Razvoj iOS aplikacije za analizu mišljenja na Twitteru

Vrzan, Danijela

Sveučilište u Rijeci – Odjel za informatiku
Jednopredmetna informatika

Danijela Vrzan

Razvoj iOS aplikacije za analizu mišljenja na Twitteru

Završni rad

Mentor: doc. dr. sc. Marija Brkić Bakarić

Rijeka, rujan 2019.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 18.06.19.

Zadatak za završni rad

Pristupnik: Danijela Vrzan

Naziv završnog rada: Razvoj iOS aplikacije za analizu mišljenja na Twitteru

Naziv završnog rada na eng. jeziku: Development of iOS application for Twitter sentiment analysis

Sadržaj zadatka: Zadatak rada je izraditi iOS aplikaciju za analizu mišljenja na Twitteru primjenom strojnog učenja i opisati tijek njene izrade. Pomoću Appleovih razvojnih okvira za strojno učenje istrenirat će se model za raspoznavanje prirodnog jezika koristeći Swift programski jezik. Primjena će biti ograničena na engleski jezik. Unosom traženih podataka aplikacija će dohvaćati određeni broj najnovijih objava na Twitteru te ih analizirati i najzastupljeniju emociju prikazati u obliku emotikona.

Mentor

doc. dr. sc. Marija Brkić Bakarić

Voditelj za završne radove

dr. sc. Miran Pobar

Zadatak preuzet:

(potpis pristupnika)

Sažetak

U prvom dijelu rada prikazane su relevantne tehnologije korištene tijekom izrade aplikacije i neki osnovni pojmovi potrebni za razumijevanje tehnologija.

U drugom dijelu, opisana je izrada aplikacije za analizu mišljenja na Twitter servisu. Aplikacija je napisana u Swift programskom jeziku koristeći Xcode integrirano razvojno okruženje (u daljnjem tekstu, engl. *IDE*). Najprije se trenira model strojnog učenja na temelju metapodataka unutar Playground sučelja, a zatim se istrenirani model uveze u Xcode projekt. Pomoću Twitter aplikacijskog programskog sučelja (u daljnjem tekstu, engl. *API*) dohvaća se 100 najnovijih tweet-ova na engleskom jeziku koje istrenirani model analizira te na temelju prosječne ocjene prikazuje prikladan emotikon na korisničkom sučelju (engl. *UI*). Korisnik aplikacije u tekstualno polje upisuje "@" ili "#" te ime osobe, naziv tvrtke ili teme čije mišljenje želi analizirati. Upis u tekstualno polje ograničen je pomoću regularnog izraza. Aplikacija je nazvana Tweetalyze.

Ključne riječi

Analiza mišljenja, Apple, iOS aplikacija, strojno učenje, Twitter

Sadržaj

1. Uvod

Na globalnoj razini Android pametni telefoni zauzimaju oko 70% udjela u tržištu pametnih telefona od čega većim dijelom u Europi, dok iPhone zauzima oko 25%. Sa više od milijardu pametnih telefona u upotrebi, od siječnja 2016. godine, iOS je druga najpopularnija mobilna platforma. U nekim regijama, koje uključuju SAD, UK i Francusku, Apple-ovi uređaji nastavljaju dominirati na tržištu pametnih telefona. Iz istog razloga, velik broj poduzeća fokusira svoj softver prvenstveno, a ponekad i isključivo na iOS platformu kao ključnu priliku za ulaganje i stratešku korist.

Apple je 2014. godine izbacio na tržište svoj programski jezik - Swift. Bez obzira na činjenicu što je dosta mlad i još uvijek u razvoju, jedan je od jednostavnijih programskih jezika za naučiti. O njegovom potencijalu govore i imena tvrtki koje ga koriste, kao što su LinkedIn, Twitter, Fitbit i Lyft, a sve veću zainteresiranost pokazuju i Facebook i Uber.

Općenito, većina Apple-ovog softvera se vodi principom jednostavnosti odnosno korisnički nastrojenim (engl. *user-friendly*) pristupom. To je upravo jedan od glavnih razloga zašto, unatoč velikoj cijeni, korisnici diljem svijeta i dalje koriste Apple-ove proizvode. U nastavku nastojimo pokazati tu jednostavnost izradom iOS aplikacije za analizu mišljenja koja koristi Apple-ov Natural Language programski okvir (u daljnjem tekstu, engl. *framework*), jedan od Core ML framework-a za izradu modela strojnog učenja u samo nekoliko linija koda.

2. Upoznavanje s relevantnim tehnologijama

2.1. Apple

Apple Inc. je američka multinacionalna tehnološka tvrtka sa sjedištem u gradu Cupertino u Kaliforniji koja dizajnira, programira i prodaje potrošačku elektroniku, računalni softver i online usluge. Smatra se dijelom Velike Četvorke (engl. *Big Four*[1]) zajedno sa tvrtkama Google, Amazon i Facebook.

Tvrtku su osnovali Steve Jobs, Steve Wozniak i Ronald Wayne 1976. kako bi razvijali i prodavali osobno računalo Apple I (slika 1) koje je razvio Wozniak. Nakon samo 12 dana, Wayne prodaje svoje dionice kolegama, a 10 godina nakon njega i Wozniak napušta Apple, no ostaje kao počasni zaposlenik. Godine 2011. Steve Jobs, zbog zdravstvenih komplikacija, napušta tvrtku te mjesto izvršnog direktora prepušta Tim-u Cook-u. Nedugo nakon odlaska Steve Jobs umire, što predstavlja kraj jednog doba za Apple. Kao i svaka tehnološka tvrtka Apple se kroz godine borio za opstanak te nije imao nimalo lagan put do uspjeha kojeg imaju danas.

Slika 1: Apple I osobno računalo (preuzeto sa: https://www.pinterest.ca/pin/300122762654261655; svibanj 2019.)

Hardverski proizvodi tvrtke uključuju iPhone pametni telefon, iPad tablet računalo, iMac osobno računalo, Macbook prijenosno računalo, iPod prijenosni uređaj za reproduciranje medija, Apple Watch pametni sat, Apple TV digitalni uređaj za reproduciranje multimedije, Apple Airpod bežične slušalice i HomePod pametni zvučnik.

Od softverskih proizvoda najpoznatiji su macOS, iOS i watchOS operacijski sustavi, iTunes za prikazivanje medija, Safari web preglednik, iCloud web servis, Xcode IDE i mnogi drugi. Također su razvili jedinstveni servis, Apple Pay, sustav bežičnog plaćanja kreditnim karticama. U srpnju 2019. godine Apple izdaje i svoju kreditnu karticu u suradnji sa američkom bankom Goldman-Sachs koja je trenutno dostupna samo u SAD-u. Kartica je posebna po tome što je napravljena od titanija i ne sadrži nikakve podatke o vlasniku kartice nego su sve informacije spremljene unutar iOS Wallet aplikacije.

Apple je danas najveća svjetska tvrtka za informacijsku tehnologiju po ukupnim prihodima te treća najveća tvrtka u svijetu po proizvodnji pametnih telefona, odmah iza Samsung-a i Huawei-a. Tvrtka također slovi kao najvrjedniji svjetski brand.

[1] *Big Four* je termin koji se koristi prilikom opisa četiri multinacionalne tehnološke tvrtke s najvećim utjecajem na društvene promjene. Termin *Big Five* pridodaje i Microsoft.

2.2. Swift

Swift je programski jezik opće namjene Apple-ovih proizvoda koji je razvila tvrtka Apple 2014. godine. Dizajniran je kako bi radio sa velikim dijelom koda napisanog u Objective-C programskom jeziku. Napravljen je pomoću LLVM prevodioca (engl. *compiler*) otvorenog koda i implementiran u Xcode od verzije 6. Swift je nastao kao rezultat dugogodišnjeg istraživanja drugih programskih jezika, implementirajući ono najbolje kako bi bio što razumljiviji, jednostavniji i intuitivniji. Apple je 2019. godine izbacio i SwiftUI alat za korisničko sučelje koji omogućuje dizajn aplikacija na deklarativan način, odnosno programer definira kako će korisničko sučelje izgledati i raditi, a SwiftUI sam razrađuje implementaciju preko same interakcije sa korisnikom.

2.3. Xcode

Xcode je integrirano razvojno okruženje tvrtke Apple za razvoj svih Apple-ovih aplikacija neovisno o hardverskom proizvodu. Prvi put je objavljen 2003. godine. Dostupan je besplatno na Mac App Store-u. Osim Swift programskog jezika, Xcode podržava izvorni kod i za druge programske jezike, a neki od poznatijih su C, C++, Java, Python i Ruby. Xcode može izraditi binarne datoteke koje sadrže kod za više arhitektura pomoću Mach-O izvršnog formata (engl. *fat binary files*). To su tvz. univerzalni binarni formati, koji omogućuju izvršavanje programskog koda i na PowerPC i na Intel-baziranim (x86) platformama, koje uključuju i 32-bitan i 64-bitan kod za obje arhitekture.

2.4. Twitter

Twitter je on-line servis za mikroblogiranje (engl. *microblogging*) i distribuciju kratkih poruka između grupe primatelja putem osobnog računala ili pametnog telefona. Uključuje aspekte društvenih mreža, kao što su Myspace i Facebook, ali su poruke ili tzv. „tweet-ovi“ ograničeni na 280 znakova i to Twitter čini drugačijim i jedinstvenim. Svaki kreirani profil na Twitter-u sastoji se od "@" znaka i imena koje osoba ili tvrtka sama odabere, npr. "@Apple". Unutar tweet-ova korisnici često referenciraju napisani tekst koristeći "#" tag-ove pomoću kojeg se mogu pretraživati tweet-ovi koristeći ključne riječi koje sadrže željene tag-ove.

2.5. Cocoa i Cocoa Touch

Cocoa i Cocoa Touch predstavljaju okruženje za razvoj aplikacija za OS X, odnosno iOS operacijske sustave. Oba sadrže Objective-C runtime i dva temeljna framework-a.

- ***Cocoa***, koji sadrži Foundation i AppKit framework-e te se koristi za izradu aplikacija koje se izvode na OS X uređajima.
- ***Cocoa Touch***, koji sadrži Foundation i UIKit framework-e te se koristi za izradu aplikacija za iOS uređaje.

Foundation framework implementira temeljnu klasu NSObject, koja definira osnovna ponašanja objekata te klase koje predstavljaju osnovne primitivne tipove, kao što su *string* i *numbers*. AppKit i UIKit framework-e koristimo prilikom izrade korisničkog sučelja aplikacije. Oba framework-a su jednaka u namjeni, ali specifična platformi. Sadrže klase za upravljanje događajima, obradu slika, obradu teksta, gumbe, tekstualna polja, dijaloge upozorenja te druge elemente korisničkog sučelja. Cocoa Touch, odnosno UIKit framework, sadrži klase specifične uređajima koji koriste ekrane na dodir (engl. *touch screen*).

2.5.1. CocoaPods

CocoaPods je upravitelj ovisnostima (u daljnjem tekstu, engl. *dependency manager*) za Swift i Objective-C Cocoa projekte. Dependency manager omogućuje jednostavnije dodavanje, brisanje, ažuriranje i upravljanje Cocoa bibliotekama treće strane otvorenog koda (u daljnjem tekstu, engl. *third-party open-source*) koje koristi aplikacija. Primjerice, umjesto pisanja vlastitog framework-a za upravljanje JSON datotekama, može se uvesti SwiftyJSON koristeći CocoaPods dependency manager.

CocoaPods se temelji na Ruby programskom jeziku koji je implementiran u macOS sustav od verzije 10.7. S obzirom da CocoaPods predstavlja opcionalan dependency manager za macOS, potrebno ga je instalirati putem terminala koristeći naredbu **gem install cocoapods.** Nakon toga, potrebna je samo inicijalizacija unutar projekta u koji uvozimo third-party biblioteku. Automatski se generira tekstualni dokument *Podfile* unutar kojeg se definira željena biblioteka ili više njih. CocoaPods zatim rekurzivno rješava ovisnosti između biblioteka, dohvaća izvorni kod za sve ovisnosti te stvara i održava Xcode workspace za izradu aplikacije.

