewStyle.Gray

view.addSubview(activityIndicator)

activityIndicator.startAnimating()

Dit creëert een activity indicator (spinner) ter grootte van 50x50 points. De positie is (0, 0) tov. het centrum van de view van de view controller, d.i. het scherm. Run de app en druk op de ‘*pause app’* knop en er verschijnt een draaiende spinner. De buttons zijn nog steeds enabled maar doen verder niets.

Het disablen van de user interface van de app gaat als volgt, voeg de volgende code aan *pauseApp* toe:

UIApplication.sharedApplication().beginIgnoringInteractionEvents()

Run de app en druk op de ‘*pause app’* button. De spinner verschijnt weer maar drukken op één van de buttons heeft geen effect: de app is disabled.

In *restoreApp* wordt de spinner vervolgens gestopt en de app weer enabled:

activityIndicator.stopAnimating()

UIApplication.sharedApplication().endIgnoringInteractionEvents()

Echter omdat de app disabled is als op de ‘*pause app’* button is gedrukt, werkt de ‘*restore app’* button niet meer. Zet de regel met *beginIgnoringInteractionEvents* daarom om in commentaar. Run de app en druk op de ‘*pause app’* en ‘*restore app’* buttons. De spinner verdwijnt nu van het scherm, vanwege de setting van het *hidesWhenStopped* attribuut.

Voeg voor het tonen van een alert de volgende code toe aan *createAlert*:

var alert = UIAlertController(

title: “Hey there!”,

message: “Are you sure?”,

preferredStyle: UIAlertControllerStyle.Alert)

self.presentViewController(alert, animated: true, completion: nil)

Merk op dat de alert in een aparte viewcontroller wordt getoond, net als de picker viewcontroller van de vorige les. Run de app en druk op de ‘create alert’ knop. De alert verschijnt, maar toont alleen de titel. Bovendien kan hij niet weg worden gehaald, dus in feite hangt de app nu. Een OK button is hard nodig.

Voeg daartoe de volgende code toe na de definitie van de *alert* variabele:

alert.addAction(UIAlertAction(title: “Ok”, style: .Default, handler: { action in

self.dismissViewControllerAnimated(true, completion: nil)

}))

Run de app: de alert laat de message nu ook zien en bevat een Ok button waarmee de alert weggeklikt kan worden.

De code staat in 8.5.

**Les 108: Login and signup**

De bouw van de Instagram kloon wordt opgedeeld in een aantal grote stukken, waarvan dit de eerste is. Code documentatie over hoe app gebruikers toegang krijgen tot hun informatie op Parse.com is te vinden in het hoofdstuk over *Users*. Dit betreft zaken als aanmelden, aanloggen, email verificatie, etc.

Voor de uitwerking wordt de code uit de vorige les gebruikt. Verwijder alle code in commentaar uit viewDidLoad en verplaats de call naar *setApplicationId* naar *didFinishLaunchingWithOptions* in *AppDelegate.swift*. De application id hoeft nl. slechts één keer te worden ingesteld, bij opstart is het beste moment daarvoor.

Haal alle code uit *ViewController.swift* weg en maak het storyboard leeg. Als eerste wordt het aanlog scherm gecreëerd. Voeg 2 labels toe:

* ‘Welcome to Instagram!” en daaronder
* “Use the form below to signup”

Hieronder twee text fields voor het invullen van usernaam en password en daaronder een Signup button. Gebruik Auto-layout met de Assistant Editor om het ook in landscape mode er goed uit te laten zien.

NB. Vink voor het password text field de optie ‘Secure Text Entry’ aan in de Attribute Inspector.

Creëer in ViewController.swift outlets ‘*usernameTextField’* en ‘*passwordTextField’* voor de text fields en een ‘*signUp’* action voor de button. Run de app om te kijken of alles er goed uit ziet.

Vervolgens wordt de logica mbt. het aanloggen toegevoegd. De ingevoerde username moet bestaan en het ingevoerde password moet daarbij horen. Er verschijnt een alert als dat niet het geval is. Om te beginnen checken we of zowel een username als password is ingevuld, maw. of niet één van beide leeg is. Voeg in functie *signUp* de volgende code toe:

var error = “”

if (username.text == ‘’ || password.text == “”) {

error = “Please enter a username and password”

}

if (error != “”) {

var alert = UIAlertController(

title: “Error in form”,

message: “error”,

preferredStyle: UIAlertControllerStyle.Alert)

alert.addAction(UIAlertAction(title: “Ok”, style: .Default,

handler: { action in

self.dismissViewControllerAnimated(true, completion: nil)

}))

self.presentViewController(alert, animated: true, completion: nil)

} else {

// Meld de gebruiker aan: volgt nog.

}

Run de app. Op zich werkt dit goed, echter staat de *auto prediction* nog aan, wat vervelend is bij het invoeren van usernaam en password. Dit halen we weg door in de *Attribute Inspector* voor beide textfields de *Correction* en *Checking* velden op *No* te zetten.

Voeg naar wens restricties toe op de ingevoerde usernaam, zoals een minimum lengte, het gebruik van specifieke karakters etc. In deze les wordt dat niet gedaan.

Vervolgens gaan we de gebruiker aanmelden op onze applicatie bij Parse.com. Gebruik hiervoor de code van de functie *myMethod* die op *Parse.com* beschikbaar is gesteld in de *Users* *– Signing Up* sectie. In deze les worden e-mail adres en phone niet gebruikt, haal die code er uit. Dit betekent jammer genoeg dat zaken als account activering via e-mail en password reset niet wordt geïmplementeerd.

De eerste opzet van de code voor het aanmelden (de *else*-tak) ziet er als volgt uit:

var user = PFUser()

user.username = username.text

user.password = password.text

user.signupInBackgroundWithBlock {

(succeeded: Bool!, signupError: NSError!) -> Void in

if (signupError == nil) {

// account kan toegevoegd worden, volgt later

} else {

println(signupError) // bij wijze van test

}

}

Ten behoeve van testen worden in de *Attributes Inspector* default waarden voor de *username* en *password* *textfields* ingevuld (in het *Text* veld).

Wanneer de app wordt gerund met deze defaultwaarden dan wordt in de eerste run een nieuwe user aangemaakt, zie daarvoor de applicatie data browser op Parse.com. In de tweede run volgt een foutmelding, omdat het account dan al bestaat. De foutmelding is echter veel te lang om in een alert box te laten zien. De al ingevulde code van Parse.com geeft echter een fout op userInfo[“error”], dat is omdat het niet zeker is dat er een element “error” bestaat in de User klasse. Vervang de *println(signupError)* door:

if (let errorString = signupError.userInfo?[“error”] as? NSString) {

// error element kan gebruikt worden

error = errorString

} else {

// probleem met error element: maak een generieke foutmelding

error = “Please try again later”

}

// toon een alert

Omdat het tonen van een alert al is geïmplementeerd voor het geval een lege username of password is ingesteld, willen we die code niet nogmaals herhalen. In plaats daarvan maken we een functie *displayAlert*:

func displayAlert(title: String, error: String) {

var alert = UIAlertController(

title: title,

message: error,

preferredStyle: UIAlertControllerStyle.Alert)

alert.addAction(UIAlertAction(title: “Ok”, style: .Default,

handler: { action in

self.dismissViewControllerAnimated(true, completion: nil)

}))

self.presentViewController(alert, animated: true, completion: nil)

}

De code na de test of error leeg is (if error != “”) verandert hierdoor:

if (error != “”) {

displayAlert(“Error in form”, error: error)

}

De code voor de alert wanneer het aanmelden mislukt:

self.displayAlert(“Could not signup”, error: error)

Run de app. Als de ingevoerde username bestaat dan volgt een alert met de foutmelding. Omdat het bij een trage internet verbinding even kan duren voordat de gebruiker is ingelogd of dat de alert verschijnt, zetten we de spinner even aan en disablen we de app.

Definieer de activity indicator globaal binnen de viewcontroller:

var activityIndicator: UIActivityIndicatorView = UIActivityIndicatorView()

Zet de code die de spinner activeert en de app disabled juist voordat de *signupInBackgroundWithBlock* wordt aangeroepen:

activityIndicator = UIActivityIndicatorView(

frame: CGRectMake(0, 0, 50, 50))

activityIndicator.center = self.view.center

activityIndicator.hidesWhenStopped = true

activityIndicator.activityIndicatorViewStyle = UIActivityIndicatorViewStyle.Gray

view.addSubview(activityIndicator)

activityIndicator.startAnimating()

UIApplication.sharedApplication().beginIgnoringInteractionEvents()

Wanneer de signup voorbij is, al dan niet met succes, moet de app weer enabled worden en de spinner gestopt. Zet de volgende code direct aan het begin van het completion block van *signupInBackgroundWithBlock*:

UIApplication.sharedApplication().endIgnoringInteractionEvents()

self.activityIndicator.stopAnimating()

De *self* prefix niet vergeten, is nodig omdat de code in een closure zit.

De signup code is hiermee kompleet, een nieuwe gebruiker kan zich aanmelden en is direct ingelogd. Nu moet er nog actie ondernomen worden voor bestaande gebruikers.

De normale manier om dit te doen is een apart inlogscherm te maken. Dit zou in dit zeer specifieke geval bijna identiek zijn aan het aanmeldscherm, ipv. ‘signup’ zou er ‘login’ in het label onder de welkomstboodschap en op de button moeten staan. Daarom wordt in de les dezelfde viewcontroller gebruikt en wordt de logica afhankelijk van wat er moet gebeuren enigszins aangepast/uitgebreid. Normaliter worden bij aanmelden meer gegevens gevraagd, zoals email adres en telefoonnummer en is het handiger aparte aanmeld- en inlogschermen te maken.

De oplossing die hier wordt gekozen is om juist onder de Signup button een label ‘Already registered ?’ te maken met een soort van ‘login/signup toggle’ button direct ernaast. Het idee hier achter is dat wanneer op deze toggle knop wordt gedrukt, het schem verandert van een login scherm naar een signup scherm en vice versa. Een aantal labels moet dan van een andere string voorzien worden om dit duidelijk te maken. Erg handig is dit niet, een apart registratie scherm is duidelijker.

Voor het uitlijnen van de ‘toggle’ button en het nieuwe label haalt Rob nog een trucje van stal. Hij creëert een nieuw dummy label dat centraal op het scherm wordt gezet. Het nieuwe label en de toggle button worden beide ten opzichte van dit label uitgelijnd, waarna het dummy label een lege string als tekst krijgt. Ik denk dat label en button ook prima zonder dit dummy label kunnen worden uitgelijnd.

Omdat het scherm voor 2 doeleinden wordt gebruikt moeten een aantal outlets/actions toegevoegd worden:

* een outlet voor het label onder de welkomstboodschap, noem het ‘*signUpLabel’*
* een outlet voor de sign up button: ‘*signUpButton’*
* een outlet voor de login button: ‘*signUpToggleButton’*
* een action voor de login button: ‘*signUpToggleButtonPressed’*
* een outlet voor het ‘already registered’ label: ‘*alreadyRegisteredLabel’*

Voor het wisselen tussen signup en login is een variabele nodig die dit bijhoudt. Definieer hiervoor de globale boolean variabele signupActive en zet deze initieel op true:

var signupActive = true

De code voor signUpToggleButtonPressed wordt nu als volgt:

if signupActive == true {

signupActive = false

signUpLabel.text = “Use the form below to log in”

signUpButton.setTitle(“Log In”, forState: UIControlState.Normal)

alreadyRegisteredLabel.text = “Not Registered?”

signUpToggleButton.setTitle(“Sign Up”, forState:UIControlState.Normal)

} else {

signupActive = true

signUpLabel.text = “Use the form below to sign up”

signUpButton.setTitle(“Sign Up”, forState: UIControlState.Normal)

alreadyRegisteredLabel.text = “Already Registered?”

signUpToggleButton.setTitle(“Log In”, forState:UIControlState.Normal)

}

Run de app en gebruik de toggle button om te checken dat het scherm wisselt van signup naar login scherm en vice versa.

De code om in te loggen moet nog geschreven worden. Op Parse.com is in de *Users sectie* te vinden dat hiervoor de functie *logInWithUsernameInBackground* gebruikt moet worden.

De functie signUp gebruiken we zowel om aan te melden als in te loggen. Het onderscheid wordt bepaald door de variabele signupActive. Creëer een if-statement en zet de bestaande signup code in de if-tak:

if signupActive == true {

// bestaande signup code

} else {

// login code

}

De login code is als volgt:

PFUser.logInWithUsernameInBackground(username.text, password.text) {

(user: PFUser!, signupError: NSError!) -> Void in

UIApplication.sharedApplication().endIgnoringInteractionEvents()

self.activityIndicator.stopAnimating()

if signupError == nil {

println(“Logged in”)

} else {

if let errorString = signupError.userInfo?[“error”] as? NSString {

error = errorString

} else {

error = “Login failed - please try again later”

}

}

De code is hiermee kompleet en te vinden in 8.6.

Uiteraard zijn de nodige aanpassingen nodig. Zo zou het bijv. handig zijn om uit te kunnen loggen, ontbreekt een pagina waarmee de accountgegevens kunnen worden bijgewerkt (e-mail, password, foto etc.),wordt bij aanmelden geen activatiemail gestuurd en kunnen op dit moment meerdere gebruikers ingelogd worden zonder dat de laatst ingelogde gebruiker wordt uitgelogd (of zorgt Parse daarvoor?).

**Les 110: Adding the user table (en 111 voor code files)**

Nu het aanmelden en inloggen werkt, wordt de Instagram app in deze les uitgebreid met de user table. Dit is een tabel die iedere ingelogde gebruiker te zien krijgt en waar in alle Instagram gebruikers zijn opgenomen. Wil je een gebruiker volgen dan tap je op zijn naam in de tabel waarna een vinkje achter zijn naam verschijnt.

Alvorens dit te bouwen nog aandacht voor een handige feature die Parse.com biedt. Parse slaat in de *Core Data* (dus lokaal) de ‘*current user’* op, d.i. de laatst ingelogde gebruiker. Wanneer de app wordt gestart is handig om automatische aangelogd te worden op basis van de opgeslagen ‘current user’. Deze wordt als volgt opgevraagd:

PFUser.currentUser() // NB println hiervan toevoegen aan viewDidLoad

Wanneer de app wordt opgestart, checken we in *viewDidLoad* of er een ‘*current user’* is, is dat het geval dan kan via een *segue* direct de *table view* met de gebruikers worden geladen. Zo niet, dan verschijnt het aanmeld-/loginscherm.

Voor nu starten we met het bouwen van het users scherm. Sleep een *navigation controller* naar het storyboard. Het mooie hiervan is dat daar een *table view controller* aan hangt, precies wat nodig is. Zet in de *Atributes Inspector* in *Simulated Metrics* de afmeting van de navigation controller op *iPhone 4 inch* en de oriëntatie op *Portrait*. Om de table viewcontroller vanuit de code aan te kunnen sturen, moet een class toegevoegd worden. Doe dit dmv. *File – New – File…* en selecteer *Cocoa Class*, noem de class *UserTableViewController* en vul *UITableViewController* in als subclass. De aangemaakte file *UserTableViewController.swift* is vrijwel leeg, zorg dat “*import UIKit”*  er alvast in staat.

Voordat we de *UserTableViewController* class gaan uitwerken, zetten we in het storyboard eerst de *Custom Class* van de table viewcontroller op de zojuist gecreëerde *UserTableViewController*. Selecteer daartoe de table viewcontroller in het storyboard (gele ikoon bovenaan) en vul dit in in de *Identity Inspector*.

In UserTableViewController.swift, kopieer een lege *viewDidLoad* en *didReceiveMemoryWarning* daarheen. Stel in de *Attributres Inspector* de table viewcontroller in als *initial viewcontroller* om te checken of deze goed werkt. Run de app, de table viewcontroller verschijnt op het scherm (zonder titel!). Merk op dat de println van de currentUser nog steeds wordt uitgevoerd.

Geef de navigation controller een naam, selecteer deze in het storyboard en vul hiertoe het Title veld in de Attributes Inspector in, noem het bijv. *UserTableNavController*. De naam hebben we nodig om ernaar te verwijzen in de code. Stel nu de navigation controller in als initial viewcontroller en run de app om te checken dat er nu een title bar boven de table viewcontroller staat.

Vervolgens creëren we nu de *segue* van de initial viewcontroller (met het inlogscherm) naar de navigation controller. Rechtsklik in het storyboard de navigation controller en sleep naar de navigation controller, selecteer ‘*show’* uit het menu dat verschijnt. Klik op de segue en vul ‘*jumpToUserTable’* in als identifier in de Attributes Inspector.

De *segue* moet uitgevoerd worden als de ‘*current user’* bestaat bij het opstarten. Dit kan echter niet in de *viewDidLoad* omdat de segue pas kan worden uitgevoerd als de view is geladen (maar nog niet op het scherm staat). Daarom moet we dit onderbrengen in *viewDidAppear*, die wordt aangeroepen na *viewDidLoad*. Vul de volgende code in in *viewDidAppear*:

If PFUser.currentUser() != nil {

self.performSegueWithIdentifier(“jumpToUserTable”, sender: self)

}

Zet de initial viewcontroller weer terug op de ‘inlog’ viewcontroller en run de app. Omdat er al een gebruiker ‘asdf’ was ingelogd, verschijnt in een glimp het inlogscherm waarna direct de table view verschijnt.

*NB Is het niet beter om de segue dan vanuit viewWillAppear te doen of werkt dat niet?*

De aansturing van de table viewcontroller wordt nu ingericht. Voeg in *UserTableViewController.swift* de volgende methoden toe:

* *numberOfSectionsInTableView* 🡪 return 1 (1 sectie)
* *numberofRowsInSection* 🡪 return 3 (bij wijze van test)
* *cellForRowAtIndexPath*:

var cell: UITableviewCell =

self.tableView.dequeueReusableCellWithIdentifier(“cell”)

as UITableViewCell

cell.textLabel?.text = “Test”

return cell

Geef het cel prototype de identifier “*cell*” in de *Attribute Inspector*. Run de app ter controle.

Voor de usernames van de app gebruikers definiëren we in UserTableViewController.swift een class-globale array ‘*users’* en initialiseren deze met een aantal testwaarden:

var users = [“Frank”, “Anita”, “Sophia”]

Hierdoor verandert *numberofRowsInSection*:

return user.count

en evenzo *cellForRowAtIndexPath*:

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

Run de app ter controle.

Nu kunnen alle app users van Parse.com gehaald worden en in de tabel worden getoond. Start door ‘users’ een ‘lege’ array van Strings te maken:

var users = [“”]

NB Deze vorm wordt gekozen om de users.count niet te laten mislukken.

In viewDidLoad (in UserTableController.swift) worden de app gebruikers binnen gehaald (zie ook documentatie op Parse.com):

var query = PFUser.query() // vraag app users op

query.findObjectsInBackgroundWithBlock({

(objects: [anyObject!], error: NSError!) -> Void in

// Verwijder alle bestaande users, in dit geval de ‘lege string’ user

self.users.removeAll(keepCapacity: true)

for object in objects {

var user:PFUser = object as PFUser

self.users.append(user.username)

}

// Laat de usernames zien in de table view

self.tableView.reloadData()

})

Run de app ter controle. Alle reeds ingevoerde users staan in de tabel.

