



SWIFT İLE iOS UYGULAMA GELİŞTİRME: YENİ BAŞLAYANLAR İÇİN



İHSAN EMRE EROL

İAÜ YENİ MEDYA UYGULAMA VE ARAŞTIRMA MERKEZİ

SWIFT İLE IOS UYGULAMA GELİŞTİRME: YENİ BAŞLAYANLAR İÇİN

İHSAN EMRE EROL

2019
İstanbul Aydin Üniversitesi Yayınları
Yeni Medya Uygulama ve Araştırma Merkezi Yayınları

İstanbul Aydin Üniversitesi Yayınları

**İAÜ YENİ MEDYA UYGULAMA VE
ARAŞTIRMA MERKEZİ**

SWIFT İLE İOS PROGRAMLAMA: YENİ BAŞLAYANLAR İÇİN

Yazar: İhsan Emre EROL

Yayın Kurulu: Prof.Dr. Hülya YENGİN (İAÜ İletişim Fakültesi Dekanı)

Prof.Dr. Özden CANKAYA (İAÜ İletişim Fakültesi Öğretim Üyesi)

Doç.Dr. Deniz YENGİN (İAÜ Yeni Medya Uygulama ve Araştırma Merkez Müdürü)

Kapak ve Sayfa Tasarım: İstanbul Aydin Üniversitesi Görsel Tasarım Koordinatörlüğü

Basım Yılı: 2019

Baskı: I

Basım Yeri: Armoni Nüans A.ş. Yukarı Dudullu Bostancı Yolu Cad. Keyap Sitesi No. 24
Ümraniye/İstanbul

E-ISBN: 978-975-2438-66-8

Copyright © İstanbul Aydin Üniversitesi

Bu yapıtıñ tüm hakları saklıdır. Yazilar ve görsel malzeme izin almadan tümüyle veya
kısмен yayımlanamaz.

Bu kitabın tüm hakları

İstanbul Aydin Üniversitesi'ne aittir.

SWIFT İLE IOS UYGULAMA GELİŞTİRME: YENİ BAŞLAYANLAR İÇİN

İçindekiler

İçindekiler	4
ÖNSÖZ	8
1. Mobil Uygulama Nedir ?.....	15
2. Mobil Uygulama Platformları	15
3. Swift Programlama Dili.....	16
4. XCODE	16
5. Storyboard	17
6. Asset Ve Kullanımlar	18
7. Segue Bağlantıları İle Sayfalar Arası Geçişler	19
8. AutoLayout.....	21
9. Algoritma Kavramı ve Algoritma Oluşturma	25
10. Değişkenler.....	27
a. Değişken Tipleri.....	27
i. String Veri Tipleri (Metinsel Veriler)	27
ii. Character Veri Tipleri (Metinsel Karakter Veriler)	27
iii. Integer Veri Tipleri (Tam Sayı Veriler)	28
iv. Float ve Double (Reel Sayı Veri Tipleri)	28
v. Bool Veri Tipleri (Mantıksal Veri Tipleri)	28
b. Tip Dönüşümleri	29
i. Tip Dönüşümleri – Sayısaldan (Integer) Metinsele (String) Dönüşüm	29
ii. Tip Dönüşümleri – Metinselden (String) Sayısala (Integer) Dönüşüm.....	29
c. Sabit Tanımlama.....	30
11. Diziler	30
a. Dictionary Diziler	32
12. Operatörler	33
a. Aritmetik Operatörler.....	33
b. Mantıksal Operatörler	34
13. Kontrol Yapıları	35
a. if Kontrol Yapısı	35
b. Switch-Case Yapısı	36
14. Döngüler	38
a. For In Döngüsü.....	38
b. While Döngüsü.....	39
c. Repeat – While Döngüsü.....	40

15. Fonksiyonlar	41
a. Temel Fonksiyonlar	42
b. Parametreli Fonksiyonlar	43
c. Geriye Veri Dönən Fonksiyonlar.....	44
d. Değişken (Belirsiz) Sayıda Parametre Alan Fonksiyonlar	46
e. Nested (İçiçe) Fonksiyonlar	48
16. Enumeration Kullanımı	49
17. Hata Yakalama ve Hata Yönetimi.....	50
18. Sınıf (Class) Yapıları ve Nesnel Programlama	52
a. Sınıf Yapıları (Class)	53
b. Özellikler ve Fonksiyonlar (Methods)	54
i. Yapıcı Metotlar - Constructors	54
ii. Yıkıcı Metotlar.....	54
c. Kalıtım (Inheritances).....	55
d. Kapsülleme (Encapsulation)	57
e. Soyutlama (Abstraction)	58
i. Protokoller ve Protocol Oriented Programlama	59
f. Çok Biçimlilik (Polymorphism)	61
19. Structure Yapıları	62
20. IBOulet ve IBAction Kavramları	64
21. Viewler	67
a. UIView	67
b. Label.....	68
c. Button	69
d. TextField.....	71
e. ImageView	72
f. ScrollView	73
g. Switch.....	74
h. Progress View.....	75
i. Segmented Control	75
j. Picker View.....	76
k. Date Picker	82
l. TableView	84
22. Extensionlar	90
23. View Animasyonları	91
a. Belirlenen Sürede Değişen Sabitler.....	92
b. Belirlenen Sürede Değişen Alpha Değeri ve Zincirleme Animasyon.....	95

24. ViewController Yaşam Döngüsü.....	97
a. init Fonksiyonu	98
b. loadView Fonksiyonu.....	98
c. viewDidLoad Fonksiyonu.....	99
d. viewDidAppear Fonksiyonu.....	99
e. viewWillAppear Fonksiyonu.....	99
f. viewWillDisappear Fonksiyonu	99
g. viewDidDisappear Fonksiyonu	100
h. viewDidUnload Fonksiyonu.....	100
i. didReceiveMemoryWarning Fonksiyonu	100
25. UIAlertController İle Açıılır Pencereler	101
26. Dosya İşlemleri	105
a. Dosya Yazma İşlemi.....	106
b. Dosya Okuma İşlemi.....	107
c. Dizin Listeleme İşlemi	107
d. Dosya Silme İşlemi.....	108
27. Network İşlemleri	109
a. Http Get İsteği.....	109
b. Http Post İsteği.....	111
c. Bir İmaj Dosyasını Okuma	112
28. Konum Tabanlı İşlemler	114
29. Cihaz Donanım İşlemleri	121
a. Kamera Erişimi ve Kullanımı.....	121
b. Mikrofon Erişimi ve Kullanımı.....	127
c. Gyroscope Erişimi ve Kullanımı.....	132
d. Manyetometre Değerlerinin Ölçümü.....	135
e. Barometre İle Rakım (Yükseklik) Tespiti	136
30. Kullanıcı Bildirimleri	139
31. WebServis Kavramı ve WebServis Kullanım Alanları	142
a. WebServis Nedir.....	142
b. RestFull Api ve İletişim Protokolleri	144
32. Harici Kütüphaneler ve Cocoapods Kullanımı.....	144
a. Harici Kütüphane Nedir ?	144
b. Paket Yöneticisi ~ Bağımlılık Yöneticisi Nedir ?	145
c. Cocoapods Kurulumu	145
d. Cocoapods İle Projeye Harici Kütüphane Dahil Edilimi	146
33. Örnek Uygulama – Yemek Tarifleri Uygulaması.....	150
a. Hazırlık.....	150

b. Projenin Oluşturulması ve Tasarımın Hazırlanması.....	151
c. Kodlama Süreci	157
34. Resimler Dizini.....	167
35. Yazım Kuralları Dizini.....	174
36. Kare Kodlar Dizini.....	174
Kaynaklar	176

ÖNSÖZ

İletişim; duygusal, düşünce ve bilgilerin akla gelebilecek herhangi bir yolla kaynaktan hedefe aktarılmasıdır. Bireyler tarafından üretilen, paylaşılan ve kitlelere sunulan iletisimler iletişim sürecini dinamikleştirmiştir. Bu dinamik yapı ise teknolojiyi arkasına alarak devamlı kendini yenilemektedir. Analog sistemlerin hızla dijitalleşmesi toplumsal iletişim reflekslerini de dönüştürmektedir. Hatta artık dijital olan bile yeni değil, güncel olarak nitelendirilmektedir.

Sanal dünya, son dönemlerde en çok tartışılan konulardan biridir. Yeni medya, sanal dünya ve gerçek dünya arasında bir köprü görevi üstlenerek toplumun yaşadığı iletişim ortamına yön vermektedir. Fakat, bireylerin sahip oldukları sanal dünyaları dijitalleşmeyle beraber toplumun da birincil uzamı durumuna gelmiştir. Huizinga'nın "*oyun, kültürden eskidir*" sözü çerçevesinde baktığımızda sürekli bir oyun içinde mücadele verdığımızı görmekteyiz. Bu oynanan oyunlar kimi zaman ekonomi, kimi zaman küreselleşme, kimi zaman da dijitalleşme oyunları olmaktadır. Bu noktada sürekli kendilerini yenileyen oyunların oyuncuları hep aynı kalmakta, tek değişken teknoloji ve teknolojinin toplumsal kültürü biçimlendirme şeklidir.

Dijitalleşmeyle beraber bireylerin birincil uzamı haline gelen sanal dünyada bireyler dijital kimliğiyle, kamusal alan sınırları içinde üretilen kültür doğrultusunda dokunarak yaşamını sürdürmektedir. Gerçekler kopyalanarak çevremizi sarmaktadır. J. Baudrillard'ın sözünü ettiği gibi, kopyalar gerçeklerin yerini almaktadır ve sanal bir dünya varlığını sürdürmektedir. Yaşanılan dünya simülé edilmekte ve kopyalanmaktadır. Bireyler de bu noktada sınırların olmadığı yaşam alanlarında gerçek ve kopya arasındaki sınırlardan habersiz sanal bir yaşam sürdürmektedirler. Gerçek dünya, dijital değişimle birlikte sanal dünya evrilmeye ve dijitalin belirleyici olma gücüyle gerçek ve sanal dünya iç içe geçmektedir.

Toplumsal kültürümüz dijitalleşmeyle birlikte kendini güncelemekte ve yenilemektedir. Yenilemelerle birlikte yeni mecralar ve buna bağlı iletişimsel

eylemler gelişmektedir. Bu noktada medya dijital bir dili temel alarak yoluna devam etmektedir. Burada üretilen iletişim pratikleri de yaşamı kolaylaştırmak amacını taşımaktadır. Bu noktadan hareketle "Swift ile Ios Uygulama Geliştirme: Yeni Başlayanlar İçin" çalışmasıyla günümüzün mobil teknolojilerinde kullanılan aplikasyonların geliştirilmesi adına eğitici bir ders kitabı hazırlanmıştır. Bu çalışmanın; ios sisteminde uygulama geliştirmek isteyenlere rehber olması amaçlanmakta ve yeni medya müfredatlarında verilen eğitimlerde kullanılması hedeflenmektedir.

İstanbul Aydin Üniversitesi Yeni Medya Uygulama ve Araştırma Merkezi bünyesinde yapılan bu çalışmada İstanbul Aydin Üniversitesi Öğretim Üyesi **Prof. Dr. Hasan Saygın katkı ve desteklerinden** dolayı teşekkür ederiz. IAÜ Yeni Medya Uygulama ve Araştırma Merkezi olarak teknoloji ve toplum ilişkisi bağlamında yaşanan, yaşanacak fayda ve zararlar incelenmeye ve bunlarla ilgili analizler devam edecektir. Kitabın yararlı olması ve araştırmaların devamı dileğiyle...

Doç. Dr. Deniz YENGİN

İstanbul Aydin Üniversitesi

Yeni Medya Uygulama ve Araştırma Merkezi

İstanbul, 2018

Yeni Medya Uygulama ve Araştırma Merkezi'nin Kuruluş Amacı, Mision ve Vizyonu...

Yeni Medya Uygulama ve Araştırma Merkezi; 2017 yılında İstanbul Aydın Üniversitesi bünyesinde, Doç. Dr. Deniz YENGİN'in öncülüğünde kurulmuştur.

Son yıllarda iletişim alanında teknolojinin gelişmesi ve çeşitlenmesiyle birlikte, "yeni medya"ya bağlı teknolojiler dünyada önemli bir endüstri koluna dönüşmüştür. Dolayısıyla yeni medya bağlamında akademik yaklaşım, kuram ve uygulama alanlarının geliştirilmesi gereksinimi doğmuştur. İAÜ Yeni Medya Uygulama ve Araştırma Merkeziyle de bu gereksinimin karşılanması amaçlanmaktadır.

İAÜ Yeni Medya Uygulama ve Araştırma Merkezi'nde bu doğrultuda bilimsel araştırmaların yapılması ve alanda yeniliklere katkı sunulması amaçlanmıştır. Son yıllarda iletişim alanında teknolojinin gelişmesi ve çeşitlenmesiyle birlikte, "yeni medya"ya bağlı teknolojiler dünyada önemli bir endüstri koluna dönüşmüştür. Dolayısıyla yeni medya bağlamında akademik yaklaşım, kuram ve uygulama alanlarının geliştirilmesi gereksinimi doğmuştur. İAÜ Yeni Medya Uygulama ve Araştırma Merkezi ile de bu gereksinimin karşılanması amaçlanmaktadır.

Merkezin çalışmalarında; bireyin alanla ilgili sorunlarına etik değerler çerçevesinde çözüm üretilmesi, dünya ve ülke gerçekleri bağlamında yeni medya okuryazarlığı ve dinamiklerinin (sosyal medya-sosyal ağlar-dijital mecralar) bilimsel olarak incelenmesi, yapılan araştırma-incelemeler sonucunda edinilen çıktıların bilimsel nitelikli rapor ve makalelerle dönüştürülerek kitlelere ulaştırılması, yeni dönüşümlerin yarattığı etkilerin araştırılması, disiplinler arası çalışmalar yapılarak yerli ve yabancı literatüre katkı sağlanması amaçlanmaktadır. Merkezin çalışmalarında; bireyin alanla ilgili sorunlarına etik değerler çerçevesinde çözüm üretilmesi, dünya ve ülke gerçekleri bağlamında yeni medya okuryazarlığı ve dinamiklerinin (sosyal medya-sosyal ağlar-dijital mecralar) bilimsel olarak incelenmesi, yapılan araştırma-incelemeler sonucunda edinilen çıktıların bilimsel nitelikli rapor ve makalelerle dönüştürülerek kitlelere ulaştırılması amaçlanmaktadır.

Bu amaç doğrultusunda merkezde; alanla ilgili bilimsel nitelikli kuramsal bilgilerin yanı sıra sosyal medya alanında özgün çalışmaların, yaratıcı projelerin ve türevlerinin hazırlanması, eğitimlerinin verilmesi, pazarlanması süreçlerine yönelik uygulama ve araştırmalar planlanmaktadır. Bu doğrultuda, özellikle “yeni medya” alanında uygulanan eğitim öğretim programlarına ve bilimsel araştırmalara yönelik akademik destek sağlanması planlanmakta, öğrencilerimizin çok yönlü gelişimini sağlayacak ve onların donanımlı bireyler olmasına katkı sağlayan çalışmaların yapılması amaçlanmaktadır. Bu amaç doğrultusunda merkezde; alanla ilgili bilimsel nitelikli kuramsal bilgilerin yanı sıra sosyal medya alanında özgün çalışmaların, yaratıcı projelerin ve türevlerinin hazırlanması, eğitimlerinin verilmesi, pazarlanması süreçlerine yönelik uygulama ve araştırmalar planlanmaktadır. Bu doğrultuda, özellikle “yeni medya” alanında uygulanan eğitim öğretim programlarına ve bilimsel araştırmalara yönelik akademik destek sağlanması planlanmaktadır.

İstanbul Aydin Üniversitesi bünyesinde kurulan Yeni Medya Uygulama ve Araştırma Merkezi; eğitim, sektör ve akademi olarak üç yapıdan oluşmaktadır. Eğitim yapısında; üniversite bünyesindeki bölüm ve merkezlere eğitim bağlamında katkı ve destek vermesi planlanmaktadır. Özellikle de yeni açılan “Yeni Medya ve İletişim Lisans Programı”yla, “Yeni Medya Yüksek Lisans Programları”的n eğitimlerinde teorik ve uygulamalı çalışmaları desteklemesi amaçlanmaktadır. Ayrıca öğrenci projelerini destekleyecek nitelikte uygulamalı seminer ve çalıştaylar da düzenlenmektedir. Öğrenciler katıldıkları seminer ve çalıştaylar sonrasında kendi ilgi alanları doğrultusunda uzman kişilerle iletişim kurarak bilgi edinmeye ve iş imkanı sağlamaktadırlar. İstanbul Aydin Üniversitesi bünyesinde kurulan Yeni Medya Uygulama ve Araştırma Merkezi; eğitim, sektör ve akademi olarak üç yapıdan oluşmaktadır. Eğitim yapısında; üniversite bünyesindeki bölüm ve merkezlere eğitim bağlamında katkı ve destek vermesi planlanmaktadır. Özellikle de yeni açılan “Yeni Medya ve İletişim Lisans Programı”yla, “Yeni Medya Yüksek Lisans Programları”的n eğitimlerinde teorik ve uygulamalı çalışmaları desteklemesi amaçlanmaktadır. Ayrıca öğrenci projelerini destekleyecek nitelikte uygulamalı seminer ve çalıştaylar da düzenlenmektedir.

Sektör yapısı ise, yeni medya alanında söz sahibi olan firmalarla iş birliği içinde projeler geliştirme ve bunları öğrenci dışı kaynağa çevirebilme amacıyla kalkınma

ajanslarından kaynak sağlanması dayanmaktadır. Ayrıca sektörel bazda, STK ve kamu kurumlarıyla oluşturulacak projelerle sosyal sorumluluk çalışmaları ve bunun uzantısı olarak da seminer-konferanslar tarzı etkinlikler düzenlenmektedir. Sektör ilişkilerinin üniversitemize dış paydaş olarak sağlanması, sektörün ihtiyacı olan işgücü istihdamına da kaynak sağlayacaktır. Bu doğrultuda etkinlikler ve çalıştaylar hem sektör bazında hem de öğrenci bazında büyük bir önem taşımaktadır. Sektör yapısı ise, yeni medya alanında söz sahibi olan firmalarla iş birliği içinde projeler geliştirme ve bunları öğrenci dışı kaynağa çevirebilme amacıyla kalkınma ajanslarından kaynak sağlanması dayanmaktadır. Ayrıca sektörel bazda, STK ve kamu kurumlarıyla oluşturulacak projelerle sosyal sorumluluk çalışmaları ve bunun uzantısı olarak da seminer-konferanslar tarzı etkinlikler düzenlenmektedir. Sektör ilişkilerinin üniversitemize dış paydaş olarak sağlanması, sektörün ihtiyacı olan işgücü istihdamına da kaynak sağlayacaktır.

