Frauke Kling
2186427
GitHub: Tranquilfeli
22.09.2016

Dokumentation

Music Player with mood-based playlists

Einleitung

Im Rahmen des Projektseminars „angewandte Informationswissenschaft“ habe ich eine iOS 10 App entwickelt.
Ziel war es, einen Musikplayer zu entwickeln, welcher den Nutzer mit seiner im iPhone abgespeicherten Musikbibliothek konfrontiert und ihm dann die Option gibt, seine Songs per Button in Playlists zu sortieren. Der Musikplayer sollte einfach in der Anwendung und schlicht im Design sein, um eine möglichst angenehme Nutzung zu garantieren.

Genutzte Sprachen und Frameworks

Die App wurde in Swift 3 geschrieben und mit Hilfe von XCode 8 umgesetzt. Swift ist eine recht junge, im Jahr 2014 von Apple veröffentlichte Programmiersprache, welche von Objective C abgeleitet wurde und auf die Apple Betriebsysteme zugeschnitten ist ("The Swift Programming Language (Swift 3): About Swift", 2016).
Um auf die im iPhone vorhandene Musikbibliothek zugreifen zu können, wurde das MediaPlayer Framework genutzt. Dies ist ein von Apple zur Verfügung gestelltes Framework, das die Arbeit mit Liedern und Videos vereinfacht.
Das Framework ist die einzige Möglichkeit, auf den systeminternen iPod der Geräte zuzugreifen ("MediaPlayer | Apple Developer Documentation", 2016).

Projektaufbau

Das Storyboard:

Die Entwicklung der UI wurde mit Hilfe des in XCode implementierten Storyboards umgesetzt. Dabei wurde ein schlichtes, funktionales Interface erarbeitet.
Es beinhaltet zwei *UIViewController,* die erste Startview (ViewController) und die zweite View zum Abspielen der Playlists (Playlist Controller).
Ein solcher ViewController wird dazu genutzt Interfacestrukturen darzustellen und Nutzerinteraktionen zu verarbeiten ("UIViewController - UIKit | Apple Developer Documentation", 2016).