De volgende stap is dat een gebruiker in de tabel kan worden getapt waarna een vinkje verschijnt achter zijn naam. Stond het vinkje er al, dan verdwijnt deze. Een aangevinkte gebruiker wordt toegevoegd aan de lijst met gevolgde personen. Hiervoor implementeren we de functie *didSelectRowAtIndexpath*:

var cell:UITableViewCell = tableView.cellForRowAtIndexPath[indexPath]!

cell.accessoryType = UITableViewCellAccessoryType.Checkmark

Eerst wordt een *cell* variabele gecreëerd die verwijst naar de geselecteerde cel. Het *uitroepteken* geeft aan dat de cell ge-unwrapped wordt, dat geeft geen probleem omdat zeker is dat de cell bestaat, anders zou hij niet geselecteerd kunnen worden. Het *vinkje* is één van de beschikbare *accessory types*, hoe simpel kan het soms zijn.

De code voor het de-selecteren van een cel is nodig om een vinkje weer weg te halen. Daarvoor moet getest worden of de cel een vinkje bevat:

if cell.accessoryType == UITableViewCellAccessoryType.Checkmark {

cell.accessoryType = UITableViewCellAccessoryType.None

} else {

cell.accessoryType = UITableViewCellAccessoryType.Checkmark

}

De volgende stap is om dit met Parse.com te synchroniseren. Hiervoor wordt een nieuwe class ‘*Followers’* op Parse.com aangemaakt. Deze klasse bevat twee *String* elementen: ‘*follower’* en ‘*following’*. Dit zijn de usernames van resp. degene die volgt en degene die gevolgd wordt. In de les wordt deze class op de Parse.com site aangemaakt, dit is tamelijk rechttoe rechtaan. Nu deze class beschikbaar is, kan de code in *didSelectRowAtIndexPath* aangepast worden. Voeg daartoe in de else-tak het volgende toe:

var followers = PFObject(className: “Followers”)

followers[“follower”] = PFUser.currentUser().username

followers[“following”] = cell.textLabel?.text

followers.saveInBackground()

Run de code en vink de bovenste twee tabel cellen aan. Kijk dan op Parse.com, ververs de data en zie dat er twee records verschijnen.

NB Het lijkt alsof Parse.com automatisch een nieuw record toevoegt wanneer bovenstaande code wordt uitgevoerd. Er is geen append opdracht oid. nodig.

Het blijkt nu dat de usernaam van de gebruiker zelf ook in de tabel wordt weergegeven. Dit is ongewenst. Er moet dus een check toegevoegd worden waarmee dat voorkomen wordt. Voeg deze toe in de *viewDidLoad* functie, in de *for-loop ‘for object in objects’*:

var user: PFUser = object as PFUser

if user.username != PFUser.currentUser().username {

self.users.append(user.username)

}

Run de code bij wijze van check. De naam van de ingelogde gebruiker (‘asdf’ in de les) komt niet meer voor in de tabel.

Nu volgt de code voor het de-selecteren van een gebruiker. Deze moet gevonden worden op *Parse.com* en uit de *Followers* class verwijderd worden. Hiervoor moet een query uitgevoerd worden, voorbeeld code daarvan is te vinden op Parse.com. Voeg in de if-tak in *viewDidLoad* de volgende code toe:

var query = PFQuery(className: “Followers”)

query.whereKey(“follower”, equalTo:PFUser.currentUser().username)

query.whereKey(“following”, equalTo:cell.textLabel?.text)

query.findObjectsInBackgroundWithBlock {

(objects:[anyObject]!, error: NSError!) -> Void in

if error == nil {

for object in objects {

object.deleteInBackground()

}

} else {

// log de foutmelding

println(“Fout bij zoeken object: \(error)”)

}

}

*Let op*: geen haakjes bij *findObjectsInBackgroundWithBlock*, is dit normaal?

*Let op 2*: de call geeft een array van objecten terug, vandaar *[anyObject]!* in rechte haken.

*Let op 3*: er kunnen klaarblijkelijk meerdere records met dezelfde volger en gevolgde vorkomen. In principe zou dit niet mogelijk moeten zijn: wanneer user A user B volgt, dan hoeft dat maar één keer in de tabel te staan!

Run de code ter controle. Wat nog moet gebeuren is dat bij opstarten van de app de gevolgde personen in de tabel aangevinkt zijn. Daarvoor creëert Rob Percival een ‘following’ array van booleans, de index is het volgnummer van de cel in de table view. Hij doet dit op een nogal omslachtige en inefficiënte manier (dat zegt hij zelf ook). Zo wordt de reloadData() voor elke user uitgevoerd, wat in feite maar één keer nodig is.

Daarom geen code daarvan in dit document.

De code is te vinden in 8.7.

**Les 112: Pull to refresh (en 113 voor code files)**

Deze les behandelt de ‘*pull to refresh’* mogelijkheid, door het window een stukje naar beneden te slepen en los te laten. Dit is tamelijk eenvoudig te implementeren. We doen dit met de UserTableViewController, om de tabel met gebruikers te verversen.

Definieer een variabele ‘*refresher’* globaal in de *UserTableViewController* class:

var refresher: UIRefreshControl!

Configureer de refresher aan het einde van de *viewDidLoad* methode:

refresher = UIRefreshControl()

refresher.attributedTitle = NSAttributedString(string: Pull to refresh user list”)

refresher.addTarget(self, action: ”refreshUserList”,

forControlEvents: UIControlEvents.ValueChanged)

self.tableView.addSubview(refresher)

De ‘*attributed string’* verschijnt bij het naar beneden slepen van het window. De functie *refreshUserList* wordt uitgevoerd wanneer de vinger van het scherm wordt gehaald. Deze heeft geen parameters, daarom is geen *dubbele punt* nodig in de naam.

De refreshUserList functie update de user list. De code daarvoor is in feite al geschreven. Deze zetten we in een aparte functie *updateUsers:*

func updateUsers {

var query = PFUser.query()

query.findObjectsInBackgroundWithBlock({

(objects: [AnyObject]!, error: NSError!) -> Void in

self.users.removeAll(keepCapacity: true)

for object in objects {

var user: PFUSer = object as PFUser

var isFollowing: Bool

if user.username != PFUser.currentUser().username {

self.users.append(user.username)

isFollowing = false

var query = PFQuery(className: “Followers”)

query.whereKey(“follower”,

equalTo:PFUser.currentUser().username)

query.whereKey(“following”,

equalTo:user.username)

query.findObjectsInBackgroundWithBlock({

(objects: [AnyObject]!, error: NSError!) -> Void in

if error == nil {

for object in objects {

isFollowing = true

}

self.following.append(isFollowing)

self.tableView.reloadData()

} else {

// log de fout

println(eror)

}

})

}

}

})

}

Vergeet deze functie niet aan te roepen in *viewDidLoad*, de code is immers naar deze functie verhuisd.

Functie *refreshUserList* ziet er als volgt uit:

func refreshUserList {

updateUsers()

refresher.endRefreshing() // anders blijft de spinner zichtbaar

}

De code is te vinden in 8.8.

**Les 115: Posting images (en 116 voor code files)**

In deze les wordt de basis functionaliteit van *Instagram* gebouwd: het toevoegen van foto’s. Ter wille van de eenvoud kunnen alleen opgeslagen foto’s toegevoegd worden, het maken van een foto met de camera is een add-on.

Hiervoor hebben we een nieuw scherm nodig. Voeg in het storyboard een standaard viewcontroller toe, zet de afmeting op 4 inch en stel in op portrait mode. Deze nieuwe viewcontroller willen we bereiken vanaf de *user table viewcontroller*. Daar zetten we een *bar button* op, noem deze ‘*Post’* (in *Attributes Inspector*) en voeg een *segue* toe vanaf deze bar button naar de nieuwe viewcontroller (kies *Show* uit het segue context menu). Het mooie hiervan is dat de nieuwe viewcontroller ook een navigatie menu bovenin heeft gekregen, de al aanwezige navigation controller is ahw. geïntegreerd in de nieuwe viewcontroller. Dit betekent dat we ‘gratis’ een ‘back’ button erbij krijgen, om terug te keren naar de user table viewcontroller.

Nu wordt de nieuwe viewcontroller geconfigureerd. Zet een *image view* er op, deze zal worden gebruikt voor de foto die zal worden ge-upload. Laat de image view een beperkt deel van het scherm in beslag nemen en maak hem vierkant (172x172 points in de les).

Plaats onder de image view een *button* ‘*Choose Image’* en een *text field*, waar een korte omschrijving van de foto in moet worden gevoerd (deze mag straks niet leeg zijn). Vul hiervan eventueel de placeholder tekst in, in de *Attributes Inspector*, bijv. ‘Describe your photo’.

Zet onder het text field een *button* ‘*Post Image’*. Wordt hier op gedrukt dan wordt de afgebeelde foto gepost. Omdat deze button een essentiële functie uitvoert, is het handig hem er onderscheidend uit te laten zien, bijv. witte letters op een blauwe achtergrond.

Alle controle elementen staan nu op het scherm. Zorg met *Auto Layout* dat het scherm in landscape mode er ook netjes uitziet. In de les lukt dat niet: het text field en de ‘post image’ button zijn niet zichtbaar. Rob geeft aan dat dit mbv. een *scrollview* opgelost kan worden, zodat in landscape mode het onderste deel van het scherm wordt bereikt door naar beneden te scrollen. In deze les laat hij het erbij zitten zodat dit scherm alleen in portrait mode goed werkt.

Om het nieuwe scherm aan te sturen moet een bijbehorende klasse worden gecreëerd. Creëer een nieuwe File, selecteer Cocoa class en noem de class ‘*postViewController’*. Selecteer *UIViewController* als *parent* *class*. Selecteer in het storyboard de nieuwe viewcontroller en vul in de *Attributes Inspector* bij Custom Class ‘postViewController’ als class in.

Zorg dat in *postViewController.swift* ‘*import UIKit’* bovenin is opgenomen (en niet ‘import Cocoa’). Kopieer een lege *viewDidLoad* en didReceiveMemoryWarning naar *postViewController.swift*. Creëer een outlets en actions:

* een *outlet* ‘*imageToPost’* voor de image view
* een *action* ‘*chooseImageButtonPressed’* voor de betreffende button
* een *outlet* ‘*imageDescriptionTextField’* voor het text field
* een *action* ‘*postImageButtonPressed’* voor de betreffende button

Voor de image view is het handig een placeholder te gebruiken, zodat er al iets op het scherm staat voordat een foto is geselecteerd. Kopieer hiertoe een neutrale .png file naar de project directory (sleep vanuit Finder, vink ‘*Copy items if needed’* aan). Selecteer de image view in het storyboard en vul de naam van deze .png file in bij *Image* in de *Attribute Inspector*. Dit is bij deze de default foto die in de image view wordt getoond.

Vul in de ‘*chooseImageButtonPressed’* functie de ‘*pick im*age’ code uit de betreffende voorgaande les:

var image = UIImagePickerController()

image.delegate = self

image.sourceType = UIImagePickerControllerSourceType.PhotoLibrary

image.allowsEditing = false

self.presentViewController(image, animated: true, completion: nil)

Merk op dat we voor deze les alleen gebruik maken van de photo library en editing niet toestaan. Pas dit naar wens aan. De call naar *presentViewController* opent de photo library waar een foto uit geselecteerd kan worden.

Voeg de protocollen *UINavigationControllerDelegate* en *UIImagePickerControllerDelegate* toe aan de *postViewController* class definitie.

Kopieer ook de didFinishPickingImage code naar postViewController.swift:

func imagePickerController(

picker: UIImagePickerController!,

didFinishPickingImage image:UIImage,

editingInfo: [NSObject : AnyObject]!) {

println(“Image selected”) // bij wijze van test

self.dismissViewControllerAnimated(true, completion: nil)

imageToPost.image = image

}

Deze functie wordt aan geroepen zodra een foto uit de photo library is geselecteerd. Run de app ter controle.

NB In de les is de *postViewController* tijdelijk als initial viewcontroller ingesteld zodat het testen sneller verloopt.

Vervolgens wordt de ‘*post image’* button functionaliteit geïmplementeerd. Voordat een foto gepost wordt moet gecontroleerd worden of er een foto geselecteerd is en of een omschrijving van de foto is ingevuld (hoewel dit niet persé verplicht hoeft te worden). Creëer voor de foto selectie een globale klasse variabele:

var photoSelected:Bool = false

Als een foto geselecteerd is kan deze op true gezet worden. Voeg dit statement toe aan de *didFinishPickingImage* code:

photoSelected = true

Functie *postImageButtonPressed* wordt dan als volgt:

var errorStr = “”

if photoSelected == false {

errorStr = “Please select an image”

} else if (imageDescriptionTextField.text == “”) {

errorStr = “Please add a description for the selected photo”

}

if error == “” {

// post de foto naar Parse.com, code volgt later

} else {

displayAlert(“Cannot post image”, errorStr)

}

Merk op dat functie *displayAlert* al eerder is gedefinieerd in *viewController.swift*. Kopieer deze functie naar *postViewController.swift*, binnen de klasse definitie. Run de app ter controle.

Nu kan de post functionaliteit toegevoegd worden. Hiervoor definiëren we een nieuwe klasse “*PostedImages*” op Parse.com. Dit doen we deze keer vanuit de code in imageButtonPressed:

var postedImage = PFObject(className: “PostedImages”)

postedImage[“title”] = imageDescriptionTextField.text

postedImage[“user”] = PFUser.currentUser().username

post.saveInBackgroundWithBlock{(success: Bool!, error: NSError!) -> Void in

if success == false {

self.displayAlert(“Could not post image”, error: “Please try again”)

} else {

let imageData =

UIImagePNGRepresentation(self.imageToPost.image)

let imageFile = PFFile(name: “image.png”, data: imageData)

post[“imageFile”] = imageFile

post.saveInBackgroundWithBlock{

(success: Bool!, error: NSError!) -> Void in

if success == false {

self.displayAlert(“Could not post image data”,

error: “Please try again”)

} else {

println(“Image posted succesfully”)

self.displayAlert(“Image posted”,

error: “successfully”)

self.photoSelected = false

self.imageToPost =

UIImage(named: “<filenaam van foto zonder

extensie>”)

self.imageDescriptionTextField.text = “”

}

}

}

}

Merk op dat de foto omschrijving en username als eerste worden gepost. Gaat dit goed dan wordt de foto zelf gepost. De reden zal waarschijnlijk zijn dat het posten van de foto redelijk wat werk is: eerst moet de PNG representatie van de foto in een variabele gevat worden. Dan moet een imagefile gedefinieerd worden en pas daarna kan de foto gepost worden. Nadat de foto gepost is moeten een aantal variabelen gereset worden voor als de gebruiker nog een foto wil posten.

Run de app ter controle. Het opslaan van de foto duurt even maar lukt wel. Check op *Parse.com* dat de foto in de nieuwe klasse is opgeslagen. De foto wordt als .png file opgeslagen en kan worden bekeken.

Een aantal variabelen moet nog geïnitialiseerd worden wanneer een nieuwe foto wordt geselecteerd. Rob doet dit in *viewDidLoad*:

photoSelected = false

imageToPost = UIImage(named: “<filenaam van foto zonder extensie>”)

imageDescriptionTextField.text = “”

Het opslaan van de foto op Parse.com duurt wel even. Bovenaan het scherm verschijnt een mini spinner ten teken dat er iets aan de gang is. Voor het mooie voegen we nog een eigen *activity indicator* toe en disablen we de applicatie tijdelijk. De code daarvoor is al eerder geschreven in *viewController.swift*. Voeg deze code in *imageButtonPressed* toe aan de *else-tak* van de controle ‘*if error !=* ‘. De variabele *activityIndicator* moet binnen de *postViewController* klasse gedefinieerd worden. Tevens moet de spinner gestopt worden en de app interactie weer toegestaan worden (1) indien de eerste *saveInBackgroundWithBlock* mislukt en (2) na de tweede *saveInBackgroundWithBlock*.

De code is te vinden in 8.9.

NB. De initial viewcontroller moet weer teruggezet worden op de ‘login’ viewcontroller. Na het succesvol posten van een foto lijkt het alsof de app het loginscherm doorloopt en daarna de user table laat zien, waarop een andere user dan voorheen aangevinkt is. Dit gebeurt ook wanneer geen foto geselecteerd wordt en op de Post knop wordt gedrukt. Het blijkt dat de *dismissViewControllerAnimated* dit veroorzaakt in de nieuwe XCode. Die moet dus weg.

**Les 116: Viewing user’s feeds (en 117 voor code files)**

In deze les wordt de Instagram app uitgebreid met de mogelijkheid om foto’s van andere gebruikers in te zien, waarmee de basis Instragram functionaliteit geïmplementeerd is.

Voordat dat gebeurt wordt eerst logout functionaliteit aan de app toegevoegd. Deze is essentieel om de app met meerdere gebruikers te kunnen testen. Hiervoor wordt onderaan op het ‘*post image’* scherm een toolbar met een *logout button* toegevoegd. Creëer in de *postViewController* een action ‘*logoutButtonPressed’*. Wanneer op de logout button wordt gedrukt moet dit richting *Parse.com* worden gemeld en vervolgens moet worden teruggesprongen naar het inlogscherm. Creëer daarvoor een *segue* van de logout button naar de login viewcontroller (selecteer ‘show’ uit het popup menu). Geef de segue in de *Attributes Inspector* de naam ‘*logoutSegue’*.

De code van functie logoutButtonPressed is dan als volgt:

PFUser.logout()

self.performSegueWithIdentifer(“logoutSegue”, sender: self)

Wat ook nog ontbreekt is dat de gebruiker na succesvol aanmelden (als nieuwe gebruiker) of inloggen (als bestaande gebruiker) het scherm met de user table te zien krijgt. Voeg in *ViewController.swift* in de *signUp* functie na beide ‘*if signupError == nil’* de volgende code toe:

self.performSegueWithIdentifer(“jumpToUserTable”, sender: self)

Let op dat dit dus twee keer moet gebeuren.

Run de code. Op zich werkt het goed, echter bevat het inlogscherm een *navigation bar*. Deze moet verwijderd worden voordat het inlogscherm op het scherm staat maar moet weer zichtbaar worden wanneer de overgang naar het ‘user table’ scherm plaatsvindt. Het weghalen doen we in *viewWillAppear* (voeg deze functie toe):

self.navigationController?.navigationBarHidden = true

Het weer terugzetten van de navigation bar doen we in *viewWillDisappear*:

self.navigationController?.navigationBarHidden = false

Nu dit in orde is wordt een nieuwe viewcontroller tbv. het bekijken van andermans foto’s op het storyboard gecreëerd. We creëren hiervoor een *table viewcontroller*. Deze zal worden bereikt vanaf de *UserTableViewController*. Creëer daarvoor een bar button ‘*view feed’* in de navigation bar van de *UserTableViewController* en een *segue* vanaf deze button naar de nieuwe table viewcontroller.