Akademik yapıda ise verilen eğitimlerin, üretilen projelerin, dijital tasarımların, uygulamalı deneyimlerden elde edilen çıktıların tartışılacığı nitel ve nicel araştırmalar yapılmakta; yürütülen bu çalışmaların güncel sorunlara da çözüm üretmesi hedeflenmektedir. Ayrıca bu üretimler, İstanbul Aydın Üniversitesi İletişim Fakültesi bünyesinde oluşturulan Uluslararası Hakemli Yeni Medya Elektronik Dergisi’nde de yayınlanmaktadır. Akademik anlamda lisans bitirme projeleri ve yüksek lisans tezleri de, merkezde yapılan uygulamalı çalışmalarla desteklenmektedir. Öğrenciler aynı zamanda dergilerde yayınladıkları makalelerde onlara koçluk sağlayacak hocalarla çalışmakta, çalışmalarını akademik anlamda temellendirme ve öğrenme fırsatı bulmaktadır. Akademik yapıda ise verilen eğitimlerin, üretilen projelerin, dijital tasarımların, uygulamalı deneyimlerden elde edilen çıktıların tartışılacığı nitel ve nicel araştırmalar yapılmakta; yürütülen bu çalışmaların güncel sorunlara da çözüm üretmesi hedeflenmektedir. Ayrıca bu üretimler, İstanbul Aydın Üniversitesi İletişim Fakültesi bünyesinde oluşturulan Uluslararası Hakemli Yeni Medya Elektronik Dergisi’nde de yayınlanmaktadır. Akademik anlamda lisans bitirme projeleri ve yüksek lisans tezleri de, merkezde yapılan uygulamalı çalışmalarla desteklenmektedir.

Merkez bünyesinde kurulan VR-LAB’ın (Virtual Reality Laboratuvarı) ise; bireyin dijital ortamlarda yaşadığı sorunsallara çözüm üretebilmesine, yeniliklerin yayılması bağlamında etik değerlere bağlı, dünya ve ülke gerçeklerine uygun yeni medya okuryazarlığı ve dinamiklerinin (sosyal medya-sosyal ağlar-dijital meclalar)

bilimsel olarak incelenmesine ve yapılan araştırmalar sonucunda çıktıların bilimsel nitelikli rapor ve makalelerle kitlelere ulaştırılmasına hizmet etmesi amaçlanmaktadır. Bu amaç doğrultusunda uygulamalı çalışmalar üretilebilmesi, üretilen bu dijital veriler ışığında analizlerin yapılması ve üretilen çalışmaların da çıktılarının alınması planlanmaktadır. Merkez bünyesinde kurulan VR-LAB’ın (Virtual Reality Laboratuvarı) ise; bireyin dijital ortamlarda yaşadığı sorunsallara çözüm üretебilmesine, yeniliklerin yayılması bağlamında etik değerlere bağlı, dünya ve ülke gerçeklerine uygun yeni medya okuryazarlığı ve dinamiklerinin (sosyal medya-sosyal ağlar-dijital mecralar) bilimsel olarak incelenmesine ve yapılan araştırmalar sonucunda çıktıların bilimsel nitelikli rapor ve makalelerle kitlelere ulaştırılmasına hizmet etmesi amaçlanmaktadır. Bu amaç doğrultusunda uygulamalı çalışmalar üretilebilmesi, üretilen bu dijital veriler ışığında analizlerin yapılması ve üretilen çalışmaların da çıktılarının alınması planlanmaktadır.

Sonuç olarak; günümüz iletişim ortamında öne çıkan bir alan olan yeni medya, gerektiği gibi doğru biçimde kullanılamamaktadır. Çeşitli marka ve ajanslar; sosyal medyayı etkili ve etkin biçimde kullanabilen, üretim yapabilecek yaratıcı, bilgili bireyleri istihdam etme konusunda rekabet halindedir. Bu bağlamda Yeni Medya Uygulama ve Araştırma Merkezi ile yeni medya alanındaki bölümlerin öğrencileri hem bilişsel anlamda gelişecek, hem de üniversite bünyesindeki Teknoloji Merkezi’nde uygulama olanağı bulacaklardır. Bu durum, öğrencilerin mesleğe bir adım önde girmelerine olanak sağlayacaktır. Öğrenciler, bireysel ve mesleki anlamda kendilerini geliştirecek ve iş imkanlarını da öğrendikleri bilgiler ve katıldıkları seminerler doğrultusunda zenginleştirereklerdir. Sonuç olarak; günümüz iletişim ortamında öne çıkan bir alan olan yeni medya, gerektiği gibi doğru biçimde kullanılamamaktadır. Çeşitli marka ve ajanslar; sosyal medyayı etkili ve etkin biçimde kullanabilen, üretim yapabilecek yaratıcı, bilgili bireyleri istihdam etme konusunda rekabet halindedir. Bu bağlamda Yeni Medya Uygulama ve Araştırma Merkezi ile yeni medya alanındaki bölümülerin öğrencileri hem bilişsel anlamda gelişecek, hem de üniversite bünyesindeki Teknoloji Merkezi’nde uygulama olanağı bulacaklardır. Bu durum, öğrencilerin mesleğe bir adım önde girmelerine olanak sağlayacaktır.

**SWIFT İLE IOS PROGRAMLAMA:
YENİ BAŞLAYANLAR İÇİN**

Mobil Uygulama Dersi – iOS Programlama

1. Mobil Uygulama Nedir ?

Mobil uygulama bir bilgisayar programıdır bir başka deyişle yazılımıdır. Mobil uygulamalar olarak nitelendirilen yazılımlar daha çok küçük, kablosuz ve taşınabilir cihazları hedeflemektedir. Burada kast edilen; tablet ve akıllı telefonlar gibi cihazlardır. Kullanıcıların günlük problemlerini kişisel mobil cihazları aracılığı ile çözmeyi hedeflerler. Bazı platformlarda harici yöntemlerle yüklenemelerine rağmen genellikle uygulama mağazaları aracılığı ile dağıtımları gerçekleştirilir. Mobil uygulamalar hybrid ve native programlananlar şeklinde ikiye ayrılır. Hybrid olarak kast edilen uygulamalar bir web framework'ü üzerine oturtulmuş yapılardır. Responsive yapıda hazırlanan bu uygulamalar bu sayede her türlü ekrana göre biçim alabilmektedir. Ayrıca hybrid uygulamalar platform bağımsızlığına sahip olmaktadır. Çünkü klasik web sayfası hazırlanması yöntemi kullanmaktadır. Native olarak hazırlanan uygulamalar ise işletim sisteminin kendi kullandığı dil ya da kullanılan dilin yapısına uygun olan dile göre derlenerek hazırlanan bir yapıyı kullanmaktadır. Bu yönü ile native programlanan uygulamalar daha performanslı ve donanım erişimini ve ihtiyacını daha kolay gideren bir yapıda hazırlanabilmektedir. Hybrid uygulamalar bir sefer yaz, heryerde çalışın mantığını taşımamasına rağmen, native programlanan uygulamalar daha az hata ve daha yüksek performans sunmaktadır.

2. Mobil Uygulama Platformları

Başta Apple ve Samsung gibi firmalar mobil cihaz piyasasında birçok mobil platform yaratmasına sebep olmuştur. Bu noktada platformdan kasıt genelde işletim sistemi olmaktadır. iOS , Android, Windows Mobile gibi işletim sistemleri mobil platformlar için geliştirilmiştir.

Biz iOS işletim sistemi üzerinde yazılımlar geliştireceğiz. iOS Apple firmasının ürettiği mobil cihazlar için (iphone ve apple tabletleri) geliştirilmiş bir işletim sistemidir. iOS apple mobil cihazlarının performansının artırılması için optimize edilmiştir. C++ ve Objective-C kullanılarak üretilmiş bir işletim sistemidir.

3. Swift Programlama Dili

Swift programlama dilinden önce apple tüm cihazlarında Objective-C isminde bir programlama dilini kullanmaktadır. 1970'lerde Smalltalk ismi verilen bir programlama dili geliştirilmektedir. Bu programlama dilinin en büyük özelliği o dönemde devrimsel bir gelişme olan nesnel yapıda olmasıydı. 1980'lerde gelindiğinde procedural bir dil olan C dili iki geliştirici ile nesnel bir yapıya kavuşturulmak için Smalltalk programlama dili ile birleştirildi. Bunun sonucunda Objective-C dili doğdu. Apple firması daha önce belirttiğimiz gibi bütün yazılımlarında Objective-C dili kullanmaktadır. Objective-C dilinin zor gramer yapısı sebebiyle yeni ve modern bir dil geliştirilmesine karar verildi. 2004 yılında Swift programlama dili bu sıkıntıyı gidermek amacıyla doğmuştur. Swift programlama dili modern yapısı ve nesnel yönelimi ile yaşamına devam etmektedir.

Swift programlama dili nesnel yönelimli bir dil olmasına rağmen, Protocol Oriented bir dil olarak tanımlanmaktadır. Nesnel yönelimli diller ile kıyaslandığında çok büyük farklılıklar yaratmamasına karşın bu özellik Swift diline daha fazla esneklik katmaktadır. Bu konu ilerleyen bölümlerde sınıf yapıları ve nesnel programlama başlığı altında ayrıca anlatılmaya çalışılmıştır.

4. XCODE

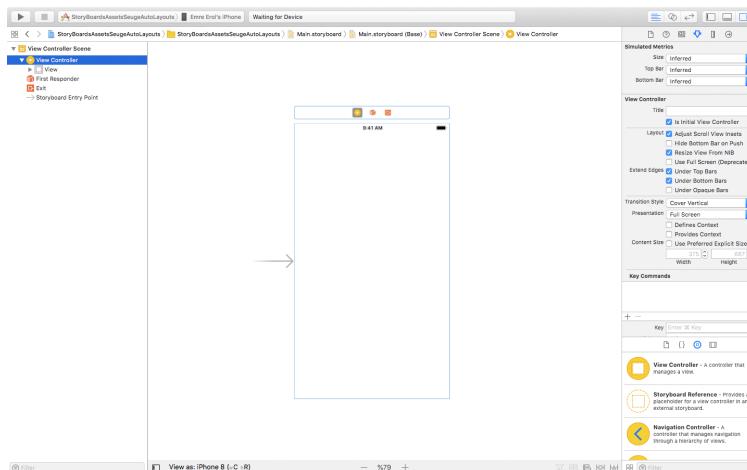
IDE; Integrated Development Environment – Tümleşik Geliştirme Ortamı. Temelde tüm IDE ler bir “gelişmiş” metin editörüdür. Geliştiriciye/Yazılımcıya üzerinde çalıştığı proje için bazı kolaylıklar sağlar.

- Kod tamamlama,
- Derleme,
- Renkli gösterim/kod boyama
- Simüle etme/çalıştırma
- Test etme ve hata ayıklama

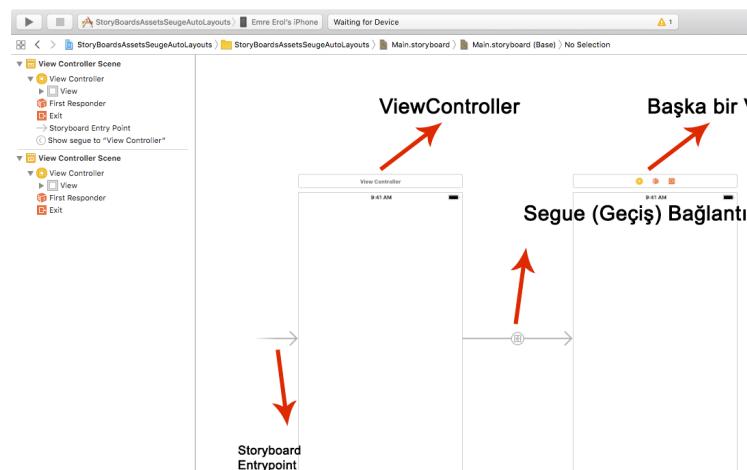
XCode 'da Apple Inc firması tarafından apple ürünleri için yazılım geliştirilmesi için yayınlanmış bir IDE dir. XCode ile özellikle Objective-C ya da Swift dili kullanılarak yazılım geliştirilebilir.

5. Storyboard

Storyboard; geliştirilen uygulama ile ilgili geliştiriciye görsel bir temsil sunan yapıdır. Bu yapı ile geliştirici geliştirmekte olduğu uygulamanın sayfaları (ViewController vb.) üzerindeki nesneleri ekleyebilir, çıkarabilir, düzenleyebilir ve konumlarını ayarlayabilir. XCode üzerinde birden fazla storyboard oluşturulabilir. Büyük projelerde bu sayede karmaşıklık azaltılabilir. Storyboard ayrıca uygulama sayfalarının arasındaki geçişleride tasarladığımız bölümdür. Sürükle bırak yöntemi ile hangi sayfadan (ViewController) hangi sayfaya (Başka bir ViewController) geçileceğini belirleyebiliriz.



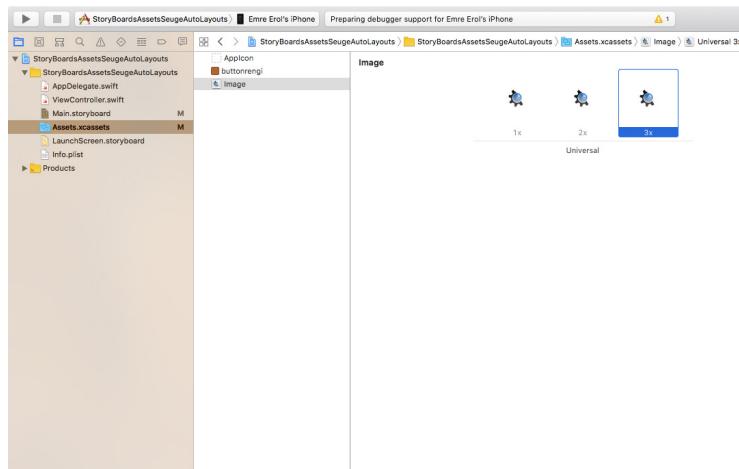
Resim 1 : StoryBoard Genel Görünüm



Resim 2 : StoryBoard Sayfa Geçişleri

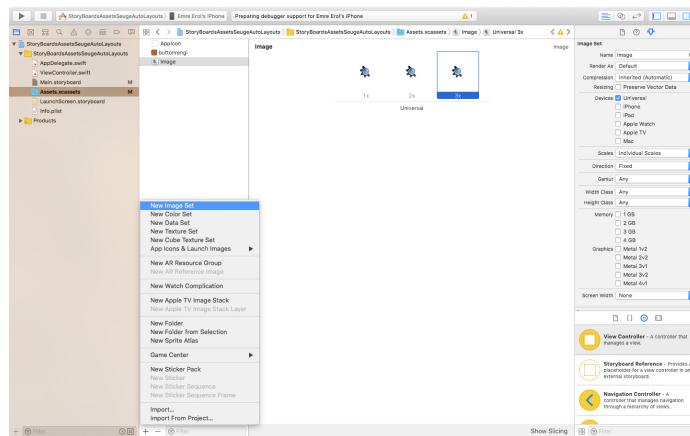
6. Asset Ve Kullanımlar

Asset proje içerisinde bazı görsel içeriklerin tutulduğu bölümdür. Bu bölüm sayesinde resim/kaplama/renkler gibi görsel içerikler proje içerişine dahil edilebilir.



Resim 3 : Asset Kataloğu

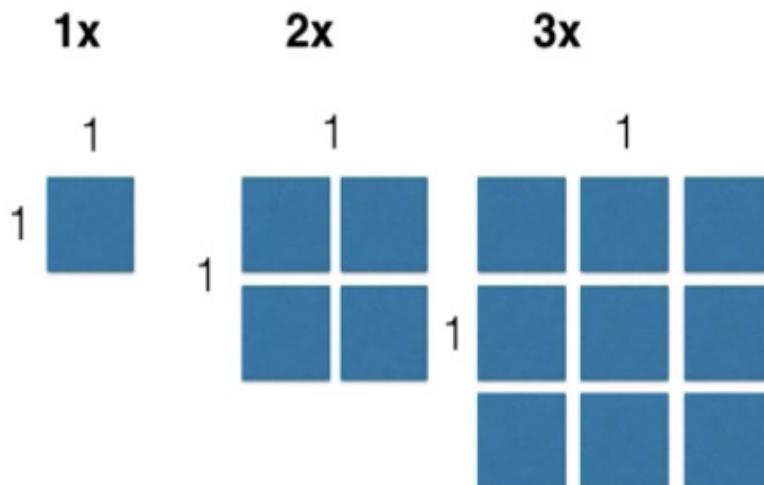
Asset kataloğu açıldıktan sonra sol alt kısımdaki artı (+) simgesinden eklenmek istenen içerik seçilerek sonrasında içerik eklenebilir.



Resim 4 : Asset Kataloğu Menüsü

Görsel eklemek istediğinizde 1x, 2x, 3x gibi bir bölümle karşılaşılır. Bu bize farklı ekran biçimlerine göre farklı kalitelerde görseller ekleyebilmemize olanak sağlar.

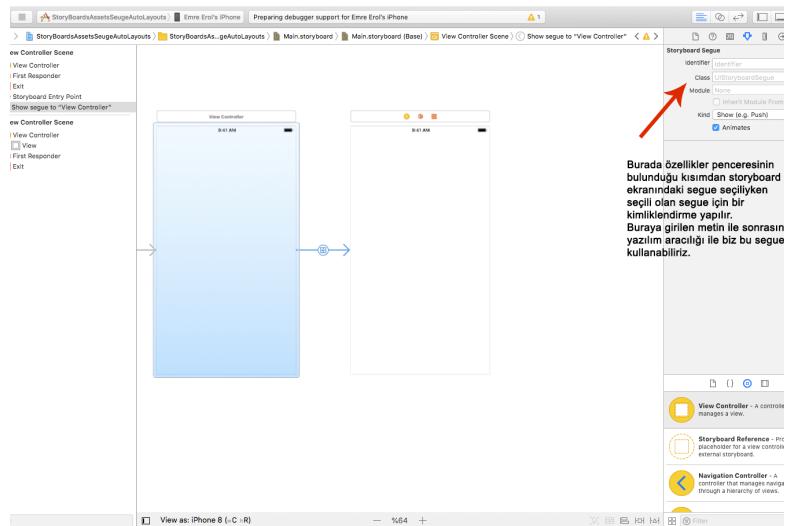
1 Pixel yer kaplayan bir görsel eklemek istediğimizi düşünelim, bu durumda; 2x yazan kısma bu görselin 2 katı büyüğündeki kopyasını, 3x yazan kısımada aynı görselin 3 katı büyüğündeki görselin eklenmesi gereklidir. Sol kısımada yer alan panel aracılığı ile eklenen içeriklerin isimleri belirlenir ve sonrasında bunlar uygulama geliştirilirken kullanılabilir.



Resim 5 : Farklı Ölçekler

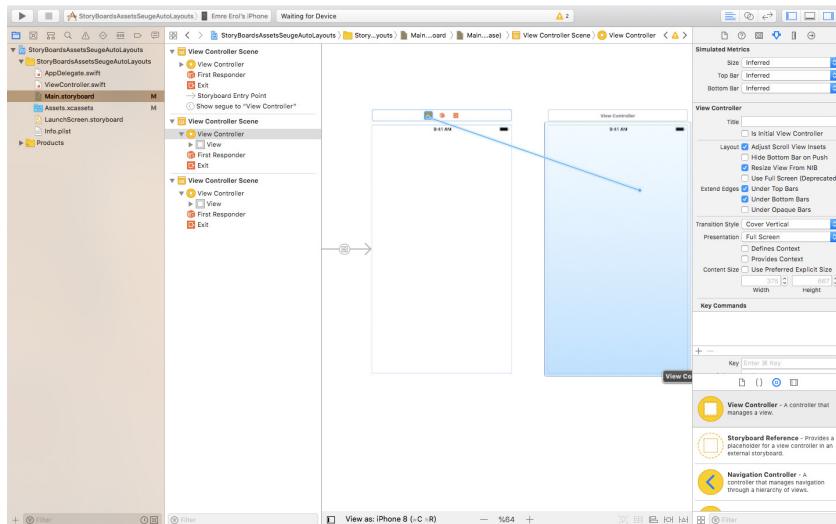
7. Segue Bağlantıları İle Sayfalar Arası Geçişler

Segue bağlantıları ile sayfalar (ViewController) arası geçişler yapılandırılır/tasarlanır.