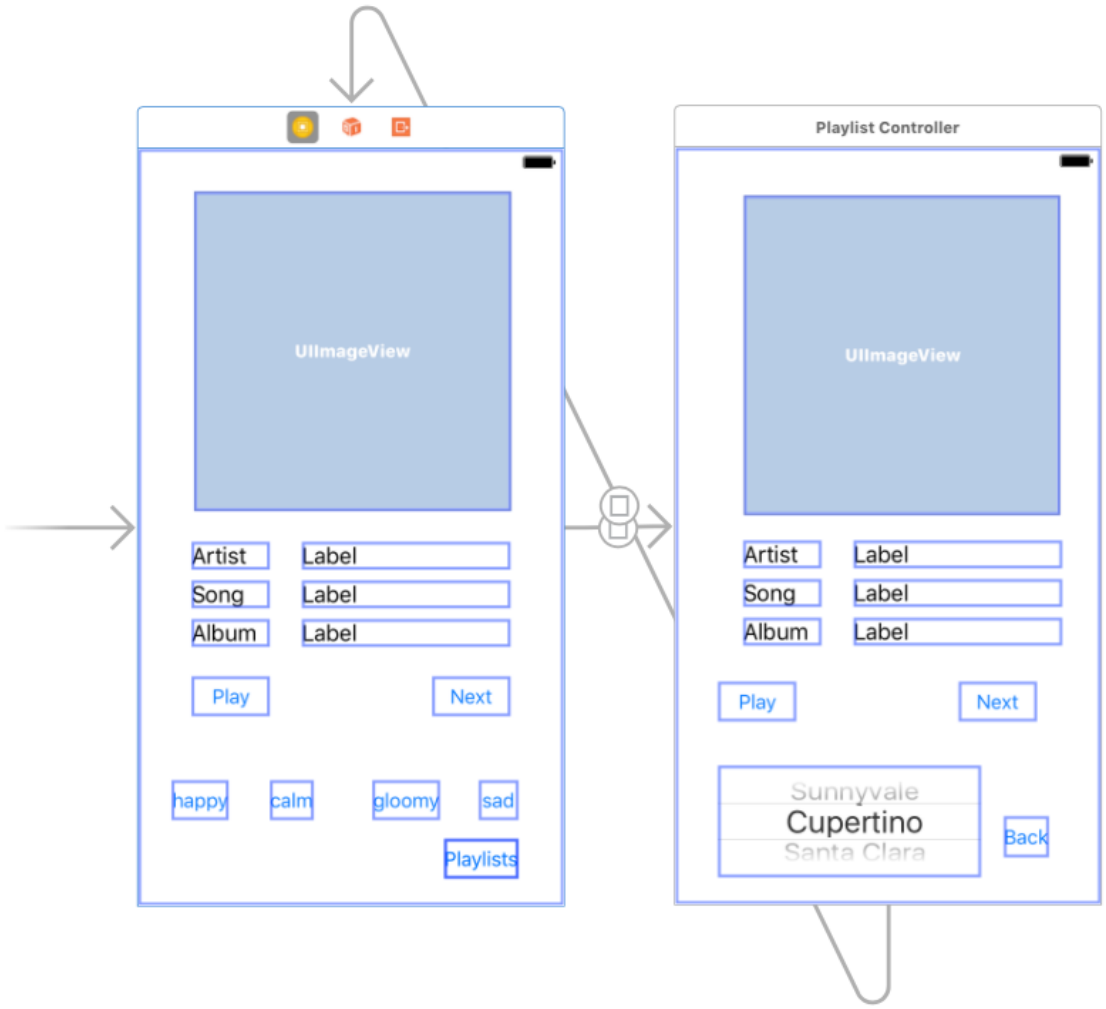

Abb. 1: Storyboard der App

Der erste View Controller beinhaltet 14 Interfacestrukturen. Eine *UIImageView* wird genutzt um das Albumcover darzustellen. Im ersten Projektplan sollten die Cover der Alben zwar nicht dargestellt werden, im Laufe der Entwicklung stellte sich jedoch heraus, dass die App ansonsten rein textbasiert sein würde.
Unter der UIImageView finden sich sechs Label. Drei davon haben fest codierte Texte, die anderen drei wurden mit den entsprechenden Informationen aus dem Media Player Framework gefüttert.
Unter den Labeln sind zwei Buttons platziert, ‚Play' und ‚Next'. Der ‚Play'-Button dient dabei auch als ‚Pause'-Button. Dies sollte eigentlich über ein sich änderndes Piktogramm dargestellt werden, wurde aber aus Zeitgründen nicht umgesetzt.
Darunter finden sich die Buttons mit denen der Nutzer seine Musikbibliothek in die entsprechenden Stimmungs-Playlists einteilen kann. Dies sind wie im Projektplan vorgestellt die Kategorien ‚happy', ‚calm', ‚gloomy' und ‚sad'.
Der Button ‚Playlists' führt über einen sog. Segue zur zweiten View, *Playlist Controller*. Segues, oder Übergänge, dienen der Transition zwischen Views ("View Controller Programming Guide for iOS: Using Segues", 2016).

Die zweite View beinhaltet sowohl das Albumcover als auch die Labels und ‚Play'- sowie ‚Next'-Buttons. Außerdem ist hier ein Media Picker platziert worden der zur Auswahl der verschiedenen Stimmungs-Playlists dient.
Ein Media Picker ermöglicht eine graphische „Übersetzung" der vier Stimmungs-Buttons in ein platzsparendes UI-Element ("MPMediaPickerController - MediaPlayer | Apple Developer Documentation", 2016).
Zuletzt wurde ein ‚Back'-Button mit einem weiteren Übergang zur ersten View implementiert. Dies ist ein sogenannter ‚Unwind'-Segue und wird im nächsten Kapitel genauer erläutert.

Dies umfasst die graphische UI der App. Geplant war, die einzelnen Elemente mit Hilfe selbst designter Piktogramme zu verschönern. Jedoch gestaltete sich dies schwierig, und da Ich nicht auf bereits vorhandene Bilder zurückgreifen wollte blieb die App so schlicht wie sie momentan ist. Für die Zukunft ist geplant, die App optisch ansprechender und selbsterklärender zu gestalten.