De nieuwe table viewcontroller laat de foto’s zien, één foto per cel. De cellen worden elk ca. 1/3 scherm groot. Selecteer hiertoe de ‘*prototype cell’* en pas de ‘*row height’* aan in de *Size Inspector* (Rob zet hem op 200).

Creëer nu een *image view* in de ‘*prototype cell’*. Maak het wat kleiner dan de afmeting van de cel zelf, zorg dat er ruimte in de cel overblijft voor 2 labels onder de foto. Omdat vooraf niet bekend is wat de oriëntatie is van de foto die wordt getoond, zetten we de *Mode* van de image view op *Aspect Fit* in de Attributes Inspector. Elke foto wordt dan in de juiste proporties afgebeeld. Zet bij wijze van test de default foto op de eerder geladen default foto voor een te posten foto. Voeg 2 labels toe onder de foto: één voor de omschrijving van de foto en één voor de username. Maak de tekst voor de username kleiner dan die van de omschrijving. Zet met *Auto Layout* de afmetingen en posities voor de image view en labels vast.

Het uiterlijk van de view feed viewcontroller ligt hiermee vast. Nu moet het beheer ervan nog geïmplementeerd worden. Creëer een nieuwe Cocoa class ‘*FeedViewController’* (File – New –File etc.) en maak deze een subclass van *UITableViewController*. Zorg dat ‘*import UIKit’* bovenin de file komt te staan. Zet in het storyboard de *Class* van de nieuwe ‘feed viewcontroller’ op *FeedViewController*. Geef de view controller een naam door in de Attribute Inspector het *Title* veld in te vullen (bijv. ‘Your Feed’).

De volgende stap is het creëren van outlets voor de image view en labels in de ‘*prototype cell’*. Echter heeft dit geen zin omdat de tabel uit een enorme hoeveelheid cellen kan bestaan en het daarmee onmogelijk is voor elke cel de outlets te definiëren. Daarom moet een nieuwe klasse gecreëerd worden die onze versie van de cel representeert. De image view en beide labels brengen we er in onder. Deze nieuwe klasse noemen we ‘*FeedCell’* en is een subclass van *UITableViewCell*. Creëer een nieuwe Cocoa Class ‘*FeedViewCell’* in XCode en maak hem een subclass van *UITableViewCell*. Zorg dat ‘*import UIKit*’ bovenaan de nieuwe file *FeedViewCell.swift* staat. Voeg outlets toe aan de *FeedViewCell* klasse voor de image view en beide labels. De klassedefinitie ziet er als volgt uit:

class FeedViewCell: UITableViewCell {

@IBOutlet var postedImage: UIImageView!

@IBOutlet var username: UILabel!

@IBOutlet var title: UILabel!

}

Stel in het storyboard de class van het *cell prototype* van de user table in op deze ‘*FeedViewCell’* klasse. Hiermee controleert de FeedViewCell klasse de cellen van de user table.

We zijn nu zover dat de code van de nieuwe *FeedViewController* geschreven kan worden. Kopieer deze over vanuit de PostViewController en pas ze aan:

* *numberOfSectionsInTableView* 🡪 return 1
* *numberOfRowsInSection* -> return 3 // bij wijze van test
* *didSelectRowAtIndexPath* hebben we niet nodig omdat we niets met de cel selectie doen
* *cellForRowAtIndexPath*, hier gebruiken we de nieuwe FeedViewCell klasse voor de cell variabele:

var myCell: FeedViewCell =

self.tableView.dequeueReusableCellWithIdentifer(“myCell”)

as FeedViewCell

myCell.title.text = “test” // bij wijze van test

myCell.username.text = “test username” // bij wijze van test

Vergeet niet de *reusable identifier* van de prototype cell in de Attribute Inspector in te stellen op “*myCell*”.

In de versie van XCode die Rob gebruikt moet de hoogte van de cell vanuit de code ingesteld worden, de setting vanuit de Attribute Inspector wordt bij hem niet overgenomen. Hiervoor is de functie *heightForRowAtIndexPath*, vul daar het volgende in:

return 227 // of een andere waarde naar keuze

Run de code bij wijze van test. De default foto verschijnt op de feed view, deze was nl. in de image view ingesteld als default.

Nu kan de content van *Parse.com* gehaald worden en gepresenteerd worden. Rob download hiervoor in eerste instantie alle beschikbare content van Parse.com wat niet echt efficiënt is. Later worden alleen de foto’s van de gevolgde personen gedownload. De data wordt opgeslagen in een aantal arrays die in *FeedViewController.swift* worden gedefinieerd:

var titles = [String]()

var userNames = [String]()

var images = [UIImage]()

var imageFiles = [PFFile]()

De *imageFiles* variabele hebben we straks nodig omdat de foto’s op Parse.com als files worden opgeslagen en dus eerst uitgepakt moeten worden.

De *numberOfRowsInSection* methode kan nu aangepast worden:

return titles.count

De data wordt in *viewDidLoad* ingeladen. Kopieer hiertoe de query code van Parse.com en pas als volgt aan:

var query = PFQuery(className: “Post”) // check of className correct is

// query.whereKey()… // nog niet nodig, komt straks

query.findObjectsInBackgroundWithBlock{

(objects: [anyObject]!, error: NSError!) -> Void in

if error == nil {

println(“Successfully retrieved (\objects.count) images”)

for object in objects {

self.titles.append(object[“title”]) as String

self.usernames.append(object[“username”]) as String

self.imageFiles.append(object[“imageFile”]) as PFFile

self.tableView.reloadData() // NIET BUITEN DE LOOP?

}

} else {

println(error)

}

}

Pas *cellForRowAtIndexPath* aan:

myCell.title.text = titles[indexPath.row]

myCell.username.text = usernames[indexPath.row]

imageFiles[indexPath.row].getDataInBackgroundWithBlock{

(imageData: NSData!, error: NSError!) -> void in

if error == nil {

myCell.postedImage.image = UIImage(data: imageData)

} else {

println(error)

}

}

De files die van Parse.com binnenkomen bevatten image data, dat met de Parse.com api methode *getDataInBackgroundWithBlock* eruit gehaald wordt. Run de app ter controle, alle plaatjes verschijnen als de data op Parse.com consistent is. In mijn geval bleken er foto’s te ontbreken bij een aantal postings waardoor een runtime error ontstond.

Nu kunnen we het aantal foto’s dat wordt gedownload beperken door alleen de foto’s van gevolgde gebruikers te tonen. Dit doen we door alle gebruikers die worden gevolgd te downloaden en per gebruiker alle foto’s vervolgens op te halen:

var getFollowedUsersQuery = PFQuery(className: “Followers”)

getFollowedUsersQuery =

whereKey(“follower”, equalTo: PFUser.currentUser().username)

getFollowedUsersQuery.findObjectsInBackgroundWithBlock{

(objects: [anyObject]!, error: NSError) -> Void in

if error == nil {

var followedUser = “”

for object in objects {

followedUser = object[“following”] as String

// Voeg hier de bestaande code in om foto’s op te halen

// Voeg nog wel een whereKey toe:

query.whereKey(“username”, equalTo: followedUser)

}

} else {

println(error)

}

}

Hiermee is de basis Instagram app klaar. Diverse verbeteringen zijn mogelijk, zoals activatie via e-mail, password recovery, het downloaden van alleen recente foto’s ipv. allemaal etc etc.

De code is te vinden in 8.10.

**Section 9 - Tinder Clone**

**Les 118: Introduction to tinder**

In deze secie wordt de Tinder app in basis nagebouwd. Hiermee kun je foto’s van anderen bekijken en je interesse al dan niet kenbaar maken. Met een swipe-right toon je interesse, een swipe-left niet. Doet de tegenpartij ook een swipe-right met jouw foto dan kun je vervolgens met elkaar afspreken. Voor de app zal mbv. de Facebook login ingelogd worden, je Facebook profielfoto verschijnt dan ook direct in Tinder.

**Les 119: Facebook login (en 120 voor code files)**

Deze les behandelt de login op Facebook vanuit de app. Hier moet het nodige voor geregeld worden, á la Parse.com integratie in de app. Voor de te bouwen app zijn beide nodig.

Op *Parse.com* is in de *Docs* sectie een stappenplan te vinden voor de integratie van Facebook en Parse.

Op de Facebook site, in het Developers gedeelte, staan de stappen vermeld om Facebook in een app te gebruiken.

Allereerst is een *Facebook account* nodig. Je kunt een apart account hiervoor aanmaken of je bestaande account gebruiken (wat is het verschil?). Zorg dat je als *Developer* bij Facebook geregistreerd bent.

Download en installeer dan de *Facebook SDK for iOS*.

Creëer nu een nieuwe *Facebook app* (‘Tinder’) op de Facebook site. Omdat we de app niet in productie zullen nemen hoeft de *namespace* niet te worden ingevuld, het mag wel en dan moet een unieke identifier worden ingevuld, bijv. “*tinderdemo-famasoft*”.

Na enige tijd verschijnt de *Dashboard pagina* met de *App Id* en de *App Secret* codes. Deze zijn beide nodig om de app met Facebook te laten werken. In mijn geval zijn de codes voor Tinder 776330049117683 en 9303ad059766e10e3815e656a6c15430.

Rob slaat in stap 3 het instellen van de *Bundle Identifier* over. Is kennelijk niet nodig?

*NB Dit is wel nodig! De app moet als ‘Native iOS’ app aangemerkt worden in Facebook en de bundle identifier die in XCode voor de app is ingevuld in info.plist moet hier overgenomen worden. Laat je dit achterwege dan mislukt het inloggen altijd (user == nil).*

Vervolgens moet het Facebook framework in de app worden geïntegreerd. Deze is standaard in de *Documents directory* te vinden. Sleep de *FacebookSDK.framework* file vanuit de Finder naar het project in XCode. Vink ‘*Copy items if needed’* aan.

De volgende stap (zie weer op de Facebook pagina) is het configureren van de ‘*info.plist’* voor Facebook. Creëer (= Add Row) in XCode in de info.plist van ons project een key genaamd ‘*FacebookAppId’*, type String en vul de ontvangen app id daar bij in.

Doe hetzelfde voor de *FacebookDisplayName* (= ‘Tinder’).

Idem voor de ‘*URL types’*, dit is echter een array. Voeg onder item 0 van ‘URL types’ een nieuwe array ‘*URL schemes’* toe. Vul hier bij item 0 de ontvangen *App Id* in voorafgegaan door ‘*fb’*.

Creëer in XCode in het project een *Bridging Header*, zoals eerder voor Parse.com is gedaan. Creëer een nieuwe Objective-C file, noem deze bijv. Facebook-Bridge. XCode herkent dit en vraagt of een *Bridging Header* moet worden geïnstalleerd. Antwoord hier bevestigend op. Voeg de regel

#import <FacebookSDK/FacebookSDK.h>

toe aan de bridging header. Gooi de gecreëerde .m file weg.

Er is geen manier om te testen of de Facebook integratie gelukt is. Dit gaan we mbv. Parse.com controleren. Integreer Parse.com in de Tinder app, zoals dat eerder in de Instagram app is gedaan, zie les 100 en de bijbehorende video. Als extra moet ook de *ParseFacebookUtils.framework* file in het project geïntegreerd worden.

Creëer een nieuwe app ‘*Tinder’* op Parse.com vanuit het dashboard. Er verschijnt een lijst met keys waarvan we de *Application ID* en *Client Key* nodig hebben. Ga naar *AppDelegate.swift* en voeg de volgende call toe aan *didFinishLaunchingWithOptions*:

Parse.setApplicationId(“<your Application ID>”, clientKey:”<your Client Key>”)

De app is nu verbonden met Parse.com en Facebook. Ga weer naar de Docs sectie op Parse.com, naar de Facebook Users hoofdstuk en volg de instructies bij stap 2. Ga naar de *Application Settings* page en selecteer de Tinder app. Zet ‘*Allow Facebook Authentication’ op ON* en voer de *Facebook App ID* en *App Secret* keys daaronder in.

Stappen 3, 4 en 5 zijn al uitgevoerd.

Voer stap 6 uit: voeg de volgende call aan *didFinishLaunchingWithOptions* toe, direct ná de *setApplicationId* call (direct ervóór schijnt nl. een foutmelding te geven):

PFFacebookUtils.initializeFacebook()

De syntax errors die verschijnen worden veroorzaakt omdat een tweetal include files ontbreken, voeg deze toe aan de *bridging header* file:

import <Parse/Parse.h>

import <ParseFacebookUtils/PFFacebookUtils.h>

Stap 7 geeft aan dat twee *application handlers* aan de app delegate moeten worden toegevoegd. Neem deze code één op één over van Parse.com en zet het onderaan de AppDelegate.swift file.

We zijn nu zover dat we de login en signup functionaliteit toe kunnen voegen. Kopieer de code hiervoor op Parse.com naar de *viewDidLoad* in *ViewController.swift* en run de app. Een foutmelding volgt omdat de permissions nog niet zijn ingesteld. Volg de link naar de betreffende Facebook pagina. Er zijn 3 soorten permissies die kunnen worden ingesteld: “*public\_profile*”, “*email*” en “*user\_friends*” (NB Rob noemt deze “likes” maar dat is incorrect). Voor onze app hebben we alleen toegang tot het publieke profiel nodig. De Facebook login code wordt dan als volgt:

var permissions = [“public\_profile”, “email”, “likes”] // NB niet alle drie nodig!

PFFacebookUtils.logInWithPermissions(permissions, {

(user: PFUser!, error: NSError!) -> Void in

if user == nil {

println(“Oeps, de gebruiker heeft de Facebook login gecancelled”)

} else if user.isNew {

println(“Gebruiker ingelogd en aangemeld bij Facebook”)

} else {

println(“Gebruiker ingelogd via Facebook”)

}

})

Run de app ter controle.

De code is te vinden in 9.2.

**Les 121: Dragging elements (en 122 voor code files)**

Deze les start met een analyse van wat er gebeurt als in Tinder een foto naar links of naar rechts wordt bewogen om dit vervolgens na te bouwen De foto maakt een kwartcirkelvormige beweging naar beneden. Het lijkt er tevens op dat de foto tijdens het bewegen wat kleiner wordt.

Creëer een nieuw single view project in XCode, noem het *Dragging*. We gaan in deze app voor het eerst een control element vanuit de code creëren. Creëer een label dat gecentreerd op het scherm staat en 200x100 points groot is:

var label: UILabel = UILabel(frame: CGRectMake(self.view.bounds.width/2 – 100,

self.view.bounds.height/2 - 50,

200, 100))

label.text = “Drag me!”

label.textAlignment = NSTextAlignment.center

self.view.addSubview(label)

Voor het draggen maken we gebruik van een *panning* (= dragging) *gesture recognizer*. Deze voegen we toe aan het label (ipv. de view wat eerder gedaan is):

var gesture = UIPanGestureRecognizer( target: self,

action: Selector(“wasDragged:”))

label.addGestureRecognizer(gesture)

label.userInteractionEnabled = true

func wasDragged(gesture: UIPanGestureRecognizer) {

println(“Dragged”) // bij wijze van test

}

Let op dat het label attribuut *userInteractionEnabled* op true wordt gezet, anders gebeurt er niets. Labels hebben normaliter nl. geen user interactie.

Vervolgens wordt het draggen, het bewegen van het label, toegevoegd. De richting en afstand (d.i. de translatie) van de beweging moet bekend zijn. Daarvoor is een referentie naar de gesture recognizer nodig, die om die reden als parameter aan functie *wasDragged* is meegegeven. Voeg de volgende code aan *wasDragged* toe:

let translation = gesture.translationInView(self.view)

var label = gesture.view!

label.center = CGPoint( x: label.center.x + translation.x,

y: label.center.y + translation.y)

gesture.setTranslationInView(CGPointZero, inView: self.view)

Het label was toegevoegd aan de gesture en daar kan aan worden gerefereerd dmv. *gesture.view*. Omdat we zeker weten dat de *gesture.view* niet *nil* is (het label hangt er immers aan) kunnen we de unwrap dmv. het uitroepteken forceren.

Vervolgens worden de coördinaten van het middelpunt van het label ingesteld.

Tot slot moet de translatie gereset worden, anders zou die bij de volgend aanroep van *wasDragged* weer worden meegenomen.

Run de code. Het label volgt nu de drag beweging. De basis dragging functionaliteit is hiermee geïmplementeerd. Op Tinder vindt echter naast een translatie ook een rotatie plaats en wijzigt de grootte van de foto enigszins (hoewel ik dat niet kan zien…). De rotatie is daarnaast groter naarmate de translatie groter is. Dit wordt eveneens geïmplementeerd.

Een rotatie implementeren we als volgt:

var rotation: CGAffineTransform = CGAffineTransformMakeRotation(1)

label.transform = rotation

De rotatie parameter wordt in *radialen* opgegeven. Een cirkel bestaat uit 2.pi radialen, zodoende komt een waarde van 1 radiaal overeen met ca. 60 graden rotatie met de klok mee. Voor het berekenen van de grootte van de rotatie gebruiken we het horizontale deel van de translatie. Na wat uitproberen stellen we het aantal radialen in op:

translation.x / 200

Run de code. Wat opvalt is dat het label wel enigszins roteert, maar niet meer naarmate verder van het middelpunt van het scherm wordt bewogen. Dit komt doordat de translatie bij elke call naar *wasDragged* wordt gereset. Daarom is een globale variabele nodig die de horizontale afstand tot het middelpunt van het scherm bijhoudt:

var xFromCenter:CGFloat = 0

Hiermee wordt de parameter gelijk aan:

xFromCenter/200

Run de code, dit ziet er zelfs nog beter uit dan wat Tinder doet!

De grootte van het label moet nu nog kleiner worden naarmate het label verder van het middelpunt van het scherm raakt. Hiervoor definiëren we een variabele ‘*stretch’* die direct gerelateerd is aan de waarde van *xFromCenter*. Na wat trial en error voor de bepaling van een goede waarde van scale ontstaat het volgende:

var scale = min(50 / abs(xFromCenter), 1)

var stretch:CGAffineTransform = CGAffineTransformScale(rotation, scale, scale)

label.transform = stretch

Merk op dat de eerste parameter zelf een transformatie is. We kunnen dus in een call twee (of zelfs meer) transformaties uitvoeren, in dit geval is dit de rotatie. De waarde van *scale* maken we niet groter dan 1, zodat de oorspronkelijke grootte van het label de maximale grootte is. De grootte moet kleiner worden naarmate de afstand tot het centrum van het scherm groter wordt, daarom delen we door *xFromCenter*. Als *scale* negatief wordt dan heeft dit het effect dat het label ondersteboven wordt weergegeven, wat ongewenst is, vandaar de *abs* functie.