Resim 6 : Segue Bağlantıları

Storyboard ekranına eklenen her sayfanın (ViewController) üzerinde sarı renkli bir ikon bulunur. Bu ikona mouse in sağ butonunun basılı tutularak sürükleşmesi ile ilgili sayfa üzerine gidilip bırakılması suretiyle segue bağlantısı oluşturma isteği üretilir. Sonrasında açılan popup penceresinden geçiş türü belirlenir. Genellikle "show" kullanılır.

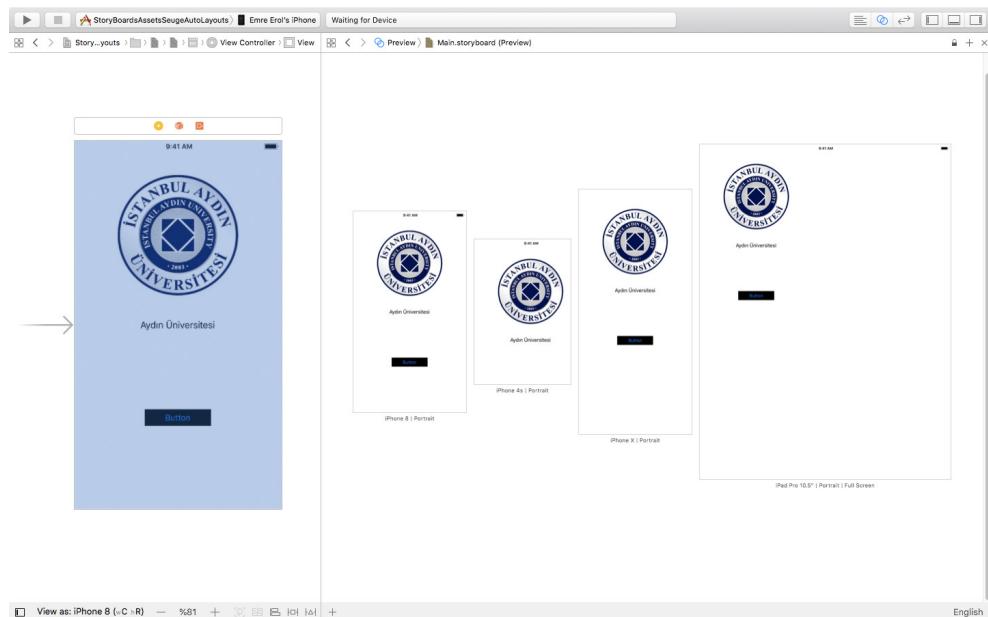


Resim 7 : Segue Bağlantıları 2

8. AutoLayout

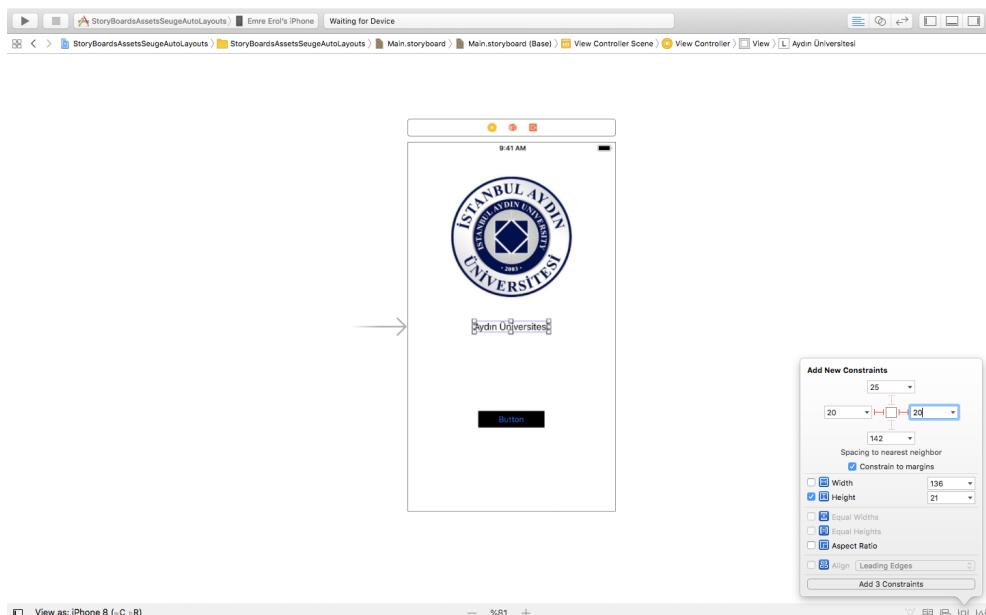
Farklı ekran büyütükleri mevcut olduğundan bir tasarım oluşturulduğunda bu tasarımın her ekran büyütüğünde aynı biçimde görüntülenmesi büyük sıkıntılar yaratmaktadır. Bu problemin çözümü için XCode geliştiricilere *AutoLayout* kavramını sunar.

AutoLayout ile bir defa hazırlanan tasarımlar tüm ekran büyütüklerinde kendi genişlik ve yüksekliklerini değiştirerek aynı biçimde görüntülenmektedir.



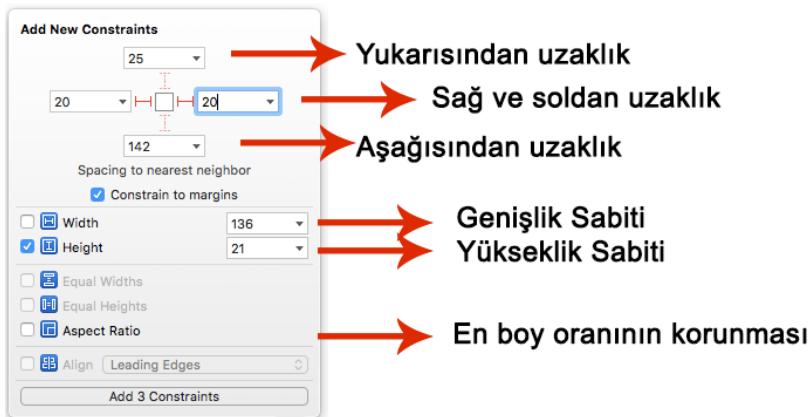
Resim 8 : AutoLayout Uygulanmamış Bir Tasarımın Diğer Ekran Tiplerindeki Yerleşimleri

AutoLayout uygulamak için tasarım üzerindeki ilgili elementler seçilerek ekranın sağ alt kısmında bulunan “Add New Constraints” butonu tıklanır. Ardından bu elemente verilebilecek sabitler oluşturulur. Bu sabitler sayesinde tasarım tüm ekran büyütüklerinde aynı görünecektir.

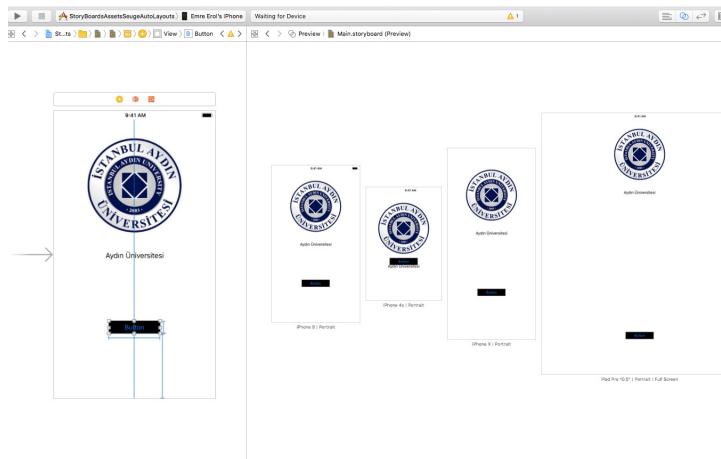


Resim 9 : AddNew Constraints Popup Penceresi

Elementin;



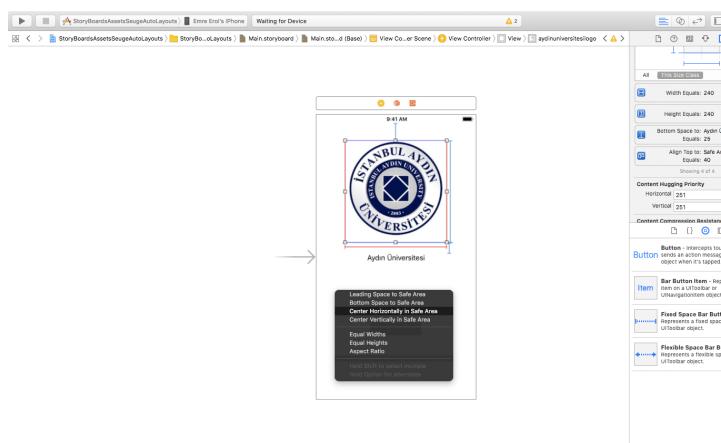
Resim 10 : Sabit Açıklamaları



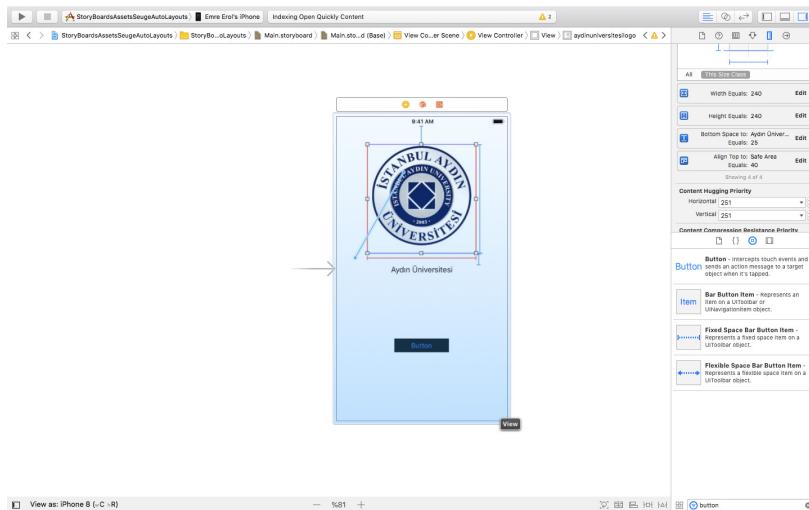
Resim 11 : AutoLayout Uygulanmış Bir Tasarımın Diğer Ekran Tiplerindeki Yerleşimleri

Elementlerin birbirlerine hizalanmalarına ilişkinde sabitler verilebilir. Örneğin; bir butonun ekranın tam ortasında yer alması için ekranın *rootView* denilen tüm viewleri içerisinde barındıran ana view elementine göre kendisini hizalaması istenebilir.

Bunun için elemente sağ tıklanarak, birbirlerine göre hizalanmak istenen diğer elementin üzerine sürüklendiğinde bırakıldığından açılan popup penceresinde verilebilecek sabitlere ilişkin seçenekler mevcuttur.

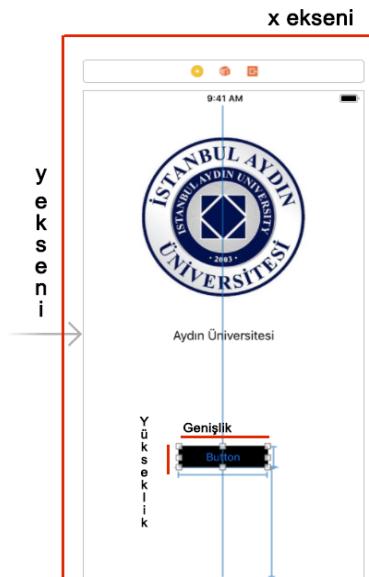


Resim 12 : İlişkilendirilmiş Sabitlerin Uygulanması



Resim 13 : İlişkilendirilmiş Sabitler Popup Penceresi

AutoLayout kullanılırken önemli olan bir elementin x eksenindeki, y eksenindeki ve genişliğini, yüksekliğini bir sabite bağlamaktır.



Resim 14 : Sabit Kavramı İçin Sabitlenecek Özelliklerin Gösterimi

9. Algoritma Kavramı ve Algoritma Oluşturma

Algoritma kelimesi çok karmaşık yapıları ifade eden bir kelimeyemiş gibi görünsede temelde günlük hayatımızda da çokça kullandığımız basit bir kavramı ifade etmektedir. Algoritma bir problemin çözümünde kullanılacak olan tüm adımlardır. Dolayısı ile problem çözme sürecinde bir parçasıdır. Problem çözme 4 adımdan oluşur;

- Problemi Anlama
- Bir çözüm yolu geliştirme
- Algoritma Yazımı
- Çözüm yolunun uygulanması ve test edilmesi

Her insanın arabaya bindiğinde yaptığı işler ve izlediği yol bellidir ya da bir asansöre bindiğinde yaptığı tüm eylemler bazı durumlarda değişse bile belli bir sistem içerisinde sıra ile gerçekleşmektedir. Arabaya bindiğimizde problem arabanın henüz çalışmıyor olusudur. Asansörde ise problem, varılmak istenen kata doğru henüz asansörün hareket etmemişidir. Tam bu noktada bu problemlerin çözümü için gerçekleştirilen eylemlerin ifadesi algoritma kelimesinin karşılığını bize vermektedir.

Algoritmalar 3 şekilde gösterilirler;

- Düz yazı ile
 - Sözde Kod ile
 - Akış Şemaları ile
- a. Algoritma Örneği - Kısa
1. Ayağa kalk
 2. Varsa çantanı al

3. Kapıya Yürü
4. Binadan Çık

b. Algoritma Örneği – Uzun

1. Ayağa Kalk
2. Telefonunu aldın mı kontrol et
3. Varsa kıyafetini giy
4. Kapıya doğru nereden yürüyeceğine karar ver
5. Kapıya doğru yürümeye başla
6. Kapı kapalı ise aç
7. Kapıdan geç
8. Merdivenlerin başına yürü
9. Merdivenlerden inmeye başla
10. Bina kapısına doğru yürümeye devam et
11. Bina kapısı kapalı ise aç
12. Bina kapısından geç

c. Algoritma Örneği – OBEB

1. Eğer $A=0$ ise, $\text{OBEB}(0,B)=B$ olacağı için $\text{OBEB}(A,B)=B$ olur ve bu noktada durulmalıdır.
2. Eğer $B=0$ ise, $\text{OBEB}(A,0)=A$ olacağı için $\text{OBEB}(A,B)=A$ olur ve bu noktada durulmalıdır.
3. A sayısını bölüm ve kalan formunda yazın ($A=B \cdot Q+R$)
4. $\text{OBEB}(A,B)=\text{OBEB}(B,R)$ olduğu için, $\text{OBEB}(A,B,R)$ 'yi Öklid Algoritmasını kullanarak bulun

d. Algoritma Örneği – İki Sayının Toplamı

1. BAŞLA
2. Birinci Sayıyı Gir
3. İkinci Sayıyı Gir
4. Girilen Sayıları Topla
5. Sayıların Toplam Değerini Yaz
6. BİTİR

10. Değişkenler

İşlenmemiş bilgi parçacıklarına Veri denir. Programcılıkta bu veriler değişkenlerde tutulur. Değişkenler ile tutulan veriler daha sonra işlenebilir ve sonuca dönüştürülebilir. Değişkenler bilgisayarların genelde RAM ismi verilen geçici hafızalarında tutulurlar. Modern programlama dilleri bize değişkenleri istediğimiz isim altında işaretleyerek kullanmamıza izin vermektedir. Çeşitli tiplerde değişkenler oluşturulabilir.

a . Değişken Tipleri

i . String Veri Tipleri (Metinsel Veriler)

Metinsel ifadelerin saklandığı veri tipleridir. String veriler çift tırnak içerisinde yazıılır.

ii . Character Veri Tipleri (Metinsel Karakter Veriler)

Metinsel ifade saklanmaktadır ancak yalnızca bir karakter barındırabilirler.

i.i. Integer Veri Tipleri (Tam Sayı Veriler)

Sadece tam sayıları tutabilen veri tipleridir. Belli bir aralıkta ki tam sayıları tutabilirler buna göre sayı aralığı değişmektedir. (Int,Int8,Int16,Int32,Int64)

iv. Float ve Double (Reel Sayı Veri Tipleri)

Virgülü sayıları saklayabilen veri tipleridir. Float 32 bit, double 64 bit saklama alanına sahiptir.

v. Bool Veri Tipleri (Mantıksal Veri Tipleri)

Yalnızca true (doğru) ya da false (yanlış) anlamındaki veriyi taşıyabilirler.

Değişkenler *var* anahtar kelimesi ile tanımlanırlar. Ardından değişkene verilecek olan isim belirlenmelidir. İsim belirlenirken türkçe karakter kullanılmamalı, boşluk bırakılmamalı ve rakamlı başlanmamalıdır. Ayrıca noktalama işaretleride kullanılmamalıdır. Ardından veri tipi (:) iki nokta üst üste kullanılarak belirlenir. Eşittir (=) operatörü ile tutulması istenen veri değişkene atanır. Tutulması istenen veri eğer metinsel ise çift tırnak arasına yazılmalıdır.

```

1 ViewController.swift
2 StoryBoardsAssetsSegueAutoLayouts
3 StoryBoardsAssetsSegueAutoLayouts
4 Created by Emre Erol on 22.09.2018.
5 Copyright © 2018 Codework. All rights reserved.
6
7
8 import UIKit
9 class ViewController: UIViewController {
10
11     var metinDegiskeni:String = "Aydın Üniversitesi"
12     var karakterDegiskeni:Character = "H"
13     var sayisalDegisken:Int = 123
14     var reelSayiDegiskeni:Float = 10.2
15     var mantiksalBooleanDegiskeni:Bool = true
16
17     override func viewDidLoad() {
18         super.viewDidLoad()
19     }
20
21 }
22
23
24
25
26
27
28
29
30

```

Resim 15 : Değişken Tanımlamalarına Örnekler

b. Tip Dönüşümleri

İhtiyaç halinde bazı veri tipleri diğer veri tiplerine dönüştürülebilir. Örneğin bir sayı tipi (Integer) metin (String) tipine ya da bir metinsel (String) veri tipi sayısal veri tipine (Integer) dönüştürülebilir.

i. Tip Dönüşümleri – Sayısalan (Integer) Metinsele (String) Dönüşüm

```
32      let sayı:Int = 12           // Tanımlanmış bir tamsayı
33      let metin:String = String(sayı) // Metinsel biçimde dönüştü
34
35
```

Resim 16 : Sayısal Değişkenin Metinsel Değişkene Dönüşümü

ii. Tip Dönüşümleri – Metinselden (String) Sayısal (Integer) Dönüşüm

```
32      let metin:String = "12"        // Tanımlanmış bir metin (iceri)
33      let tamsayı:Int? = Int(metin) // Tamsayı biçimde dönüştü
34
35
```

Resim 17 : Metinsel Değişkenin Sayısal Değişkene Dönüşümü

Dikkat edilirse Metinsel (String) tipten Sayısal (Integer) tipe dönüşüm yapılırken dönüşüm sonucunda atanacak olan yeni tanımlanan değişkenin sonuna soru işaretçi (?) konulduğu görülecektir. Burada soru işaretçi değişkenin opsiyonel olduğunu. Tip dönüşümü sonucunda bir değer dönmeyebilir. Bu ihtimalden dolayı soru işaretçi kullanılır. Örneğin bir harf int ‘e çevrilemez. Bu durumda değişken nil değerini alacaktır. Bir değişkenin nil değeri alabilmesi için soru işaretçi “?” ile opsiyonel olarak tanımlanması gereklidir.

Bu değişkenler kullanılırken nil olmadığı kontrol edilerek kullanılmalıdır. Eğer opsiyonel bir değişkenin nil olmadığından eminsek değişkeni kullanırken ünlem işaretçi “!” değişkenin sonuna konularak kullanıma devam edilir.