2.6. Strojno učenje

Umjetna inteligencija (engl. *artificial intelligence*) je opći pojam za primjenu bilo koje tehnologije koja računalu omogućuje imitiranje ljudskog ponašanja koristeći logiku, if-then pravila i stabla odlučivanja. Grana umjetne inteligencije koja se bavi oblikovanjem i usavršavanjem algoritama na temelju empirijskih[2] podataka, zove se strojno učenje (engl. *machine learning*). U prošlosti, računala su mogla raditi samo ono što im je bilo napisano u kodu, za što su bila programirana. Strojno učenje omogućuje da se taj spektar mogućnosti proširi i to na način da računalo samo uči na temelju zadanih parametara detektirajući uzorke i primjenjujući te uzorke u predviđanju budućih podataka ili za obavljanje definiranih radnji. U početku će griješiti, no kao i čovjek, s vremenom postane sve bolji dok na kraju ne nauči zadanu radnju.

2.6.1. Core ML

Core ML je framework za strojno učenje koji se koristi na svim Apple-ovim proizvodima (iOS, macOS, tvOS i watchOS) za brzo predviđanje ili zaključivanje s jednostavnom integracijom unaprijed naučenih modela strojnog učenja. Najveća prednost ovog tipa strojnog učenja su rezultati u gotovo stvarno vrijeme (engl. *real-time*), tj. uz malo kašnjenja (engl. *low latency*). Nema slanja podataka pozivanjem API-a i čekanja na odgovor, svi podatci se nalaze na uređaju i mogu se koristiti izvan mreže (engl. *offline*).

Vision framework za raspoznavanje fotografija u stvarnom vremenu, *Natural Language* framework za predviđanje teksta, *SoundAnalysis* framework za identifikaciju govornika i *Speech* framework za pretvaranje zvuka u tekst, predstavljaju inovacije koje omogućuje strojno učenje pomoću Core ML-a.

2.6.2. Create ML

Godine 2018. Apple je objavio novi framework za strojno učenje, Create ML, koji omogućuje programerima kreiranje i učenje prilagođenih modela strojnog učenja preko macOS Playground sučelja integriranog unutar Xcode-a. Sučelje omogućuje treniranje modela u stvarnom vremenu bez predznanja o strojnom učenju. Model se trenira tako da mu se prikažu reprezentativni primjeri na temelju kojih se uči raspoznavanju uzorka. Tako istrenirani model pomoću Core ML-a implementira se u aplikaciju.

[2] Podaci temeljeni na iskustvu, dobiveni istraživanjem ili pokusima, ne teorijom.

2.7. Obrada prirodnog jezika

Obrada prirodnog jezika (u daljnjem tekstu, engl. *Natural Language ili NL*) je grana strojnog učenja pomoću koje se računalo uči razumijevanju, analiziranju, manipulaciji i potencijalnom generiranju prirodnog jezika. Najpoznatiji primjer je Google Translate koji prevodi tekst sa jednog jezika na drugi. Obrada prirodnog jezika se smatra izuzetno teškim problemom računalne znanosti. Sama priroda ljudskog jezika čini računalnu obradu teškom. Primjerice, neke riječi su višeznačne što je računalima teško za razumjeti. Bile su potrebne godine učenja prije nego je Google Translate dostigao razinu koju posjeduje danas, iako se još uvijek mogu naći riječi koje izgube značenje u samom prijevodu.

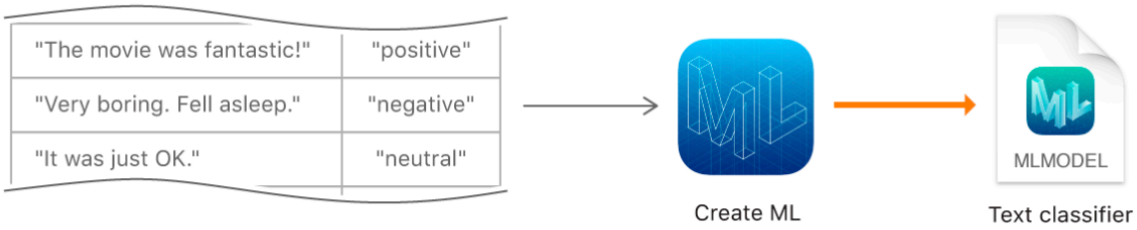
2.7.1. Natural Language

Za analizu teksta i izvlačenje specifičnih metapodataka prirodnog jezika, Apple je objavio svoj Natural Language framework, koji se uz pomoć Create ML-a može koristiti za izradu i implementaciju prilagođenih modela prirodnog jezika. Mogućnosti framework-a su identifikacija jezika, segmentacija teksta u riječi, rečenice i odlomke, izdvajanje entiteta određenih tipova (imena organizacija i ljudi), lematizacija[3] te analiziranje dijela govora tj. identificiranje dijela govora kojem trenutna riječ pripada. Uvođenjem Core ML-a u kod aplikacije, automatski uvodimo sve ostale framework-e strojnog učenja među kojima je i NL framework.

[3] Postupak svođenja riječi na osnovni oblik.

3. Izrada modela strojnog učenja

Prije izrade same aplikacije, potrebno je istrenirati model strojnog učenja i naučiti ga raspoznavati pozitivne, negativne i neutralne komentare, odnosno tweet-ove na Twitteru. Za treniranje modela potrebni su metapodatci (slika 2). Na stranici Carnegie Mellon University[4] nalazi se dokument koji sadrži 988 tweet-ova u csv formatu u 3 kategorije: Pos (163 tweet-a koji sadrže pozitivno mišljenje), Neg (316 tweet-ova koji sadrže negativno mišljenje) i Neutral (509 tweet-ova koji sadrže neutralno mišljenje, odnosno ne sadrže emociju). Dokument je izradila tvrtka Sanders Analytics, a sadrži tweet-ove i mišljenja o tvrtki Apple. Tweet-ove su zatim pokazali ljudima i zabilježili njihove emocije. Koristit ćemo navedeni dokument za treniranje modela strojnog učenja zbog brzine i jednostavnosti. Skidanje tweet-ova i analiziranje mišljenja je zahtjevan i dugotrajan proces.



Slika 2: Treniranje modela pomoću metapodataka

Apple je omogućio svim programerima, sa ili bez ikakvog predznanja o strojnom učenju, jednostavnu izradu modela putem macOS Playground-a, Apple-ovog interaktivnog sučelja integriranog u Xcode IDE. Za razliku od modela prepoznavanja slika, koji ima live pregled izrade modela, za prirodni jezik to još nije implementirano.

Izrada modela započinje uvozom metapodataka za treniranje koji moraju biti spremljeni u tabličnom ključ-vrijednost formatu (u daljnjem tekstu, engl. *key-value*) kao csv ili json dokument. U ovom slučaju koristimo csv dokument u kojem svaki redak tablice sadrži par ključeva - class i text. Vrijednosti ključeva su ulazni podatci za treniranje modela. Kreiramo novu konstantu *let data* koju inicijaliziramo kao instancu MLDataTable strukture koja kao

[4] http://boston.lti.cs.cmu.edu/classes/95-865-K/HW/HW3/

parametar prima csv dokument (slika 3). Strukturu moramo označiti ključnom riječi *try* koja sprječava nastavak izvršavanja koda ukoliko dođe do greške prilikom otvaranja dokumenta.

```
let data = try MLDataTable(contentsOf: URL(fileURLWithPath: "../../twitter-sanders-apple3.csv"))
```

Slika 3: Uvoz csv dokumenta

MLDataTable je Create ML verzija tablice u kojoj svaki redak predstavlja entitet (npr. knjiga) s karakteristikama koje se mogu promatrati. Svaki stupac (MLDataColumn) predstavlja obilježje tog entiteta, kao npr. ime knjige ili autora. U našem slučaju entitet je tweet, a obilježja class i text, gdje class predstavlja 3 vrste mišljena o tweet-u: Pos, Neg i Neutral (slika 4).

class	text
Pos	Now all @Apple has to do is get swype on the iphone and it will be crack. Iphone that is
Pos	@Apple will be adding more carrier support to the iPhone 4S (just announced)
Pos	Hilarious @youtube video - guy does a duet with @apple 's Siri. Pretty much sums up the love affair! http://t.co/8ExbnQjY
Pos	@RIM you made it too easy for me to switch to @Apple iPhone. See ya!
Pos	I just realized that the reason I got into twitter was ios5 thanks @apple
Pos	I'm a current @Blackberry user little bit disappointed with it! Should I move to @Android or @Apple @iphone
Pos	The 16 strangest things Siri has said so far. I am SOOO glad that @Apple gave Siri a sense of humor! http://t.co/TWAeUDBp via @HappyPlace
Pos	Great up close & personal event @Apple tonight in Regent St store!
Pos	From which companies do you experience the best customer service aside from @zappos and @apple?
Pos	Just apply for a job at @Apple hope they call me lol
Pos	RT @JamaicanIdler: Lmao I think @apple is onto something magical! I am DYING!!! haha. Siri suggested where to find whores and where to h ...
Pos	Lmao I think @apple is onto something magical! I am DYING!!! haha. Siri suggested where to find whores and where to hide a body lolol
Pos	RT @PhillipRowntree: Just registered as an @apple developer... Here's hoping I can actually do it... Any help greatly appreciated!
Pos	Wow. Great deals on refurbed #iPad (first gen) models. RT: Apple offers great deals on refurbished 1st-gen iPads http://t.co/ukWOKBGd @Apple
Pos	Just registered as an @apple developer... Here's hoping I can actually do it... Any help greatly appreciated!
Pos	! Currently learning Mandarin for my upcoming trip to Hong Kong. I gotta hand it to @Apple iPhones & their uber useful flashcard apps
Pos	Come to the dark side @gretcheneclark: Hey @apple if you send me a free iPhone I will publicly and ceremoniously burn my #BlackBerry.
Pos	Hey @apple if you send me a free iPhone (any version will do) I will publicly and ceremoniously burn my #BlackBerry.
Pos	Thank you @apple for Find My Mac - just located and wiped my stolen Air. #smallvictory #thievingbastards
Pos	Thanks to @Apple Covent Garden #GeniusBar for replacing my MacBook keyboard/cracked wristpad during my lunch break today out of warranty.
Pos	@DailyDealChat @apple Thanks!!
Pos	iPads Replace Bound Playbooks on Some N.F.L. Teams http://t.co/2UXAWKwf @apple @nytimes
Pos	@apple..good ipad
Pos	@apple @siri is efffing amazing!!
Pos	Amazing new @Apple iOs 5 feature. http://t.co/jatFVfpM
Pos	RT @TripLingo: We're one of a few Featured Education Apps on the @Apple **Website** today sweet! http://t.co/0yWvbe1Z

Slika 4: Isječak iz twitter-sanders-apple3.csv dokumenta

Nakon uvoza tablice i kreiranja instance MLDataTable strukture, potrebno je rasporediti tablične podatke na testne i podatke za treniranje. Uobičajena je raspodjela u omjeru 80:20 ili 90:10 u korist podataka za treniranje. Dijelimo podatke u dvije tablice koristeći metodu *randomSplit(by:seed:)* (slika 5). Drugi parametar ove metode nam služi ukoliko se pojavi potreba za pozivanjem identične nasumične raspodjele koju smo koristili prilikom treniranja modela, a najčešće za testiranje i ispravljanje kvarova (engl. *debugging*).

```
let(trainingData, testingData) = data.randomSplit(by: 0.9, seed: 5)
```