Het laatste wat we voor onze Tinder app willen gebruiken is of het label voorbij een bepaald punt op het scherm wordt bewogen. In dat geval moet er nl. iets gebeuren: ve naar links betekent ‘niet leuk’, ver naar rechts is ‘leuk’. Dit testen we aan de hand van de x-coördinaat van het centrum van het label:

if label.center.x < 100 { // stel waarde naar wens in

println(‘Does not like’)

} else if label.center.x > self.view.bounds.width-100 {

println(‘Likes’)

}

Als laatste herstellen we de positie, oriëntatie en afmetingen van het label. Wanneer bijv. een foto in Tinder vóór het overschrijden van de grens wordt losgelaten omdat de gebruiker zich bedenkt, moet de foto weer in zijn oorspronkelijke staat getoond worden. Hiervoor moeten we testen op het eind van de drag beweging, dus op de status van de gesture. Voeg aan het eind van *viewDidLoad* de volgende code toe:

if gesture.state = UIGestureRecognizerState.ended {

label.center = CGPointMake(self.view.bounds.width/2,

self.view.bounds.height/2)

scale = max(abs(xFromCenter)/50, 1)

rotation = CGAffineTransformMakeRotation(0)

stretch = CGAffineTransformScale(rotation, scale, scale)

label.transform = stretch

}

Hier wordt het midden van het label op het midden van het scherm gezet.

De scale wordt geïnverteerd tov. wat hij was, dus delen door 50 en het minimum van het resultaat en 1 nemen.

De rotatie wordt op 0 gezet, dit is dus de rotatie tov. de oorspronkelijke oriëntatie.

De code is te vinden in 9.3.

**Les 123: Push notifications (en 124 voor code files)**

In deze les worden de voorwaarden behandeld om push notifications vanuit een app te versturen en wordt een eenvoudig code voorbeeld daarvan gegeven. Push notifications kunnen overigens op de simulator verzonden, maar niet ontvangen worden. Ontvangen kan alleen op een iOS device en daarvoor is een Apple Developer account nodig.

De voorbereidingen voor het versturen van push notifications zijn tamelijk omslachtig. Op Parse.com is een ‘*iOS Push Notifications’* tutorial te vinden waarin dit stap voor stap wordt uitgelegd. Alleen bijzonderheden worden hier vermeld.

* Stap 1.1 Generating a certificate request:

Selecteer Save to disk bij de aanvraag van het certificaat. Ik heb een aparte folder ‘Certificaten’ onder mijn home dir (Famasoft op OWC) hiervoor aangemaakt.

* Stap 1.2 Creating an App ID:

Eenmaal ingelogd op de Apple Developer site, selecteer ‘Certificates, Identifers & Profiles’, vervolgens Identifers onder IOS Apps. Klik hier op + (toevoegen). Voeg bij Name ‘Tinder Demo’ (of iets naar keuze) in. Selecteer Explicit App ID en vul bij Bundel ID ‘com.famasoft.tinderdemo’ in, dit mag naar wens wat anders zijn, zolang het maar uniek is. Scroll verder en vink Push Notifications aan onder Services. Er verschijnt nu een overzichtscherm, check of alles klopt en klik op Submit.

* Stap 1.3 Configuring your App ID for Development Push Notifications

Selecteer het overzichtscherm met App ID’s (Identifiers – App IDs) en selecteer de TinderDemo. Klik op Edit en scroll naar beneden, naar het Push Notifications deel. Onder ‘Development SSL Certificate’ klik op ‘Create Certificate…’ Hier wordt om de in stap 1.1 gecreëerde .csr file gevraagd. Klik op Continue en in het volgende scherm op Choose File… Selecteer de in stap 1.1 gecreëerde file en klik op Generate. Wanneer het certificaat gegenereerd is save het dan bij het eerste certificaat. Dubbelklik op het certificaat in de Finder om het in KeyChain te installeren. KeyChain opent automatisch, zoek het certificaat daar op. We gaan het certificaat nu exporteren naar .p12 format via Archief – Exporteer Onderdelen… Geef het certificaat een naam naar keuze en sla het bij de andere certificaten op, vul een password in indien gewenst, die hoeft echter niet. Vervolgens wordt om het login password gevraagd. Vul dit in en check in de Finder of het certificaat is opgeslagen.

Nu ben ik toch alles weer in detail aan het beschrijven. Zie voor het vervolg, vanaf stap 2, de Parse.com website. Geef het in stap 2 te genereren provisioning profile een naam die aansluit bij het doel, bijv. ‘Tinder Dev Profile’ en sla het op bij de certificaten. Het profiel moet nog geïnstalleerd worden, dit gebeurt automatisch door in de Finder erop te dubbelklikken. In XCode is het in de Preferences – Accounts terug te vinden.

Nu moet de app op Parse.com nog verder geconfigureerd worden, ervan uitgaande dat daar een Tinder app is gecreëerd. Selecteer de Tinder app op Parse.com, ga naar de Push sectie en zet de switch Client Push Enabled op Yes. Vervolgens moet het .p12 certificaat dat eerder is gecreëerd geladen worden bij ‘Apple Push Certificates’.

Nu kan in XCode de app voorbereid worden op het gebruik van Push notifications. Open de *info.plist* file en pas de bundle identifier aan aan wat eerder is ingevoerd en op Parse.com te zien was na laden van het .p12 certificaat.

Open vervolgens de *Build Settings*, ga naar Code Signing en check de Code Signing Identity. Hier moet de Apple Development iOS Push Services identity voor onze Tinder app gekozen worden.

Nu kan de code voor de push notifications ingevuld worden. In *AppDelegate.swift*, voeg aan het eind van *didFinishLaunchingWithOptions* de volgende code toe:

var pushSettings: UIUserNotificationSettings =

UIUserNotificationSettings(forTypes: .Alert, categories: nil)

application.registerUserNotificationSettings(pushSettings)

application.registerForRemoteNotifications()

Run de app. Deze zal nu via een dialoog box om toestemming vragen voor het versturen van push notifications. Antwoord Ja.

Er zijn nu 2 methoden nodig die ook in *AppDelegate.swift* geïmplementeerd moeten worden:

* *didRegisterForRemoteNotificationsWithDeviceToken*, deze wordt aangeroepen als de registratie succesvol is verlopen.
* *didFailToRegisterForRemoteNotificationsWithError*, voor als de registratie is mislukt.

Zet in beide een println() statement, resp “success” en “fail”. Run de app vanuit de Simulator en er zal “fail” verschijnen, dit is omdat de Simulator geen push notifications kan ontvangen. Maar zenden kan wel en daarvoor voegen we in de *ViewController.swift* in *viewDidLoad* de volgende code toe:

var push = PFPush()

push.setMessage(“This is a test”)

push.sendPushInBackgroundWithBlock({

(isSuccessful: Bool!, error: NSError!) -> Void in

println(isSuccessful)

})

Run de app ter controle. De “fail” verschijnt nog steeds en erna verschijnt true ten teken dat de notificatie succesvol is verstuurd. Dit is niet vanuit de app te controleren, maar op Parse.com wel. Selecteer daar de Tinder App en selecteer het Push tabblad. Bij succes zijn daar de verzonden push berichten te zien.

Het is overigens ook mogelijk direct van Parse.com een push bericht te sturen naar een aangesloten device.

De code is te vinden in 9.4.

**Les 125: Login & signup (en 126 voor code files)**

Hoewel het inloggen in onze Tinder app via Facebook werkt vindt daarvan geen terugkoppeling plaats. Dat wordt in deze les in de app ingebouwd. Ook het geval dat de gebruiker geen Facebook account heeft wordt ondervangen. Tevens wordt er gevraagd naar welk geslacht de interesse uit gaat. Verder wordt ingegaan hoe informatie bij Facebook vandaan gehaald kan worden.

Creëer een *button* op het storyboard voor de Facebook login. Geef deze het uiterlijk van de Facebook login button, zoek daarvoor een foto op het internet. Centreer deze op het scherm.

Voeg een *label* ‘Tinder’ toe bovenaan het scherm, kopieer hier naar wens het originele Tinder logo in. Creëer *constraints* in *Auto layout* voor label en button.

Creëer een *action ‘signIn’* voor de login button. Verplaats de Facebook login code naar deze *signIn* functie. In de *viewDidLoad* komt een stuk code die checkt of de gebruiker al is ingelogd:

if PFUser.currentUser() != nil {

// definitieve code volgt nog in een volgende video

println(“user logged in”)

}

Wanneer de gebruiker is ingelogd dan wordt deze naar de Tinder hoofdpagina geleid, dat komt in een volgende video. Voor nu moet de user interface reageren op de drie mogelijkheden bij het inloggen in signIn.

Ten eerste kan de Facebook login worden gecancelled. De hoofdpagina verschijnt dan – komt in een volgende video – en het is netjes een melding te laten zien. Creëer een *label* boven de inlog button en zet de tekst ‘*The login was unsuccessful, please try again*’ er in. Maak de kleur rood en zet de *alpha* op 0, zodat het initieel niet zichtbaar is. Voeg een *outlet* ‘loginCancelledLabel’ toe en voeg in signIn na de ‘*if user == nil*’ de volgende code toe:

self.loginCancelledLabel.alpha = 1

Maak het label onzichtbaar direct aan het begin van *signIn*:

self.loginCancelledLabel.alpha = 0

De situatie dat een nieuwe user zich aanmeldt in Tinder vergt wat meer werk. Hiervoor maken we een aanmeldscherm waar naar meer informatie wordt gevraagd. Creëer een nieuwe ‘*signup’* viewcontroller en creëer een *segue* ‘signUpSegue’ van de inlogbutton naar deze signup viewcontroller. In functie *signUp* trigger een segue na de ‘else if user.isNew’:

self.performSegueWithIdentifier(“signUpSegue”, sender: self)

Om dit te kunnen testen moeten we de Tinder app van het door voor Tinder gebruikte Facebook account verwijderen. Doe dit door naar het Facebook profiel te gaan en via de Instellingen de Apps op te vragen. Daar staat Tinder tussen, verwijder deze en klaar is Kees.

Om helemaal zeker te zijn verwijder ook de in Parse.com bij Tinder aangemelde user. Ga hiervoor naar Parse.com, selecteer de Tinder app, ga naar Core, selecteer de aangemelde gebruiker (als het goed is is het er één) en verwijder het record.

Dit betekent dat we niet langer op Tinder zijn aangelogd en bij het opstarten van onze Tinder app in de ‘*else if user.isNew’* tak terechtkomen. Test dit ter controle. De gebruiker moet zich expliciet via Facebook aanmelden waarna het nog lege ‘signup’ scherm verschijnt.

Om niet voor elke test de accounts te verwijderen is het handiger de nieuwe ‘signup’ viewcontroller de initial viewcontroller te maken.

We kunnen nu verder met het inrichten van het ‘sign up’ scherm. Voeg hier aan toe:

* Een label ‘Sign Up’ bovenaan het scherm.
* Een image view voor het gebruikersprofiel midden op het scherm, geef het een redelijke grootte en zet de Mode op Aspect Fit zodat de foto niet vervormd wordt weergegeven. In deze les wordt de Facebook gebruikt, een leuke uitbreiding is om de gebruiker zelf een foto te laten kiezen dmv. een image picker, zoals dit voor de Instagram app is geïmplementeerd.
* Een switch om de interesse voor mannen danwel vrouwen kenbaar te maken. Niet voor beide, dat is weer een leuke uitbreiding.
* Zet een label boven de switch met de tekst ‘I am interested in…’, een label ‘Men’ links van de switch en een label ‘Women’ rechts van de switch.
* Plaats een button ‘Sign Up’ hieronder om het Sign Up scherm af te sluiten en de gegevens op te slaan.

De controle elementen staan hiermee op het scherm. Voor de besturing moet een viewcontroller gecreëerd worden. Creëer een Cocoa Class file, noem deze SignUpViewController en maak het een subclass van UIViewController. Selecteer de ‘sign up’ view controller in het storyboard en vul SignUpViewController in bij Custom Class in de Attributes Inspector.

Creëer outlets/actions:

* een outlet ‘*genderSwitch’* vor de switch
* een outlet ‘*profilePic’* voor de image view
* een action ‘*signUp’* voor de sign up button

In *viewDidLoad* halen we de profielfoto van de gebruiker op Facebook op en geven deze weer in de image view:

var FBSession = PFFacebookUtils.session()

var accessToken = FBSession.accessTokenData.accessToken

let url = NSURL(String: “<url>”+accessToken) // Zie code voor <url>

let urlRequest = NSURLRequest(URL: url)

NSURLConnection.sendAsynchronousRequest(

urlRequest, queue: NSOperationQueue.mainQueue(),

completionHandler: { (response, data, error) in

let image = UIImage(data: data)

self.profilePic.image = image

// sla de foto op in Parse.com

var user = PFUser.currentUser()

user[“image”] = data

user.save()

})

Run de app om te testen. De profielfoto wordt van Facebook opgehaald en opgeslagen op Parse.com. Dit laatste doen we zodat Facebook niet elke keer benaderd moet worden om de profielfoto van de gebruiker op te halen.

Bij wijze van test kan het ophalen van de profielfoto in commentaar gezet worden en de volgende testcode toegevoegd worden direct buiten de closure:

var user = PFUser.currentUser()

let image = UIImage(data: user[“image”])

profilePic.image = image

Run en constateer dat de profielfoto wordt weergegeven, omdat deze was opgeslagen op Parse.com.

De volgende stap is het ophalen van het geslacht van de gebruiker. Dit is publieke informatie op Facebook en kunnen we opvragen. Het is onderdeel van de profielinfo op Facebook, die als volgt opgehaald wordt. Voeg aan het eind van *viewDidLoad* de volgende code toe:

FBRequestConnection.startForMeWithCompletionHandler({

(connection, result, error) in

println(result)

})

Dit drukt de profielinfo af. Dat is een dictionary waarvan gender een element is. Maak de definitie van variabele *user* globaal zodat de info bewaard blijft en sla de gender info er in op:

self.user[“gender”] = result[“gender”] //’self’ omdat in closure

self.user[“name”] = result[“name”]

self.user.save()

Run de app. Deze crashed. De reden is dat de call naar *PFUser.currentUser()* plaatsvindt voordat Parse goed en wel geïnitialiseerd is. Zet de call over naar *viewDidLoad* en de fout verdwijnt. Run de app en check dat de gender en name op Parse.com zijn opgeslagen.

Tot slot moet de switch nog afgevangen worden. Dat doen we in de *signUp* functie, dat is immers het moment dat de gebruiker zijn gegevens heeft ingevoerd. Een switch heeft twee waarden: *on* en *off*. On komt overeen met de rechterstand, in dit geval geeft dit een voorkeur voor vrouwen aan. Voeg de volgende code aan *signUp* toe:

var user = PFUser.currentUser()

if genderSwitch.on {

user[“interestedIn”] = “female”

} else {

user[“interestedIn”] = “male”

}

user.save()

Run de app en speel met de switch. Check het resultaat op Parse.com.

De volgende stap is dat na de sign up de Tinder hoofdpagina verschijnt middels een segue daarnaartoe. Dat komt in een volgende les aan de orde.

De code is te vinden in 9.5.

**Les 127 Geolocation and adding users (en 128 voor code files)**

Om de app goed te kunnen testen zijn meerdere gebruikers nodig. Tevens moet de (actuele) locatie bekend zijn, zodat Tinder de dichtstbijzijnde kan vinden.

Voor het bepalen van de locatie wordt *GeoLocation* gebruikt van *Parse.com*. In de documentatie op Parse.com wordt vermeld dat *GeoLocation* onderwater gebruik maakt van *Core Location*, dat betekent dat om dit te laten werken de 2 bekende properties in info.plist moeten worden opgenomen. *GeoLocation* maakt het werken met locaties nog iets eenvoudiger, maar wat mij betreft had het net zo goed direct met *Core Location* kunnen gebeuren. Nu is er nog een extra afhankelijkheid van Parse.com in de app ingebouwd.

De viewcontroller voor de hoofdpagina van Tinder wordt er hier vast bijgemaakt.

Creëer een viewcontroller op het storyboard en creëer een bijbehorende Cocoa class ‘*TinderViewController’*, een subclass van *UIViewController*. Maak deze viewcontroller de initial viewcontroller tbv. snel kunnen testen.

In de *viewDidLoad* van de *TinderViewController* wordt de actuele locatie van de gebruiker bepaald. Dat is de juiste plek om te doen, elke keer op het moment dat de Tinder hoofdpagina verschijnt moet de actuele locatie van de gebruiker bekend zijn, niet wanneer wordt ingelogd. Dat is de reden dat deze viewcontroller nu al wordt aangemaakt. Voeg de volgende code toe aan *viewDidLoad*:

PFGeoPoint.geoPointForCurrentLocationInBackground {

(geoPoint: PFGeoPoint!, error: NSError!)

if error == nil {

println(geoPoint) // bij wijze van test

}

}

Voor het bepalen van de actuele locatie moeten aan de *info.plist* de 2 bekende Core Location properties worden toegevoegd, zoals dat eerder gebeurd is:

* *NSLocationWhenInUseUsageDescription*, met een begeleidende tekst (bijv. ‘Tinder needs to know your location’)
* *NSLocationAlwaysUsageDescription*, met een begeleidende tekst

Run de app. De actuele locatie wordt afgedrukt als een latitude en longitude coördinaat. Deze locatie slaan we op in Parse.com. zodat hij voor andere gebruikers beschikbaar is. Voeg de volgende code toe na de *println*:

var user = PFUser.currentUser()

user[“location”] = geoPoint

user.save()

Run de app en check op Parse.com dat de locatie wordt opgeslagen.

In het vervolg van de les bouwt Rob code waarmee een zestal gebruikers worden toegevoegd. Van ieder van hen zijn links naar hun profielfoto beschikbaar. Het zijn allemaal vrouwen, anders zou Rob ze niet zien vanwege zijn voorkeur voor vrouwen. Voor het toevoegen wordt een functie *addPerson* gebouwd met één *String* parameter die de URL van de foto bevat:

func addPerson(urlString: String) {

var newUser = PFUser()

let url = NSURL(String: urlString)

let urlRequest = NSURLRequest(URL: url)

NSURLConnection.sendAsynchronousRequest( urlRequest,

queue: NSOperationQueue.mainQueue(),

completionHandler: { (response, data, error) in

newUser[“image”] = data

newUser[“gender”] = “female” // allemaal vrouwen!

var lat = Double(37 + i)

var long = Double(-122 + i)

i += 10

var location = PFGeoPoint(latitude: lat, longitude: lon)

newUser[“location”] = location

newuser.username = “\(i)”

newuser.password = “password”

newuser.signup() // Niet vergeten!!!

}

}

Variabele i wordt buiten de functie gedefinieerd en is initieel 0. Voeg in *viewDidLoad* 6 calls naar addPerson toe met een geldige link als parameter. Zie de code in 9.6 voor de details.

Run de app om de gebruikers toe te voegen. Doe dit eenmalig! Anders worden dezelfde gebruikers meerdere keren toegevoegd. Zet de *addPerson* calls daarna weer in commentaar.

**Les 129: Swiping users (en 130 voor code files)**

In de vorige les zijn een aantal testgebruikers toegevoegd. Deze zijn op Parse.com in de User tabel te vinden. Wat opvalt is dat gebruiker 16 (in mijn geval) geen image heeft opgeslagen. Dit kan in de praktijk ook voorkomen. We gaan hiervoor een placeholder in de plaats gebruiken. Zoek een willekeurige foto die geschikt is als placeholder en importeer in het project, in de les heeft deze de naam ‘*placeholder.jpg’*.