Der Code:

Bei der Erstellung eines Projekts in XCode 8 werden dem Entwickler von der IDE drei Dateien vorgegeben. Hierbei handelt es sich um *AppDelegate.swift, ViewController.swift* und *Main.storyboard*.
Für dieses Projekt wurde die *AppDelegate.swift*-Datei nicht verändert. Diese Datei legt fest, wie sich das Programm in bestimmten Situationen verhalten soll. Es gab für die Musikplayer-App nur eine interessante Funktion die geändert hätte werden müssen: ***func*** *applicationDidEnterBackground.* Diese gibt vor, wie sich die App verhalten soll wenn der Home-Bildschirm aufgerufen wird oder eine andere App geöffnet wird. Da das Media Player Framework Zugriff auf die interne Music-App des iPhones ermöglicht, ist dies bereits eingebaut. Die App spielt also weiterhin Musik ab wenn Sie geschlossen wird.

ViewController.swift:

```
import UIKit
import MediaPlayer



class ViewController: UIViewController {

    @IBOutlet weak var m_nextButton: UIButton!
    @IBOutlet weak var m_playButton: UIButton!

    @IBOutlet weak var m_SongTitel: UILabel!
    @IBOutlet weak var m_artist: UILabel!
    @IBOutlet weak var m_album: UILabel!

    @IBOutlet weak var m_AlbumCover: UIImageView!

    @IBOutlet weak var m_happy: UIButton!
    @IBOutlet weak var m_calm: UIButton!
    @IBOutlet weak var m_gloomy: UIButton!
    @IBOutlet weak var m_sad: UIButton!

    var mediaItems = MPMediaQuery.songs().items
    var query = MPMediaQuery.songs()
    var m_play = false
```

Abb. 2: Variablendeklaration

Die im Storyboard erstellten UI-Elemente müssen mit den Dateien verbunden werden. Dies geschieht indem man per Ctrl-Klick vom betreffenden Element auf den Code zieht. Dort öffnet sich ein Interface in welchem man auswählen kann, ob es sich um eine Variablendeklaration handelt oder eine Funktion erstellt werden soll. Wie in Abbildung 2 zu sehen, implementiert die Klasse ViewController das UIKit (Standard) und das MediaPlayer Framework.
Innerhalb der Klasse wurden 13 Variablen deklariert. Es handelt sich hierbei um die bereits vorgestellten Buttons und drei Variablen die mit dem MediaPlayer Framework in Verbindung stehen.
Um den Musikplayer zu erstellen müssen zunächst die auf dem iPhone gespeicherten Songs abgerufen werden. Dies geschieht mit einer Query, welche auf interne Methoden des Frameworks zugreift ("MPMediaItem - MediaPlayer | Apple Developer Documentation", 2016). Die im iPhone gespeicherten Songs stellen MediaItems dar welche über diese Suchanfrage gefunden werden. Sie können dann später mit einem eigens erstellen MusicPlayer-Objekt abgespielt werden.

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

    let mediaCollection = MPMediaItemCollection(items: mediaItems!)
    playerData.m_player.setQueue(with: mediaCollection)
    }
```

Abb. 3: Funktion viewDidLoad()

Die Funktion viewDidLoad() wird von XCode bei der Erstellung des Projekts in der ersten ViewController-Datei miterstellt. Sie wird dann üblicherweise vom Programmierer überschrieben und somit verändert.
Hier wurde eine mediaCollection erstellt und der eigens erstellte Musikplayer wurde angewiesen, die so erstellte Playlist abzuspielen wenn er genutzt wird.
Der Musikplayer wurde in einem sogenannten ***Struct*** initialisiert. Dies wurde in der Datei *DataModel.swift* definiert. Hierzu später mehr.