Slika 5: Nasumična raspodjela podataka na testne i podatke za treniranje modela

Nakon toga kreiramo novu konstantu *let sentimentClassifier* i inicijaliziramo je kao instancu MLTextClassifier strukture, specifične za Natural Language framework, metodom koja za parametre prima podatke za treniranje koje smo prethodno definirali te nazive stupaca tablice, text i label - koji predstavljaju ključeve tablice (slika 6). Ako sada pokrenemo kod, treniramo model strojnog učenja. U samo 3 reda koda uvezli smo dokument čije podatke koristimo za treniranje i testiranje modela, definirali smo nasumičnu raspodjelu tih podataka i kreirali model strojnog učenja.

```
let sentimentClassifier = try MLTextClassifier(trainingData: trainingData, textColumn: "text",
    labelColumn: "class")
```

Slika 6: Definiranje modela

Idući korak je dobivanje povratne informacije o točnosti istreniranog modela. Tijekom treniranja modela, Create ML odvaja 5% testnih podataka koje koristi za validaciju modela tijekom faze treniranja. Podatci o validaciji omogućuju ocjenjivanje točnosti na primjerima na kojima model nije bio treniran. Pozivanjem *trainingMetrics* i *validationMetrics* svojstava MLClassifierMetrics strukture možemo izračunati točnost treniranja i točnost validacije modela, koristeći *classificationError* svojstvo. Jednostavnim matematičkim izračunom dobijemo vrijednosti u postotku (slika 7).

```
let trainingAccuracy = (1.0 - sentimentClassifier.trainingMetrics.classificationError) * 100
let validationAccuracy = (1.0 - sentimentClassifier.validationMetrics.classificationError) * 100
```

```
100
62
```

Slika 7: Postotak točnosti treniranja i validacije

Prosljeđivanjem testnih podataka *evaluation(on:)* metodi koja vraća MLClassifierMetrics instancu, možemo izračunati učinak treniranog modela testirajući ga na rečenicama koje nikada prije nije vidio. Pomoću dobivene MLClassifierMetrics instance, koristeći *classificationError* svojstvo, izračunamo točnost procjene istreniranog modela (slika 8). Točnost se bazira na metapodatcima iz csv datoteke koju smo definirali na početku. Dobivena točnost iznosi 69.79%.

```
let evaluationMetrics = sentimentClassifier.evaluation(on: testingData)
let evaluationAccuracy = (1.0 - evaluationMetrics.classificationError) * 100
```

Slika 8: Izračun točnosti procjene modela

Pretpostavit ćemo da je točnost od oko 70% zadovoljavajuća u našem slučaju, zbog relativno malog broja metapodataka koje koristimo i činjenice da se klasifikacija tweet-ova bazira na mišljenjima ljudi koji će u većini slučajeva subjektivno ocijeniti tweet te se mišljenje o istom tweet-u razlikuje od osobe do osobe.

Pozivanjem *prediction(from:)* metode koja predviđa klasifikaciju za dani tekst, možemo testirati model unutar Playground-a kako bi provjerili njegovu točnost (slika 9).

```
try sentimentClassifier.prediction(from: "@Apple is a terrible company!")                    "Neg"
try sentimentClassifier.prediction(from: "I would say @kfc has the best spicy wings! Wouldn't you?")  "Pos"
try sentimentClassifier.prediction(from: "The best University in the world! #UniversityOfRijeka")     "Pos"
try sentimentClassifier.prediction(from: "I think @Tesla is ok.")                            "Neutral"
```

Slika 9: Analiza mišljenja danog teksta

Iz dobivenih rezultata možemo zaključiti kako model dosta dobro ocjenjuje dani tekst, tweet-ovi koje smo smislili i dali modelu na procjenu su klasificirani onako kako bi ih i mi sami klasificirali.

Koristeći *MLModelMetadata(author:shortDescription:version:)* metodu sa tri parametra, autora modela, kratki opis i verziju, definiramo metapodatke o modelu te ih spremimo unutar *let metaData* konstante. Metoda ima više opcionalnih parametara, ali je većina u ovom slučaju nepotrebna.

Kada smo zadovoljni sa procjenom točnosti modela, spremimo model koristeći *write(to:metadata:)* metodu (slika 10) koja ima dva parametra. Prvi se odnosi na lokaciju na računalu gdje želimo spremiti gotov model dok drugi predstavlja metapodatke o modelu koje smo definirali unutar *let metaData* konstante. Spremljeni model možemo uvesti u aplikaciju koju ćemo izraditi u nastavku.

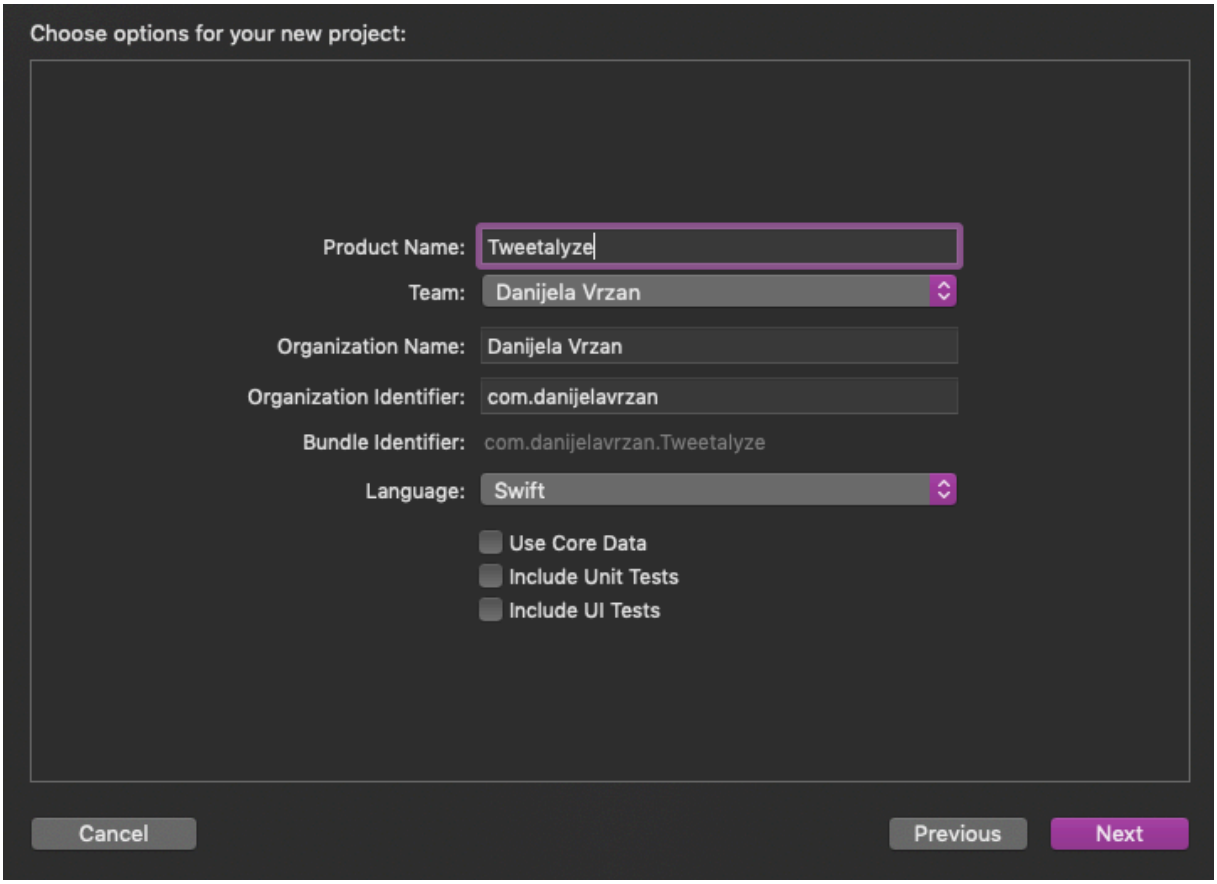
```
let metaData = MLModelMetadata(author: "Danijela Vrzan", shortDescription:
    "A model trained to classify sentiment based on Tweets", version: "1.0")

try sentimentClassifier.write(to: URL(fileURLWithPath:
    "../TweetalyzeSentimentClassifier.mlmodel"), metadata: metaData)
```

Slika 10: Izvoz modela strojnog učenja za analizu mišljenja

4. Izrada aplikacije

Aplikacija je izrađena u Xcode IDE-u. Odabiremo *iOS Single View app* i unosimo osnovne podatke o projektu (slika 12) kao što je naziv aplikacije, naziv tima ili programera koji razvija aplikaciju, jedinstveni ID organizacije ili programera. *Bundle Identifier* predstavlja jedinstveni ID aplikacije u Apple-ovom eko sustavu. Za ID organizacije, odnosno ID programera preporuča se koristiti obrnuti naziv domene (engl. *reverse domain name*)[5], koji se onda automatski generira kao *Bundle Identifier* s nastavkom imena aplikacije.

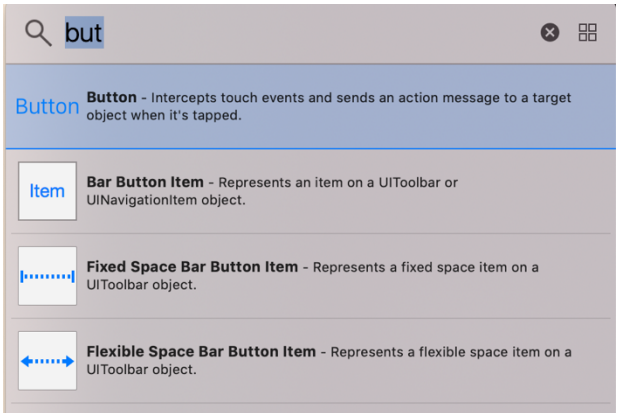
4.1. Izrada korisničkog sučelja



Slika 11: Kreiranje nove aplikacije u Xcode-u

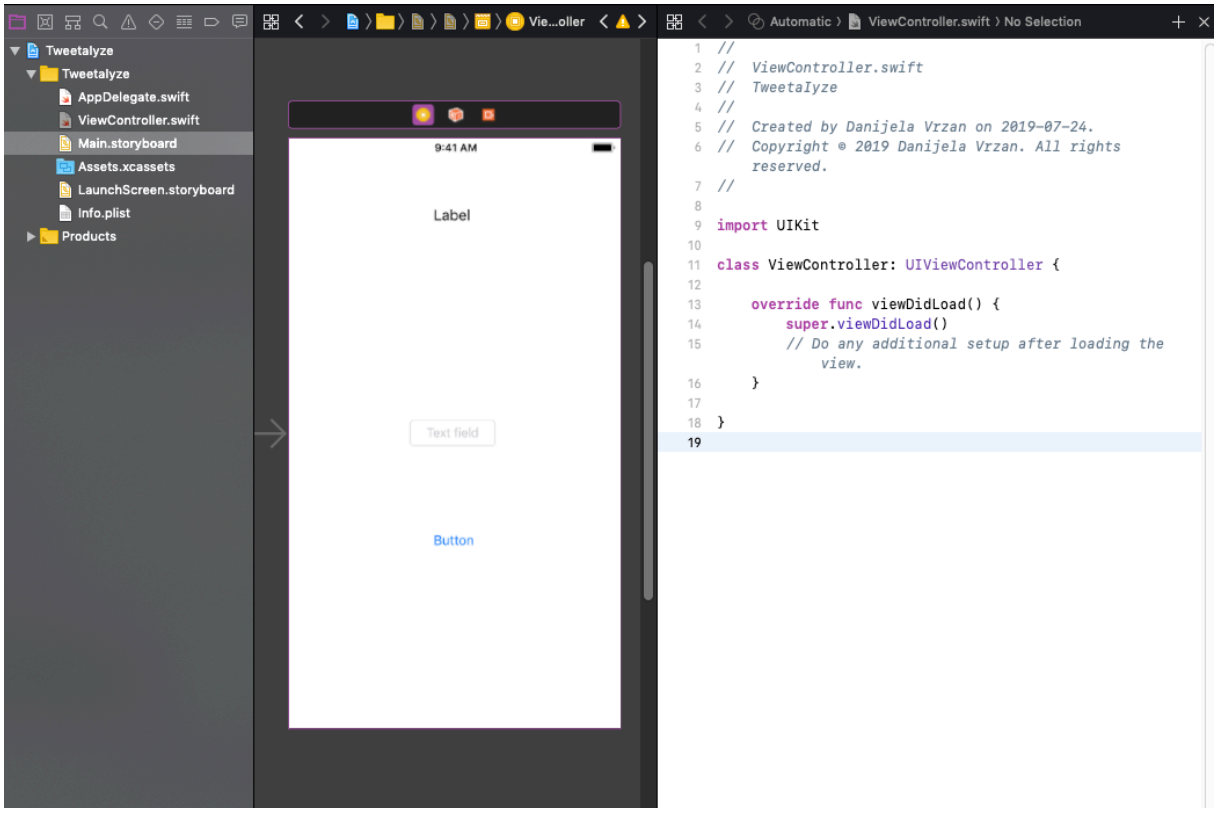
Prije samog programiranja, potrebno je u Main.storyboard datoteku koja predstavlja izgled aplikacije dodati elemente korisničkog sučelja aplikacije (slika 12). Elemente dodajemo putem grafičkog sučelja, te ih pozicioniramo na *ViewController*.