Nu de placeholder beschikbaar is kan een begin gemaakt worden met de code voor het swipen. Kopieer hiertoe de *wasDragged* functie van les 121 (Dragging) naar *TinderViewController.swift*, dit wordt een klassemethode. Doe dit ook voor de *xFromCenter* variabele, dit wordt een globale klassevariabele. Kopieer de code uit de *viewDidLoad* functie van les 121 waarin label en gesture worden gedefinieerd over naar de *viewDidLoad* van de *TinderViewController*.

Ipv. een label hebben we een image nodig. Vervang de label en gesture code als volgt:

var userImage:UIImageView = UIImageView(frame:

CGRectMake(self.view.bounds.width/2 -100,

self.view.bounds.height/2 – 50,

200, 100))

userImage.image = UIImage(named: “placeholder.jpg”)

self.view.addSubview(userImage)

var gesture = UIPanGestureRecognizer(target: self,

action: Selector(“wasDragged”))

userImage.addGestureRecognizer(gesture)

userImage.userInteractionEnabled = true

Run de code ter controle. De foto gaan we full screen maken, het is wel handig als je iemand herkent ;-) Tevens moeten de proporties correct blijven. Vervang de definitie van *userImage* door:

var userImage: UIImageView = UIImageView(frame:

CGRectmake(0, 0, self.view.bounds.width,

self.view.bounds.height))

userImage.contentMode = UIViewContentMode.ScaleAspectFit

De code voor als de gesture beëindigd is moet ook worden aangepast aan de situatie dat nu een image ipv. een label wordt gebruikt. Merk op dat in de les nog steeds de variabele label wordt gebruikt, terwijl het nu een image view representeert:

if gesture.state == UIGestureRecognizerState.Ended {

label.removeFromSuperview()

var userImage: UIImageView = UIImageView(frame:

CGRectmake(0, 0, self.view.bounds.width,

self.view.bounds.height))

userImage.image = UIImage(named: “placeholder.jpg”)

userImage.contentMode = UIViewContentMode.ScaleAspectFit

self.view.addSubview(userImage)

var gesture = UIPanGestureRecognizer(target: self,

action: Selector(“wasDragged”))

userImage.addGestureRecognizer(gesture)

userImage.userInteractionEnabled = true

xFromCenter = 0

}

Run de app ter controle. De placeholder kan heen en weer bewogen worden. De volgende stap is de data van Parse.com te gebruiken. Het is de bedoeling dat de gebruikers die het dichtstbij zijn als eerste worden getoond. Hiervoor heeft Parse.com een functie beschikbaar, zie in de Docs sectie bij *GeoPoints*. Dit gaat dmv. een query shereKey:nearGeoPoint, hiermee kunnen de <n> dichtstbijzijnde gebruikers opgezocht worden. Deze lijst met gebruikers wordt opgesteld direct nadat de eigen locatie bepaald is, dit gebeurt in *viewDidLoad* van de *TinderViewController*:

var query = PFUser.query()

query.whereKey(“location”, nearGeoPoint: geoPoint)

query.limit = 10 // bij wijze van test, we hebben ook maar 6 gebruikers

query.findObjectsInBackgroundWithBlock({ (users, error) -> Void in

for user in users {

println(user) // test, definitieve code volgt

}

})

De huidige query zoekt 10 users op. Dit kan enige tijd duren waardoor het handiger is om in kleine stapjes de gebruikersinfo binnen te halen. Dit is een verbetering van de app die in de les niet verder wordt uitgewerkt.

De informatie die we nodig hebben bestaat uit de username en de foto. Hiervoor creëren we 2 globale arrays binnen de *TinderViewController* klasse:

var userNames = [String]()

var userImages = [NSData]()

var currentUserIndex // nodig om de volgende gebruiker te laten zien

Vervang de *println(user)* nu door:

self.userNames.append(user.username)

self.userImages.append(user[“image”] as imageData)

Er moeten nog een aantal punten verholpen worden. Zo krijgt een gebruiker zichzelf te zien en wordt er niet naar de geslachtsvoorkeur gekeken. Voeg hiervoor de volgende *whereKey* statements toe aan de query:

query.whereKey(“username”, notEqualTo: PFUser.currentUser().username)

query.whereKey(“gender”, equalTo: PFUser.currentUser()[“interestedIn”])

Het opgeven van verschillende keys zorgt ervoor dat deze als een AND worden gecombineerd, zodat alleen de resultaten die aan alle drie de queries voldoen er uit komen.

Dan moet de volgende gebruiker verschijnen wanneer iemand is weggeveegd. Vervang in het stuk code nadat de gesture is beëindigd het volgende statement:

self.currentUserIndex++

userImage.image = UIImage(data: self.userImages[self.currentUserIndex])

Let op het gebruik van ‘*self’*, is nodig omdat de code in een closure zit (NB Dit klopt niet maar het is ook weer geen syntax fout).

De 3 *whereKeys* achter elkaar doen het echter niet bij Rob, hij krijgt slechts één resultaat terug terwijl dit er meer moeten zijn (alle ingevoerde dames voldoen aan het zoekprofiel). Hij lost dit op door alleen de *nearGeoPoint* query te laten staan en in de ‘*for user in users*’ loop te checken of de gevonden gebruiker aan alle criteria voldoet. Omdat nu veel meer data wordt opgehaald is dit een stuk minder efficiënt, maar het werkt wel.

Een ander probleem is dat er geen check is op het aantal geregistreerde gebruikers. Dit betekent dat *currentUserIndex* groter kan worden dan het aantal elementen in *userNames* en *userImages* waardoor de app crashed. Dit kan eenvoudig ondervangen worden door:

if self.currentUserIndex < self.userNames.count

toe te voegen bij aan de code direct na het beëindigen van de gesture.

De code is te vinden in 9.7.

**Les 131: Accepting & rejecting users (en 132 voor code files)**

Op dit punt kunnen foto’s naar links of rechts bewogen worden, maar daar gebeurt verder niets mee. In de database moet bijgehouden worden welke gebruikers geaccepteerd en afgewezen zijn. Dat is het onderwerp van deze les.

Creëer hiertoe op Parse.com in de user database twee extra kolommen:

* Accepted, selecteer als type ‘array’
* Rejected, ook een array

Beide houden per gebruiker alle andere gebruikers vast die geaccepteerd danwel afgewezen zijn. Aan de hand van de swipe beweging kan de bijbehorende logica geïmplementeerd worden. Op Parse.com is in de *Docs* sectie (onder *Objects-Updating Objects*) info te vinden over het toevoegen van objecten aan een array. We gaan de functie *addUniqueObjectForkey*, deze voegt alleen objecten toe die nog niet in de array voorkomen, dit is precies wat nodig is.

Zoek de code op waar de beslissing Accepted / Rejected wordt gemaakt en pas deze als volgt aan:

if ‘swipe naar links’ { // = afgewezen

// code indien gekozen

PFUser.currentUser().addUniqueObject(usernames[currentUserIndex],

forKey: “rejected”)

PFUser.currentUser().save()

currentUserIndex++

} else {

// code indien gekozen

PFUser.currentUser().addUniqueObject(usernames[currentUserIndex],

forKey: “accepted”)

PFUser.currentUser().save()

currentUserIndex++

}

Run de code en controleer dat de ‘*accepted’* en ‘*rejected’* kolommen worden bijgewerkt op Parse.com.

Gebruikers die al geaccepteerd of afgewezen zijn moeten niet telkens terugkomen op het scherm. Die code moet nog worden ingebouwd. Dat doen we in de loop waar door de gedownloade users wordt gelopen. Definieer juist buiten de loop twee arrays voor het vasthouden van de geaccepteerde en afgewezen gebruikers:

var acceptedUsers = PFUser.currentUser()[“accepted”]

var rejectedUsers = PFUser.currentUser()[“rejected”]

De test of een gebruiker al voorkomt in de ‘*accepted’* of ‘*rejected’* list kan als volgt:

if !contains(accepted, user.username) && !contains(rejected, user.username)

Run de code ter controle. De code is te vinden in 9.8.

**Les 133: Contacting users (en 134 voor code files)**

In deze les wordt de Tinder app afgebouwd door gebruikers de mogelijkheid bieden contact met elkaar op te nemen. Dat gebeurt in de vorm van e-mail, helaas geen chat dus.

Op dit moment wordt bij het aanmelden alleen het publieke profiel bij Facebook opgevraagd. Het email adres moet apart opgevraagd worden.Pas in ViewController.swift de initialisatie van de variabele *permissions* aan:

var permissions = [“public\_profile”, “email”]

Om deze toe te voegen gaan we de bestaande Tinder gebruikers tijdelijk uitloggen en weer inloggen. Voeg in *viewDidLoad* van *ViewController.swift* in de ‘*if PFUser.currentUser() != nil*’ de volgende code toe:

PFUser.logOut()

Run de app ter controle en zie dat het geladen Facebook profiel het email adres bevat.

Haal de *PFUser.logOut()* weer weg. Voeg in *SignUpViewController*.*swift* in *viewDidLoad* na de ‘*user[“name”] = result[“name”]*’ het volgende statement toe:

user[“email”] = result[“email”]

Verwijder het eigen user account op Parse.com en run de app. Check op Parse.com dat de gegevens bij het account zijn opgeslagen.

De volgende stap is de overgang van de *SignUpViewController* naar de *TinderViewController*. Creëer de *segue* in het storyboard en geef de segue een naam, bijv. ‘*goToMainScreen’*. Voeg in de functie *signUp* na de *user.save()* het volgende statement toe:

self.performSegueWithIdentifier(“goToMainScreen”, sender: self)

In de les wordt de app gerund en crashed deze omdat in de *viewDidLoad* van de *TinderViewController* de variabelen *accepted* en *rejected* leeg zijn, omdat het eigen account opnieuw is aangemaakt. Ik heb daar zelf al een oplossing voor gecodeerd die hier op controleert – tenslotte is dit de beginsituatie van iedere gebruiker. Dit blijkt bijna identiek te zijn aan de oplossing in de les. Ik sla dit over.

Dan nu over naar het contacteren van andere Tinder gebruikers. Dit gaan we via email realiseren. Zet om te beginnen een *navigation bar* boven aan de *TinderViewController* en noem deze ‘*Tinder’*. Voeg daar een *bar button ‘Contacts’* aan toe. Voeg vervolgens een *table viewcontroller* voor de Tinder contacten toe in het storyboard. Zet hier ook een *navigation bar* bovenin, hernoem de titel tot ‘*Contacts’* en vul in de Attribute Inspector ‘*Back’* in bij de *Back button*. Selecteer het prototype cel en noem dit ‘*contactsCell’*.

Voeg een *segue* ‘*contacts’* toe van de Contacts button naar de Contacts viewcontroller. Creëer een nieuwe Cocoa class ‘*ContactsViewController’* voor de aansturing van de Contacts viewcontroller, dit is een subclass van *UITableViewController*. Vul in het storyboard *ContactsViewController* in bij *Custom Class* in de Identity Inspector.

Om te kunnen chatten moeten gebruikers elkaar wederzijds eaccepteerd hebben. Dit kunnen we manueel regelen op Parse.com door de eigen accountnaam in de ‘Accepted’ kolom in te vullen bij andere gebruikers die jij zelf geaccepteerd hebt. Nog beter is om gewoon bij alle andere gebruikers jouw eigen accountnaam in de ‘Accepted’ kolom in te vullen.

In *viewDidLoad* van de *ContactsViewController* kan nu de tabel met gebruikers ingevuld worden waarmee je een match hebt. Dit kan mbv. een query op de user tabel:

var query = PFUser.query()

query.whereKey(“accepted”, equalTo: PFUser.currentUser().username)

query.wherekey(“username”,

containedIn(PFUser.currentUser() [“accepted”] as [AnyObject])

query.findObjectsInBackgroundWithBlock{ (results, error) -> Void in

if results != nil {

println(results) // bij wijze van test

}

}

Hierbij worden eerst de records geselecteerd waarbij jouw eigen username voorkomt in de ‘*accepted’* kolom, maw. komen de gebruikers naar voren die jou geacepteerd hebben.

De tweede zoeksleutel zoekt de door jouzelf geaccepteerde gebruikers. Let op dat deze als *[AnyObject]*, dus als een array, wordt teruggegeven. Het opgeven van 2 whereKeys werkt als een AND operatie.

Run de app ter controle dat inderdaad de gegevens van alleen de matches getoond worden. Van deze gegevens hebben we de foto en het email adres nodig. Creëer hiervoor 2 globale array variabelen userImages en userEmailAddresses:

var userImages = [NSData]()

var userEmailAddresses = [String]()

Vervang de *println(results)* door:

for result in results {

self.userImages.append(result[“image”] as NSData)

self.userEmailAddresses.append(result[“email”] as String)

self.tableView.reloadData()

}

De functie *numberOfRowsInSection* wordt nu:

return userImages.count

In *cellForRowAtIndexPath* zetten we het email adres in het *text* field en de foto in de *image view*:

let cell = tableView.dequeueReusableCellWithIdentifier("contactsCell",

forIndexPath: indexPath) as UITableViewCell

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

cell.imageView?.image = UIImage(data: userImages[indexPath.row])

Voordat we de code runnen hebben we emailadress nodig van de andere gebruikers. Deze vullen we manueel in op Parse.com. Vul hier naar wens bestaande of niet bestaande testadressen in.

Run de app ter controle. Ik loop hier tegen iets onverklaarbaars aan. Het resultaat van de query is bij mij altijd hetzelfde, wat ik ook met de data doe. Zelfs als ik een gebruiker verwijder komt deze toch weer terug in de query. Ik krijg wel een juist resultaat als ik de simpele *equalTo* voorwaarde gebruik.

Tot slot wordt getoond hoe een email naar een emailadres kan woren gestuurd. Hiervoor implementeren we de functie *didSelectRowAtIndexPath*:

var url = NSURL(string: “mailto” + userEmailAdresses[indexPath.row]

+ “?subject=Hi!”)

UIApplication.sharedApplication().openURL(url)

Dit is de standaard structuur van een url waarmee een mailadres wordt aangeduid. Het subject is hier ook nog meegenomen.

Vanuit de simulator kunnen helaas geen mail gestuurd worden, maar wanneer de app vanaf een device draait dan zal de Mail app worden geopend.

De code is te vinden in 9.9.

**Section 10 – Games and Sprite Kit**

**Les 136: Hello World with Sprite Kit (en 137 voor code files)**

In deze les wordt een begin gemaakt met de ontwikkeling van Flappy Bird. Creëer een nieuw project, selecteer *Game* en in het volgende scherm *SpriteKit*, dit is Apple’s belangrijkste framework voor de ontwikkeling van games.

Het main storyboard bevat een *game viewcontroller*. SpriteKit werkt met *scenes*, deze worden in een viewcontroller gecreëerd. In de *viewDidLoad* van de viewcontroller – in de file *GameViewController.swift* – wordt de game scene geladen, die in de file *GameScene.swift* staat. De scene wordt geconfigureerd en geactiveerd.

In de game scene wordt in de functie *didMoveToView* een label ‘Hello World’ gecreëerd en toegevoegd aan de view.

In de functie *touchesBegan* wordt op het moment dat op het scherm getapt wordt op die positie een sprite ‘*Spaceship’* op de aanraakplek getoond, die vervolgens om zijn as gaat roteren.

Run de app om te zien wat er gebeurt. Haal daarna de code die we niet nodig hebben weg: maak *didMoveToView* en *touchesBegan* leeg.

De code is te vinden in 11.1.

**Les 138: Adding and animating sprites (en 139 voor code files)**

In deze les wordt een eerste animatie gemaakt van een ‘vliegende’ Flappy Bird. Haal hiervoor de plaatjes op van ‘iosdeveloperscourse.com/coursecontent/flappybirdimages.zip’. Unzip en kopieer de img dir met de plaatjes naar het project. Vink ‘copy items if needed’ aan.

Als start voegen we de eerste sprite toe aan de app. Definieer in *GameScene.swift* de klasse variabele *bird*:

var bird = SKSpriteNode()

Voeg aan in *didMoveToView* de volgende code toe:

var birdTexture = SKTexture(imageNamed: “img/flappy1.png”)

bird = SKSpriteNode(texture: birdTexture)

bird.position = CGPoint(x: CGRectMidX(self.frame), y: CGRectMidY(self.frame))

self.addChild(bird)

In SpriteKit wordt met ‘*texture’* een image bedoeld. De functie *addChild* voegt een ‘*node’* toe aan de scene. Run de app om te zien dat Flappy Bird in het midden van het scherm wordt getoond.

De volgende stap is om Flapy Bird te laten ‘flappen’. Creëer hiervoor een tweede texture:

var birdTexture1 = SKTexture(imageNamed: “img/flappy2.png”)

Creëer een animatie waarin beide plaatjes afwisselend getoond worden gedurende 0.1 sec. Voeg de volgende code ná de *bird.position* assignment en vóór de *addChild* toe:

var animation = SKAction.animateWithTextures( [birdTexture, birdTexture2],

timePerFrame: 0.1)

var makeBirdFlap = SKAction.repeatActionForever(animaion)

bird.runAction(makeBirdFlap)

Run de app: we hebben nu een vliegende Flappy Bird.

NB. Bij mij werden de twee plaatjes niet goed geladen. Het bleek dat in de filenaam de “*/img*” verwijderd moet worden, ondanks dat de plaatjes wel degelijk in de img directory staan. De code van Rob is hetzelfde, de plaatjes staan daar ook in de img dir en daar werkt het wel. Enige verschil is dat zijn img dir icon blauw is, bij mij is het geel. Lijkt alsof ze een andere betekenis in XCode hebben. Heel vreemd dit…

De code is te vinden in 11.2.

**Les 140: Animating the background (en 141 voor code files)**

In deze les wordt de achtergrond toegevoegd aan de app. Definieer een klasse variabele *background*:

var backgound = SKSpriteNode()

Voeg de volgende code toe aan het begin van *didMoveToView*:

var backgroundTexture =

SKTexture(imageNamed: “bg.png”) // geen “img/” ervoor!!

background = SKSpriteNode(texture: backgroundTexture)

background.position = CGPoint( x: CGRectMidX(self.frame),

y: CGRectMidY(self.frame))

background.size.height = self.frame.height

self.addChild(background)

Let op dat de hoogte van het achtergrond plaatje gelijk wordt gemaakt aan de hoogte van het scherm. Dit werkt zowel in portrait als landscape mode. De breedte van de achtergrond laten we nu voor wat het is.

Run de app, we hebben nu een vliegende Flappy Bird voor een achtergrond.

NB. De achtergrond moet vóór de Flappy Bird worden getekend, doe het andersom en we krijgen een flikkerende Flappy Bird te zien wat niet de bedoeling is. Later blijkt dat we dit kunnen oplossen door de het *zPosition* attribuut zodanig in te stellen dat Flappy vóór de achtergrond zichtbaar is.