C . Sabit Tanımlama

Swift dilinde sabit tanımlamaları yapılabilir. Sabitler aldıkları tanımlandıklarında aldıkları veriyi ya da ilk aldıkları veriyi program sonuna kadar değiştirmeden tutan değişkenlerdir.

let anahtar kelimesi kullanılarak tanımlanırlar. *let* anahtar kelimesinden sonra normal bir değişken tanımlar gibi diğer detaylar girilir.

Örnek : let sabit:Int = 12



Kare Kod 1 : <https://github.com/ios-kitap/Degiskenler>

Değişken tanımlamaları ve tip dönüşümleri ile ilgili tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

11. Diziler

Bir değişken içerisinde birden fazla veri saklamak istendiğinde dizilere başvurulur. Dizilerdeki temel unsur içerisinde barındırdığı verilerin aynı tipten olmasıdır.

Dizilerde bulunan veriler indeks numarasına göre çağırılır ve kullanılır. Burada dikkat edilmesi gereken husus dizilerin indeks numarasının sıfırdan başlayıp ardışık olarak artmasıdır.

Bir dizi oluşturulurken normal değişken tanımlanır gibi *var* anahtar kelimesi

kullanılır. Ardından bu dizi değişkenine bir isim verilir. Eşittir (=) operatörü ile bu dizinin içerisinde tutulacak olan verilerin ataması yapılır. Sonrasında köşe parantezler ([]) içerisinde değişkenler aralarında virgule olacak biçimde yazılır.

```

25 //Dizi Tanımlamaları-----
26 var birDizi = ["Aydın", "Üniversitesi", "Sosyal", "Bilimler", "Enstitüsü"]
27
28 var baskaBirDizi = [1,2,92,2,5,432]

```

Resim 18 : Dizi Tanımlanması

Diziye yeni bir eleman eklenmek istendiğinde dizi adı yazılır ve nokta işareti ile bu dizinin metotlarına ulaşılır. *append* kelimesi ardından parantezler arasına girilen veri diziye eklenebilir.

Aynı işlemi *insert* kelimesi ile yaptığımız taktirde istedigimiz konuma veriyi ekleyebiliriz. Ancak *insert* kelimesini kullandığımızda bu defa parantez içerisine birde eklenecek olan konumun indeks numarası girilmelidir.

```

60 //Diziye Eleman Ekleme (Sonuna)
61 birDizi.append("Eklenecek Olan Yeni Eleman")
62 var eklenecekOlanStringTipindeEleman:String = "Bu şekilde de diziye eleman eklenebilir"
63 birDizi.append(eklenecekOlanStringTipindeEleman)
64
65 //Diziye Eleman Ekleme (İstediğimiz konuma)
66 birDizi.insert("İstediğimiz Konuma Eklenecek Olan Eleman", at: 2)

```

Resim 19 : Append ve Insert Metotları

Diziden eleman çıkarmak için yine işlem yapılacak olan dizinin adı yazılır ve nokta işareti ile bu dizinin metotlarına ulaşılır. *remove* kelimesi ile birlikte parantezlerin arasında *at:* kelimesi konulur. Buradaki *at:* kelimesi çıkarılmak istenen elemanın indeks numarasının girileceği kısımdır. *at:* ardından çıkarılmak istenen dizinin indeks numarası girilir. Yalnızca *removeFirst()* metodu kullanıldığı taktirde dizinin ilk elemanı çıkarılabilir. Yalnızca *removeLast()* metodu kullanıldığı taktirde dizinin son elemanı çıkarılabilir.

```

66 birDizi.insert("İstediğimiz Konuma Eklenecek Olan Eleman", at: 2)
67
68 //Diziden Eleman Çıkarma
69 birDizi.remove(at: 2)
70 birDizi.removeFirst()
71 birDizi.removeLast()
72
73 //Dizi içinde dizi
74 var dizideindeDizi:[String] = ["birDizi", "ikiDizi", "ucDizi"]

```

Resim 20 : Diziden Eleman Çıkarma İşlemi

Bazı durumlarda dizi içinde dizide oluşturulabilir.

```
74     var diziIcindeDizi = [birDizi,baskaBirDizi] as [Any]
75     var digerSeklidleDiziIcindeDizi = [
76         "uyeler": ["Ahmet","Mehmet","Ali"],
77         "sehirler": ["İstanbul","Ankara","İzmir","Bursa"]
78     ]
79
80     var ulasilanDizi = diziIcindeDizi[0] ⚠ Initialization of variable 'ulasilanDizi'
81     var ulasilanBaskaBirDizi = digerSeklidleDiziIcindeDizi["sehirler"] ⚠ Initialization of variable 'ulasilanBaskaBirDizi'
```

Resim 21 : Çok Boyutlu Dizi

a. Dictionary Diziler

Dictionary dizilerin normal dizilerden farkı indeks numarası yerine geliştiricinin belirlediği anahtar kelimelerin kullanılmasıdır.

```
29
30     //Dictionary Diziler
31     var dictionaryDizi = [
32         "Ahmet":13,
33         "Mehmet":24,
34         "Hakan":53
35     ]
36     //Dizi Tanımlamaları-----
```

Resim 22 : Dictionary Dizi Tanımlanması

`dictionaryDizi["Ahmet"]` komutu çalıştırıldığında dizinin ilgili verisine ulaşılmış olur. Sonuç bu durumda 13 olacaktır.



Kare Kod 2 : <https://github.com/ios-kitap/Diziler>

Dizi tanımlamaları ile ilgili tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

12. Operatörler

a. Aritmetik Operatörler

Tüm programlama dillerinde olduğu gibi swift programlama dilinde de temel matematik işlemlerini gerçekleştirmek için kullandığımız operatörlere aritmetik operatörler denir. Program boyunca ihtiyaç duyulan noktalarda aritmetik operatörler kullanılarak aritmetik işlemler gerçekleştirilebilir. Basitçe aşağıdaki tabloda gösterilmiştir;

+	Toplama	İki sayısal verinin toplam sonucunu verir.
-	Çıkarma	İki sayısal verinin çıkarma sonucunu verir.
/	Bölme	Solundaki Float ya da Double tipindeki verinin sağındakiFloat ya da Double tipindeki veriye bölümünü verir. Sonuç yine Float ya da Double tipindedir.
*	Çarpma	İki değişkenin çarpım sonucunu verir.
%	Mod	Soldaki verinin sağdaki veriye bölümünden kalan sonucu verir.
+=	Kendisiyle Toplama	Değişkende tutulan sayısal değeri verilen sayı ile toplayıp yine değişkene atar.
-=	Kendisinden Çıkarma	Değişkende tutulan sayısal değeri verilen sayıdan çıkarıp yine değişkene atar.
=	Atama ya da eşitleme operatörü	Sağdaki veriyi solundaki değişkene atar, eşitler.

Tablo 1 : Aritmetik Operatörler

b. Mantıksal Operatörler

Program esnasında toplanan verilerin mantıksal olarak programın akışını etkilediği noktalarda mantıksal operatörler kullanılır. Mantıksal operatörler Boolean veriler eşliğinde kullanılabilmektedir. Genellikle iki verinin birbiri ile karşılaştırılması durumunda karar verme aşamalarında karşımıza çıkarlar. Bu karşılaştırmaların sonucu mantıksal operatörler Boolean veriler döner. Dönen true ya da false veriye göre karar verilir ve program o koşulda çalışmasına devam eder.

==	Eşit mi? (Mantıksal Operatör)	İki değişkenin eşit olup olmadığını sonucunu mantıksal veri olarak döner. Eşitse true, eşit değilse false döner.
>	Büyükür (Mantıksal Operatör)	Solundaki değişkenin sağındaki değişkenden büyük olup olmadığını kontrol eder. Büyükse true döner, değilse veya eşitse false döner.
>=	Büyük Eşittir (Mantıksal Operatör)	Solundaki değişkenin sağındaki değişkenden büyük veya eşit olup olmadığını kontrol eder. Büyükse veya eşitse true döner, değilse false döner.
<	Küçükür (Mantıksal Operatör)	Solundaki değişkenin sağındaki değişkenden küçük olup olmadığını kontrol eder. Küçükse true döner, değilse veya eşitse false döner.
<=	Küçük Eşittir (Mantıksal Operatör)	Solundaki değişkenin sağındaki değişkenden küçük veya eşit olup olmadığını kontrol eder. Küçükse veya eşitse true döner, değilse false döner.
!	Değil (NOT) (Mantıksal Operatör)	Mantıksal sonucu tersine çevirir. True false dönüşür, false true ya dönüşür.
&&	VE (AND) (Mantıksal Operatör)	Birden fazla karşılaştırma işlemlerinde dönen iki değerin true olup olmadığını kontrol eder. İki değerde true ise sonucu true olarak döner. Değilse false olarak döner.
	VEYA(OR) (Mantıksal Operatör)	Birden fazla karşılaştırma işlemlerinde dönen iki değerden herhangi birinin true olup olmadığını kontrol eder. İki değerden biri veya ikiside true ise true döner. İki değerde false ise false döner.

Tablo 2 : Mantıksal Operatörler



Kare Kod 3 : <https://github.com/ios-kitap/Operatorler>

Aritmetik operatörler ile ilgili tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

13. Kontrol Yapıları

a. if Kontrol Yapısı

Karşılaştırmalı mantıksal kontrollerde *if else* yapıları kullanılır. Karşılaştırma işlemi barındırdığından burada mantıksal operatörler kullanıldığından; operatörün ürettiği sonucun *true* ya da *false* olmasına göre karar verilir ve program verilen karar doğrultusunda işleyişine devam eder.

```
//  
93     //if else  
94  
95     if 21 < 35 {  
96         print("21 küçüktür 35'ten")  
97     }  
98  
99     let ahmetinYasi:Int = 41  
100    let mehmetinYasi:Int = 36  
101  
102    if ahmetinYasi == mehmetinYasi {  
103        print("Ahmet'in yaşı Mehmet'in yaşına eşittir.")  
104    }else if ahmetinYasi < mehmetinYasi {  
105        print("Ahmet'in yaşı küçüktür Mehmet'in yaşından.")  
106    }else{  
107        print("Ahmet'in yaşı büyütür Mehmet'in yaşından")  
108    }  
109
```

Resim 23 : *if else* Kontrol Yapısı

If kelimesi, eğer anlamına gelmektedir. Else kelimesi de değilse anlamına gelmektedir. If kelimesinden sonra kontrol edilecek koşul kontrol edilmek istenen mantıksal operatör ile yazılır. Koşul sağlanmadığı taktirde karşılaştırma operatörü *false* değer üretecektir. Bu durumda ilk küme parantezleri ({ }) arasında bulunan kod parçası çalışmaz. Eğer koşul sağlanıyorsa karşılaştırma operatörü *true* değer üretecek ve o kısımdaki kod parçası çalışacaktır. Koşul sağlanmadığı taktirde eğer kontrol yapısında mevcut bir *if-else* basamağı mevcut ise bu basamakta yer alan koşul kontrol edilir. Bu basamakta yer alan operatörün ürettiği değere göre bu kod bloğu çalıştırılır ya da çalıştırılmaz. Son olarak kontrol yapısında bir *else* basamağı mevcut ise, hiçbir koşul sağlanmadığı durumda (bir diğer deyişle tüm karşılaştırma operatörleri *false* ürettiyse) *else* bloğundaki kod parçası çalıştırılacaktır.

```
if koşul {  
    // Kod Bloğu 1. Basamak  
}  
} else if koşul {  
    // Kod Bloğu 2. Basamak  
}  
} else {  
    // Kod Bloğu son basamak. Hiçbir koşul geçerli değilse bu  
    // kod bloğu çalışacaktır.  
}
```

Yazım Kuralı 1 : if else Yapısı

b. Switch-Case Yapısı

Bir diğer kontrol yapısı olan *switch-case* bir değişkene göre birden fazla olasılığın söz konusu olduğu durumlarda kullanılabilir.

```
109 | //Switch-Case
110
111
112     switch ahmetinYasi {
113         case 23:
114             print("Ahmet 23 Yaşında")
115         case 41:
116             print("Ahmet 41 Yaşında")
117         default:
118             print("Hata!")
119     }
120
```

Resim 24 : Switch-Case Kontrol Yapısı

Switch kelimesi ile *switch* kontrol yapısı kullanılmak istediği belirtilir. Ardından kontrol edilecek değişken yazılır. Küme parantezleri ({ }) içerisinde koşullar belirtilir. *Case* kelimesi kullanılarak değişkenin alabileceği koşullar belirtilir ve iki nokta üst üste ile kod parçasına geçilir. Bir sonraki *case* kelimesine kadar kod bloğu devam eder. Son olarak *default*: kelimesi ile hiçbir koşul sağlanamadığı taktirde çalıştırılacak olan kod bloğu yazılır.

```
switch KontrolEdilecekDeğişken {
    case KoşulDeğeri1:
        // Kontrol Edilecek Değişken koşul değeri 1 e eşitse bu kod
        // bloğu çalışır.

    case KoşulDeğeri2:
        // Kontrol Edilecek Değişken koşul değeri 2 ye eşitse bu kod
        // bloğu çalışır.

    default:
        // Hiçbir koşul sağlanmadığı taktirde bu kod bloğu çalışır.

}
```

Yazım Kuralı 2 : Switch Case Yapısı



Kare Kod 4 : <https://github.com/ios-kitap/KontrolYapilari>

Kontrol yapıları ile ilgili tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

14. Döngüler

Programlamada tekrar edilmesi gereken işlemler olduğunda döngüler kullanılır. Döngüler sayesinde onlarca satır ile yapılabilecek işlemler birkaç satır ile kolayca gerçekleştirilebilmektedir.

a. For In Döngüsü

Başlangıç ve bitiş değerlerini belirlediğimiz döngü biçimdir. Dizilerde de kullanılabilir.

```
123
124      //Döngüler-----
125
126          //For In Döngüsü
127          for sayı in 1...5 {
128              print(sayı)
129          }
130
131          for diziElemani in birDizi {
132              print(diziElemani)
133          }
134
```

Resim 25 : For In Döngü Yapısı

For kelimesi ile bir for döngüsü yaratılmak istediği belirtilir. Ardından bir döngü değişkeni tanımlanır. Bu döngü değişkeni duruma göre belirtilen başlangıç ve bitiş değerlerini taşıyacak olan değişkendir. In kelimesi ile döngü değişkeninin bağlı olduğu dizi ya da aralık belirtilir. Sonrasında küme parantezi ({ }) arasına kod bloğu yazılır. Belirtilen aralık tamamlanana kadar kod bloğu tekrar çalıştırılır. Döngü değişkeni her çalışmada değişecektir. Değişken bir diziye bağlı ise her çalışmada dizinin diğer elemanına geçiş yapar.

```
for Döngü Değişkeni in Döngü Aralığı / Dizi {
    // Tekrar edecek döngü kodbloğu
}
```

Yazım Kuralı 3 : For in Döngü Yapısı

b. While Döngüsü

While döngüsünde bir döngü değişkeni bulunmaz. Bunun yerine yalnızca koşul taşımaktadır. Bu koşul sağlandığı sürece kod parçası tekrar çalıştırılacaktır. While döngüsü genellikle döngünün kaç defa çalıştırılacağı belli olmayan durumlarda kullanılır. Döngü bir koşula bağlanarak koşulun program bağlamında değişip değişmediği her kod bloğu çalışmasından önce kontrol edilir. Koşul ilk başta sağlanıyorsa, döngü hiç çalışmamayabilir.

```
135
136      //While Döngüsü
137      var i = 0
138      while (i<5) {
139          print(i)
140          i+=1|
141      }
142
```

Resim 26 : While Döngüsü

```
while Koşul {  
    // Döngünün tekrar edecek kod bloğu  
}
```

Yazım Kuralı 4 : While Döngü Yapısı

While anahtar kelimesi ile bir *while* döngüsü yaratılmak istediği belirtilir. Ardından döngünün çalışma koşulu mantıksal operatörler ile veya boolean değerler ile girilir. Küme parantezi ({}) arasına yazacağımız her kod döngünün tekrar edecek olan kod bloğu olacaktır.

c. Repeat – While Döngüsü

Repeat-while döngüsü çalışma mantığı açısından while döngüsüne çok benzemektedir. *While* döngüsünden farkı kontrol işlemi döngünün sonunda yapılmaktadır. Bu özelliğinden dolayı, döngü en az bir kez kontrol satırına kadar çalışmaktadır. Eğer koşul sağlanmıyorsa bu durumda döngüden çıkmaktadır.

```
3 //Repeat-While Döngüsü  
4 var i = 0  
5  
6 repeat {  
7     print(i)  
8     i += 1  
9 }while i < 10  
10 print("Son")
```

Resim 27 : Repeat - While Döngüsü

```
repeat {  
    // Döngünün tekrar edecek kod bloğu  
}  
} while Koşul
```

Yazım Kuralı 5 : Repeat - While Döngü Yapıları

Repeat anahtar kelimesi ile *repeat-while* döngüsü yaratılmak istediği belirtilir. Ardından küme parantezleri ({}) ile belirttiğimiz kısım döngünün tekrar edecek olan kod bloğudur. Küme parantezinin kapatımından hemen sonra while anahtar kelimesi konularak ardından boolean değer ile ya da mantıksal operatörler kullanılarak döngünün çalışma koşulu yazılır.



Kare Kod 5 : <https://github.com/ios-kitap/Donguler>

Döngü yapıları ile ilgili tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

15. Fonksiyonlar

Fonksiyonlar (metotlar veya prosedürler) yazılım kavramının erken dönemlerinin hemen arkasından ortaya çıkan yapılardır. C gibi Prosedürel programlama dilleri, fonksiyonlar üzerine kuruludur. Bu programlama dilleri kullanılarak geliştirilen

büyük projelerde kod yiğinları arttıkça içinden çıkılmaz bir hal almaya başlar. Bu durumdan kurtulmanın ilk aşaması fonksiyonlara başvurmaktır.

Ayrıca program içerisinde yürütülen işlemlerin bazıları özel durumlarda ve sadece istenildiği zamanlarda kullanılmak istenebilir. Ana program döngüsü dışında harici olarak gruplanan bu işlemler sadece gerektiği yerde çağrıldığı taktirde o kısımlarda çalışacak ve ana program döngüsü içerisine dahil edilecektir.

Fonksiyonlar yapılımak istenen bir işin ayrı bir program parçası haline getirilip istenildiğinde çalıştırılarak sonuçlar elde edilmesine yararlar.

a. Temel Fonksiyonlar

Fonksiyon kendi kod bloğuna verilen işlemleri yerine getirir ve işlemleri tamamlar. Kendi kod bloğuna dışarıdan herhangi bir veri almaz.

```
3 let a = 4
4 let b = 3
5
6 func topla(){
7     let sonuc = a + b
8     print(sonuc)
9 }
10
11 topla()
```

Resim 28 : Temel Fonksiyon Tanımlanması ve Kullanımı

```
func fonksiyonAdı() {
    // Fonksiyon kod blogu.
}
```

Yazım Kuralı 6 : Temel Fonksiyon Yapısı

func anahtar kelimesi ile fonksiyon tanımlanmak istediği belirtilir. Ardından bu fonksiyona daha sonra fonksiyon kullanılırken çağrıılacak olan isim girilir. Sonrasında normal parantezler açılıp kapatılır. Bu parantezlerin varlık sebebi, parametreli fonksiyonlar başlığında ayrıca anlatılacaktır. Her zamanki gibi küme parantezi açılması ile fonksiyonun çağrıılması ile çaplışturılacak olan kod bloğumuz başlar ve küme parantezinin kapatılması ile kod bloğu sona erer.

b. Parametrelî Fonksiyonlar

Parametrelî fonksiyonlar çağrıldıklarında içlerine veri alabilen fonksiyonlardır. Bazı durumlarda bu verilerin kullanımı istege bağlı biçimde getirilebilmektedir.