[5] Uobičajeni način imenovanja vezan uz registriranu domenu npr. imamo aplikaciju MyApp i registriranu domenu example.com, onda koristimo com.example.MyApp za opis aplikacije



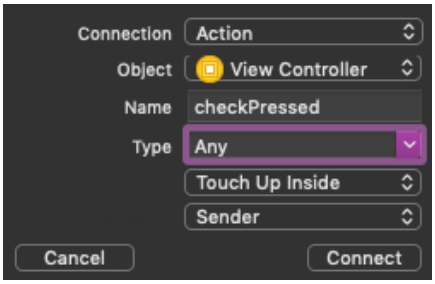
Slika 12: Biblioteka objekata (engl. Object Library)

ViewController je instanca UIViewController klase koja upravlja hijerarhijom prikaza UIKit aplikacije. ViewController.swift je dokument koji sadrži sav kod vezan za jedan pogled (engl. *view*) unutar Main.storyboard datoteke (slika 13). Ako napravimo još jedan pogled unutar Main.storyboard datoteke, instanciramo novi pogled UIViewController klase koji nazovemo SecondViewController.swift, koji upravlja novim pogledom aplikacije.

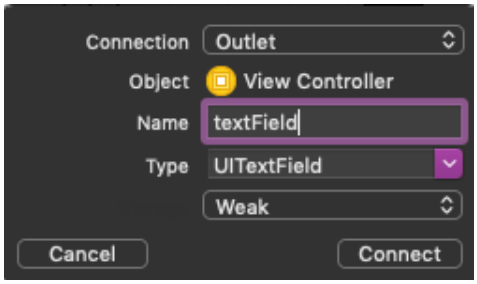


Slika 13: Glavni dijelovi grafičkog sučelja Xcode-a

Nakon dodavanja elemenata gumba, unosa teksta i tekstualne oznake, unutar pogleda, označimo element te povučemo i spustimo u ViewController.swift datoteku unutar *ViewController* klase. Tu inicijaliziramo funkcije (slika 14) i varijable (slika 15) potrebne za aplikaciju. Napravit ćemo varijable za sva 3 elementa sučelja te po jednu funkciju za gumb i unos teksta (slika 16).



Slika 14: Kreiranje funkcije



Slika 15: Kreiranje varijable

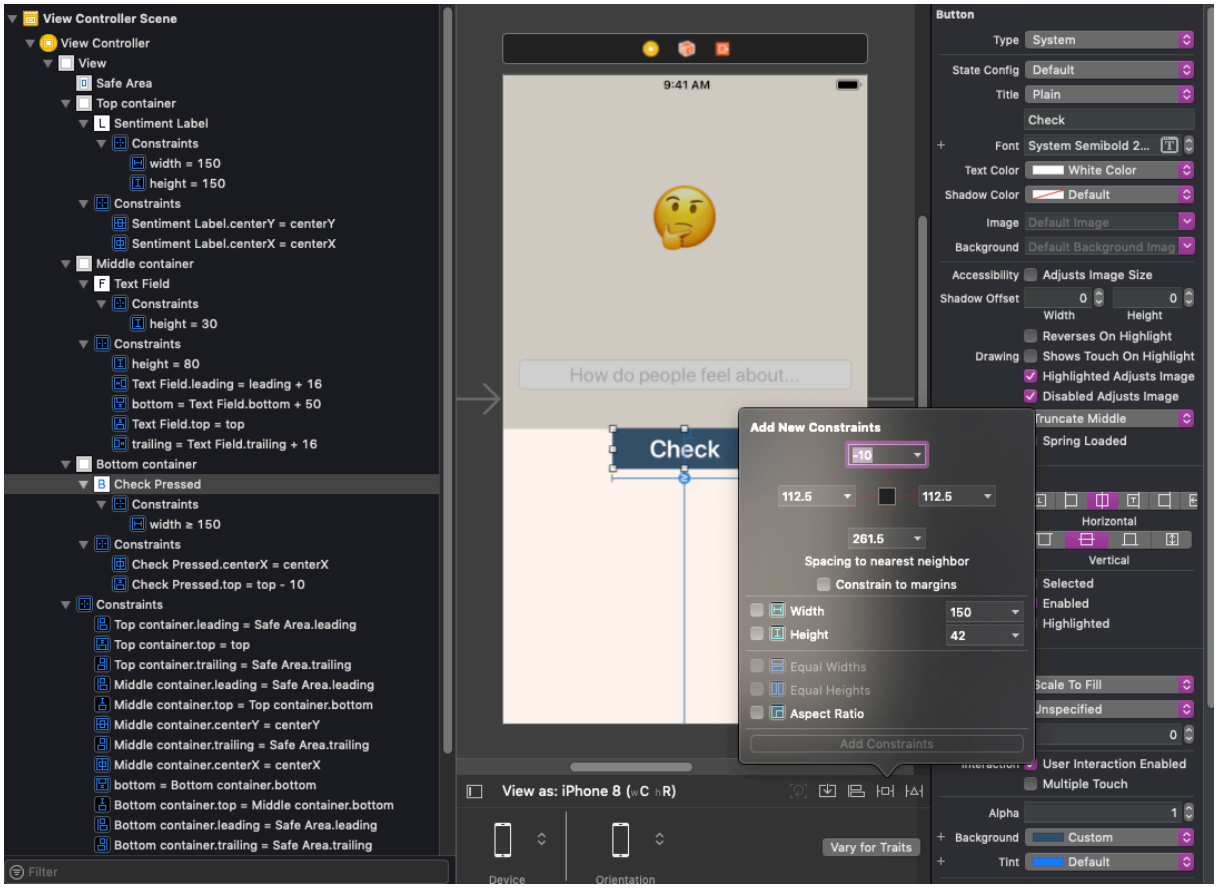
```
class ViewController: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var sentimentLabel: UILabel!
    @IBOutlet weak var checkPressed: UIButton!

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

    @IBAction func textFieldEditingDidChange(_ sender: UITextField) {

    }

    @IBAction func checkPressed(_ sender: Any) {

    }
```

Slika 16: Povezivanje elemenata korisničkog sučelja sa kodom aplikacije

4.2. Dizajn korisničkog sučelja

Idući korak koji možemo odmah napraviti je dizajn korisničkog sučelja koji je u ovom slučaju jednostavan. Potrebno je namjestiti ograničenja automatskog izgleda (engl. *auto-layout constraints*) pojedinih elemenata pogleda za različite veličine zaslona iPhone-a. Iz knjižnice objekata dodat ćemo 3 pogleda, koja ćemo nazvati Top, Middle i Bottom containers, a sadržavat će elemente korisničkog sučelja.

Ovi pogledi se koriste za jednostavniji razmještaj elemenata na glavnom pogledu aplikacije. Nakon što namjestimo njihova ograničenja unutar margina glavnog pogleda, namjestimo ograničenja elemenata tako da im razmještaj na glavnom pogledu bude razmjeran svim veličinama zaslona (slika 18).



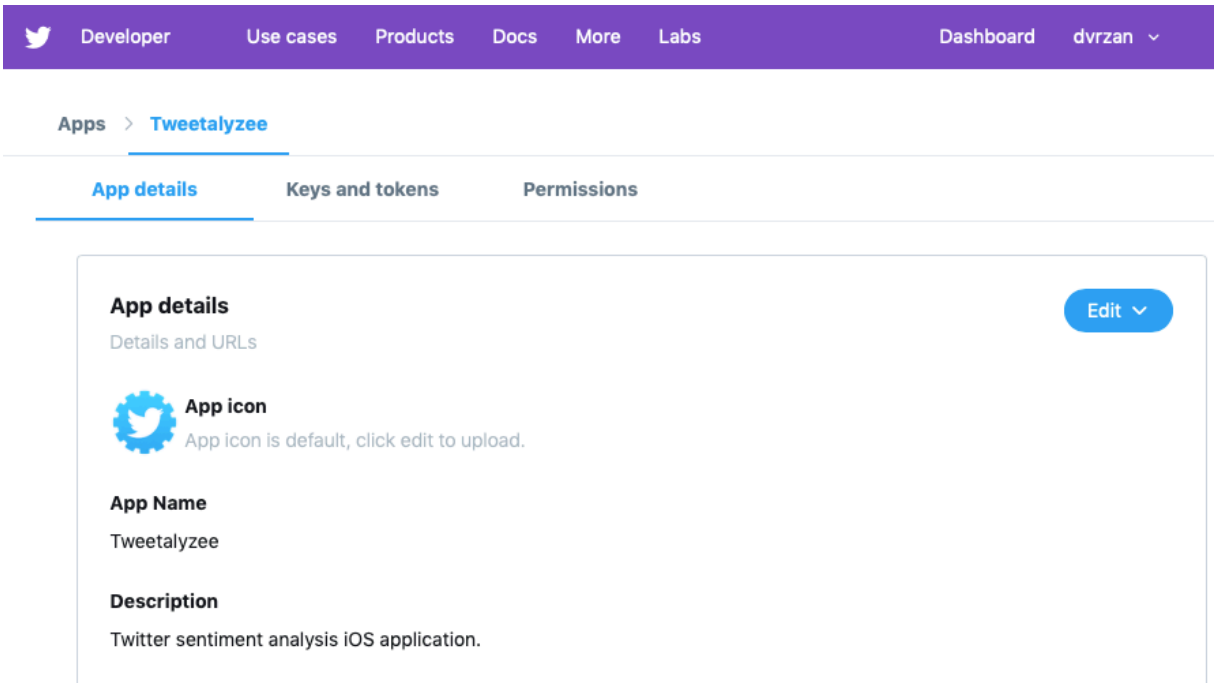
Slika 17: Dizajn glavnog pogleda aplikacije

Ako radimo aplikaciju za iPhone 7 i ne namjestimo ograničenja automatskog izgleda za druge veličine zaslona, elementi sučelja će ostati iste veličine i na istom mjestu. Ako otvorimo aplikaciju na iPhone 4S ekranu, elementi aplikacije će biti odrezani, dok će na iPhone Xs Max ekranu biti premali i na krivoj poziciji.

Kada smo zadovoljni ograničenjima ekrana, dizajniramo izgled elemenata aplikacije, što uključuje boju pozadine, boju i veličinu teksta, način unosa teksta te druga potrebna i željena svojstva.

4.3. Twitter developer account

Aplikacija povlači live tweet-ove sa Twitter-a pomoću Twitter API-a. Da bi dobili pristup tweet-ovima moramo registrirati Twitter Developer račun. Twitter ima više vrsta API-a za različite tipove korisnika: Standard API, Ads API, Twitter enterprise API i Twitter publisher tools and SDKs. Kako se svi osim standardnog naplaćuju, koristit ćemo besplatni standardni API koji nam omogućava dohvaćanje maksimalno 100 tweet-ova, što je za naše potrebe sasvim dovoljno. Uspješnom registracijom Twitter Developer računa dobivamo pristup Twitter API ključevima. Da bi dobili ključeve pomoću kojih se spajamo na Twitter i dohvaćamo tweet-ove, potrebno je na Twitter Developer kontrolnoj ploči (engl. *dashboard)* kreirati aplikaciju (slika 18) unutar koje će se automatski generirati potrošački ključevi (engl. *Consumer Keys)* koji se koriste za provjeru autentičnosti zahtjeva upućenih prema Twitter platformi.