De achtergrond gaan we naar links bewegen zodat het lijkt alsof Flappy Bird naar rechts beweegt. Een eerste poging is om de achtergrond in stapjes van 10 pixels per 0.1 seconde naar links te laten bewegen. Voeg de volgende code toe aan *didMoveToView*, direct onder de initialisatie van de *background* variabele:

var moveBackground = SKAction.moveByX(-10, y:0, duration:0.1)

var moveBackgroundForever = SKAction.repeatActionForever(moveBackground)

background.runAction(moveBackgroundForever)

Run de app. Het lijkt alsof de vogel vliegt, echter omdat we een enkel achtergrondplaatje hebben beweegt deze van het scherm en vliegt Flappy Bird op een grijs scherm. Wat ook opvalt is dat de achtergrond weliswaar netjes in het midden van het scherm staat, maar omdat de breedte in landscape mode kleiner is dan de schermbreedte ontstaan er links en rechts 2 grijze banden. Dit betekent dat de achtergrond een aantal keren naast elkaar op het scherm gezet moet worden, dit aantal wordt bepaald door de breedte van het beeldscherm.

Verder kunnen we de achtergrond wel naar links bewegen, maar op een gegeven moment komt toch de grijze band weer terug omdat de naast elkaar geplakte plaatjes uiteindelijk van het scherm verdwijnen.

De oplossing is om de plaatjes als afzonderlijke nodes (sprites) naast elkaar op het scherm te zetten en ze ieder met dezelfde snelheid naar links te laten bewegen over hun volledige breedte. Op het moment dat die beweging klaar is, moet stante pede het plaatje weer op het scherm gezet worden. Doordat de plaatjes netjes aangelijnd achter elkaar geplaatst zijn, verschijnt er geen grijze achtergrond op het scherm. Het opvolgen van acties wordt een ‘*action sequence’* genoemd, via de functie sequence kunnen meerdere acties sequentieel gecombineerd worden.

De code wordt nu als volgt:

var t:NSTimeInterval = NSTimeInterval(backgroundTexture.size().width/100)

var moveBackground =

SKAction.moveByX(-backgroundTexture.size().width,

y: 0, duration: t)

var replaceBackground = SKAction.moveByX(-backgroundTexture.size().width,

y: 0, duration: 0) // let op: 0 = stante pede!

var moveBackgroundForever =

SKAction.repeatActionForever(

SKAction.sequence([moveBackground, replaceBackground]))

var nBackgrounds:Int =

Int(self.frame.width / backgroundTexture.size().width) + 2

for var i:CGFloat=0; i<2; i++ {

background = SKSpriteNode(texture: backgroundTexture)

background.position =

CGPoint(x: (backgroundTexture.size().width) \* (1/2 + i),

y: CGRectMidY(self.frame))

background.size.height = self.frame.height

background.runAction(moveBackgroundForever)

self.addChild(background)

}

De variabele *nBackgrounds* is het aantal achtergrond plaatjes dat naast elkaar wordt getoond. De ‘+2’ is te verklaren door de beweging naar links van de plaatjes. Past de eerste net niet op het scherm (net te klein) dan is sowieso een tweede ernaast nodig. Echter na het beëindigen van de beweging past deze ook net niet meer op het scherm waardoor toch een stuk grijze achtergrond rechts zou verschijnen. Daar is het tweede extra achtergrondplaatje voor nodig.

Run de app, we hebben nu een vliegende Flappy Bird op een continu naar links bewegende achtergrond.

De code is te vinden in 11.3.

**Les 142: Controlling sprites (en 143 voor code files)**

In deze les gaan we Flappy Bird op en neer laten bewegen. Door op het scherm te tappen krijgt hij ahw. een schop omhoog. Omdat Flappy gewicht heeft valt hij naar beneden. Het is dus zaak af en toe te tappen zodat hij in de lucht blijft.

SpriteKit heeft hiervoor een ingebouwde ‘*physics engine’*, waarmee bijv. zwaartekracht op sprites kan worden toegepast waardoor de bewegingen er natuurlijk uit zien. Voeg juist vóór de *addChild(bird)* instructie de volgende code toe:

bird.physicsBody = SKPhysicsBody(circleOfRadius: bird.size().height/2)

bird.physicsBody.dynamic = true

bird.physicsBody.allowsRotation = false

De *physics body* heeft de vorm van een cirkel, dit benadert de vogelfiguur goed. Het *dynamic* attribuut geeft aan dat zwaartekracht toegepast moet worden. Omdat cirkelvormige objecten de neiging hebben te roteren zetten we dat uit. Ik zag geen verschil totdat de vogel tegen een pijp (met een physics body) botst waarna hij kantelde en op een gegeven moment ging draaien.

Run de app. De vogel valt direct naar beneden en verdwijnt van het scherm. Om te voorkomen dat hij van het scherm verdwijnt gaan we een nieuwe sprite definiëren: de grond. Wanneer de vogel hier op terecht komt stuitert hij een paar keer tot hij blijft liggen. Dit wordt automatisch door SpriteKit verzorgd!

Voeg onder de *addChild(bird)* de volgende code toe:

var ground = SKNode()

ground.position = CGPointMake(0,0)

ground.physicsBody =

SKPhysicsBody(rectangleOfSize:CGSizeMake(self.frame.size.width,1))

ground.physicsBody.dynamic = false

self.addChild(ground)

Merk op dat geen texture nodig is voor de grond, het is ahw. een onzichtbare lijn.

Run de app. De vogel blijft nu na een paar keer stuiteren op de grond liggen.

Om de Flappy Bird weer de lucht in te krijgen tappen we op het scherm. Dit geeft de vogel een impuls omhoog waarna hij door de zwaartekracht weer terugvalt. Voeg de volgende code toe:

bird.physicsBody.velocity = CGVectorMake(0,0)

bird.physicsBody.applyImpulse(CGVectorMake(0,50))

Hiermee wordt eerst de dalende beweging gestopt door de snelheid op 0 te zetten. In de praktijk blijkt de beweging van Flappy hierdoor beter te controleren. Vervolgens krijgt Flappy een ram omhoog. De waarde van 50 kan naar wens ingesteld worden, ik vond zelf een waarde van 100 prettiger werken.

Ik heb zelf nog een ceiling toegevoegd zodat Flappy beperkt omhoog kan bewegen.

De code is te vinden in 11.4.

**Les 144: Spawning sprites (en 145 voor code files)**

In deze les worden de pijpen toegevoegd aan de scenes. Deze bevinden zich recht tegen over elkaar onder en boven aan het scherm met een ruimte ertussen, hier moet Flappy tussendoor vliegen. Deze ruimte stellen we in op 4 Flappy Birds groot. Om de 3 seconden (instelbaar) verschijnt rechts een stel pijpen op het scherm dat met de achtergrond mee beweegt. De plek waar de ruimte zit varieert en is in principe random. Wel is het zo dat de ruimte niet beneden een kwart en boven driekwart van het scherm zit.

Voor het verschijnen van de pijpen gebruiken we een timer, deze wordt in *didMoveToView* gedefinieerd:

var timer =NSTimer.scheduledTimerWithTimeInterval(3, target: self,

selector: Selector(“makePipes”), userInfo: nil, repeats: true)

We gaan nu de pijpen in eerste instantie midden op het scherm zetten, met een gat ertussen ter grootte van 4 Flappy Birds precies halverwege het scherm:

let gapHeight = bird.size.height \* 4

var pipe1Texture = SKTexture(imageNamed: “pipe1.png”)

var pipe1 = SKSpriteNode(texture: pipe1Texture)

pipe1.position = CGPoint( x: self.frame.width/2,

y: self.frame.height/2 + pipe1.size.height/2

+ gapHeight/2)

self.addChild(pipe1)

var pipe2Texture = SKTexture(imageNamed: “pipe2.png”)

var pipe2 = SKSpriteNode(texture: pipe2Texture)

pipe2.position = CGPoint( x: self.frame.width/2,

y: self.frame.height/2 - pipe2.size.height/2

- gapHeight/2)

self.addChild(pipe2)

Run de app ter controle dat de ruimte tussen de pijpen in vertikale zin halverwege het scherm zit.

De positie van de ruimte tussen de pijpen gaan we variëren om het spel spannend te maken. In de les gaat Rob uit van maximaal een kwart schermhoogte naar boven of beneden vanaf het huidige middelpunt van het scherm. Om het onvoorspelbaar te maken gebruiken we daarvoor een random variabele *movementAmount*, die maximaal een halve schermhoogte groot kan zijn, gemeten van het laagste punt tot het hoogste punt:

var movementAmount = arc4random() % UInt32(self.frame.height/2)

Het eerste probeersel gaat uit van de positie halverwege de hoogte van het scherm. Omdat *movementAmount* wordt berekent vanaf de laagste positie, d.i. een kwart schermhoogte, moeten we voor de berekening van de *pipeOffset* die uitgaat van halverwege de hoogte van het scherm dus nog een kwart schermhoogte aftrekken. Voor beide pijpen wordt de positie nu als volgt bepaald:

var pipeOffset = CGFloat(movementAmount) – self.frame.height/4

pipe1.position = CGPoint( x: self.frame.width/2,

y: self.frame.height/2 + pipe1.size.height/2

+ gapHeight/2 + pipeOffset)

pipe2.position = CGPoint( x: self.frame.width/2,

y: self.frame.height/2 + pipe2.size.height/2

+ gapHeight/2 + pipeOffset)

Run de app een aantal keer en constateer dat de ruimte tussen de pijpen nog steeds even groot is maar van positie wisselt.

Wat opvalt is dat Flappy flikkerend zichtbaar is. Het flikkeren van Flappy wordt veroorzaakt doordat zowel de pijp als Flappy dezelfde ‘*Z-positie’* heeft, dwz. op dezelfde ‘diepte’ zitten. Default is deze waarde 0 voor ale sprites (of nodes). Door die van Flappy groter te maken, komt deze vóór de pijp te liggen:

bird.zPosition = 10

De volgende stap is om de pijpen van rechts naar links te laten bewegen. De startpositie is ruim rechts van het scherm (en niet halverwege zoals nu het geval is). Wanneer de pijpen links uit beeld zijn verdwenen halen we ze weg zodat geen kostbaar geheugen wordt gebruikt:

var movePipes = SKAction.moveByX(-self.frame.width\*2, y: 0,

duration: NSTimeInteval(self.frame.width/100))

pipe1.position = CGPoint( x: self.frame.width \*1.5, // rechts van het scherm!

y: self.frame.height/2 + pipe1.size.height/2

+ gapHeight/2 + pipeOffset)

pipe1.runAction(movePipes)

pipe2.position = CGPoint( x: self.frame.width \* 1.5, // rechts van het scherm!

y: self.frame.height/2 + pipe2.size.height/2

+ gapHeight/2 + pipeOffset)

pipe2.runAction(movePipes)

In tegenstelling tot de beweging van de achtergrond is de beweging van de pijpen iets eenmaligs. De herhaling wordt veroorzaakt door de timer, maar dan hebben we een nieuw stel pijpen dat van rechts naar links beweegt.

Run de app ter controle. De pijpen bewegen van rechts naar links en elke 3 seconden komt er een nieuw stel pijpen in beeld. Flappy kan nu echter dwars door de pijpen heen vliegen en dat is niet de bedoeling. Dit lossen we op door de pijpen te voorzien van een *physics body*:

pipe1.physicsBody = SKPhysicsBody(rectangleOfSize: pipe1.size)

pipe1.dynamic = false

pipe2.physicsBody = SKPhysicsBody(rectangleOfSize: pipe2.size)

pipe12.dynamic = false

De code is te vinden in 11.5.

**Les 146: Collisions (en 147 voor code files)**

In deze les wordt de botsing van Flappy met een pijp en de grond afgehandeld. Botst Flappy tegen een pijp of komt hij op de grond, dan wordt het spel gestopt: de beweging van de achtergrond en pijpen stopt, Flappy kan niet meer bewogen worden en er worden geen nieuwe pijpen meer gecreëerd.

We hebben hiervoor een mechanisme nodig om de botsing van Flappy met een paal te detecteren. Dit is ingebouwd in SpriteKit en wel in de SKPhysicsContactDelegate, deze voegen we toe aan de klasse definitie van GameScene:

class GameScene: SKScene, SKPhysicsContactDelegate {

Zet de physics contact delegate op self:

self.physicsWorld.contactDelegate = self

In *SpriteKit* wordt onderscheid gemaakt tussen het maken van *contact* en *botsingen*. *Contact* is het elkaar raken van twee objecten, een *botsing* volgt op contact en de detectie ervan wordt gebruikt om de impact op de beweging van beide objecten te bepalen, dat doet SpriteKit overigens automatisch.

Stel bijv. dat in een space game een raket wordt afgeschoten op een ruimteschip. Wanneer de raket contact maakt met het ruimteschip dan wordt het ruimteschip vernietigd. Er is dan geen sprake van een botsing. Dit is wel het geval wanneer twee ruimteschepen elkaar raken: dan is er zowel sprake van contact als een botsing.

In SpriteKit geef je aan of je geïnteresseerd bent in het contact en botsingen tussen twee objecten. Hiertoe worden de objecten ingedeeld in *objectcategoriën*, in een space game heb je bijv. de categoriën rocket ship, missile, asteroid en planet. Per object leg je vast tot welke object categorie deze behoort. Hiervoor wordt een *bitmask* gebruikt, waarin elk bit een object categorie representeert. Zo kun je in principe dus meerdere object categoriën koppelen aan een object (maar dan wordt het wel ingewikkeld…). De object categorie wordt vastgelegd in de *physics body* van een object, bijv.

let spaceShipObjectCategory = 1

let missileObjectCategory = 1 << 1

let asteroidObjectCategory = 1 << 2

let planetObjectCategory = 1 << 3

spaceship.physicsBody.categoryBitMask = spaceShipObjectCategory

missile.physicsBody.categoryBitMask = missileObjectCategory

Vervolgens leg je per object vast of je geïnteresseerd bent wanneer contact wordt gemaakt of gebotst met andere objecten. De physics body bevat hiervoor de attributen *contactTestBitMask* en *collisionBitMask*:

spaceship.physicsBody.contactTestBitMask =

spaceShipObjectCategory + asteroidObjectCategory + planetObjectCategory

missile.physicsBody.contactTestBitMask =

spaceShipObjectCategory + asteroidObjectCategory + planetObjectCategory

spaceship.collisionBitMask =

spaceShipObjectCategory + asteroidObjectCategory +

planetObjectCategory

missile.physicsBody.collisionBitMask = 0

Hieruit valt op te maken dat contact tussen een missile en een spaceship moet worden gedecteerd, het is dan voldoende om bij één van beide aan te geven dat contact met de andere object categorie moet worden gemeld door SpriteKit. Wanneer in dit voorbeeld twee missiles elkaar raken, dan hoeven we dit niet te weten.

Tevens is af te leiden dat botsen voor een missile niet interessant is: bij het minste contact explodeert het missile nl. Er is geen sprake dat het geraakte object er anders door gaat bewegen, of het wordt vernietigd (zoals een spaceship) of er gebeurt niets (zoals bij een planeet).

Met deze achtergrond info definiëren we voor Flappy Bird twee object categoriën:

let birdObjectCategory = 1

let otherObjectCategory = 1 << 1

We maken dus alleen onderscheid tussen de vogel en andere objecten (pijpen en achtergrond), meer onderscheid is voor het spel niet relevant. NU voeren we de object categoriën in voor de verschillende sprites en stellen de contact en collision bitmasks in voor Flappy:

bird.physicsBody.categoryBitMask = birdObjectCategory

bird.physicsBody.contactTestBitMask = otherObjectCategory

bird.physicsBody.collisionBitMask = otherObjectCategory

background.physicsBody.categoryBitMask = otherObjectCategory

pipe1.physicsBody.categoryBitMask = otherObjectCategory

pipe2.physicsBody.categoryBitMask = otherObjectCategory

Het is niet nodig om ook voor de background en pipes de contact en collision bitmasks in te stellen, het is voldoende om geïnformeerd te worden wanneer Flappy met één van beide contact maakt, andersom is overbodig.

De *SKPhysicsContactDelegate* biedt een functie *didBeginContact* die wordt aangeroepen wanneer twee objecten contact maken en via de contact bitmask is aangegeven dat we dat willen weten:

func didBeginContact(contact: SKPhysicsContact!) {

println(“Contact”) // bij wijze van test

}

In ons geval hoeven we niet precies te weten wie er contact maakt met wie, de gevolgen zijn nl. hetzelfde. Vliegt Flappy tegen een pijp aan of valt hij op de grond dan is het spel over. We doen dus niets met de ‘*contacts’* parameter.

Run de app ter controle, wanneer Flappy de grond of een pijp raakt wordt daar melding van gegeven.

De volgende stap is om iets met de contact melding te doen: de achtergrond en de pijpen moeten stoppen met bewegen en de controle over Flappy wordt disabled. Voor dit laatste definiëren we een globale klassevariabele gameOver:

var gameOver:Bool = false

Voeg in *DidBeginContact* het volgende statement toe:

gameOver = true

Voeg in *touchesBegan* de volgende code toe:

if !gameOver {

gameOver = true

bird.physicsBody?.velocity = CGVectorMake(0, 0)

bird.physicsBody?.applyImpulse(CGVectorMake(0, 50))

}

Dit betekent dat alleen wanneer gameOver false is, de vogel bewogen kan worden door op het scherm te tappen. Run de app ter controle.

Wat opvalt is dat Flappy erg hard naar beneden valt. Hier kunnen we dankzij SpriteKit eenvoudig wat aan doen, door de ‘gravity’ instelling in te stellen. Dit is een globale instelling van de gameScene class:

self.physicsWorld.gravity = CGVector(0, -5)

De gravity is een vector, wij zetten de ‘dx’ op 0 zodat er geen horizontale zwaartekracht werkt op Flappy (de andere objecten zijn ongevoelig gemaakt voor de zwaartekracht). De ‘dy’ staat op -5, wat betekent dat de zwaartekracht naar beneden is gericht (want negatief) en ongeveer de helft is van wat normaal is, de default is 9,81 m/s2.

Het stoppen van de beweging van achtergrond en pijpen is wat lastiger. We hebben een referentie nodig naar beide, maar omdat de pijpen elke 3 seconden worden bijgemaakt weten we in principe niet welke er op enig moment bestaan. We kunnen dit oplossen door een nieuwe *node* te maken, die in feite een groep van nodes is. Alle objecten die we willen stilzetten voegen we toe aan deze node, die we vervolgens als geheel stilzetten. Definieer een globale klass variabele *movingObjects*:

var movingObjects = SKNode()

Aan het begin van didMoveToView, voeg deze node toe aan de scene:

self.addChild(movingObjects)

Vervolgens voegen we alle bewegende objecten toe aan movingObjects in plaats van direct aan de game scene. Vervang de self.addChild(xxx) statements voor de achtergrond en de pijpen door:

movingObjects.addChild(xxx)

Let op dat dit niet voor de grond en de lucht gebeurt. De snelheid van movingObjects zetten we op 0 in *didBeginContact*:

movingObjects.speed = 0

As simple as that. Voeg tenslotte nog in makePipes een check op de waarde van gameOver toe, zodat niet nodeloos pijpen worden gecreëerd als het spel is afgelopen.

Run de app ter controle. De code is te vinden in 11.6.

**Les 148 Scoring and game controls (en 149 voor code files)**

Wat nog ontbreekt aan het spel is het bijhouden van de score en de mogelijkheid overnieuw te beginnen bij een ‘game over’. Dat wordt in deze les ingebouwd.