Bu fonksiyonlar çağrıldıklarında, fonksiyon hazırlanırken kendisine belirtilen bazı verileri, fonksiyonun kod bloğu çalıştırılırken kullanılmak üzere talep eder. Sonrasında bu veriler fonksiyon çalıştırıldığında fonksiyon içerisinde kullanılır.

```
18 let adsoyad = "Emre Erol"
19
20 func isimYazdir(isim:String) {
21     print("Merhaba \(isim)")
22 }
23
24 isimYazdir(isim: adsoyad)
```

Resim 29 : Parametrelî Fonksiyon Tanımlanması ve Kullanımı

```
func someFunction(parametreAdı: parametreTipi) {
    // Fonksiyonun içeriği bu kısımda yer alır.
}
```

Yazım Kuralı 7 : Parametrelî Fonksiyon Yapısı

Tüm yazım şekli temel fonksiyonlarla aynı olmasına karşın burada normal parantezler arasında fonksiyon parametresi tanımlanır. Parantez içerisinde, parametrenin ismi ve iki nokta üst üste ile ayrılmış biçimde parametrenin tipi belirtilir.

Fonksiyon çağrılığında ise fonksiyonun adı ve normal parantezler arasında parametre ismi girilerek iki nokta üst üste ile bu parametrenin alacağı veri girilir. Veri direkt olarak kullanılabilir gibi istenirse değişkenin ismini de kullanılabilir.

Parametreli fonksiyonlarda birden fazla parametre girişi de kullanılabilir. Bu durumda her parametre belirtiminden sonra virgül (,) konularak parametreler başka bir parametrenin daha belirtildiği işaretlenir. Çağırıldığından da tanımlanmasında olduğu gibi veriler virgül ile ayrılarak normal parantezler içerisinde alınır.

```

1
2
3 func fonksiyon(veri1:Int,veri2:String,veri3:Double) {
4
5 }
6
7

```

Resim 30 : Birden Çok Parametre Alan Fonksiyon Tanımlanması ve Kullanımı

c. Geriye Veri Dönən Fonksiyonlar

Bazı durumlarda fonksiyonların bir sonuca ulaşmaları ve bu sonucu geriye döndürmeleri gerekebilir. Örneğin; bir toplama işleminin sonucunu ya da erişim sağlanan bir resmin işleminden geçirilerek değiştirilmiş resmin sonucunu geriye, fonksiyon dışarısına alınması gerekebilir. Bu fonksiyonlar hazırlanırken geriye döndürülecek olan verinin hangi tipte olacağı belirlenir. Daha sonra kod bloğu içerisinde *return* anahtar kelimesi ile o tipteki veri döndürülür.

```
func veriDonenFonksiyon() -> DonecekVeriTipi {
    return DonecekDeger
}
```

```
let alinandeger = veriDonenFonksiyon()
```

Yazım Kuralı 8 : Geriye Veri Dönen Fonksiyon Tanımlanması ve Kullanımı

Geriye veri dönen fonksiyonların tanımlaması; parametre girişinin belirtildiği normal parantezin kapanışı ile kod bloğunu başlatan küme parantezinin açılışı arasında yapılır. Bu kısıma yazılan tire (-) işaretile hemen ardından gelen büyükür işaretile (>) kodu okuyan açısından bir ok görünümü kazandırılarak fonksiyonun geriye veri döndüreceği belirtilir. Hemen ardından geriye dönecek olan verinin tipinin yazılması ile geriye veri döndüren fonksiyon tanımlaması tamamlanır.

Fonksiyon tanımlaması sırasında yapılan bu işlemlerden sonra kod bloğu içerisinde fonksiyonun çağrıldığında çalıştırılacak olan tüm işlemler yazılır. Bu fonksiyonlarda return anahtar kelimesi kullanılması zorunludur. Eğer kod bloğu içerisinde return anahtar kelimesi bulunmazsa hata alınacaktır. Yapılan tüm işlemler sonucunda geriye döndürülecek olan veri tipinde bir veri return anahtar kelimesinin sağına yazarak geri döndürme işlemi yapılır.

Return anahtar kelimesi bir fonksiyon içerisinde birden fazlada kullanılabilir. Hazırlanan fonksiyon, kod bloğu içerisinde *If else* gibi kontrol ifadeleri taşıyorsa her koşulda farklı veri döndürürmek istenebilir. Bu durumda birden fazla *return* anahtar kelimesi kullanılmasının bir sakıncası yoktur.

```
1
2
3 func islemYapilmisi(deger:Int)->String{
4     if deger == 1 {
5         return "İşlem yapılmış"
6     }else if deger == 2{
7         return "İşlem yapılmamış"
8     }else{
9         return "Bir hata meydana gelmiş"
10    }
11 }
12 }
```

Resim 31 : Birden Fazla Return Anahtar Kelimesinin Kullanımı

Bazı durumlarda fonksiyonların birden fazla veri döndürmeleri gerekebilir. Normalde veri döndüren fonksiyonlarda yapılan işlemlerin aynısı bu fonksiyonlarda da yapılır. Ancak bu defa geriye döndürülecek olan veriler, parantez içerisinde ve anahtar “kelime:veri tipi” formatı ile belirtilir. Sonrasında fonksiyon çağrııldığından sonuçlar bu anahtar kelimeler ile ayrı ayrı alınabilir.

```
39
40 let x = 21
41 let y = 12
42
43 func dortIslem() -> (toplam:Int,cikarma:Int,carpma:Int,bolme:Float) {
44     let toplamSonucu = x + y
45     let cikarmaSonucu = x - y
46     let carpmaSonucu = x * y
47     let bolmeSonucu:Float = Float(x) / Float(y)
48     return (toplamSonucu,cikarmaSonucu,carpmaSonucu,bolmeSonucu)
49 }
50
51 print("Toplam Sonucu \(dortIslem().toplam)")
52 print("Toplam Sonucu \(dortIslem().carpma)")
53 print("Toplam Sonucu \(dortIslem().cikarma)")
54 print("Toplam Sonucu \(dortIslem().bolme)")
```

Resim 32 : Birden Fazla Veriyi Geri Döndüren Fonksiyonlar

d. Değişken (Belirsiz) Sayıda Parametre Alan Fonksiyonlar

Variadic fonksiyonlar olarak karşımıza çıkan bu fonksiyon türü aslında gelişmiş bir parametrelî fonksiyondur. Daha önce bahsettiğimiz parametrelî fonksiyonlar, içerisinde bizim belirlediğimiz sayıda parametre alabilmekteydi. Ancak ihtiyaç halinde bu fonksiyonlar; içerisine, aynı tipte olmak kaydı ile belirsiz sayıda parametre alacak biçimde tanımlanabilirler. Bu sayede bir fonksiyonun içerisinde ne kadar parametre gireceğinin bilinmediği hallerde bu formatta bir tanımlama yapılarak çözüme ulaşılabilir.

```

1
2
3 func meyveleriYaz(_ meyveler:String...){
4     for meyve in meyveler {
5         print(meyve)
6     }
7 }
8
9 meyveleriYaz("elma", "armut", "üzüm")
10
11

```

Resim 33 : Değişken Sayıda Parametre Alan Fonksiyonların Tanımlanması ve Kullanımı

```

func fonksiyonAdı (_ parametreAdı : ParametreVeriTipi ...) {
    // Fonksiyon kodbloğu.
}

```

Yazım Kuralı 9 : Değişken Sayıda Parametre Alan Fonksiyonlarının Tanımlanması ve Kullanımı

Variadic fonksiyonlar tanımlanırken parametre tanımlaması kısmında; alt tire (`_`) ile parametre tanımlamaları başlanır. Burada alt tire fonksiyon çağrıldığında parametre girişi yapılrken parametre adını kullanmamak için bir işaret görevi görmektedir. Sonrasında yine parametrenin adı ve iki nokta üst üste ile beraber fonksiyonun içerisinde olması gereken verilerin parametrelerin veri tipi belirtilir. Tip belirlemesinden sonra konulan üç nokta (`...`) ile bu verinin bilinmeyen sayıda olacağı ifade edilir.

Variadic fonksiyonlar çağrıldığında ise parametreler yine parametre parantezlerinin içerisinde bu defa parametre adı kullanılmadan sadece tüm parametrelerin arasında virgül kullanılarak sırasıyla girilirler.

e. Nested (İçiçe) Fonksiyonlar

Şimdiye kadar kullandığımız tüm fonksiyonlar bir şekilde ana program döngüsü içerisinde kullanılabilen fonksiyonlardır. Ancak swiftin bize sunduğu bir diğer avantaj ise nested fonksiyonlar olarak karşımıza çıkan iç içe fonksiyonlardır.

Bu fonksiyonlar fonksiyonlarının içerisinde tanımlanırlar ve ana program döngüsüne kapalıdır. Diğer fonksiyonlar gibi istenildiği yerde kullanılamazlar. Yalnızca tanımlandıkları fonksiyon içerisinde çağrılp kullanılabılırler.

```
1
2
3 func selamlamaMesajıYaz(_ mesaj: String) {
4
5     //Nested Fonksiyon Tanımlanıyor!
6     func merhabaEkle() {
7         print("Merhaba! \(mesaj)")
8     }
9
10    merhabaEkle()
11 }
12
13 selamlamaMesajıYaz("Geliştirici")
14
```

Resim 34 : Nested Fonksiyonların Tanımlanması ve Kullanımı



Kare Kod 6 : <https://github.com/ios-kitap/Fonksiyonlar>

Fonksiyon yapıları ile ilgili tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

16. Enumeration Kullanımı

Enumeration' lar birbiriyle mantıksal olarak bağlı bulunan verilerin gruplanarak proje içerisinde güvenli biçimde bu verilere erişilmesine ve kullanılmasına olanak sağlayan yapılardır. Dizilerde olduğu gibi indeks numarası ile erişim sağlanabildiği gibi, tanımladıkları isimle de erişim sağlanabilir.

Enumeration' ların temelinde verilerin mantıksal olarak grupperlendirilmesi yatar. Örnek vermek gerekirse eğer, bir pusulanın üstündeki ana yönler ya da takvimdeki bütün aylar, haftanın 7 günü enumeration grupplaması için uygun olacaktır.

```
enum Enumerationİsmi {  
    // Tanımlamalar bu bölüme yapılmalıdır.  
}
```

Yazım Kuralı 10 : Enumeration Tanımlaması

```
1 enum yonler {  
2     case kuzey  
3     case guney  
4     case dogu  
5     case batı  
6 }  
7  
8 let secilenYon = yonler.guney  
9  
0 switch secilenYon {  
1     case .kuzey:  
2         print("Kuzey yönü seçilmiş!")  
3     case .guney:  
4         print("Güney yönü seçilmiş!")  
5     case .dogu:  
6         print("Doğu yönü seçilmiş!")  
7     case .batı:  
8         print("Batı yönü seçilmiş!")  
9     default:  
0         print("Böyle bir yön yok!")  
1 }
```

Resim 35 : Coğrafi Yönlerin Enumeration Biçiminde Tanımlanması ve Kullanımı

17. Hata Yakalama ve Hata Yönetimi

Bir proje geliştirilirken dikkatle düşünülmesi gereken en önemli konulardan bir diğeri, hata yönetimidir. Projelerde beklenen ya da beklenmeyen bir hata meydana geldiğinde, yazılımın nasıl davranışacağını belirlemek, yazılımın kalitesini artıracaktır. Kuşkusuz geliştiricinin tecrübe ve yeteneği ile doğru orantılı bir kavramdır ancak swift programlama dili bize bu yeteneği kolaylıkla sürdürmemiz için gereken esnekliği sağlamaktadır.

Bir hatayı yakalamadan önce o hatayı tanımlamak doğru bir yaklaşımdır. Bazı hatalar önceden sistem içerisinde tanımlı olmasına karşın *enum* tanımlamaları ile kendi hatalarımızı tanımlayabiliriz.

```
42 enum kalemHata: Error {  
43     case murekkepYok  
44     case kapakKapali  
45 }
```

Resim 36 : Enumeration İle Hata Tanımlamaları

Fonksiyon içerisinde meydana gelecek olan hatalar fonksiyonların hataları fırlatması ile yakalanabilir. Bir fonksiyon tanımlanırken kullanılan *throws* anahtar kelimesi fonksiyonun meydana gelme ihtimali olan bir hatayı fırlatacağı anlamına gelir. Sonrasında istenmeyen durumun bulunduğu satırda *throw* anahtar kelimesi kullanılarak hata daha önce belirlenen enum tanımlaması ile birlikte fırlatılabilir.

```
47 func yaziYaz() throws {  
48     if murekkep == 0 {  
49         throw kalemHata.murekkepYok  
50     }  
51  
52     if !kapak {  
53         throw kalemHata.kapakKapali  
54     }  
55  
56     print("Yazı Yazıyor...")  
57 }
```

Resim 37: Fonksiyon İçerisinde Hata Fırlatma

Fonksiyon programın herhangi bir yerinde çağrııldığında hata yakalama işlemi *do - try - catch* bloğu uygulanarak yakalanabilir. Bu bölümde eğer fonksiyon hata fırlatırsa, çalışan satır *try* satırından *catch* kod bloğuna atılır ve çalışma buradan devam eder.

```
59  do{
60    try yaziYaz()
61    print("Yazı yazıldı!")
62  }catch kalemHata.murekkepYok {
63    print("Mürekkep yok, yazı yazılamadı.")
64    murekkepDoldur() //Mürekkebi dolduralım
65  }catch kalemHata.kapakKapali {
66    print("Kapak kapalı. Yazı yazılamıyor.")
67    kapakAc() //Kapağı açalım
68 }
```

Resim 38 : Do-Try-Catch Bloğu İle Fırlatılan Hatanın Yakalanması

```
do {

    // Devam eden kodlar..

    try HataFırlatanFonksiyon()

    // Devam eden kodlar..

} catch yakalananHata {

    // Eğer hata tipi yukarıdaki gibiysse

    // Bu kısımdaki kodlar çalıştırılır.

} catch YakalananBaskaBirHata {

    // Eğer hata tipi yukarıdaki gibiysse

    // Bu kısımdaki kodlar çalıştırılır.

} catch {
```

// Hata meydana gelmişse ancak tipi

// Bilinmiyor yada önemsiz ise bu

// kısımdaki kodlar çalıştırılır

}

Yazım Kuralı 11 : Hata Yakalama Do-Try-Catch Yazım Kuralı

18. Sınıf (Class) Yapıları ve Nesnel Programlama

Nesne yönelimli programlama olarak kaynaklarda rastlanabilecek olan nesnel programlama, temelinde bir programlama yaklaşımıdır. Prosedüral programlama dillerinin, geliştiricilere yaşattığı sıkıntıların sonucunda ortaya çıkmıştır. O dönemlerde görece büyük projelerin sadece fonksiyonlar ile yazılması ile projelerde büyük bir kod karmaşıklığına sebebiyet vermiştir. Ayrıca bu biçimde geliştirilen projelerde daha sonrasında yapılacak olan bakım ya da güncelleme gibi projenin yayınlanması sonrasında karşı karşıya kalınan durumlarda, geliştiriciler devasa kod yoğunlarının arasından çökmemeliyordu. Doğal olarak bu da yazılımların maliyetlerini ciddi oranda artırıyordu.

Tüm bu problemlerin çözümüne ilişkin olarak atılan öneri ise Object Oriented Programming (OOP) ile programlama yaklaşımı oldu.

Nesnel programlama yaklaşımı ortaya atılırken gerçek dünya örnek alındı. Gerçek dünya da herşeyin bir nesne (Object) olmasından yola çıkılarak bu durum yazılıma yansıtıldı.

Gerçek hayatı nesneleri düşünelim. Örneğin bir koltuğu ele aldığınızda, koltuğun bazı nitelik özelliklerinin olduğunu fark edersiniz. Koltuğun rengi, kumaşı, yumuşaklık seviyesi gibi onu niteleyici özellikleri (attributes) vardır. Ayrıca nesnelerin ki bizim örneğimizde koltuğun bazı fonksiyonları da (methods) mevcut olabilir. Bazı durumlarda koltuk yatağına dönüştürmektedir. Bunun dışında koltuğu oluşturan aslında başka nesneler vardır. İçerisinde ahşap, demir, iplik gibi başka nesneler kullanılmıştır. Bu nesneler, koltuk özellik ve fonksiyonlarına sahip olmasalarda, kendi özellik ve fonksiyonlarını koltuk nesnesi içerisinde de korumaktadır. Dolayısı ile nesneyi oluşturan diğer küçük nesneler ile arasında da kalıtımsal (Inheritance) bir bağ gözle çarpmaktadır.

Yukarıdaki örnekteki gibi, nesnel programlama hayatın içinden örnek alınarak, programlamayı daha kolay hale getirmek ve bu yönde bakım maliyetlerini düşürmek amacıyla oluşturulmuştur.

Bir kaç temel kavramı bulunmaktadır;

a. *Sınıf Yapıları (Class)*

Öncelikle bahsettiğimiz nesneleri oluşturmak için sınıf yapıları *class* anahtar kelimesi kullanılarak hazırlanır. Bu sınıf yapıları aslında yazılımdaki nesnelere karşılık gelmektedir. Sınıflar hazırlanıktan sonra yazılım içerisinde türetilerek (Derivative) kullanılırlar. Sınıflar içerisinde nesnelerin özellikleri ve fonksiyonları değişkenler ve metotlar olarak hazırlanır. Bu değişken ve metotlar hazırlanırken, erişim hiyerarşileri ile iç ve dış erişime açık olup olmadıkları belirlenir.

```
class SınıfAdı {  
    // Bu sınıfa ait özellik ve fonksiyonlar  
}
```

Yazım Kuralı 12 : Sınıf Tanımlanması

```
1 class sınıf {  
2     //Özellikler  
3     var birOzellik:String!  
4     var baskabirOzellik:Int!  
5  
6     //Fonksiyonlar ~ Metotlar  
7     func sınıf(){  
8  
9     }  
10    func birFonksiyon(){  
11  
12    }  
13    func baskaBirFonksiyon(){  
14  
15    }  
16  
17    }  
18 }
```

Resim 39 : Swift Dilinde Sınıf Yapısı Özellikler ve Fonksiyonlar

b. Özellikler ve Fonksiyonlar (Methods)

Hazırlanan sınıflar değişkenler ve fonksiyonlardan oluşur. Aslında bu durum nesnel programmanın temelini oluşturur. Bir konu ile ilgili ya da bir nesne ile ilgili tüm değişkenleri ve fonksiyonları gruplamak, nesnel programmanın temel mantığıdır.

i. Yapıcı Metotlar - Constructors

Bir sınıf ilk türetildiği-üretildiği anda yapıcı metotlar çalıştırılırlar. Bu özelliği itibarıyle yapıcı metotlar, içlerinde bulunduğu sınıfların hazırlık metotudur denilebilir. Örneğin; bir resim ile ilgili işlem yapan bir sınıf hazırladığımızı varsayıyalım. Bu sınıf ilk oluşturulurken, yapıcı metot kullanılarak işlem yapılacak olan resim sınıf içerisine atanabilir. Ya da sınıf ilk türetildiğinde bazı işlemler yapılmak istenebilir. *init* anahtar kelimesi ile tanımlanırlar ve bu metotlar geriye veri döndüremezler. Varsayılan olarak public erişime sahiptirler.

```
1
2
3 class ogrenci {
4
5     var ogrenciAdi:String
6
7     init(ogrenciAdi:String) {
8         self.ogrenciAdi = ogrenciAdi
9         print("Yapıcı metot ile dışarıdan alınan öğrenci adı : \(self.ogrenciAdi)")
10    }
11
12
13 }
14
15
16 let student = ogrenci(ogrenciAdi: "Emre Erol")
17
```

Resim 40 : Bir Sınıf İçerisinde Yapıcı Metot Tanımlanması ve Kullanımı

ii. Yıkıcı Metotlar

Bazı durumlarda türetilen sınıflara ihtiyaç kalınmadığı an da işlemler yapılması gerekebilir. Bu durumlarda ise yıkıcı metotlar kullanılır. *deinit* anahtar kelimesi ile tanımlanırlar.