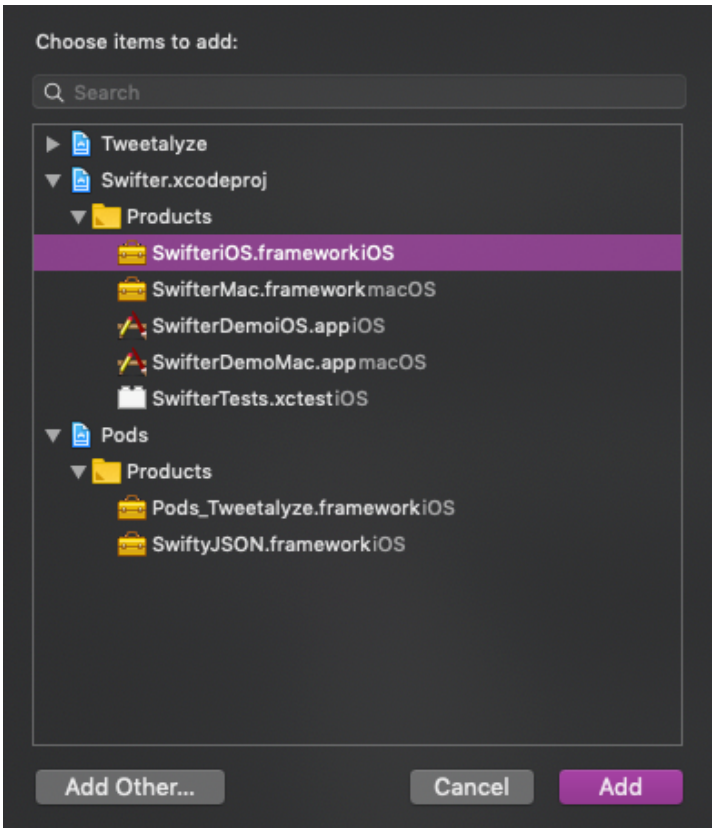


Slika 18: Twitter Developer Dashboard

4.4. Funkcija za dohvaćanje tweet-ova

4.4.1. Swifter

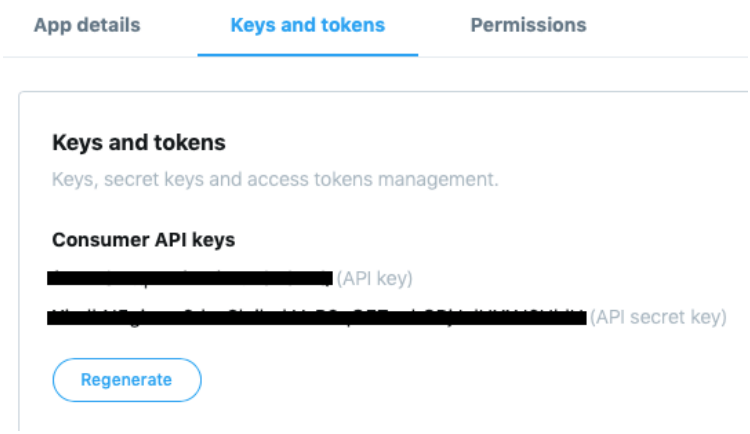
Swifter[6] je Twitter framework za iOS i OS X napisan u Swift programskom jeziku. Iako postoji framework za iOS napisan od strane Twitter-a, pokazao se kompliciranim za implementaciju te je nekolicina programera odlučila napraviti jednostavniji framework u želji da svima olakša korištenje Twitter API-a. Swifter projekt se nalazi na GitHub-u te ga je potrebno preuzeti i zatim uvesti Swifter Xcode projekt u Xcode projekt aplikacije. Nakon toga na *General* tab-u projekta pod *Embedded Binaries*, odaberemo SwifteriOS.framework (slika 19).



Slika 19: Uvoz SwifteriOS.framework datoteke u projekt

Uvezemo SwifteriOS framework u ViewController.swift. Inicijaliziramo novu konstantu *let swifter* kao instancu Swifter framework-a koja prima dva parametra: *consumerKey* i *consumerSecret*. Ti parametri predstavljaju Twitter API ključeve koji su generirani prilikom izrade nove Twitter aplikacije unutar Twitter Developer kontrolne ploče (slika 20).

[6] https://github.com/mattdonnelly/Swifter



Slika 20: Twitter API ključevi

API ključevi se nikada ne upisuju u tekstualnom formatu u sam kod aplikacije. Twitter API ključevi koji se naplaćuju, direktno su povezani sa kreditnom karticom osobe ili kompanije koja ih plaća. Problem se javlja prilikom korištenja GitHub-a i sličnih source control platformi, ako je kod aplikacije javno dostupan. Opća je praksa čuvati API ključeve na sigurnom mjestu. Iz tog razloga pokazat ćemo kako sakriti API ključeve. Napravimo novi dokument tipa *Property List* u projektu koji nazovemo ApiKeys.plist. Property List je jednostavan key-value tip dokumenta u koji upišemo nazive API ključeva i spremimo string ključa.

Umjesto upisivanja API ključeva u tekstualnom formatu u samom kodu, pozvat ćemo ih iz ApiKeys.plist dokumenta. Da bi to mogli napraviti potrebno je definirati funkciju koja će dohvatiti API ključeve iz ApiKeys.plist dokumenta te vratiti njihovu vrijednost prema zadanim parametrima. Funkciju valueForAPIKey(named:) sa parametrom koji prima naziv API ključa iz ApiKeys.plist datoteke definiramo unutar nove datoteke ApiKeys.swift (slika 21).

```
func valueForAPIKey(named keyname:String) -> String {
    let filePath = Bundle.main.path(forResource: "ApiKeys", ofType: "plist")
    let plist = NSDictionary(contentsOfFile:filePath!)
    let value = plist?.object(forKey: keyname) as! String
    return value
}
```

Slika 21: ApiKeys.swift datoteka

Umjesto upisivanja API ključa u tekstualnom formatu pozivamo funkciju valueForAPIKey(named:) (slika 22) i time onemogućavamo čitanje API ključa neautoriziranim osobama.

```
let swifter = Swifter(consumerKey: valueForAPIKey(named: "API_KEY"),
    consumerSecret: valueForAPIKey(named: "API_SECRET_KEY"))
```

Slika 22: Instanca Swifter framework-a sa Twitter API ključevima

Prilikom stavljanja koda na GitHub, izuzmemo ApiKeys.plist dokument tako da ga stavimo unutar .gitignore dokumenta koji Git-u daje informacije o tome koje dokumente treba izuzeti prilikom stavljanja koda na GitHub.

Za dohvaćanje tweet-ova koristit ćemo *searchTweet(using:)* funkciju Swifter klase. Obavezan parametar je *using* parametar koji predstavlja traženi upit, odnosno ukoliko tražimo tweet-ove koji sadrže @Apple, napisat ćemo string "@Apple" kao argument. Pokrenemo li kod, dobit ćemo tekst u JSON formatu (slika 23) svih tweet-ova koji sadrže "@Apple".


```
{
  "id" : 1.155854942260945e+18,
  "in_reply_to_user_id" : null,
  "retweet_count" : 0.0,
  "lang" : "en",
  "id_str" : "1155854942260944896",
  "source" : "<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for
     iPhone</a>",
  "contributors" : null,
  "metadata" : {
    "result_type" : "recent",
    "iso_language_code" : "en"
  },
  "full_text" : ".@Apple Why do we have the capability to send talking unicorn versions of
     ourselves to each other, but we cannot bold, italicize, or underline iMessages?",
  "in_reply_to_status_id_str" : null,
  "truncated" : false,
  "in_reply_to_user_id_str" : null,
  "place" : {
    "contained_within" : [

    ],
    "full_name" : "Dallas, TX",
    "place_type" : "city",
    "bounding_box" : {
      "coordinates" : [
        [
          [
```


Slika 23: Isječak json-a sa prikazom teksta tweet-a

Opcionalni parametri koje koristimo su *lang* parametar koji definira jezik na kojem tražimo tweet-ove, *count* parametar u kojem definiramo broj tweet-ova koje dohvaćamo, *tweetMode* koji definira duljinu tweet-ova koje dohvaćamo te *success* parametar koji sprema traženi rezultat i metapodatke u JSON formatu.

U našem slučaju argument koji prosljeđujemo *searchTweet(using:)* funkciji treba biti string koji se pročita iz textField elementa koji smo ranije definirali. To ćemo napraviti tako da inicijaliziramo novu konstantu *let searchText* kao string koji se nalazi u text polju textField elementa. Kako je *textField.text* opcionalan string, odnosno tekstualno polje za unos se može ostaviti prazno, stavit ćemo ga u if blok koji će se izvršiti samo ako textField polje nije prazno. Sada, umjesto string-a, u *using* parametar upišemo naziv konstante *searchText* (slika 24). Za opcionalni parametar *count* također ćemo definirati globalnu konstantu *let tweetCount*.

```
func getTweets() {
    if let searchText = textField.text {
        swifter.searchTweet(using: searchText, lang: "en", count:
            tweetCount, tweetMode: .extended, success: { (results,
            metadata) in

            //Kod

        }) { (error) in
            print("Twitter API Request error, \(error)")
        }
    }
}
```

Slika 24: Dohvaćanje tweet-ova sa Twitter-a

Uvozom gotovog modela strojnog učenja u projekt, Xcode je automatski generirao TweetalyzeSentimentClassifier klasu i sve potrebne funkcije te klase. U dokumentu klase pronađemo funkciju za predviđanje koja prima string parametar (slika 25). Prvo uvezemo CoreML framework i kreiramo novi objekt *let sentimentClassifier* navedene klase, kako bi mogli koristiti funkciju unutar koda.

```
    /**
        Make a batch prediction using the structured interface
        - parameters:
           - inputs: the inputs to the prediction as [TweetalyzeSentimentClassifierInput]
           - options: prediction options
        - throws: an NSError object that describes the problem
        - returns: the result of the prediction as [TweetalyzeSentimentClassifierOutput]
    */
    func predictions(inputs: [TweetalyzeSentimentClassifierInput], options:
        MLPredictionOptions = MLPredictionOptions()) throws ->
        [TweetalyzeSentimentClassifierOutput] {
        let batchIn = MLArrayBatchProvider(array: inputs)
        let batchOut = try model.predictions(from: batchIn, options: options)
        var results : [TweetalyzeSentimentClassifierOutput] = []
        results.reserveCapacity(inputs.count)
        for i in 0..<batchOut.count {
            let outProvider = batchOut.features(at: i)
            let result =  TweetalyzeSentimentClassifierOutput(features: outProvider)
            results.append(result)
        }
        return results
    }
}
```

Slika 25: Funkcija za predviđanje generirana prilikom uvoza modela u projekt

Da bi mogli predviđati mišljenje na temelju tweet-ova, moramo parsirati JSON koji dobijemo kao rezultat poziva Twitter API-a i iz njega izvući vrijednost svojstva *full_text* (slika 23) koji sadrži string tweet-a. Najjednostavniji način je koristeći SwiftyJSON framework.

4.4.2. SwiftyJSON

Može se koristiti izvorno svojstvo Swift-a koje uključuje dekodiranje JSON-a koristeći enkoder i dekoder, ali s obzirom da nam treba samo jedna informacija iz JSON-a, na ovaj način izbjegavamo izradu strukture i smanjujemo količinu koda. SwiftyJSON uvozimo preko CocoaPods-a. Preko terminala odemo u mapu u kojoj se nalazi projekt, sa **pod init** komandom kreiramo Podfile dokument u koji upišemo framework koji želimo koristiti (slika 26). Instaliramo ga sa **pod install** komandom.