```
@IBAction func playMusic(_ sender: AnyObject) {

    if(m_play == false)
    {
        playerData.m_player.play()
        updateLabels()
        m_play = true

    }else
    {
        playerData.m_player.pause()
        m_play = false
    }

}
```

Abb. 4: playMusic()

Um Musik abspielen zu können wurde die *playMusic*-Funktion implementiert. Diese prüft zunächst ob der Musikspieler aktiv ist. Ist er dies nicht, wird der im Struct initialisierte Musikplayer mit der Methode *play()* aufgefordert zu spielen. Sollte der Nutzer beim Druck auf den Button bereits Musik spielen, wird der Player gestoppt. Außerdem findet sich hier die Funktion *updateLabels().*

```
func updateLabels() {

    m_SongTitel.text = playerData.m_player.nowPlayingItem?.title
    m_artist.text = playerData.m_player.nowPlayingItem?.artist
    m_album.text = playerData.m_player.nowPlayingItem?.albumTitle
    m_AlbumCover.image = playerData.m_player.nowPlayingItem?.artwork?.image(at: CGSize(width: 300, height: 300))
}
```

Abb. 5: updateLabels()

Diese Funktion weist den im Storyboard erstellten Labels die entsprechenden Informationen über den gerade spielenden Song zu. Sie wird für jeden abgespielten Song neu aufgerufen.

```
@IBAction func nextSong(_ sender: AnyObject) {
    playerData.m_player.skipToNextItem()
    updateLabels()
}
```

Abb. 6: nextSong()

Mit dem Klick auf den ‚Next'-Button triggert der Nutzer die in Abbildung 6 gezeigte Methode. Dabei ruft der Musikplayer die Methode *skipToNextItem()* auf und die Labels mit den Songinformationen werden aktualisiert.

DataModel.swift:

Wie bereits erwähnt musste ein sogenanntes Struct implementiert werden um dort den Musikplayer zu initialisieren und einige Daten zu speichern. Dies war notwendig um Daten, die in mehreren Views gebraucht werden übergreifend speichern zu können.

```
import Foundation
import MediaPlayer

struct playerData{

    static var m_happyList = Set<MPMediaEntityPersistentID>()
    static var m_calmList = Set<MPMediaEntityPersistentID>()
    static var m_gloomyList = Set<MPMediaEntityPersistentID>()
    static var m_sadList = Set<MPMediaEntityPersistentID>()

    static let m_player = MPMusicPlayerController.systemMusicPlayer()
}


enum PlayListType {
    case HAPPY
    case SAD
    case GLOOMY
    case CALM
}
```

Abb. 7: DataModel.swift

In Abbildung 7 ist zu sehen, dass die Variablen als statisch initialisiert wurden. Der Musikplayer wurde als sog. systemMusicPlayer erstellt, dieser gewährt den Zugriff auf die interne Musikbibliothek. Wir finden hier auch die Sets für die einzelnen Playlisten und einen Switch Case für Playlisttypen.

```
func setSongQueue(listType:PlayListType)
{

    playerData.m_player.stop()

    let idList:Array<MPMediaEntityPersistentID>

    switch listType{

    case PlayListType.HAPPY:
        idList = Array(playerData.m_happyList)
        break

    case PlayListType.SAD:
        idList = Array(playerData.m_sadList)
        break

    case PlayListType.GLOOMY:
        idList = Array(playerData.m_gloomyList)
        break
    case PlayListType.CALM:
        idList = Array(playerData.m_calmList)
        break