Bir yıkıcı metodun kullanılabilmesi için o sınıfın türetildiğinde atandığı değişkenin nil değeri alabilir biçimde işaretlenmesi gerekir.

```
1
2
3 class ogrenci {
4
5     var ogrenciAdi:String
6
7     init(ogrenciAdi:String) {
8         self.ogrenciAdi = ogrenciAdi
9         print("Yapıcı metot ile dışarıdan alınan öğrenci adı : \(self.ogrenciAdi)")
10    }
11
12    deinit {
13        print("Türetilmiş sınıf ortadan kaldırılıyor!")
14    }
15
16 }
17
18
19 var student:ogrenci? = ogrenci(ogrenciAdi: "Emre Erol")
20 student = nil //deinit yıkıcı fonksiyonu çalıştırıldı!
21
```

Resim 41 : Bir Sınıf İçerisinde Yıkıcı Metot Tanımlanması ve Kullanımı

c. Kalıtım (Inheritance)

Kalıtım ile nesnelerden nesneler üretilmesi prensibi ortaya konulmuştur. Bu sayede daha önceden hazırlanmış olan sınıf yapıları içerisinde yer alan özellik ve fonksiyonları, yeni bir sınıf oluşturulurken tekrar yazılmadan kullanılabilmektedir. Bir nevi kalıtımsal olarak yukarıdan tüm bu özellik ve fonksiyonları yeni oluşturulan sınıf almaktadır. Bu durum yeni oluşturulan sınıfın kendisine ait yeni özellik ve fonksiyonların oluşturulmasını engellemez.

```
1
2 class insan {
3
4     let ayakSayisi:Int = 2
5     let kolSayisi:Int = 2
6
7     func yürü(){
8         print("Yürüyor")
9     }
10
11    func koş(){
12        print("Koşuyor")
13    }
14
15 }
16
17 class superKahraman : insan {
18     func uç(){
19         print("Uçuyor")
20     }
21 }
22
23 let superman = superKahraman()
24
25 superman.yürü()
26 superman.koş()
27 superman.uç()
```

Resim 42 : Sınıflarda Kalıtım Örneği

Resim 48' de görünen örnekte bir insan sınıfı oluşturulmuştur. Bu insan sınıfının yürümeye ve koşmaya yarayan iki fonksiyonu bulunmaktadır. İnsan sınıfının hemen altında ise "superKahraman" isminde bir sınıf oluşturulmuş ve iki nokta üst üste ile insan sınıfından kalıtım aldığı belirtilmiştir. Ek olarak superKahraman sınıfına uçma fonksiyonu eklenmiştir. Sınıflar tanımlandıktan sonra da hemen alt kısmında bir "superKahraman" sınıfı superman ismi ile türetilmiştir. Görüldüğü gibi "superman" nesnesi, "insan" sınıfından türetilmediği halde yürüme ve koşma

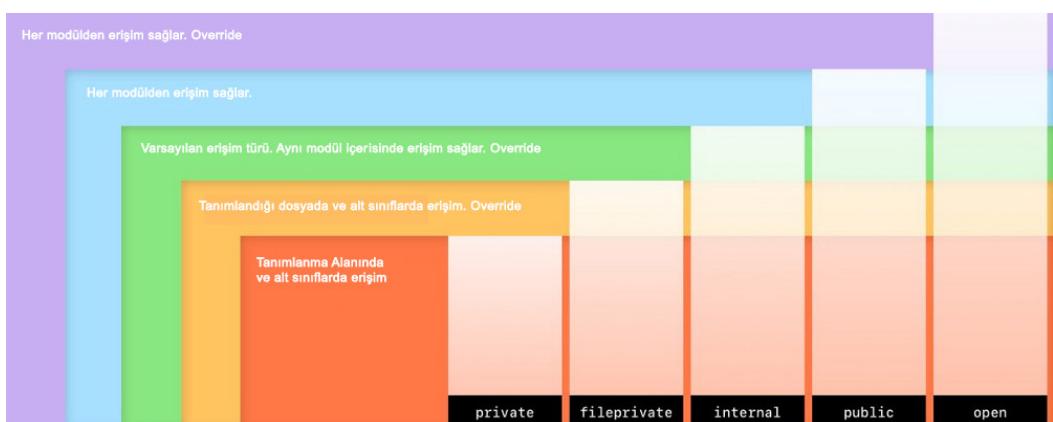
eylemlerini gerçekleştirebilmektedir. Çünkü bu metodları “superKahraman” sınıfı kalıtımsal olarak insan sınıfından almaktadır. Ancak aynı zamanda uçma fonksiyonunu da kullanabilmektedir.

```
class SınıfAdı : Kalıtım Alınacak Sınıf {  
    // Bu sınıfa ait özellik ve fonksiyonlar burada yer alır  
}
```

Yazım Kuralı 13 : Kalıtım Yapısı

d. Kapsülleme (Encapsulation)

Sınıf yapıları başlığı altında bahsettiğimiz erişim hiyerarşileri kapsülleme başlığı altında incelenmektedir. Burada ifadeye konu olan erişim hiyerarşisinden kasıt, sınıfların kendilerinin, sınıf içerisinde oluşturulan nesnenin özelliklerinin ve fonksiyonlarının diğer nesneler tarafından erişilebilir olup olmadığıdır. Kapsülleme bazı anahtar kelimeler aracılığıyla yapılır ve değişken ve fonksiyon tanımlamalarının başında bu anahtar kelimelerinin (public, private, fileprivate, vb...) kullanımı ile hazırlanır.



Resim 43 : Kapsülleme ve Erişim Kontrol Grafiği

e. Soyutlama (*Abstraction*)

Soyutlama abstract sınıf yapılarını anlatmak için kullanılan bir kavramdır. Soyutlama ile “hazırlanacak olan” sınıf yapılarının yanlışca iç yapısı ve erişim hiyerarşisi belirlenir. Ancak detaylar belirtilmez. Bu yönüyle soyut (*Abstract*) sınıflar türetilmez ve normal sınıflar gibi kullanılamazlar. Çünkü onlar soyut nesnelerdir.

Bir kalemi hayal ettiğinizi varsayıyalım. Hayal ettiğiniz bir kalem ile yazı yazamazsınız. Ancak bu kalemi hayalinizden esinlenerek hayal olmaktan çıkarıp gerçek bir kalem üretirseniz bu durumda o gerçek olan kalemle yazabilirsiniz. İşte *abstract* sınıflar ve soyutlama kavramı hazırlayacağımız sınıfların hayali tarafını ifade etmektedir. Soyut sınıflardan genişletilen (*extend*) gerçek sınıflar, soyut sınıflar içerisinde belirtilen özellikleri alarak oluşmak zorundadırlar.

Soyutlama kavramı hem geliştirici için bir sınırları belirleme kolaylığı sunarken hem de başka geliştiricilerin hazırlanan bu yapıları daha kolay anlamlandırmamasına yarar.

Ne var ki swift programlama dili şu anda soyutlama özelliğinden yoksundur. Yoksun olmasının en önemli sebebi, Swift programlama dilinin hem Nesnel Yönetimli hemde Protokol Yönetimli bir dil olmasına yatkınlıdır. Ancak soyutlama bir tasarım deseni kullanılarak yapılmaya çalışılmaktadır. Boş fonksiyonlardan ve özelliklerden oluşan genel bir sınıf tanımlanarak bu sınıf kalıtım yolu ile başka bir sınıfta türetilir. Sonra içi boş olan fonksiyonların polimorfizm ile override edilerek (geçersiz kılınarak) yeniden yazılır.

Bir başka soyutlama yöntem olarakda *protocol* ismi verilen parçaların kullanılmasıdır. Protokoller Swift programlama nesnel yönetimli bir dil olmasının yanısıra protokol yönetimli bir dil olması sonucunda, dile eklenmiş yapılardır.

Aşağıda öncelikle tasarım deseni metodу ile nasıl soyut sınıf hazırlanabileceğine ilişkin örnekler ekran görüntüleri ile anlatılmaya çalışılmıştır.

```
1
2 //Abstract-Soyut Sınıf
3 class Hayvan {
4     func sesCikar() {}
5 }
6
7 //Gerçek Sınıf
8 class Kedi: Hayvan {
9     override func sesCikar() {
10         print("miyav")
11     }
12 }
13
14 //Gerçek Sınıf
15 class Ordek: Hayvan {
16     override func sesCikar() {
17         print("vakvak")
18     }
19 }
20
```

Resim 44 : Soyut Sınıf Tanımlama Örneği

41 numaralı resimde görüldüğü üzere en başta tanımlanan “Hayvan” “ismindeki sınıf “sesCikar” isimli bir fonksiyonu barındırmaktadır. Ancak dikkat edilirse fonksiyonun hiçbir işlem yapmadığı ve boş olduğu görülmektedir. Sonrasında ise “Hayvan” isimli sınıfın başka sınıflar kalıtım alınarak üretilmiştir ve içi boş olan “sesCikar” isimli fonksiyon override edilerek iç yapısı ayrı ayrı istenildiği gibi yazılmıştır. Bu nokta da diğer sınıflar gerçek bir sınıf olma özelliği gösterirken, “Hayvan” isimli sınıf yalnızca bir şablon görevi görmektedir. Dolayısı ile soyut bir yapısı vardır.

i. Protokoller ve Protocol Oriented Programlama

Daha öncede ifade edildiği gibi swift programlama dili hem nesnel yönelimli (Object Oriented) hemde protokol yönelimli (Protocol Oriented) bir yapıya sahiptir. Protokoller nesnel programmanın temel kavramı olan soyutlama

ozelligini tam olarak karşılamasada, kısmen soyut sınıflarımıçasına üretilip kullanılabilmektedirler.

Aşağıda; Resim 41 üzerinde gerçekleştirilen tasarım deseni ile üretilmiş soyut sınıf hazırlama yönteminin aynısı protokoller kullanılarak hazırlanan biçimi ekran görüntüsü ile anlatılmaya çalışılmıştır.

```
1 protocol kediSesi {
2     func sesCikar()
3 }
4
5 protocol ordekSesi{
6     func sesCikar()
7 }
8
9 class Kedi : kediSesi {
10    func sesCikar() {
11        print("Miyav")
12    }
13 }
14
15 class Ordek : ordekSesi {
16    func sesCikar() {
17        print("vakvak")
18    }
19 }
```

Resim 45 : Protokoller

Ekran görüntüsünde görüldüğü gibi, en başta *protocol* anahtar kelimesi ile Hayvan isminde bir protokol tanımlanmıştır. Bu protokol içerisinde bir fonksiyon barındırmaktadır ve sonrasında üretilen bütün sınıflar Hayvan protokolünden kalıtım almış, bir başka deyişle genişletilmiş ve içerisindeki fonksiyonlar yeni yaratılan sınıflara eklenmiştir.

Resim 41' de tasarım deseni ile yapılan örnek protokoller ile ifade edilmeye çalışıldığından çok büyük farklılıklar olmadığıotope çarpar. Ancak protokollerin varlık sebebi kullanım esnekliklerinden gelmektedir. Burada asıl amaç, nesnel programmanın getirdiği özellik olan kalıtım yapısının daha kullanışlı bir hale getirilmeye çalışılmasıdır. Bir sınıfın kalıtım alıp o sınıfı türettiğimizde kalıtım alınan sınıfın bütün özellik ve fonksiyonları yeni türetilen

sınıf içerisinde barındırılmaktadır. Oysa kalıtım alınan sınıfın bazı özellik ve fonksiyonları yeni yaratılan ve türetilen sınıf için gereksiz olabilir. Bu durumun ortadan kaldırılması protokol yönelikli olarak hazırlanan bir yapıda istenilen protokoller kullanılırken, istenmeyen protokoller yeni yaratılan sınıfa dahil edilmeyerek durumuna bağlıdır.

f. Çok Biçimlilik (Polymorphism)

Çok biçimlilik kavramı görece diğer başlıklara nazaran daha kolay anlaşılabilen ve kullanıcılar tarafından更容易理解的和易于使用的是“多态”而不是“重载”。也就是说，多态是面向对象编程中一个重要的概念，它指的是同一个方法可以在不同的类中实现不同的行为。而重载则是指同一个类中可以有多个同名的方法，它们根据参数的不同而有不同的行为。两者都是为了提高代码的可读性和可维护性。

Araba isimli bir sınıf oluşturduğumuzu düşünelim. Bu sınıf arabaların tüm ortak özelliklerini barındırır biçimde olacaktır. 4 tekeri, direksiyonu, kapıları, motoru gibi özellikleri bulunur. Ayrıca kapıların açılması, direksiyonun dönmesi, motorunun çalışması gibi de fonksiyonları olacaktır. Şimdi araba sınıfından kalıtım olarak elektrikli araba sınıfı hazırladığımızı düşünelim. Yine kapıları açılacaktır, yine direksiyonu dönecektir ancak motorunun çalışma prensibi değiştiğinden motor fonksiyonu geçersiz kılınıp yeniden hazırlanacaktır. Bu örneğin yazılım alanında kullanılması çok biçimliliktir.

```

1
2 class Araba {
3
4     let tekerSayisi = 4
5     let kapiSayisi = 4
6
7     func motorCalis() {
8         print("Yakit Gerekiyor")
9     }
10
11    func direksiyonDon(){
12        print("Direksiyon dönüyor")
13    }
14 }
15
16 class elektrikliAraba : Araba{
17     /*
18     Kalıtım yolu ile alınan motorCalis() fonksiyonu
19     elektrikliAraba sınıfında override ile
20     geçersiz kılınarak tekrar yazılıyor.
21     */
22     override func motorCalis() {
23         print("Pil gerekiyor...")
24     }
25 }
26

```

Resim 46 : Çok Biçimlilik Örneği ve Override Edilen Bir Fonksiyon



Kare Kod 7 : <https://github.com/ios-kitap/Classes>

Sınıf yapıları ve nesnel programlama ile ilgili tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

19. Structure Yapıları

Swift Programlama dilinde sınıf yapılarına çok benzer başka bir yapı daha bulunmaktadır. Bu yapılar *struct* anahtar kelimesi ile tanımlanan structure' lardır.

Structure' lar da sınıflar gibi tanımlanırlar. Yalnızca sınıflarda kullanılan *class* anahtar kelimesi yerine *struct* anahtar kelimesi kullanılır ve ardından structure için bir isim girilir. Sonrasında kod bloğu için küme parantezi açılır ve kapatılana kadar olan structure' in iç yapısını oluşturur.

```
struct StructureAdı {  
    // Özellik ve fonksiyonlar bu bölüme yazılır.  
}
```

Yazım Kuralı 14 : Structure Yapısı

Structure' lar kullanıcıları açısından da sınıflar ile benzerlik gösterirler. Örneğin sınıflar gibi türetilerek kullanılırlar. İçerinde özellikler ve fonksiyonlar tanımlanabilir. İlk değer atamaları için *init* yapıcı fonksiyonu kullanılabilir

Ancak bazı sınıf özellikleri structure yapılarında kullanılamazlar ki bunların en önemlisi kalıtmıdır.

Bu özellikleri itibarı ile structure' lar swift programlama dilinin sınıflardan bir numara küçük olan yapı taşıdır denilebilir.

```

14
15 struct Work {
16     var location = ""
17     var units = [String]()
18     let length:Int
19
20     func last() ->String{
21         return units[units.count-1]
22     }
23
24 }
25

```

Resim 47 : Structure Tanımlanması Örneği

Structure tanımlaması yapıldıktan sonra, sınıflar gibi türetilerek yazılım içerisinde kullanılırlar. Ayrıca içersindeki özellik ve fonksiyonlara erişimde yine sınıflardaki gibi türetilen değişken üzerinden nokta operatörü aracılığı ile yapılmaktadır.

```

2
3 //Structure Tanımlanıyor
4 struct KullanılacakRenkler{
5
6     var renk1:String
7     var renk2:String
8     var renk3:String
9
10    func varsayılanRenkleriAl() ->(renk1:String,renk2:String,renk3:String){
11        let r1 = "#000000"
12        let r2 = "#000000"
13        let r3 = "#000000"
14        return (r1,r2,r3)
15    }
16 }
17
18 //Structure Türetiliyor
19 var renkler = KullanılacakRenkler(renk1: "#FFFFFF", renk2: "#FF0000", renk3: "#454545")
20
21 //Structer özelliklerine erişiliyor
22 let renk1 = renkler.renk1
23 let renk2 = renkler.renk2
24 let renk3 = renkler.renk3
25
26 //Structure fonksiyonlarına erişiliyor
27 let alınanRenkler = renkler.varsayılanRenkleriAl()
28

```

Resim 48 : Structure Yapısı Türetimi, Özellik ve Fonksiyonlarının Kullanımı



Kare Kod 8 : <https://github.com/ios-kitap/Structures>

Structure yapısı ile ilgili tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

20. IBOutlet ve IBAction Kavramları

Viewler konusuna geçmeden önce IBOutlet ve IBAction kavramlarına değinilmesi gerekmektedir. Xcode arayüz tasarımda birçok kolaylık sağlamaktadır ve bu kolaylıklardan biride IBOutlet ve IBAction bağlantılarıdır.

IBOutlet *interface builder outlet* kelimelerinin kısaltılmıştır. IBOutlet storyboard ekranında geliştiricinin tasarladığı tüm arayüz elementlerinin swift dosyalarına olan bağlantılarını bir değişken üzerinden düzenler. Storyboard ekranı Xcode üzerinde açık vaziyetteken ve ilgili viewcontroller ekranı seçili durumdayken *assistant editor*'e geçiş yapıldığında; seçili olan viewcontroller ekranına bağlı olan swift dosyası ekranın sağ tarafında açılır. Storyboard ekranından sürükle bırak yöntemi ile kodların arasına götürdüğümüz arayüz elementine erişim için bir anahtar kelime girmemiz beklenir. Bu noktada bağlantı şeklinin “IBOutlet” olmasına dikkat edilir ve işlem tamamlandığında artık o arayüz elementine swift kodları ile erişme imkanına sahip olunur.