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'Tweetalyze' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for Tweetalyze

pod 'SwiftyJSON'

end
```

Slika 26: Podfile dokument

Da bi koristili CocoaPods u projektu, umjesto Xcode projekta koji smo otvarali do sada, moramo otvoriti Xcode workspace projekt u kojem se nalaze svi dokumenti uključujući i novi Pods dokument. Deklariramo novi niz *var tweets* koji prima niz TweetalyzeSentimentClassifierInput ulaznih podataka. S obzirom da dohvaćamo 100 tweet-ova, unutar for petlje parsiramo *full_text* iz JSON-a, koji predstavlja tweet te svaki parsirani tweet dodajemo u niz koji smo prethodno deklarirali (slika 27). Tako dobiveni niz tweet-ova možemo koristiti u funkciji za predviđanje mišljenja.

```
var tweets = [TweetalyzeSentimentClassifierInput]()

for i in 0..<self.tweetCount {
    if let tweet = results[i]["full_text"].string {
        let tweetClassification =
            TweetalyzeSentimentClassifierInput(text: tweet)
        tweets.append(tweetClassification)
    }
}
```

Slika 27: Dio funkcije za dohvaćanje tweet-ova

4.5. Funkcija za predviđanje mišljenja

Deklariramo novu konstantu *let predictions*, u koju spremimo rezultat poziva funkcije *predictions(inputs:)*, kojoj kao argument prosljeđujemo niz tweet-ova koje smo prethodno parsirali iz JSON-a. Za rukovanje pogreškom (engl. *error handling)* moramo staviti kod u do-catch blok (slika 28).

```
do {
    let predictions = try self.sentimentClassifier.predictions(inputs: tweets)
} catch {
    print("There was an error with prediction, \(error)")
}
```

Slika 28: Do-catch blok za rukovanje pogreškom

Kako bi mogli prikazati mišljenje na korisničkom sučelju, za svih 100 tweet-ova, moramo implementirati bodovni sustav. Deklariramo novu varijablu *var sentimentScore* kojoj dodamo +1 za svaki pozitivno ocijenjen tweet, a oduzmemo -1 za svaki negativan tweet. Neutralno ocijenjeni tweet-ovi predstavljaju 0 pa ih ne računamo. Sve to stavimo u for petlju koja za svakoj predviđenoj oznaci, dodijeli određeni broj bodova te krajnji rezultat spremi u *var sentimentScore* varijablu (slika 29).

```
var sentimentScore = 0

for p in predictions {
    let sentiment = p.label
    if sentiment == "Pos" {
        sentimentScore += 1
    } else if sentiment == "Neg" {
        sentimentScore -= 1
    }
}
```

Slika 29: Bodovni sustav za prikaz prosječnog mišljenja

4.6. Funkcija za prikaz mišljenja na korisničkom sučelju

Umjesto prikaza broja spremljenog u *sentimentScore* varijabli, implementirat ćemo prikaz emotikona u skladu sa dobivenim bodovima. Emotikoni su Unicode simboli te ih prikazujemo na ekranu preko elementa tekstualnog polja koji smo definirali na početku unutar Main.storyboard dokumenta. Povezali smo element sa kodom te definirali varijablu *var sentimentLabel* (slika 30).

```
@IBOutlet weak var sentimentLabel: UILabel!
```

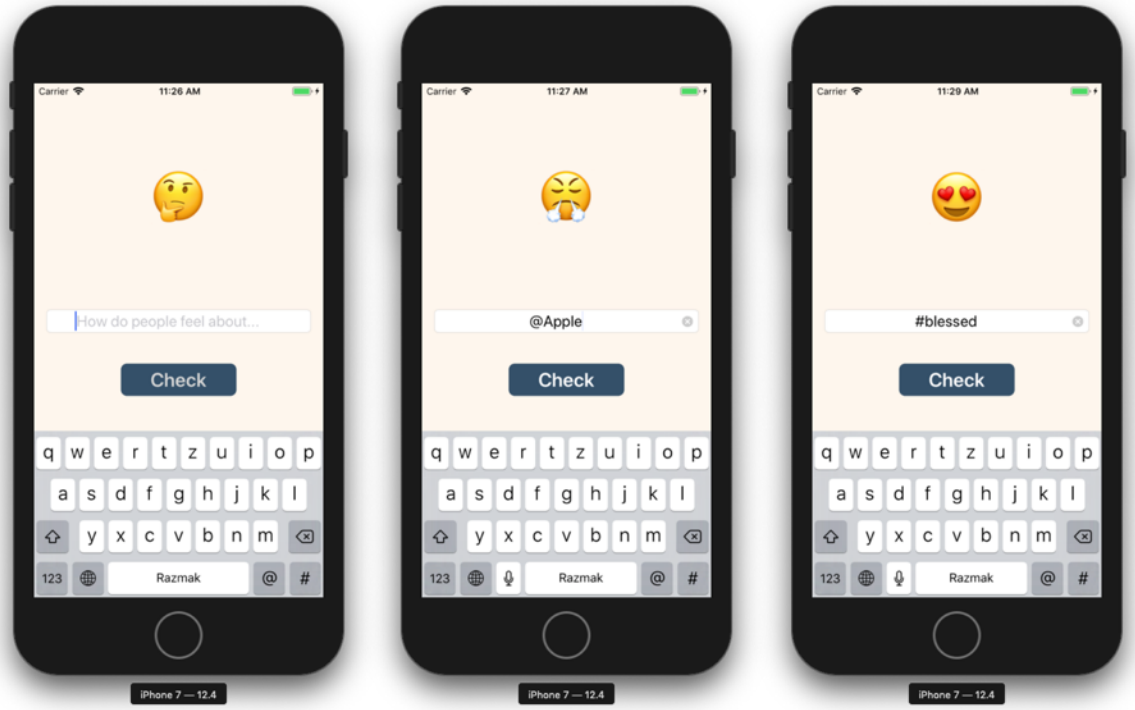
Slika 30: Generirana varijabla pilikom povezivanja elementa sučelja s kodom

Funkcija za prikaz mišljenja sadrži if-else petlju koja unutar svake grane uspoređuje *sentimentScore* varijablu sa danim bodovnim granicama za svaki emotikon. Pozivom *text* svojstva varijable *sentimentLabel* unutar petlje definiramo emotikon koji se prikazuje za određeni rezultat mišljenja (slika 31).

```
if sentimentScore > 20 {
    self.sentimentLabel.text = "😍"
} else if sentimentScore > 10 {
    self.sentimentLabel.text = "😃"
} else if sentimentScore > 0 {
    self.sentimentLabel.text = "🙂"
} else if sentimentScore == 0 {
    self.sentimentLabel.text = "😐"
} else if sentimentScore > -10 {
    self.sentimentLabel.text = "🙁"
} else if sentimentScore > -20 {
    self.sentimentLabel.text = "😤"
} else {
    self.sentimentLabel.text = "😡"
}
```

Slika 31: Funkcija za prikaz mišljenja putem emotikona

Sve što nam preostaje je pokrenuti kod i testirati mišljenje na temelju string-a koji upišemo u element unosa teksta (slika 32). Na kraju, umjesto spaghetti koda koji smo napisali, refaktoriramo kod u funkcije, pozovemo ih na pravom mjestu te dobijemo završni kod (vidi Dokumentaciju).



Slika 32: Prikaz rada aplikacije

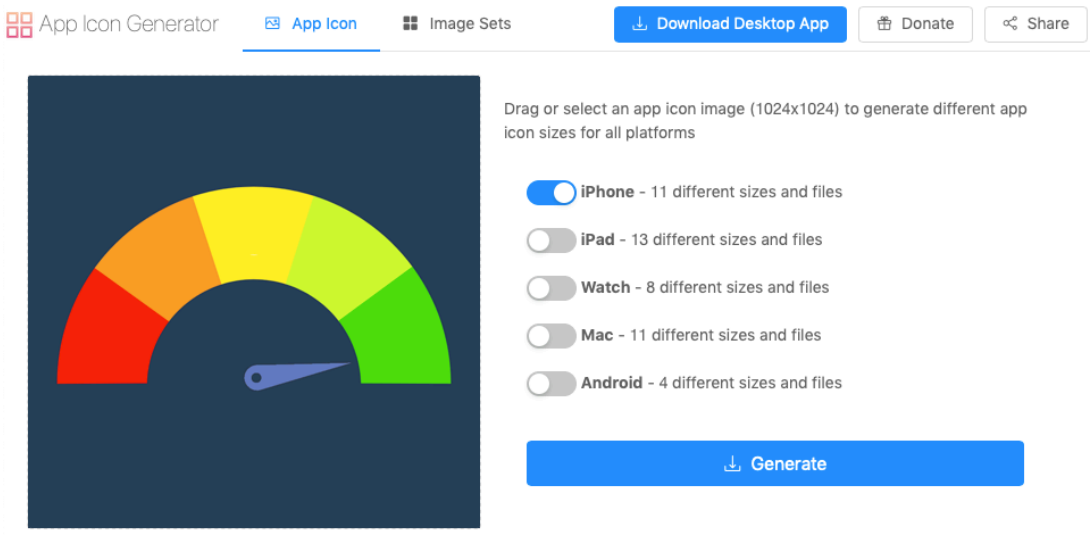
4.7. Završne postavke

Završne postavke na aplikaciji koje nam nedostaju su dodavanje ikone aplikacije i uređivanje zaslona prilikom pokretanja aplikacije (u daljnjem tekstu, engl. *launch screen*). Osim toga, u glavnim postavkama Xcode projekta, orijentaciju ekrana ograničimo na portretni način. Također odabiremo razvoj (engl. *deployment)* aplikacije za iPhone, jer nam nije plan razvijati aplikaciju i za iPad uređaje.

4.7.1. Ikona aplikacije

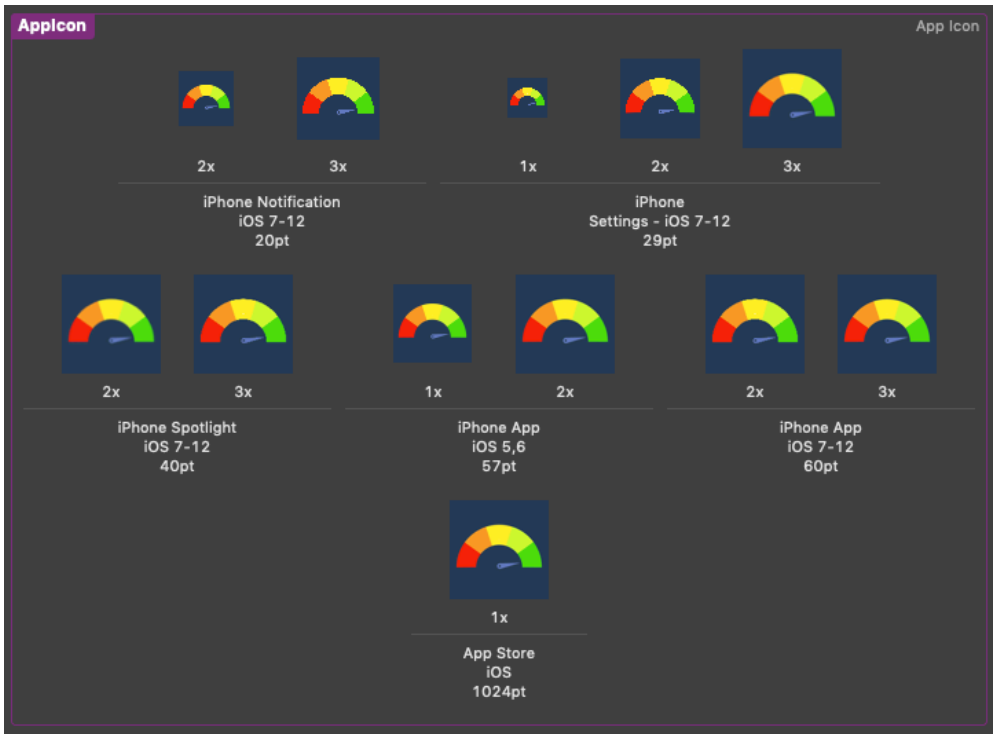
Najjednostavniji način izrade ikone aplikacije je preko stranice Canva[7] koja sadrži različite predloške dizajna. Dizajniramo novu PNG sliku veličine 1024x1024 piksela kao preporučenu dimenziju za ikonu aplikacije. Napravljenu sliku zatim uvezemo u web stranicu za automatsko generiranje ikona[8] (slika 33).

[7] https://www.canva.com
[8] https://appicon.co



Slika 33: Generiranje ikone za Xcode projekt

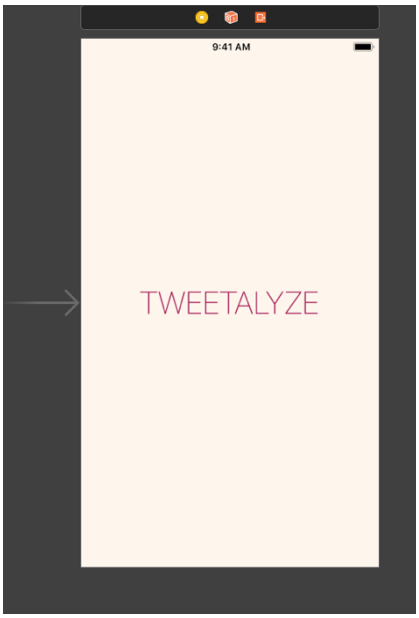
Generiramo slike za iPhone i web stranica će automatski imenovati ikone prema Apple-ovom standardu unutar Assets.xcassets datoteke. Sve što treba napraviti je zamijeniti već postojeću datoteku sa novom generiranom koja sadrži sve potrebne dimenzije ikone aplikacije (slika 34).



Slika 34: Uvoz generiranih ikona aplikacije u Xcode projekt

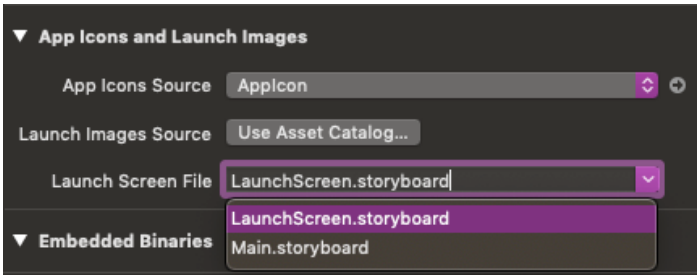
4.7.2. LaunchScreen.storyboard

Izrada launch screen-a aplikacije je obavezan dio izrade same aplikacije. Apple u dokumentaciji upozorava da izrada launch screen-a nije prilika za umjetničko izražavanje, nego je namijenjen povećanju percepcije bržeg otvaranja aplikacije. Preporuka je izbjegavati logo, slike pa čak i tekst, ali malo se aplikacija drži tog načela. Launch screen bi trebao biti identičan početnom zaslonu same aplikacije. Napravit ćemo jednostavan launch screen sa tekstom imena aplikacije unutar LaunchScreen.storyboard dokumenta koji je predviđen upravo za tu namjenu i već se nalazi unutar projekta (slika 35).



Slika 35: Launch screen aplikacije

Sve što nakon toga treba napraviti je, u općim informacijama projekta, namjestiti ime dokumenta koji sadrži launch screen (slika 36).



Slika 36: Postavljanje otvaranja aplikacije na launch screen datoteku

5. Dodatne funkcionalnosti

5.1. Ograničenje pogrešnog unosa - regularni izraz

Prilikom unosa string-a u polje za unos, javlja se problem ukoliko napišemo nevažeći string. Želimo da unos bude ograničen tako da je upit važeći samo kada se upiše "@" ili "#" na prvom mjestu, a zatim mala i velika slova ili brojevi. To možemo napraviti implementirajući regularni izraz koji će ograničiti unos teksta na točno ono što želimo.

Napravimo novu datoteku unutar projekta naziva String+Regex.swift. Ovaj način imenovanja standardan je za dodavanje novih funkcionalnosti već postojećoj klasi, strukturi, enumeraciji ili protokolu, tzv. ekstenzija (engl. *extension*). Iz imena je vidljivo da dodajemo regularni izraz String strukturi (slika 37).