Voor de score definiëren we een globale variabele score. Het presenteren van de score gaat mbv. een label:

var score = 0

var scoreLabel = SKLabelNode()

Met *SpriteKit* is het niet mogelijk om zelf labels op het storyboard te zetten, daarvoor is de *SKLabelNode* ontworpen. De score wordt verhoogd wanneer Flappy tussen de pijpen is gevlogen zonder deze te raken. Dat is niet te bepalen met de huidige nodes. Voor de ruimte tussen de pijpen creëren we een nieuwe node. Dit doen we in functie *makePipes*, direct na de creatie van de nodes:

var gap = SKNode()

gap.position = CGPoint(x: self.frame.width \*1.5, // rechts van het scherm!

y: self.frame.height/2 + pipeOffset)

gap.physicsBody = SKPhysicsBody(rectangleOfSize:

CGSizeMake(pipe1.size.width, gapHeight))

gap.runAction(moveAndRemovePipes)

gap.physicsBody.dynamic = false

movingObjects.addChild(gap)

Voor de positie van de gap gebruiken we de coördinaten van de pijpen. De X-coördinaat is identiek, de Y-coördinaat is het vertikale midden van het scherm plus de *pipeOffset*. De gap moet met de pijpen meebewegen en verwijderd worden als deze van het scherm verdwijnen. De zwaartekracht mag geen invloed uitoefenen op de gap.

Wat we nu willen bereiken is dat gedetecteerd wordt dat Flappy en de gap contact met elkaar maken, maar dat Flappy door de gap heen kan vliegen en dus niet terug stuitert. Hiervoor moet de collision bitmask van Flappy (de bird) en de gap gelijk aan elkaar zijn, zo beweert Rob. Dit klinkt heel vreemd, de Apple docs stellen dat wanneer twee objecten contact met elkaar maken, en AND operatie wordt uitgevoerd op de collision bit mask van de eerste en de category bit mask van de tweede. Is de uitkomst positief dan heeft de botsing impact op het eerste object, anders niet. De code van Rob werkt alleen maar omdat hij de gapObjectCategory op 0 zet (0 << 3 is nul), waardoor de collision bitmasks van bird en gap beide 0 zijn, hierdoor wordt er niets gedaan als beide met elkaar botsen.

Definieer een gapObjectCategory voor de gap en initialiseer gap daarmee:

let gapObjectCategory = 1 << 2 // aan begin van *didMoveToView*

gap.physicsBody?.caterogyBitMask = gapObjectCategory

Stel de collision bit masks in van de bird en de gap:

bird.physicsBody?.collisionBitMask = gapObjectCategory

gap.physicsBody?.collisionBitMask = gapObjectCategory

En stel tot slot de contact bit mask van gap in op de birdObjectCategory, odat we geïnformeerd worden wanneer Flappy de gap raakt.

NB Omgekeerd kan ook, als één van beide objecten maar het andere object in zijn *contact bit mask* heeft opgenomen.

De functie *didBeginContact* moet aangepast worden, omdat er nu meerdere contact mogelijkheden zijn moeten we testen of Flappy de gap raakt of dat hij tegen een pijp aan vliegt of de grond raakt:

if contact.bodyA.categoryBitMask == birdObjectCategory ||

contact.bodyB.categoryBitMask == gapObjectCategory {

score ++

} else {

gameOver = true

movingObjects.speed = 0

}

Het *scoreLabel* moet nog worden geconfigureerd en bijgewerkt. De configuratie doen we in *didMoveToView*:

scoreLabel.fontName = “Helvetica”

scoreLabel.fontSize = 60

scoreLabel.text = “0” // De initiële score

scoreLabel.position = CGPointMake(self.frame.size.width/2,

self.frame.size.height-70)

scoreLabel.zPosition = 10

self.addChild(scoreLabel)

Het label staat midden bovenin het scherm, de hoogte is zo gekozen dat er nog 10 punten over blijven tot aan de bovenrand.

Werk *scoreLabel* bij in didBeginContact:

scoreLabel = “(\score)

Run de app ter controle. Het werkt, maar ik zie de score niet in landscape mode.

Tot slot moet het nog mogelijk worden gemaakt om het spel te herstarten na een ‘game over’. De constatering van dat het spel is afgelopen gebeurt in didBeginContact. Daar laten we als eerste een ‘game over’ label zien. Definieer het gameOverLabel aan het begin van didMoveToView:

var gameOverLabel = SKNode()

En voeg de volgende code toe na de ‘gameOver = true’ in didBeginContact:

scoreLabel.fontName = “Helvetica”

scoreLabel.fontSize = 30

scoreLabel.text = “Game over! Tap to play again”

scoreLabel.position = CGPointMake(self.frame.size.width/2,

self.frame.size.height/2)

scoreLabel.zPosition = 10

self.addChild(scoreLabel)

Wanneer op het scherm wordt getapt moet het spel opnieuw starten. Dit handelen we af in touchesBegan, als gameOver true is, dan betekent dit dat na een ‘game over’ op het scherm getapt is:

if (!gameOver) {

// bestaande code

} else {

gameOver = false

score = 0

scoreLabel.text = “0”

movingObjects.removeAllChildren()

makeBackground()

movingObjects.speed = 1

bird.position = CGPoint(x:self.frame.width/2,

y:self.frame.height/2)

bird.velocity = CGVectorMake(0, 0)

}

Alle objecten die zijn ondergebracht in *movingObjects* worden verwijderd, naast de pijpen zit de *background* daar echter ook bij en die moet bij een nieuw spel dus opnieuw worden gecreëerd. Omdat we de code voor het configureren van *background* niet opnieuw helemaal willen uitschrijven, wordt dit in een functie *makeBackground* ondergebracht.

De snelheid van movingObjects was bij de ’game over’ op 0 gezet. Deze moeten we terugzetten op 1, anders beweegt de achtergrond niet bij opnieuw opstarten.

Flappy wordt weer op zijn initële positie neergezet. Zijn snelheid moet op 0 gezet worden. Stel dat Flappy op de grond was gevallen dan heeft hij nog steeds die vertikale snelheid. Wordt de game daarmee opnieuw opgestart dan valt Flappy vrijwel direct op de grond.

Het gameOverLabel moet nog van het scherm gehaald worden. Rob stelt dat we deze niet zomaar kunnen verwijderen en daarvoor een globale ‘label holder’ gereëerd moet worden, waarin het gameOverLabel wordt opgenomen. Deze werkt identiek aan movingObjects:

var labelHolder: SKSpriteNode() // globale klasse variabele

De labelHolder wordt aan het begin van didMoveToView toegevoegd:

self.addChild(labelHolder)

Het label zelf wordt in didBeginContact bij een ‘game over’ aan labelHolder toegevoegd (in plaats van aan de view):

labelHolder.addChild(gameOverLabel)

Ik heb een eenvoudiger oplossing. Het gameOverLabel is globaal gedefinieerd en kan eenvoudig van de view weggehaald worden;

self.removeChildrenInArray([gameOverLabel])

Dit blijkt prima te werken. Run de app ter controle. Wanneer Flappy op de grond valt kan het spel opnieuw opgestart worden. Vliegt hij echter tegen een pijp dan volgt een runtime error. Dit komt omdat er het contact van Flappy met de pijp twee keer wordt doorgegeven, één keer vanuit Flappy bekeken, en één keer vanuit de pijp bekeken. Het punt is dat hierdoor in *didBeginContact* het *gameOverLabel* twee keer wordt gecreëerd, omdat er niet wordt gekeken naar de actuele waarde van *gameOver*. Door een ‘if !gameOver’ statement om de code te zetten wordt dit verholpen.

De code is te vinden in 11.7.

Er zijn diverse verbeteringen en uitbreidingen mogelijk. Denk aan het variëren van de snelheid, de gap grootte, de afstand tussen de pijpen, bonus objecten, een opstartscherm, een knop om het spel te starten zodat het niet direct begint, het opslaan van de scores lokaal en/of in de cloud voor meerdere gebruikers, het toevoegen van muziek, etc. etc.

NB Nog een handige tip voor het debuggen van physics. Voeg in *GameViewController.swift* de volgende regel toe ná de instelling van *showsNodeCount*:

skView.showsPhysics = true

Dan zullen de contouren van alle fysieke objecten worden getoond.

**Sectie 11 – Snapchat clone**

**Les 150: Creating a user table (en 151 voor code files)**

In deze les wordt gestart met de bouw van een Snapchat kloon. Hiermee kunnen foto’s naar andere aangesloten Snapchat gebruikers gestuurd worden die kort zichtbaar zijn op het scherm en dan definitief verdwijnen.

Rob gebruikt hiervoor een premature versie van Tinder waarvan het storyboard nog leeg is.

Creëer als eerste een nieuwe applicatie op Parse.com. Ga hiervoor naar het *Dashboard*, noem de app Snapchat en kopieer de *Application ID* en *client ID* uit de lijst met keys die verschijnt naar *AppDelegate.swift*, naar de call naar *Parse.setApplicationId*.

Vervolgens wordt de login gebouwd. Dit gebeurt in de les wel heel simpel: het invoeren van een bestaande user id is voldoende om in te loggen. Bestaat de opgegeven user id niet, dan volgt een aanmeld proces. Het handigst is om dit vanuit de Tinder app te kopiëren, aangezien het daar minder simplistisch wordt opgezet dan hier.

Voeg op het storyboard een *label* toe voor de titel (‘Snapchat’), een *label* ‘Enter a username…’, een *text field* ‘username’ en een *button* ‘sign in’. Creëer in de main viewcontroller een *outlet* ‘*username’* voor het text field en een *action* ‘*signIn’* voor de button.

Voor de login gebruiken we standaard code op Parse.com, zie hiervoor in de *Docs* sectie onder *Users – Signing Up*. We gebruiken de functie *loginUsernameInBackground*, met “mypass” als default password omdat we geen password gebruiken, maar dit had natuurlijk net zo makkelijk met een extra password field gebouwd kunnen worden. De eerste opzet voor de login code in *signIn* is als volgt:

PFUser.logInWithUsernameInBackground(username.text, password: “mypass”) {

(user: PFUser!, error: NSError!) -> Void in

if user != nil {

// gebruiker is ingelogd: redirect naar de user tabel, volgt nog

println(“logged in”)

} else {

println(error)

// meld aan als nieuwe gebruiker

}

}

De code voor het aanmelden is als volgt:

var user = PFUser()

user.username = self.username.text //self omdat code in closure zit

user.password = “mypass”

user.signUpInBackgroundWithBlock {

(succeeded: Bool!, error: NSError!) -> Void in

if error == nil {

// Nieuwe gebruiker aangemeld en geregistreerd op Parse.com

println(“signed up”)

} else {

println(error) // er is nu echt iets fout gegaan

}

}

Run de app ter controle. Het handige van deze manier van inloggen is dat we gemakkelijk testgebruikers kunnen aanmaken, in plaats van een apart aanmeld scherm zoals voor de Tinder app.

De volgende stap is de bouw van de user tabel. Creëer op het storyboard een *table viewcontroller*. Creeer een nieuwe klasse “*UserTableViewController*” (subclass van *UITableViewController*) die gebruikt wordt om de table viewcontroller te beheren. Vul in het storyboard de *Custom Class* in van de table viewcontroller. Creëer in het storyboard een segue van de initiële viewcontroller naar de table viewcontroller, noem de segue ‘showUserTable’ (bij *Identifier* in de *Attribute Inspector*).

Voeg in signIn direct na de ‘*if user != nil’* (= na succesvol inloggen) de segue toe:

self.performSegue(“showUserTable”, sender: self)

Doe dit ook na de ‘*if error == nil’* (= na succesvol aanmelden).

Als de gebruiker al was ingelogd dan willen we dat de user tabel direct verschijnt. Dit regelen we in *viewDidAppear.* Het kan niet in viewDidLoad omdat een segue pas kan worden uitgevoerd als de view waarvandaan dat gebeurt op het scherm moet staan. Voeg een override voor viewDidAppear toe:

if PFUser.currentUser() != nil {

self.performSegueWithIdentifer(“showUserTable”, sender: self)

}

We kunnen nu de user tabel verder configureren. Zet een navigation bar bovenaan de table viewcontroller, geef het de titel ‘Snapchat users’ oid.

In viewDidLoad in UserTableViewController.swift gaan we de Snapchat gebruikers van Parse.com ophalen. Haal de standaard code hiervoor weer op van Parse.com, deze staat in de Docs sectie onder Users – Querying. Voeg de volgende code toe aan viewDidLoad:

var query = PFUser.query()

query.whereKey(“username”, notEqualTo: PFUser.currentUser())

var users = query.findObjects()

for user in users {

println(user.username) // bij wijze van test

}

De query levert alle Snapchat gebruikers op behalve de huidige ingelogde gebruiker zelf. Run de app, alle gebruikers behalve jij zelf worden in de console log afgedrukt. Merk op dat XCode een warning geeft ‘a long running operation is being executed on the main thread’. Dit komt omdat we nu de synchrone variant van findObjects gebruiken. In tegenstelling tot de asynchrone variant findObjectsInBackgroundWithBlock wordt nu op het resultaat van de query gewacht. In dit geval niet erg omdat we de tabel pas kunnen vullen als de gebruikers geladen zijn.

Voor de user tabel creëren we een globale klassevariabele in UserTableViewController:

var userArray:[String] = [] // initieel leeg

Voeg na de ‘println(user.username)’ het volgende statement toe:

self.userArray.append(user.username) // selt, want in closure

We kunnen nu de table viewcontroller functies implementeren:

* numberOfSectionsInTableView:

return 1

* numberOfRowsInSection:

return userArray.count

* cellForRowAtIndexPath:

let cell = … // code overnemen, pas de reusable identifier aan: “cell”

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

Pas de identifier van de table view cell in het storyboard ook aan: “cell”.

Run de app ter controle. De gebruikers verschijnen, in de volgorde waarin ze zijn ingevoerd. De volgorde kan aangepast worden door dit in de query op te geven, zie daarvoor de documentatie op Parse.com.

De code is te vinden in 11.1a (let op !).

**Les 152: Sending images (en 153 voor code files)**

In deze les wordt het versturen van een foto naar een andere Snapchat gebruiker behandeld. Dit is de gebruiker die uit de tabel wordt geselecteerd. Het versturen wordt gerealiseerd door een upload naar Parse.com.

Voor de selectie van een foto c.q. het maken van een foto met de camera gebruiken we de code van de Instagram app en wel de functies *imagePickerController*-*didFinishPickingImage* en *pickImage*. Hiervoor moeten de *UINavigationControllerDelegate* en *UIImagePickerControllerDelegate* toegevoegd worden aan de viewcontroller.

Voeg de twee functies toe aan UserTableViewController.swift:

* *imagePickerController-didFinishPickingImage*:

println(“image selected”)

self.dismissViewControllerAnimated(true, completion: nil)

// voeg hier de code voor de upload naar Parse.com toe

* *pickImage*:

var image = UIImagePickerController()

image.delegate = self

image.sourceType = UIImagePickerControllerSourceType.PhotoLibrary

image.allowsEditing = false

self.presentViewController(image, animated: true, completion: nil)

Omdat we de simulator gebruiken worden de foto’s uit de photo library geselecteerd. Op de iPhone kan de camera gebruikt worden.

De *pickImage* functie moet worden aangeroepen als een gebruiker uit de user tabel wordt geselecteerd. Hiervoor implementeren we de functie *didSelectRowAtIndexPath*:

pickImage(self)

activeRecipient = indexPath.row

De variabele activeRecipient is een globale klasse variabele die de index van de geselecteerde gebruiker vasthoudt.

Run de app ter controle dat een foto geselecteerd kan worden wanneer een gebruiker is geselecteerd.

De volgende stap is de upload van de geselecteerde foto naar Parse.com. Gebruik hiervoor de standaard code van Parse.com in de Docs sectie Objects – Saving Objects en maak deze op maat in *imagePickerController-didFinishPickingImage*:

var imageToSend = PFObject(className: “image”)

imageToSend[“image”] = UIImageJPEGRepresentation(image, 0.5)

imageToSend[“senderUsername”] = PFUser.currentUser()

imageToSend[“recipientUsername”] = userArray[activeRecipient ]

imageToSend.save()

Deze code creëert automatisch een nieuwe tabel “*image*” mocht deze nog niet bestaan. De foto wordt als jpeg opgeslagen op Parse.com. Als compressie ratio wordt 0.5 gebruikt, omdat foto’s erg groot kunnen worden kan het wellicht nog minder (bijv. 0.01). De *save()* functie wordt gebruikt voor de opslag van de foto. Het is wellicht raadzaam om *saveInBackgroundWithBlock* te gebruiken en in het block een alert of label te laten zien.

Run de app ter controle. Rob loopt bij de eerste foto tegen en probleem aan: de foto is groter dan 128 kB en wordt niet opgeslagen. Hij heeft hiervoor een aardige workaround. Door de foto in een file op te sturen is de limiet veel groter. Hij maakt hiervoor een nieuwe key ‘photo’ aan:

imageToSend[“photo”] = PFFile(name: “image.jpg”,

data: UIImageJPEGRepresentation(image, 0.5))

Run de app ter controle. De foto wordt nu wel opgeslagen, verifieer dit op Parse.com.

De code is te vinden in 11.2a.

**Les 154: Receiving images (en 155 voor code files)**

In deze les wordt de ontvangst van een foto door een Snapchat gebruiker geïmplementeerd. Dit gebeurt door te checken of een foto is ontvangen en een alert naar de betreffende gebruiker te sturen. Om dit te testen voegen we een logout button toe, zodat we de foto vanuit het andere account kunnen ontvangen.

Creëer een *bar button item ‘logout’* in de navigation bar van de table viewcontroller. Rob creëert vervolgens een *segue* ‘logout’ van deze logout button naar de initiële viewcontroller. Dit had natuurlijk ook met een *action outlet* gekund, maar dit is weer wat anders. Deze keuze heeft tot gevolg dat de functie *prepareForSegue* overridden moet worden in de *UserTableViewController* klasse:

override func prepareForSegue(segue: UIStoryBoardSegue, sender: AnyObject?) {

if segue.identifier == ‘logout’ {

PFUser.logout()

}

}

Voor het checken of een foto is ontvangen maakt Rob gebruik van een timer. Elke x seconden/minuten wordt dan gecheckt of er een foto is ontvangen. De timer wordt gecreëerd in de *viewDidLoad* van de *UserTableViewController*:

timer = NSTimer.scheduledTimerWithTImeInterval(5, target: self,

selector: Selector(“checkForMessages”), userinfo: nil, repeats: true)

NB timer is een globale klasse variabele:

var timer = NSTimer()

Creëer de functie checkForMessages:

func checkForMessages() {

println(“Checking for messages…”)

}

Voeg nu een query toe aan *checkForMessages* om na te gaan of er voor de ingelogde gebruiker nog ongelezen berichten zijn. Wordt iets gevonden dan is het per definitie nog niet gelezen, anders zou het weg zijn gegooid:

var query = PFQuery(className: “image”)

query.whereKey(“recipientUsername”, equalTo: PFUser.currentUser().username)

query.findObjectsInBackgroundWithBlock( {

(results:[AnyObject]!, error: NSError) -> Void in

if results.count > 0 {

println(“You have messages waiting…”) // bij wijze van test

// download foto’s één voor één en zet op scherm

}

})

In tegenstelling tot Rob gebruik ik een asynchrone query, wat aan te bevelen is. Run de app ter controle.