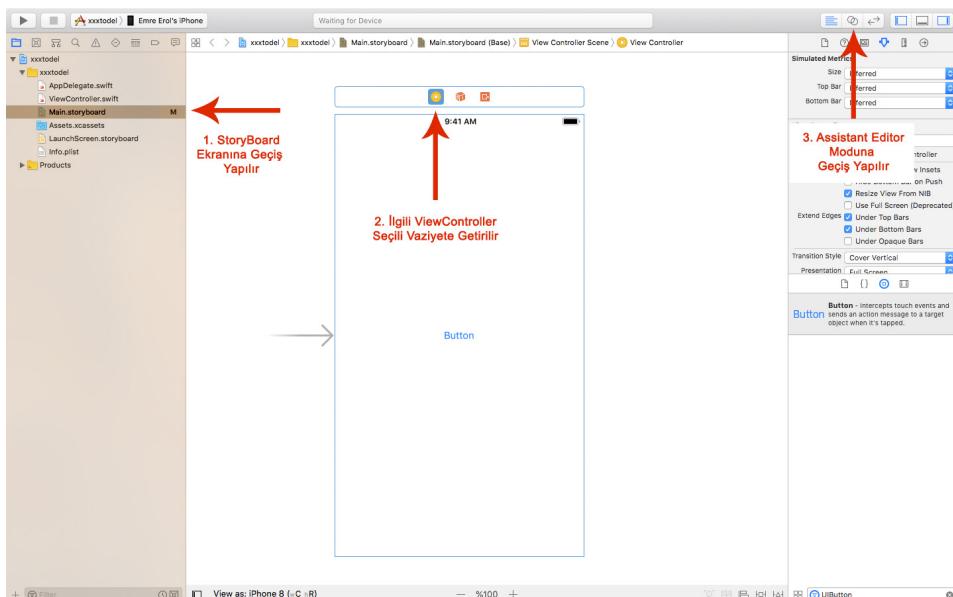
IBAction ise *interface builder action* kelimelerinin kısaltılmış ifadesidir. IBOutlet gibi oluşturulan IBAction da tek fark erişim için anahtar kelime girmemiz gereken alanda bağlantı şeklinin IBAction biçiminde olmasıdır. Bu sayede XCode bize bir fonksiyon oluşturmakta ve kullanıcı kontrolleri bu fonksiyon aracılığı ile yönetilmektedir.

Her arayüz elemanı IBAction bağlantısı kuramayabilir. Button elementi basılmak üzere tasarlanmış bir *view* olduğu için bu nesnenin IBAction bağlantısı mevcuttur. Ancak *label* elementi yalnızca

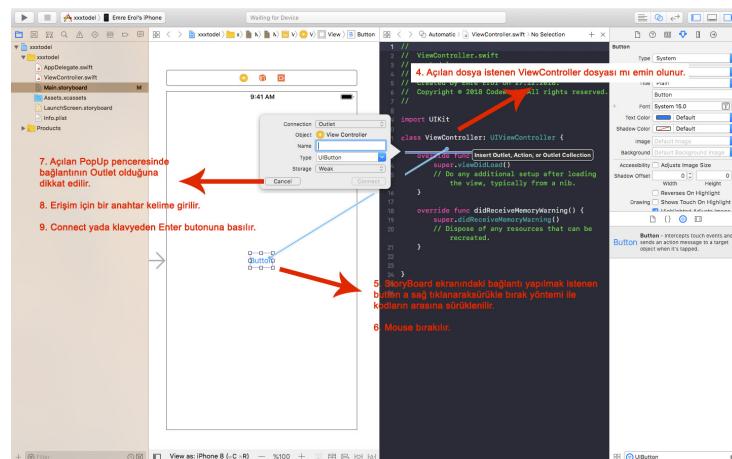
metin göstermek üzere tasarıldığından herhangi bir action bağlantısı kuramaz. Ancak outlet bağlantısı kurabilir. Bu outlet bağlantısı ile swift kodları ile bu label a erişim sağlanarak o arayüz elementinin özellikleri değiştirilebilir veya kendi sınıfından gelen metodları çalıştırılabilir.

Bir button arayüz elemanın IBOulet bağlantısı sırasıyla aşağıdaki gibi yapılabilir;

1. Storyboard ekranına geçiş yapılır.
2. İlgili ViewController seçili vaziyete getirilir.
3. Assistant Editor moduna geçilir.
4. Sayfanın sağ kısmında ViewController a bağlı olan swift dosyasının açıldığından emin olunur.
5. StoryBoard ekranında bağlanılmak istenen arayüz elemanın (Button) Mouse ile üzerindeyken sağ tıklanılarak sürükleme işlemi gerçekleştirilmeye başlanır. Mouse sağ kısmındaki kod kısmına götürülür.
6. ViewController sınıfının içerisindeyken Mouse bırakılır.
7. Açılan popup penceresinde bağlantı şeklinin Outlet olmasına dikkat edilir.
8. Bu arayüz elemanına erişim için bir anahtar kelime girilir.
9. Connect butonuna yada klavyeden enter tuşuna basılır.



Resim 49 : IBOutlet Adımları 1



Resim 50 : IBOOutlet Adımları 2

Eğer IBAction oluşturulmak istenirse bu durumda 7. Adımda açılan popup penceresinde bağlantı biçimi IBAction olarak değiştirilir ve anahtar kelime girilerek connect butonuna basılır ya da klavyeden enter tuşuna basılır.

Bu işlemlerin sonrasında @IBOutlet başlayan bir değişken ya da @IBAction kelimesi ile başlayan bir fonksiyon swift kodlarımız arasında XCode tarafından bizim için oluşturulur.

```

9 import UIKit
10
11 class ViewController: UIViewController {
12
13     @IBOutlet weak var btn: UIButton!
14
15     @IBAction func btnTouched(_ sender: Any) {
16
17     }
18
19
20
21
22     override func viewDidLoad() {
23         super.viewDidLoad()
24         // Do any additional setup after loading
25         // the view, typically from a nib.
26     }
27
28     override func didReceiveMemoryWarning() {
29         super.didReceiveMemoryWarning()
30         // Dispose of any resources that can be
31         // recreated.
32
33
34 }
35

```

Resim 51 : IBOOutlet ve IBAction Bağlantıları Oluşturulmuş Bir Görünüm



Kare Kod 9 : <https://github.com/ios-kitap/IBOutletIBAction>

IBOutlet ve IBAction kavramlarına ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

21. Viewler

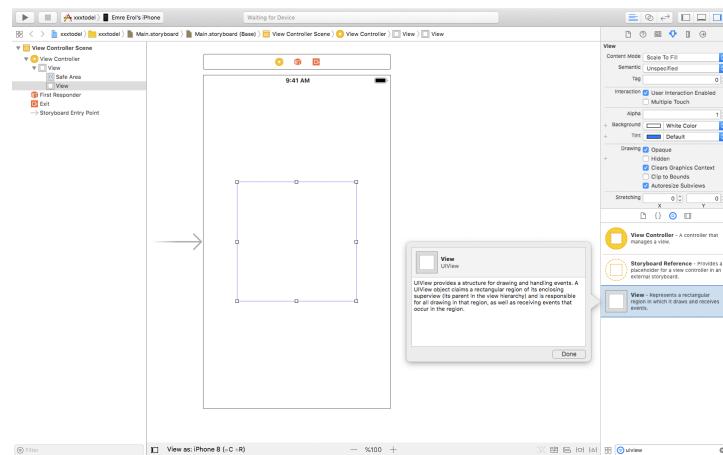
XCode ile bir geliştirme yaparken en çok rastlanılan ve kullanılan birimler viewlerdir. Çünkü viewler kullanıcı arayüzlerini ifade ederler ve tüm kullanıcı bileşenleri viewlerden türetilmiştir. Ham bir view nesnesi mevcuttur. Bu view nesnesine ek özellikler kazandırılarak *UIImageView*, *UITableView* gibi yeni arayüz elementleri üretilmiştir.

Bu üretilen yeni komponentlerin tamamı birbirinden farklı kullanım amaçlarına ve kullanım yöntemlerine sahiptir. Hepsinin ortak noktası ise kullanıcı etkileşimi için üretilmiş olmasıdır.

a. *UIView*

UIView temelde uygulama içerisindeki dörtgen bir alanı kontrol etmemize yarayan bir komponenttir. Ancak aslında diğer tüm komponentlerin üst yapısı diyeBILECEĞİMİZ bir nesnedir. Çünkü diğer tüm nesneler *view* nesnesinden kalıtım yolu ile üretilmiştir.

UIView başka nesneler üretmek için geliştirilen uygulamalarda geliştirici tarafından da kullanılabilir.

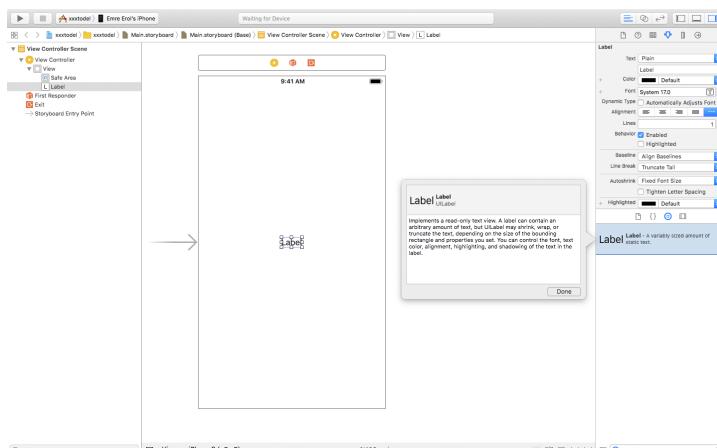


Resim 52 : UIView

b. Label

Label ya da nesne adıyla *UILabel*; bir veya birden fazla satırı sadece okunabilen, değiştirilemeyen yani salt okunur olan metinlerin gösterilmesi için kullanılır. İçerisinde barındırdığı hazır metodlar sayesinde, metne ilişkin bazı özellikler değiştirilebilir. Font stili, karakter büyülüğu vb. metinlere ilişkin özelliği label metodlarını kullanarak değiştirilip güncellenebilir.

Label yalnızca IBOutlet ile swift koduna bağlantı kurabilir.



Resim 53 : Label

IBOutlet bağlantısı kurmuş *label* elementinin bazı özelliklerini aşağıdaki tabloda gösterilmiştir;

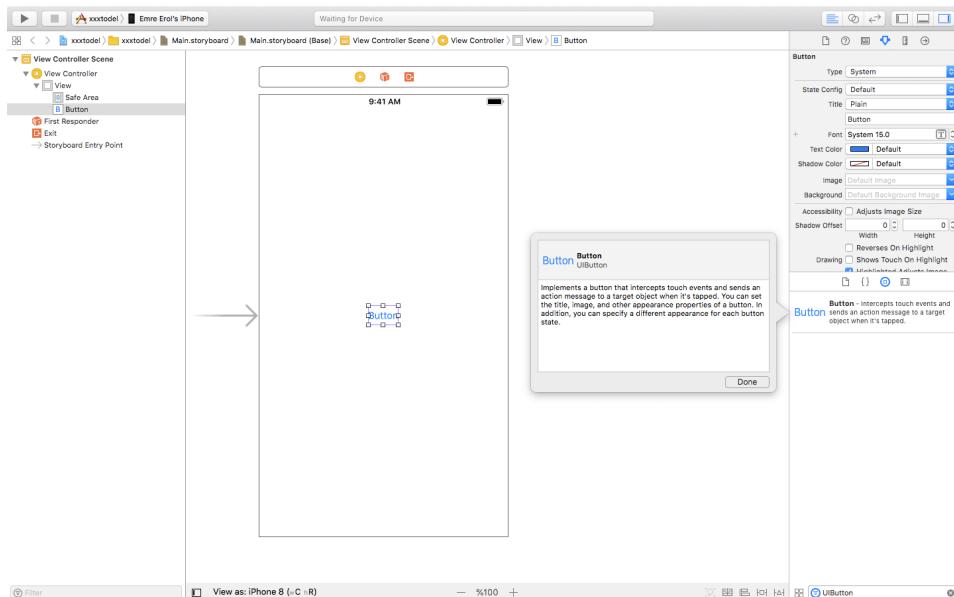
text	Label elementinin içinde göstereceği String türündeki metni belirlemek için kullanılır.
color	Label elementinin içinde gösterilen metnin rengini kontrol eder.
font	Label elementinin içinde gösterilen metnin fontunu kontrol eder.
alignment	Label elementinin içerisinde gösterilern metnin hizasını belirler.
lines	Label elementinin içindeki satır sayısını belirler. 0 olarak belirlendiğinde sınırsız sayıda satır olabileceği ifade edilir.

Tablo 3 : Label Elementinin Bazı Özellikleri

c. Button

Şüphesiz buton en fazla kullanılan viewlerden birisi olarak karşımıza çıkar. Buton kullanıcıların ekrana dokunması ile eylemlerin gerçekleştirilemesini ifade eden bir elementtir. Normalde geliştiriciler hiç buton kullanmadan da ekrana dokunulup dokunulmadığını programlama yöntemleriyle anlayıp yine işlemler yapabilirler. Ancak burada butonun başka bir özelliği ortaya çıkmaktadır. Çünkü buton kullanılan bir alanda kullanıcıda o butona basıldığı takdirde bazı eylemlerin gerçekleşeceği hakkında bilgilendirilir.

Ayrıca buton elementi geliştiriciye dokunma işlemlerinin takibi için kolaylıklar sağlar. Böylelikle geliştirici çok fazla kod yazmadan kullanıcının dokunma eylemlerini takip edip gerekli işlemleri yapabilir.



Resim 54 : Button

Button elementinin sağladığı hazır metod ve özellikler sayesinde geliştirici hızlı biçimde butonunu tasarlayıp davranış biçimini programlayabilir. Button bir ikon taşıyacak mı ya da içerisinde bir metin mi barındıracak ve eğer bir metin barındıracak sa bu metnin font stili, karakter boyutu ne olacak gibi sorulara cevap verebilir. Bunlar dışında tasarımsal bazı değişiklikleride hızlı biçimde gerçekleştirilebilir.

Button elementi hem IBOutlet hem IBAction bağlantısı kurabilir. IBAction bağlantısı ile oluşturulan fonksiyon ile kullanıcı butona dokunma eylemini gerçekleştirdiğinde yapılacak olan işlemler hazırlanır.

IBOutlet bağlantısı kurmuş Button elementinin bazı özellikleri aşağıdaki tabloda verilmiştir;

Title	Buttonun metnini kontrol eden özelliktir.
Image	Button üzerinde görsel kullanılmak istendiğinde kullanılabilen özelliktir.
Background	Buttonun arkaplan görseli kontrol edilir.

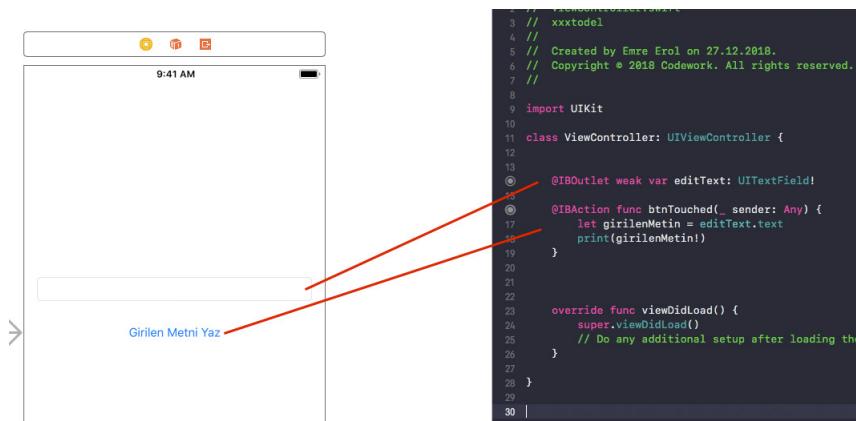
Tablo 4 : Button Özellikleri

d. TextField

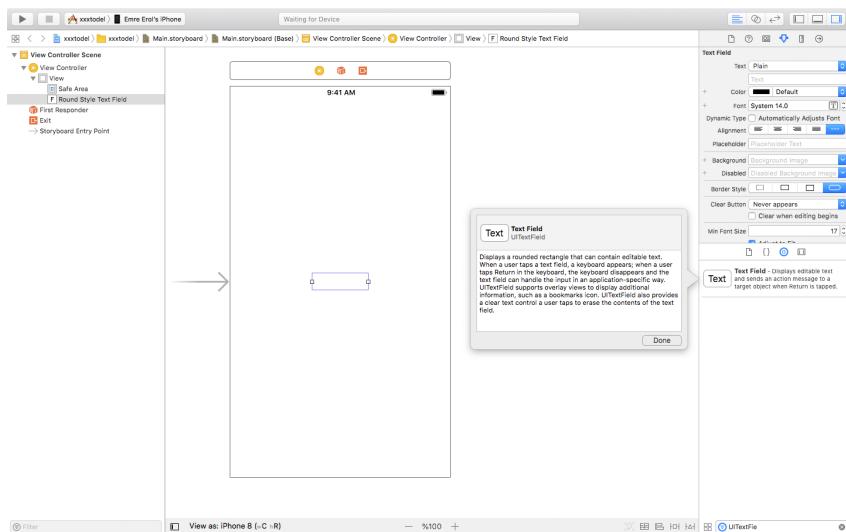
TextField ya da nesne adı ile *UITextField* ise kullanıcıların içerisinde istediği metni girebildikleri alan olarak ifade edilebilir. *Textfield*'lar da kullanıcıların dokunma eylemlerini takip ederler. Dokunma eylemi gerçekleştiğinde otomatik olarak kullanılan *textfield* tipine göre cihaz klavyesi görünür olur ve kullanıcı klavyeyi kullanarak *textfield* içerişine istediği metni girebilir.

Yine diğer tüm view elementlerinde olduğu gibi, geliştirici *textfield* yapısına ilişkin birçok özellik ve metodu kullanarak değişiklikler yapabilir ve fonksiyonelitesini artırıp azaltabilir.

IBOutlet bağlantısı kurulmuş bir *textfield* elementinin içerisindeki metni print fonksiyonu ile XCode üzerinde konsola yazdırın fonksiyon örneği aşağıda verilmiştir;



Resim 55 : TextField Kullanım Örneği



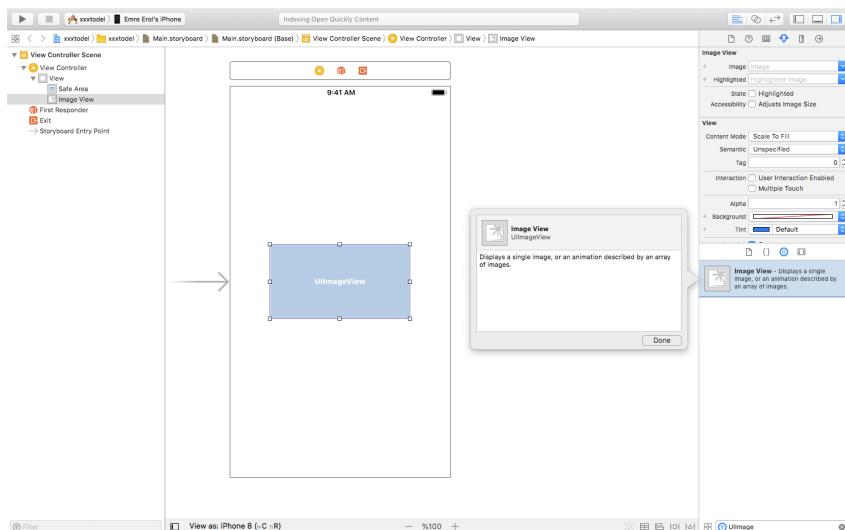
Resim 56 : TextField

e. ImageView

ImageView içerisinde görsel barındırmak ve göstermek üzere tasarlanmış bir view türüdür. İçerisinde barındırdığı imajın görünüm seçeneklerine ilişkin özellikleri bulunur. Barınacak ve görüntülenecek olan imaj internetten çekilerek ya da proje içerisinde alınabilir.

“Attributes” penceresine geçiş yapılarak *Image* bölümünden daha önceden projenin asset kataloğuna eklenmiş olan görseller seçilerek *ImageView* içinde gösterilebilir.

ImageView yalnızca IBOulet bağlantı kurabilen bir arayüz elementidir.