```
import UIKit

extension String {
    var isValidInput: Bool {
        let checkUserInputRegex = "^[@][A-z0-9]+|^[#][A-z0-9]+"
        return NSPredicate(format: "SELF MATCHES %@",
            checkUserInputRegex).evaluate(with: self)
    }
}
```

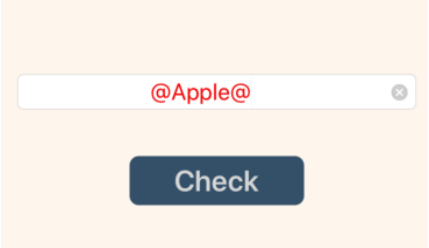
Slika 37: Ograničavanje unosa teksta regularnim izrazom

Iduća stvar koju treba napraviti je povezati element unosa teksta sa ViewController.swift dokumentom tako da kreiramo novu funkciju testFieldEditingDidChange(sender:) (slika 38).

```
@IBAction func textFieldEditingDidChange(_ sender: UITextField) {
    if let userInput = sender.text, userInput.isValidInput == true {
        textField.textColor = .black
        return checkPressed.isEnabled = true
    } else {
        textField.textColor = .red
        return checkPressed.isEnabled = false
    }
}
```

Slika 38: Pozivanje funkcije za ograničenje unosa teksta i onemogućavanje gumba

Unutar generirane funkcije, koja se poziva ukoliko se tekst unutar polja za unos teksta promijeni, inicijaliziramo novu konstantu *let userInput* kao *tekst* svojstvo UITekstField klase. Pozivamo *isValidInput* varijablu i provjeravamo vraća li *true* ili *false.* Ako vraća *true*, omogućujemo gumb *Check* koji pokreće analiziranje string-a ili ako vraća *false* i upisani string ne zadovoljava zadani regularni izraz, gumb *Check* za analiziranje unosa će biti onemogućen dok god uneseni tekst ne bude ispravan (slika 39). Kao dodatni pokazatelj krivog unosa, boja krivo unesenog teksta je crvene boje dok god uneseni tekst ne zadovolji regularni izraz. Također, gumb *Check* dok god je u onemogućenom stanju ima tekst u sivoj boji.



Slika 39: Prikaz krivog unosa na korisničkom sučelju

5.2. Pokazatelj aktivnosti

Prilikom testiranja aplikacije, dogodilo se da se za drugačiji pojam pretrage prikazalo isto mišljenje nekoliko puta za redom bez naznake da se išta dogodilo. Isto mišljenje ne znači krivo mišljenje, ali ne postoji indikacija da se dogodila pretraga jer isti emotikon cijelo vrijeme stoji na ekranu. Iz tog razloga, implementiramo pokazatelj aktivnosti (engl. *activity indicator*), ikonu koja se prikaže preko ekrana prilikom pretrage i učitavanja podataka.

To ćemo također napraviti koristeći ekstenziju, pa napravimo novi dokument unutar projekta kojeg nazovemo UIViewController+ActivityIndicator.swift. Ekstenzija dodaje novu funkcionalnost UIViewController klasi koja upravlja pogledima aplikacije. Dodajemo kod koji kreira pokazatelj aktivnosti te upravlja njegovim prikazom na ekranu unutar pogleda (slika 40).

```
var vSpinner : UIView?

extension UIViewController {
    func showSpinner(onView : UIView) {
        let spinnerView = UIView.init(frame: onView.bounds)
        spinnerView.backgroundColor = UIColor.init(red: 0.5, green: 0.5,
            blue: 0.5, alpha: 0.5)
        let ai = UIActivityIndicatorView.init(style: .whiteLarge)
        ai.startAnimating()
        ai.center = spinnerView.center

        DispatchQueue.main.async {
            spinnerView.addSubview(ai)
            onView.addSubview(spinnerView)
        }

        vSpinner = spinnerView
    }

    func removeSpinner() {
        DispatchQueue.main.async {
            vSpinner?.removeFromSuperview()
            vSpinner = nil
        }
    }
}
```

Slika 40: Dio koda za pokazatelj aktivnosti

Pozivamo *showSpinner(onView:)* funkciju unutar ViewController.swift datoteke jer želimo da se pokazatelj aktivnosti prikazuje na glavnom pogledu prilikom pretraživanja pritiskom na gumb *Check*. Stoga, dodajemo dio koda unutar funkcije *checkPressed(sender:)* (slika 41).

```
@IBAction func checkPressed(_ sender: Any) {
    self.showSpinner(onView: self.view)
    getTweets()
}
```

Slika 41: Pozivanje funkcije za prikaz pokazatelja aktivnosti

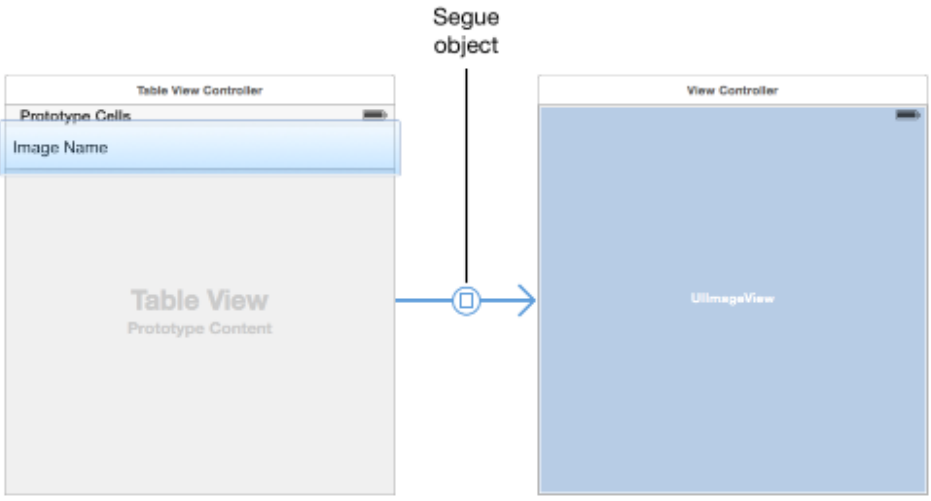
Implementiramo deaktivaciju pokazatelja aktivnosti pozivom funkcije *removeSpinner()* unutar funkcije za predviđanje mišljenja. Funkciju pozivamo prije nego se prikaže emotikon na ekranu, odnosno prije poziva funkcije *showSentimentUI(with:)* (slika 42).