De volgende stap is het downloaden en bekijken van de foto’s totdat ze allemaal bekeken zijn. Rob implementeert dit door de foto’s in een loop te downloaden en vol op het scherm te zetten, over de user tabel heen. Parse.com biedt de klasse *PFImageView* voor het beheren van op Parse.com opgeslagen foto’s en plaatjes, wat . De code wordt dan als volgt:

for image in results {

var imageView:PFImageView = PFImageview()

imageView.file = image[“photo”] as PFFile

imageView.loadInBackground( { (photo, error: NSError!) -> Void in

if error == nil {

var displayedImage =

UIImageView(frame: CGRectMake(0, 0,

self.view.frame.width, self.view.frame.height))

displayedImage.image = photo

displayedImage.contentMode =

UIViewContentMode.ScaleAspectFit

self.view.addSubview(displayedImage)

} else {

println(“Error downloading image \(error)”)

}

})

}

NB. Ik krijg zoals veel anderen een syntax error op *PFImageView*. Dit kan opgelost worden door het *ParseUI* framework toe te voegen aan het project en het statement ‘*import ParseUI’* toe te voegen. Het toevoegen van de headerfile in de bridging header werkt bij mij niet.

Run de app ter controle. De foto’s – zo die er zijn – worden zo groot mogelijk, maar wel passend (*ScaleAspectFit*) op het scherm afgebeeld. Een aandachtspunt is dat dit gewoon over de tabel heen gaat, waarbij interactie met de tabel gewoon mogelijk blijft. Het is beter om dit te vermijden. Dit lost Rob op door een achtergrond view over de tabel heen te leggen, met een *alpha* waarde die zo groot is dat de tabel donker wordt maar nog net zichtbaar is. Doordat de view over de user tabel heen ligt, is er geen interactie mogelijk met de user tabel. Over de achtergrond view wordt vervolgens de foto getoond. Voeg na de ‘*if error == nil’* de volgende code toe:

var backgroundView =

UIImageView(frame: CGRectMake(0, 0,

self.view.frame.width, self.view.frame.height))

backgroundView.backgroundColor = UIColor.blackColor()

backgroundView.alpha = 0.8

self.view.addSUbView(backgroundView)

Run de app ter controle, de achtergrond is nu donker. Maar ik kan nog steeds uitloggen, dus de logout knop is toch nog beschikbaar, vreemd… Erg mooi is het allemaal nog niet, alles wordt over elkaar heen gekwakt. Dit verbeteren we door een foto 5 seconden te tonen. Hiervoor gebruiken we weer een *timer*. Voeg deze na de weergave van een foto toe:

self.timer = NSTimer.scheduledTimerWithTimeInterval(5, target: self,

selector: Selector(“hideMessage”), userInfo: nil, repeats: false)

Functie *hideMessage* moet de achtergrond view en de foto van het scherm afhalen. Rob lost dit op door beide views dezelfde tag id te geven nl. 3 (een willekeurig getal) en deze met één statement van het scherm te halen. In *checkForMessages*:

backgroundView.tag = 3

displayedImage.tag = 3

Functie *hideMessage*:

for subViews in self.view.subViews {

if subView.tag == 3 {

subView.removeFromSuperview()

}

}

Run de app ter controle. Bij mij ziet het er nog steeds niet uit, als er meerdere plaatjes klaar staan dan worden die over elkaar heen getoond. Begrijp niet waarom, zoek ik nog uit.

Om het af te maken laten we voordat de foto’s getoond worden een alert zien en laten we per foto zien wie deze heeft getuurd. De alert laten we zien nadat de foto’s zijn gedownload, dus ná de *loadInBackground*:

var senderUsername = image[“senderUsername”]

var alert = UIAlertController(title: “You have a message”,

message: “Message from \(senderUsername)”,

preferredStyle: UIAlertControllerStyle.Alert)

alert.addAction(UIAlertAction(title: “OK”, style: UIAlertActionStyle.Default,

handler: {

// voeg in dit blok de code voor tonen van achtergrond en foto

}))

self.presentViewController(alert, animated: true, completion: nil)

Run de app ter controle. De alert verschijnt, na drukken op OK verschijnt een foto. Een aantal punten zijn nog niet in orde:

* in de alert verschijnt “Message from (Optional)<username>”
* de alert blijft maar verschijnen

Het eerste punt is eenvoudig te verhelpen door te testen of ‘image[“senderUsername”] gelijk is aan nil voordat de assignment plaatsvindt:

var userName:String = “unknown Snapchat user”

if image[“senderUsername”] != nil {

senderUsername = image[“senderUsername”]

}

Het tweede punt wordt verholpen door de foto na vertonen van Parse.com te verwijderen.

De code is te vinden in 11.3a.

**Section 12 – Submitting your app to the App Store**

**Les 156: App icons and launch images**

In deze les worden alle nodige launch images en app icons gecreëerd. Rob doet dit voor de CatYears app die eerder gemaakt is. Hij heeft hiervoor een afbeelding beschikbaar die als basis voor de diverse launch images en app icons worden gebruikt.

Apple geeft op developer.apple.com de nodige richtlijnen voor launch images en app icons. Zo moet bijv. voor de iPad een launch image met een resolutie van 2048x1536 (landscape) en één met een resolutie van 1536x2048 (portrait) worden gemaakt. Rob gebruikt Adobe Premiere CS hiervoor, ik heb op zijn advies Gimp (gratis) geïnstalleerd en dat lijkt het prima te doen. De basis zit in het aanpassen van de canvas grootte en het schalen van de foto. Bij het schalen wordt de foto als geheel in meer of minder pixels gestopt, bij het aanpassen van de canvas grootte wordt een deel weggesneden.

Wanneer de nodige launch images en app icons gereed zijn kunnen ze in het project ingebracht worden. Klik in XCode in de project explorer op images.xassets en selecteer Launch Image. In het scherm rechts hiervan verschijnen frames voor alle mogelijke launch images die in het project ingebracht moeten worden. Aard en aantal van deze frames worden bepaald door a)voor welk device de app bedoeld is (iPhone en/of iPad) en b)of de app in portrait en/of landscape mode kan opstarten.

Selecteer de afbeeldingen naar de bijbehorende launch image frames en doe dit evenzo voor de app icons.

**Les 157: Screenshots**

Om een app in de App Store te krijgen moeten er screenshots van mee worden geleverd. Liefst zoveel mogelijk zodat duidelijk is wat de app allemaal kan.

Op de iPhone en iPad kan een screenshot worden gemaakt door de Power en Home button tegelijk in te drukken. Op de Simulator kan dit door CMD-S in te drukken of in menu File – Save screen shot te selecteren.

**Les 158: Submitting your app (en 159 voor further reading)**

Hiervoor heb je een betaald iOS developer account nodig. Log in op de Apple developer site en ga naar itunesconnect.apple.com, daar kun je al je apps beheren.

Voor het inzenden van een app is een provisioning profile nodig. Hiermee kan alleen jij apps in de Appstore publiceren. Ga hiervoor terug naar de Apple Developer Center op developer.apple.com en log daar in met je Apple Developer account.

Een certificate signing request is nodig, waarmee een certificaat kan worden aangemaakt voor de computers die je gebruikt. Dit gaat mbv. het Keychain Access programma. Selecteer hier in het menu Keychain Access – Certificate Assistant – Create a Certificate from a Certificate Authority. Vul in het volgende scherm je e-mail adres, een naam voor het certificaat (bijv. Frank de Groot – iOS key), laat CA Email Address leeg en selecteer Saved to disk. Sla de certificate request op in een folder naar keuze.

Ga op de Apple Developer site verder met de aanmaak van het provisioning profile. Laad het zojuist gegenereerde certificate request en klik Generate. Download het gegenereerde certificaat naar een folder naar keuze. Dubbelklik in de Finder op het zojuist gegenereerde certificaat, waardoorhet certificaat wordt aan Keychain Access toegevoegd. Klik op Done en het certificaat verschijnt in het volgende overzicht van gegenereerde certificaten.

Selecteer links in hetzelfde scherm Provisioning Profiles – Distribution en klik op het + teken rechts en selecteer dan App Store onder Distribution. Ad Hoc kan ook als je eerst nog op een aantal devices wil testen. In het volgende scherm wordt de App Id gevraagd. Als je geen zaken als push notifications gebruikt kan volstaan worden met een *wildcard id.* Het provisioning profile dat gecreëerd wordt kan dan ook voor andere iOS apps gebruikt worden die geen specifieke zaken gebruiken zoals push notifications, in-app purchases en Game Center. Heb je dergelijke zaken wel nodig dan kan in het menu links op het scherm een App Id daarvoor worden aangemaakt dat op dit scherm dan uit de dropdown box kan worden geselecteerd.

In het volgende scherm moet het certificaat dat zojuist is gecreëerd worden geselecteerd.

Vul in het volgende scherm een naam voor het provisioning profile in, bijv. ‘Franks iOS Provisioning Profile’. Klik op OK en het provisioning profile wordt gegenereerd en kan worden gedownload, het wordt opgeslagen met de zojuist ingegeven naam, met extensie ‘.*mobileprovision’*. Dubbelklik in de Finder op deze file en selecteer ‘*Add to Library’*. In de video verschijnt het in de iPhone Configuration Utility, echter wordt deze niet meer gebruikt, zie hiervoor in de betreffende Instagram les.

Ga nu weer naar iTunes Connect en selecteer *Manage Your Apps* en selecteer . Vul nu de Appname in, deze moet uniek zijn voor de AppStore! Vul bij *SKU Number* de bundle identifier in die in XCode voor de app is te vinden. Selecteer bij *Bundle Id* de wildcard app id in die eerder is geselecteerd. Vul bij *Bundle Id Suffix* het *SKU Number* in, deze moeten nl. identiek zijn.

In het volgende scherm worden de beschikbaarheidsdatum en prijsstelling ingegeven. Daarna volgt een scherm met *Version Number* (moet overeen komen met die in XCode voor deze app), *Copyright*, *Primary* en *Secondary Category* en een lijst met *Rating* indicatoren. Vul in Metadata een zo goed mogelijke en ‘selling’ beschrijving in van de app. Vul bij *Review Notes* zaken in die de Apple reviewer nodig heeft, zoals login gegevens.

Een stukje verder moeten de screenshots voor de diverse iPhone en iPad formaten opgegeven worden. Er wordt automatisch gecheckt of het opgegeven screenshot formaat past bij het betreffende device formaat. Klik dan op *Save* en in het volgende scherm op *View Details* en klik daarna op *Ready to Upload Binary*. Beantwoord de 3 2-keuze vragen en klik Save. Klik vervolgens door tot een scherm verschijnt waarin de Status op ‘Waiting for Upload’ is.

Ga hiervoor terug naar XCode en zet het target device op iOS device, bijna linksboven op het scherm. Selecteer nu uit het Product Menu de optie Archive. Er verschijnt nu een melding dat er geen signing identity is gevonden. Klik dan op Fix Issue waarna een lijst met beschikbare accounts verschijnt. Selecteer de meest voor de hand liggende (is voor mij niet duidelijk). Vervolgens verschijnt mogelijk de melding dat geen provisioning profiles zijn gevonden, klik hier ook op Fix Issue. Onze app wordt vervolgens gearchiveerd waarna het Organizer scherm verschijnt. Dit kan in XCode ook vanuit het Windows menu – Organizer worden opgeroepen, waarin de lijst met gearchiveerde apps zijn opgenomen. Klik hier op Submit… en selecteer in de pop-up het Development team.

De les stopt hier omdat ten tijde van het maken van de les met XCode 6 Beta werd gewerkt waarmee het nog niet mogelijk was om de app in de App Store te krijgen. De melding ;No matching provisioning profiles found for xxx.app”.

In een nieuwe versie van XCode kan dit wel en verschijnt het iTunes Connect login scherm, waarna de app geselecteerd moet worden voor de upload. Apple heeft vervolgens ca. 10 dagen nodig voor de controle van de app waarvan via e-mail bericht zal verschijnen.

**Section 13 – Creating a website for your app**

**Les 160: Installing Wordpress**

In deze sectie wordt een website gecreëerd voor de lancering van de app. Hiervoor gebruikt Rob een thema van appz.themeshaker.com. De website zelf gaat hij met het CMS WordPress bouwen.

Rob gebruikt EcoWebhosting als provider voor de website, ik heb daar een gratis account gecreëerd dat 1 jaar gratis blijft. Log daar in en vul in het volgende scherm jouw domain name in. Deze moet uniek zijn! Is dat het geval dan wordt een bevestigingsmail gestuurd en wordt het hosting package voor je gecreëerd. Dit duurt ca. 1 uur, dus tijd voor koffie.

Selecteer een uur later ‘View hosting passwords’ links op het scherm. Klik vervolgens op de opgegeven domain waar je automatisch wordt ingelogd. Het Control Panel van jouw website verschijnt nu. Alle onderdelen op dit scherm zijn een jaar lang gratis beschikbaar.

Scroll een stuk naar beneden en selecteer ‘Word Press Blog’. Dit gaan we installeren. Als je eigenaar bent van het opgegeven domain, dan kun je dat later gebruiken, anders zul je het met het opgegeven IP adres moeten doen. Klik op Install No w. Een melding dat het CGI script is geïnstalleerd verschijnt nu. Klik op de link die zeker werkt – die met het IP adres – voor verder installatie van WordPress. Selecteer ‘Single’ in het volgende scherm. Vul in het volgende scherm de Site Title, Username (zelf te kiezen), Password en E-mail adres in en klik op Install WordPress.

Na installatie verschijnt het WordPress login scherm. Log daar met de zouist ingegeven credentials in. Het WordPress Dashboard verschijnt nu waarmee de website kan worden gebouwd. Dat gebeurt in de volgende les.

**Les 161 – Setting up your theme**

**Sectie FdG – Overige onderwerpen**

**1. Integratie GooglePlus**

Op de Google Developers site wordt uitgelegd hoe Google Plus kan worden geïntegreerd in een iOS app, zie <https://developers.google.com/+/mobile/ios>.

Op Medium.com is een uitgewerkt voorbeeld hiervan te vinden: <https://medium.com/swift-programming/google-in-swift-a0943352dfe9>.

In beide gevallen wordt vermeld dat in de Build Settings bij Other Linker Flags de optie ‘-ObjC’ moet worden toegevoegd. Deze geeft echter onresolved symbols voor het Parse framework. Weglaten is echter geen optie omdat het GooglePlus framework dan niet werkt. De oplossing is om 2 opties toe te voegen:

* -force\_load GooglePlus.framework/GooglePlus
* -force\_load GoogleOpenSource.framework/GoogleOpenSource

De path referentie start bij de .proj file, in dit voorbeeld staan beide frameworks dus in de projectdirectory.

Ik heb googlePlus uiteindelijk niet geintegreerd omdat het niet lekker samen wilde met Parse. Er waren een aantal tegenstrijdige acties nodig. Omdat Parse makkelijker in gebruik is, heb ik besloten af te zien van GooglePlus.

**2. Unwind segues**

Een unwind segue is een segue waarmee kan worden teruggesprongen in de view controller hierarchie. De viewcontroller waarvandaan wordt gesprongen wordt uit de viewcontroller hiërarchie verwijderd, waardoor het geheugen wordt vrijgegeven. In een app waar alle kanten op kan worden genavigeerd is dit de manier om te implementeren. Een perform segue stapelt de nieuwe viewcontroller nl. op de reeds actieve viewcontrollers, waardoor op een gegeven moment het geheugen op is.

De implementatie is als volgt. Stel dat button A is opgenomen in viewcontroller A. Als op deze button wordt gedrukt moet worden teruggesprongen naar viewcontroller B, welke viewcontroller A heeft geïnitieerd (dat is wel de voorwaarde).

Start met het creëren van een action item in de viewcontroller B waarnaar je terug wilt springen, bijv:

@IBAction func unwindToViewControllerA(segue: UIStoryboardSegue) {

// code die vóór het terugspringen wordt uitgevoerd

}

Rechtsklik in het storyboard op button A en sleep naar de Exit knop van viewcontroller A waar deze button in is opgenomen (de rechter knop boven in viewcontroller A). Er verschijnt nu een menu waarin o.a. de functie *unwindToViewControllerA* is opgenomen. Selecteer deze en klaar is Kees.

Functie *unwindToViewControllerA* wordt aangeroepen nog voordat de segue wordt uitgevoerd, zodat nog kan worden gerefereerd aan de viewcontroller waarvandaan wordt gesprongen. Dit maakt het mogelijk om afsluitende acties uit te voeren.

De viewcontroller wordt automatisch dismissed door Cocoa, in diverse voorbeelden wordt dit expliciet in de code gedaan maar dit lijkt alleen in iOS 8.0 nodig te zijn, lijkt op een bug.

**3. View animatie met Auto Layout**

Doel is om een menu met buttons bij een tap op het scherm te laten verschijnen met een animatie: het menu is initieel onzichtbaar en verschijnt binnen een 0,5 seconde onderaan op het scherm. Bij nogmaals tappen buiten het menu moet het menu weer naar beneden zakken.

Hier is redelijk veel over te vinden op internet. Het enige dat voor mij goed werkte was om de Auto Layout constraint aan te passen en vervolgens in een animatie de layout aanpassing te forceren.

In geval van dit menu moet een outlet voor de bottom constraint gemaakt worden. In *viewWillAppear* wordt de waarde van de constraint op de negatieve hoogte van de view gezet, zodat de view geheel onder het scherm verdwijnt:

portraitMenuViewBottomConstraint.constant =

-self.portraitMenuView.bounds.height

Wordt op de map getapt dan wordt het menu getoond, afhankelijk van de oriëntatie van het device wordt het portrait danwel het landscape menu getoond met een animatie:

func checkTap(gestureRecognizer: UIGestureRecognizer) {

// Toon het menu afhankelijk van de orientatie van het scherm

if UIDeviceOrientationIsLandscape(UIDevice.currentDevice().orientation) {

// Het landscape menu wordt getoond danwel verborgen door de bottom

// constraint aan te passen

self.landscapeMenuViewRightEdgeConstraint.constant =

(menuIsVisible ? -self.landscapeMenuView.bounds.width-16 : -16)

UIView.animateWithDuration(0.5, animations: {

self.view.layoutIfNeeded()

})

} else {

// Niet landscape interpreteren we als portrait, hoewel er in principe nog

// meer mogelijkheden zijn (bijv. unknown)

// Het portrait menu wordt getoond danwel verborgen door de bottom

// constraint aan te passen

self.portraitMenuViewBottomConstraint.constant =

(menuIsVisible ? -self.portraitMenuView.bounds.height : 0)

UIView.animateWithDuration(0.5, animations: {

self.view.layoutIfNeeded()

})

}

menuIsVisible = !menuIsVisible

}