    }
```

Abb. 8: setSongQueue(PlayListType)

In der Datei *PlaylistController.swift* findet dieser Nutzung. Die Funktion setSongQueue(PlayListType) wird über den Media Picker in der zweiten View aufgerufen. Im PlayListType ist ein Set an Songs, bzw ihre sogenannte *PersistentID* gespeichert. Über diese PersistentID werden die Songs dann abgespielt sobald man in der zweiten View die entsprechende Playlist auswählt und Play drückt.

Anwendungsanleitung und Probleme

Um die App nutzen zu können muss ein iPhone 5s oder SE mit iOS 10 vorhanden sein. Auf anderen iPhones wird das Layout der App nicht korrekt angezeigt. Dies ist der Tatsache zu schulden, dass die Implementierung des Autolayouts von XCode meinen Horizont übersteigt (Ich versuchte, ich scheiterte). Um die Entwicklung nicht weiter zeitlich zu behindern wurde die App nur für die oben genannten Maße entwickelt.
Nachdem das Projekt von Git heruntergeladen und via XCode auf dem Gerät installiert wurde erfolgt die Ausführung via XCode automatisch. Dann muss der App-Entwickler verifiziert und für das Gerät zugelassen werden. Bei der ersten Nutzung öffnet sich ein Pop-Up das darüber informiert, dass das Gerät auf die Musikbibliothek zugreift. Unflätliche Formulierungen innerhalb des Pop-Ups sind der Tatsache zuzuschreiben, dass mit der Umstellung auf iOS 10 eine Reihe von Vorgaben über Privacy und Nutzung bestimmter Daten von Apple vorgegeben wurden. So musste z.B. explizit angegeben werden was die App mit den Kamera-Daten vorhat, auch wenn der Code der App die Kamera nicht im

Geringsten berührt. Hat man der App also Zugriff auf die Musikbibliothek gewährt, wird man mit der ersten View konfrontiert.
Hier klickt der User nun auf „Play“. Ab diesem Zeitpunkt darf er sich der Aufgabe widmen, seine Musikbibliothek in die vier zur Verfügung stehenden Stimmungen einzuteilen. Dies geschieht indem er einen der vier Buttons drückt während das Lied läuft. Die Lieder werden immer nur einmal für eine der Playlists gespeichert da Sie in einem Set gespeichert werden. Wenn der Nutzer fertig ist kann er auf den Button ‚Playlists‘ drücken. Dort kann er dann über den Media Picker seine Stimmung anwählen und Musik hören. Dabei kann er die App auch schließen und andere Anwendungen öffnen.

Reflexion

Wurde das Ziel erreicht? Wenn nicht, wieso? Was ist schief gelaufen? Wieso? Lösungen?

Die grundliegende Idee der App so wie sie im Projektplan vorgestellt wurde, wurde umgesetzt. Allerdings bin ich mit der optischen Aufmachung nicht zufrieden. Es stellte sich jedoch wesentlich schwerer als gedacht heraus, vernünftigte Assets selbst zu designen. Für die Zukunft würde ich dies gerne umsetzen, dazu muss ich mir jedoch ein komplett neues Set an Skills erarbeiten.
Außerdem ist die App nicht für alle iOs 10 iPhones nutzbar, da das AutoLayout mit seinen Constraints eine zu große Herausforderung für mich darstellte. Ich habe versucht, mir über StackOverflow Wissen anzulesen, und die dort vorgeschlagenen Tipps umzusetzen, aber schlussendlich hat dies nur Zeit aus dem Projekt gezogen und nicht funktioniert. Dies wäre der erste Ansatzpunkt wenn die App weiter entwickelt wird.

Literaturverzeichnis

The Swift Programming Language (Swift 3): About Swift. (2016). Developer.apple.com. Retrieved 22 September 2016, from https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/

MediaPlayer | Apple Developer Documentation. (2016). Developer.apple.com. Retrieved 22 September 2016, from https://developer.apple.com/reference/mediaplayer

UIViewController - UIKit | Apple Developer Documentation. (2016). Developer.apple.com. Retrieved 22 September 2016, from https://developer.apple.com/reference/uikit/uiviewcontroller

View Controller Programming Guide for iOS: Using Segues. (2016). Developer.apple.com. Retrieved 22 September 2016, from https://developer.apple.com/library/content/featuredarticles/ViewControllerPGforiPhoneOS/UsingSegues.html

MPMediaPickerController - MediaPlayer | Apple Developer Documentation. (2016). Developer.apple.com. Retrieved 22 September 2016, from https://developer.apple.com/reference/mediaplayer/mpmediapickercontroller

MPMediaItem - MediaPlayer | Apple Developer Documentation. (2016). Developer.apple.com. Retrieved 22 September 2016, from https://developer.apple.com/reference/mediaplayer/mpmediaitem