```
self.removeSpinner()
showSentimentUI(with: sentimentScore)
```

Slika 42: Pozivanje funkcije za deaktivaciju pokazatelja aktivnosti

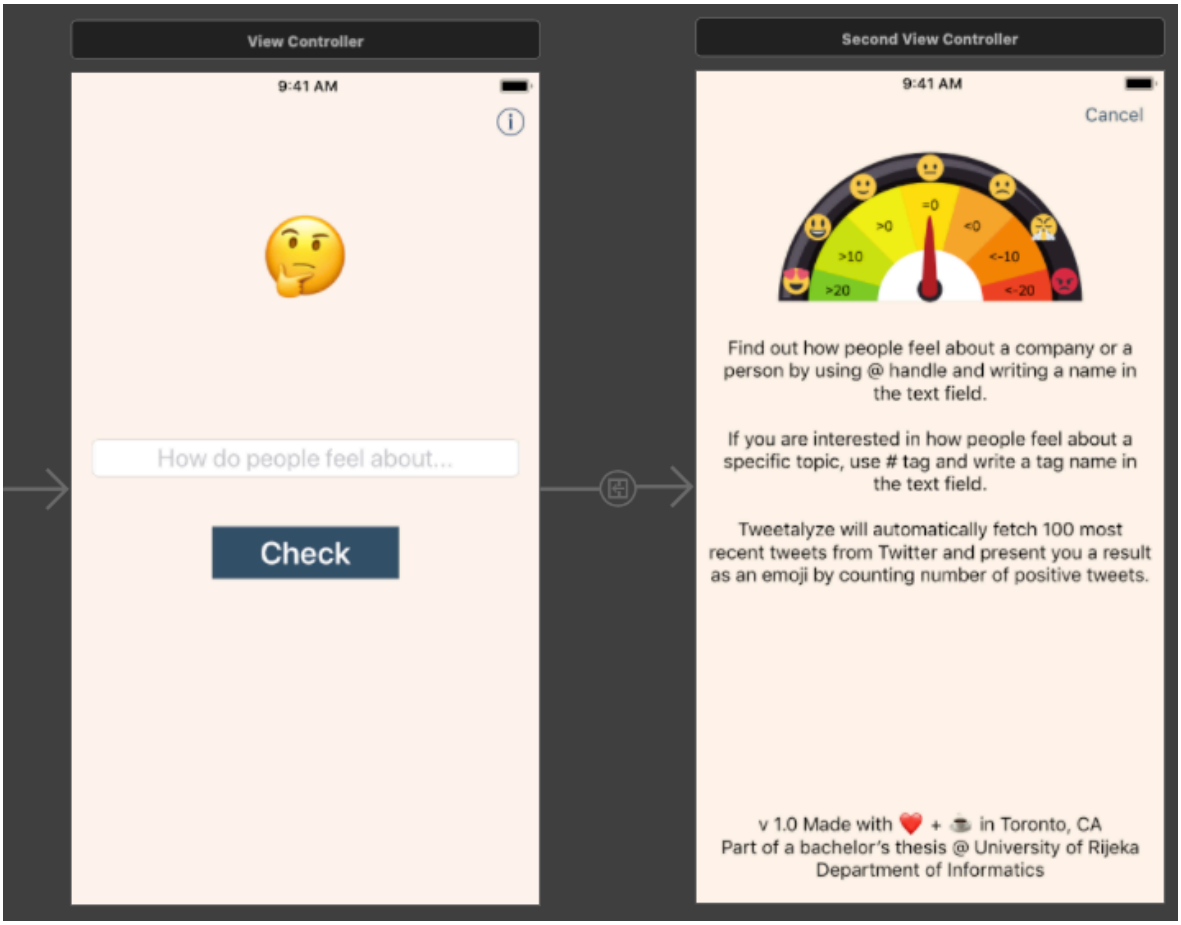
5.3. Gumb za informacije i novi pogled

Ako itko osim samog programera otvori aplikaciju, iz samog početnog ekrana nije jasna smisao i način korištenja aplikacije. Kreirat ćemo novi pogled aplikacije unutar Main.storyboard dokumenta i novi dokument SecondViewController.swift koji će sadržavati sav kod vezan za drugi ekran aplikacije. Kreirali smo dva pogleda, no moramo ih nekako spojiti da bi iz glavnog pogleda došli do pogleda koji sadrži informacije. Koristit ćemo segue. Segue je način spajanja dva ili više pogleda preko gumba, definiranjem načina tranzicije, odnosno sa kojeg pogleda preko segue objekta prelazimo na koji pogled (slika 43).



Slika 43: Kreiranje segue objekta

Kreirat ćemo novi sistemski gumb tipa Info Light koji je predefinirani gumb za informacije unutar iOS sustava. Preko gumba kreiramo segue pomoću kojeg definiramo prijelaz s jednog ekrana na drugi. Unutar drugog ekrana definiramo novi gumb *Cancel* koji će zatvoriti trenutni pogled i vratiti nas natrag na glavni pogled (slika 44).



Slika 44: Prikaz gotove Tweetalyze aplikacije

U ovom trenutku, izradili smo gotovu aplikaciju za analizu mišljenja na Twitter-u, sa ograničenjem unosa teksta i novim ekranom sa informacijama o načinu korištenja aplikacije koja koristi vlastiti model strojnog učenja.

6. Zaključak

Izrada modela strojnog učenja u Apple-ovom eko sustavu za nekoga tko se nije nikada susreo sa sličnim je dosta jednostavna. Model se trenira i testira unutar samog Playground-a što eliminira potrebu za izradom aplikacije. Nakon što je zadovoljen postotak točnosti, trenirani model se jednostavno izveze i uveze u novu ili postojeću aplikaciju za daljnje korištenje.

Problem koji se ukazuje kod analize mišljenja je subjektivnost interpretacije teksta od strane ljudi te sarkazam, koji se na različite načine analizira. Netko pročita tekst kao pozitivan, a netko će za isti tekst reći da je negativan. Računalo se trenira na metapodatcima koje su napravili ljudi, ali samo ne razumije ni prirodni jezik ni sarkazam.

Postotak uspješnosti treniranja modela potencijalno se može povećati treniranjem na većem broju metapodataka ili iskušavanjem drugih metoda strojnog učenja.

7. Popis slika

8. Literatura

1. Apple Inc., https://www.britannica.com/topic/Apple-Inc, 21. svibanj 2019.
2. Swift, https://swift.org, 22. svibanj 2019.
3. Create emotion with color in UX design, https://uxplanet.org/create-emotion-with-color-in-ux-design-446a3766b085, 23. svibanj 2019.
4. Color hunt, https://colorhunt.co, 23. svibanj 2019.
5. SwiftUI, https://developer.apple.com/xcode/swiftui/, 18. srpanj 2019.
6. Xcode, https://developer.apple.com/xcode/ide/, 18. srpanj 2019.
7. Twitter Inc., https://www.britannica.com/topic/Twitter, 18. srpanj 2019.
8. Hrvatski jezični portal, http://hjp.znanje.hr/, 19. srpanj 2019.
9. Što je to „machine learning“ ili strojno učenje?, https://pcchip.hr/helpdesk/sto-je-to-machine-learning-ili-strojno-ucenje/, 19. srpanj 2019.
10. Machine Learning, https://developer.apple.com/machine-learning/, 19. srpanj 2019.
11. Introduction to Machine Learning on mobile, https://medium.com/@dmennis/introduction-to-machine-learning-on-mobile-36845619c56, 19. srpanj 2019.
12. Understand Core ML on iOS in 5 minutes, https://medium.com/@dmennis/understand-core-ml-on-ios-in-5-minutes-bc8ba5411a2d, 19. srpanj 2019.
13. Create ML, https://developer.apple.com/documentation/createml, 19. srpanj 2019.
14. Natural Language Processing (NLP) for Machine Learning, https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b, 19. srpanj 2019.
15. Human Interface Guidelines, https://developer.apple.com/design/human-interface-guidelines/ios/overview, 22. srpanj 2019.
16. CocoaPods, https://github.com/CocoaPods/CocoaPods, 25. srpanj 2019.

9. Dokumentacija

```
//  ViewController.swift
//  Tweetalyze
//
//  Created by Danijela Vrzan on 2019-07-24.

import UIKit
import SwifteriOS
import CoreML
import SwiftyJSON

class ViewController: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var sentimentLabel: UILabel!
    @IBOutlet weak var checkPressed: UIButton!{
        didSet{
            checkPressed.layer.cornerRadius = 8
        }
    }

    let tweetCount = 100
    let sentimentClassifier = TweetalyzeSentimentClassifier()

    let swifter = Swifter(consumerKey: valueForAPIKey(named: "API_KEY"),
     consumerSecret: valueForAPIKey(named: "API_SECRET_KEY"))

    override func viewDidLoad() {
        super.viewDidLoad()
        checkPressed.isEnabled = false
    }

    @IBAction func textFieldEditingDidChange(_ sender: UITextField) {
        if let userInput = sender.text, userInput.isValidInput == true {
            textField.textColor = .black
            return checkPressed.isEnabled = true
        } else {
            textField.textColor = .red
            return checkPressed.isEnabled = false
        }
    }

    @IBAction func helpPressed(_ sender: Any) {
        performSegue(withIdentifier: "goToSecondScreen", sender: self)
    }

    @IBAction func checkPressed(_ sender: Any) {
        self.showSpinner(onView: self.view)
        getTweets()
    }
```

```
    //Funkcija za dohvaćanje tweetova

    func getTweets() {
        if let searchText = textField.text {
            swifter.searchTweet(using: searchText, lang: "en", count:
     tweetCount, tweetMode: .extended, success: { (results, metadata) in

                var tweets = [TweetalyzeSentimentClassifierInput]()

                for i in 0..<self.tweetCount {
                    if let tweet = results[i]["full_text"].string {
                        let tweetClassification =
     TweetalyzeSentimentClassifierInput(text: tweet)
                        tweets.append(tweetClassification)
                    }
                }
                self.predictSentiment(with: tweets)
            }) { (error) in
                print("Twitter API Request error, \(error)")
            }
        }
    }


    //Funkcija za predviđanje mišljenja

    func predictSentiment(with tweets:
[TweetalyzeSentimentClassifierInput]) {
        do {
            let predictions = try
     self.sentimentClassifier.predictions(inputs: tweets)

            var sentimentScore = 0

            for p in predictions {
                let sentiment = p.label
                if sentiment == "Pos" {
                    sentimentScore += 1
                } else if sentiment == "Neg" {
                    sentimentScore -= 1
                }
            }
            self.removeSpinner()
            showSentimentUI(with: sentimentScore)
        } catch {
            print("There was an error with prediction, \(error)")
        }
    }
```

```
    //Funkcija za prikaz mišljenja na korisničkom sučelju

    func showSentimentUI(with sentimentScore: Int) {
        if sentimentScore > 20 {
            self.sentimentLabel.text = "😍"
        } else if sentimentScore > 10 {
            self.sentimentLabel.text = "😃"
        } else if sentimentScore > 0 {
            self.sentimentLabel.text = "🙂"
        } else if sentimentScore == 0 {
            self.sentimentLabel.text = "😐"
        } else if sentimentScore > -10 {
            self.sentimentLabel.text = "😕"
        } else if sentimentScore > -20 {
            self.sentimentLabel.text = "😤"
        } else {
            self.sentimentLabel.text = "😡"
        }
    }

}


//  SecondViewController.swift
//  Tweetalyze
//
//  Created by Danijela Vrzan on 2019-07-30.

import UIKit

class SecondViewController: UIViewController {

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

    @IBAction func backPressed(_ sender: Any) {
        self.dismiss(animated: true, completion: nil)
    }
}


//  ApiKeys.swift
//  Tweetalyze
//
//  Created by Danijela Vrzan on 2019-07-29.

import Foundation

func valueForAPIKey(named keyname:String) -> String {
    let filePath = Bundle.main.path(forResource: "ApiKeys", ofType:
     "plist")
    let plist = NSDictionary(contentsOfFile:filePath!)
    let value = plist?.object(forKey: keyname) as! String
    return value
}
```

```swift
//  String+Regex.swift
//  Tweetalyze
//
//  Created by Danijela Vrzan on 2019-08-06.

import UIKit

extension String {
    var isValidInput: Bool {
        let checkUserInputRegex = "^[@][A-z0-9]+|^[#][A-z0-9]+"
        return NSPredicate(format: "SELF MATCHES %@",
     checkUserInputRegex).evaluate(with: self)
    }
}



//  UIViewController+ActivityIndicator.swift
//  Tweetalyze
//
//  Created by Danijela Vrzan on 2019-08-06.

import UIKit

var vSpinner : UIView?

extension UIViewController {
    func showSpinner(onView : UIView) {
        let spinnerView = UIView.init(frame: onView.bounds)
        spinnerView.backgroundColor = UIColor.init(red: 0.5, green: 0.5,
     blue: 0.5, alpha: 0.5)
        let ai = UIActivityIndicatorView.init(style: .whiteLarge)
        ai.startAnimating()
        ai.center = spinnerView.center

        DispatchQueue.main.async {
            spinnerView.addSubview(ai)
            onView.addSubview(spinnerView)
        }

        vSpinner = spinnerView
    }

    func removeSpinner() {
        DispatchQueue.main.async {
            vSpinner?.removeFromSuperview()
            vSpinner = nil
        }
    }
}
```