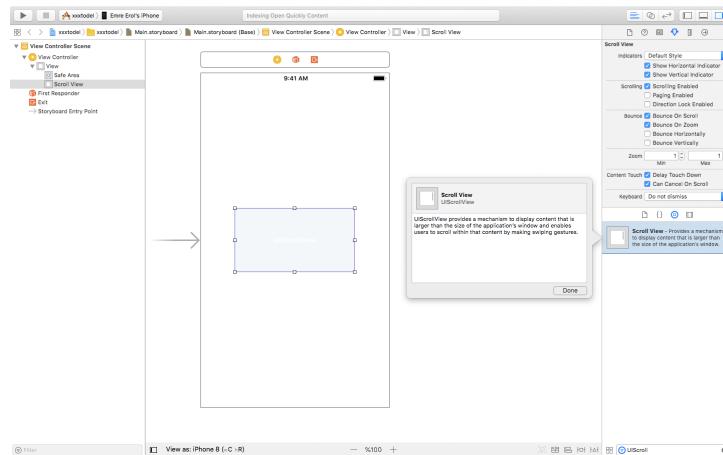


Resim 57 : ImageView

f. ScrollView

Bazı durumlarda hazırlanan tasarımlar ekran dışına taşıacak büyüklüklerde olabilir. Ekran dışına taşan tasarım elementlerinin ekranda görüntülenebilmesi için o kısma kaydırma özelliği eklenmelidir. Bu gibi durumlarda *scrollview* bir taşıyıcı olarak kullanılarak tüm tasarım *scrollview* içerisinde yapılır. Bu sayede artık kullanıcı ekran dışına taşan görüntüyede erişim sağlayabilir.

Scrollview kaydırma işlevi dışında yakınlaştırma ve uzaklaştırma işlemide yapabilme kabiliyetine sahiptir.

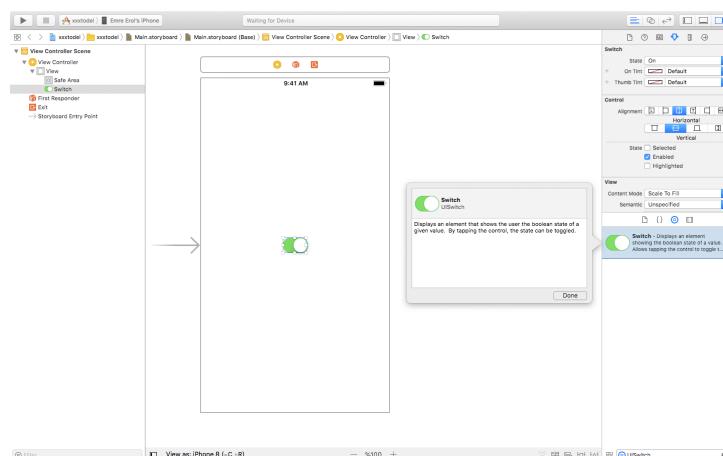


Resim 58 : ScrollView

g. Switch

Web platformlarında sıkça kullanılan checkbox ‘ın mobil platforma uyarlanmış ve mobil platform için kullanışlı olduğu varsayılan biçimi switch elementidir.

Nesne adı ile *UISwitch* binary bir yapıya sahiptir. Yalnızca açık ve kapalı konumlarında bulunabilir.

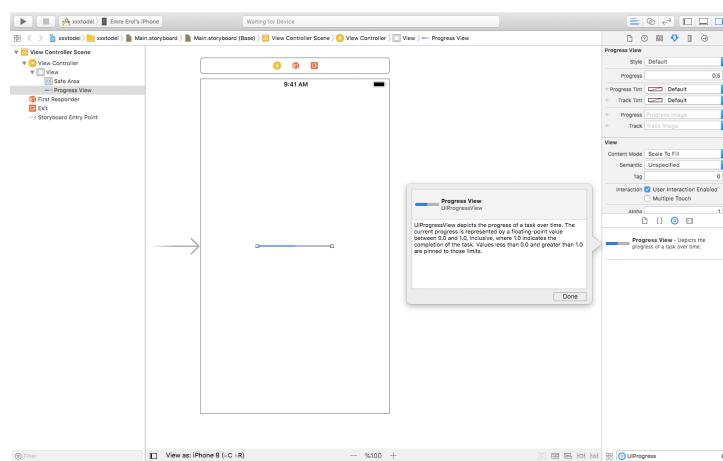


Resim 59 : Switch

b. Progress View

Yapılan bir işlemin zaman içerisindeki süreci ile ilgili kullanıcıyı bilgilendirmek için kullanılan view türüdür. 0 ile 100 arasında değer alarak kullanılmaktadır.

ProgressView yalnızca IBOutlet bağlantı kurabilir.

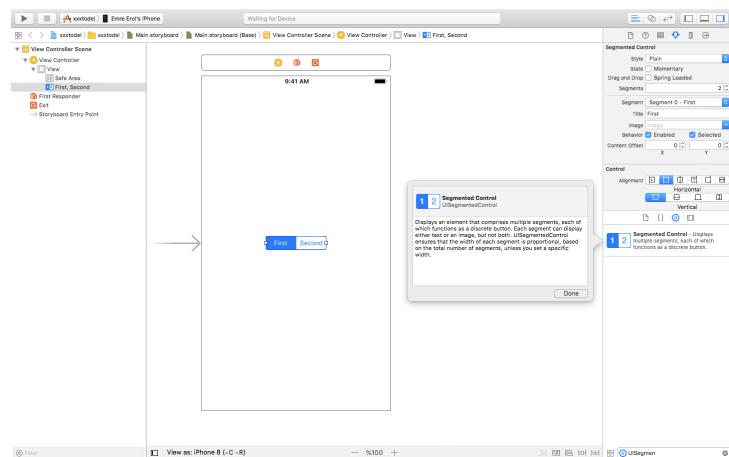


Resim 60 : *ProgressView*

i. Segmented Control

Yatay biçimde sıralanmış bölümlerden oluşan bir buton takımı olarak nitelendirebileceğimiz *segmented control* içerisinde sınırsız sayıda bölüm bulundurabilir. Bu bölümler içerisinde metin ya da imaj içerikler kullanılabılır.

Segmented kontrol kullanıcı dokunma eylemlerini takip eden bir yapıya sahiptir ve kullanıcı bu bölümlerden herhangi birine dokunduğunda seçili olan bölüm dokunduğu bölüm olarak değişmektedir. Ardından geliştiricinin o bölüm için hazırladığı işlemler gerçekleştirilir.



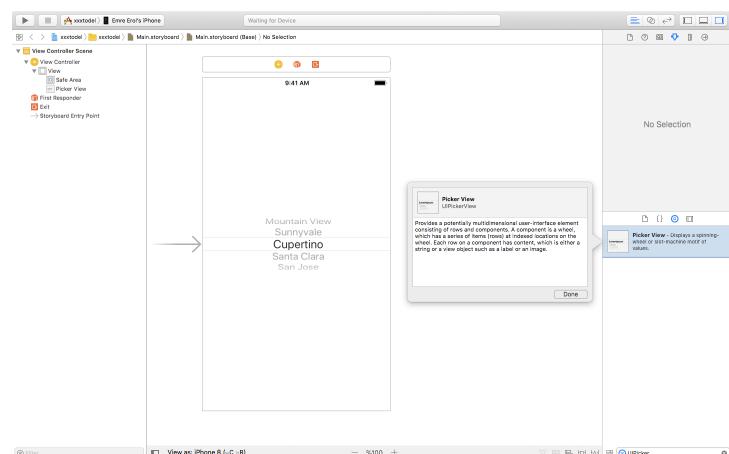
Resim 61 : Segmented Control

j. Picker View

Web platformlarında combobox ya da select diye tabir edilen yapıların ios platformuna özgü biçimidir. İçerisine bir dizi aracı ile verilen girdileri kullanıcıya seçim yapması için bir panel içerisinde gösterir.

Genellikle *textview*'ler ile birlikte kullanılır ve kullanıcının yaptığı seçim ilgili *textview*'e yazılır.

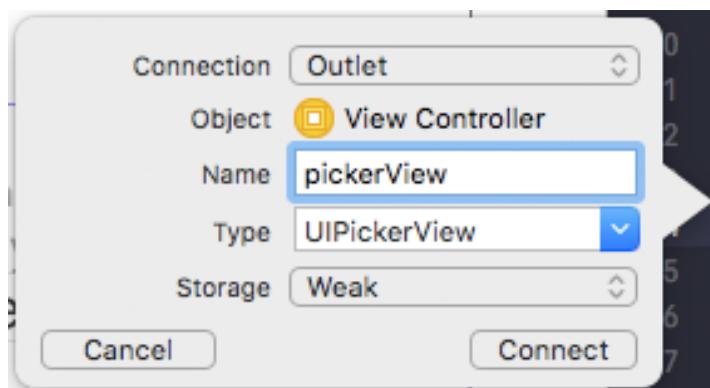
Birden fazla kolonlu bir yapıya sahip olabilir. Bu sayede tek seferde birden fazla bilgi kullanıcından alınabilir.



Resim 62 : PickerView

Kullanımı diğer view' ler kadar olmayan *pickerview* örneği aşağıda adım adım anlatılmaya çalışılmıştır.

Öncelikle storyboard ekranına bir *pickerview* elementi eklenir. Ardından “assistant editör” yardımı ile bu *pickerview* için bir IBOulet bağlantısı oluşturulur.



Resim 63 : PickerView IBOulet Bağlantısı

Sonrasında *pickerview* elementini eklediğimiz viewcontroller ekranına bağlı olan swift dosyası üzerinde gerekli kodları yazılmasına başlanabilir. Bunun için ilgili swift dosyasına geçiş yapılır.

Viewcontroller ekranına bağlı olan swift dosyasının sınıf tanımlamasının yapıldığı bölümde *pickerview* elementinin *UIPickerViewDelegate* ve *UIPickerViewDatasource* protokollerinden kalıtım alınır.

```

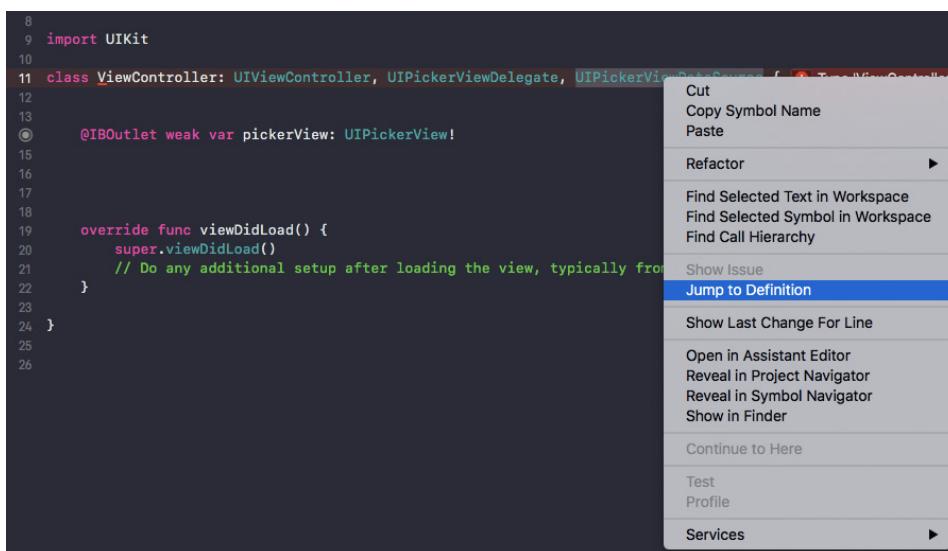
6
7 import UIKit
8
9 class ViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {
10
11     @IBOutlet weak var pickerView: UIPickerView!
12
13
14
15
16
17
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         // Do any additional setup after loading the view, typically from a nib.
22     }
23
24 }
25
26

```

Resim 64 : UIPickerViewDelegate ve UIPickerViewDatasource Kalıtımı

Bu aşamada viewcontroller sınıf tanımlamasının bulunduğu satırda bir hata mesajı görüntülenecektir. Bu hata mesajı kalıtım alınan protokol doğrultusunda bazı fonksiyonların sınıf içeresine dahil edilmesine ilişkin bir mesaj içerir.

Bu fonksiyonları sınıf içeresine dahil etmek için klavyenizde viewcontroller sınıf tanımlaması yapılan satır üzerinde *UIPickerViewDatasource* üzerine mouse ile sağ tıklanır ve açılan popup menüsünden “Jump To Definition” butonuna tıklanarak bu protokollerin tanımlandığı *UIPickerView* sınıf dosyasına geçiş yapılır.



Resim 65 : PickerView Sınıf Dosyasına Geçiş

Açılan kod dosyası *UIPickerView* elementinin sınıf dosyasıdır. Bu dosya içerisinde dahil ettiğimiz *UIPickerViewDatasource* ve *UIPickerViewDelegate* protokollerinin tanımlamalarında bulunur. Bu protokollerin içerisinde tanımlı olan bazı fonksiyonlar buradan alınıp viewcontroller ekranına bağlı olan swift dosyası içeresine dahil edilerek kullanılmalıdır. Kullanılacak olan fonksiyonlar *UIPickerViewDatasource* protokolü altında *numberOfComponents* ve *numberOfRowsInComponent*; *UIPickerViewDelegate* protokolü altında da *titleForRow* fonksiyonudur. Bu fonksiyonlar func anahtar kelimesinden başlanarak seçilerek kopyalanıp view controller ekranına bağlı olan swift dosyasınalaştırılarak dahil edilmelidir.

```

50 public protocol UIPickerViewDataSource : NSObjectProtocol {
51
52
53     // returns the number of 'columns' to display.
54     @available(iOS 2.0, *)
55     public func numberOfComponents(in pickerView: UIPickerView) -> Int
56
57
58     // returns the # of rows in each component..
59     @available(iOS 2.0, *)
60     public func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int
61 }

```

Resim 66 : UIPickerViewDataSource Protokolü ve Dahil Edilecek Fonksiyonlar

```

63 public protocol UIPickerViewDelegate : NSObjectProtocol {
64
65
66     // returns width of column and height of row for each component.
67     @available(iOS 2.0, *)
68     optional public func pickerView(_ pickerView: UIPickerView, widthForComponent component: Int) -> CGFloat
69
70     @available(iOS 2.0, *)
71     optional public func pickerView(_ pickerView: UIPickerView, rowHeightForComponent component: Int) -> CGFloat
72
73
74     // these methods return either a plain NSString, a NSAttributedString, or a view (e.g UILabel) to display the row for the
75     // for the view versions, we cache any hidden and thus unused views and pass them back for reuse.
76     // If you return back a different object, the old one will be released. the view will be centered in the row rect
77     @available(iOS 2.0, *)
78     optional public func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String?
79
80     @available(iOS 6.0, *)
81     optional public func pickerView(_ pickerView: UIPickerView, attributedTitleForRow row: Int, forComponent component: Int) ->
82         NSAttributedString? // attributed title is favored if both methods are implemented
83
84     @available(iOS 2.0, *)
85     optional public func pickerView(_ pickerView: UIPickerView, viewForRow row: Int, forComponent component: Int, reusing view:
86         UIView
87
88     @available(iOS 2.0, *)
89     optional public func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int)
90 }

```

Resim 67 : UIPickerViewDelegate Protokolü ve Dahil Edilecek Fonksiyonlar

Pickerview içerisinde yer alacak ve görüntülenecek olan verilerde bir dizi değişken tanımlaması ile tutulabilir.

```

10 class ViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {
11
12
13
14     let renkler = ["Kırmızı", "Mavi", "Yeşil"]
15
16     @IBOutlet weak var pickerView: UIPickerView!
17
18     func numberOfComponents(in pickerView: UIPickerView) -> Int{
19
20     }
21
22     func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int{
23
24     }
25
26     func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String?{
27
28     }

```

Resim 68 : ViewController Sınıfına Dahil Edilmiş Protokol Fonksiyonları
ve PickerView dizi Değişkeni

Dahil edilen *numberOfComponents* fonksiyonu pickerView içerisinde görüntülenecek olan kolon sayısının belirlendiği fonksiyondur. *numberOfRowsInComponent* kaç satır bulunacığının belirlendiği fonksiyondur. *TitleForRow* fonksiyonu ile de satır içerisinde görüntülenmek istenen metin belirlenebilir. Bu belirlemeler için fonksiyonların hepsinin geriye veri döndüren fonksiyonlar olduğuna dikkat edilmelidir. Bahsedilen bilgiler fonksiyonlarda return anahtar kelimesi kullanılarak verilerin geri döndürülmesi ile belirlenebilir.

numberOfComponents fonksiyonunun kod bloğuna return anahtar kelimesi ile tam sayı (Int) tipinde 1 yazılarak tek kolonlu bir pickerview olduğu belirlenir.

numberOfRowsInComponent fonksiyonunun kod bloğunada yine return anatar kelimesi ile pickerview içerisinde görüntülenmesi için hazırlanan dizi değişkenin eleman sayısı belirlenir.

Son olarak *titleForRow* fonksiyonunda dizi değişkenin içindeki elemlara erişilerek return anahtar kelimesi ile döndürülür. Burada ihtiyaç olan dizi indeks numarası fonksiyonun parametreleri arasında *row* anahtar kelimesi ile belirlenmiş olan değişkendir.

```

17     func numberOfComponents(in pickerView: UIPickerView) -> Int{
18         return 1
19     }
20
21     func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int{
22         return renkler.count
23     }
24
25     func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String?{
26         return renkler[row]
27     }
28
29

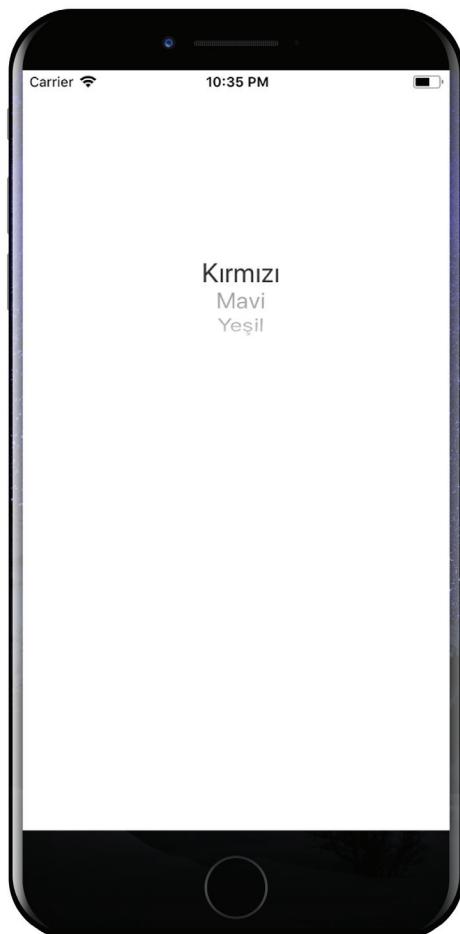
```

Resim 69 : Dahil Edilmiş Fonksiyonların Kullanımları

Son olarak viewcontroller sınıfı içerisinde *override* edilerek ilk yaratıldığı anda bulunan *viewDidLoad* fonksiyonu kod bloğu içerisinde IBOutlet bağlantısı oluşturduğumuz pickerview elementinin *delegate* ve *datasource* kaynakları belirlenir. Burada *self* kelimesi kullanılarak pickerview içerisinde gösterilecek olan veriler doldurulurken, *pickerview* elementinin içerisinde bulunduğu sınıfın kaynaklarının kullanılacağıının belirlenir.

```
30
31     override func viewDidLoad() {
32         super.viewDidLoad()
33         // Do any additional setup after loading the view.
34         pickerView.delegate = self
35         pickerView.dataSource = self
36     }
37
```

Resim 70 : viewDidLoad Fonksiyonunda Picker View Elementinin Kaynaklarının lenmesi

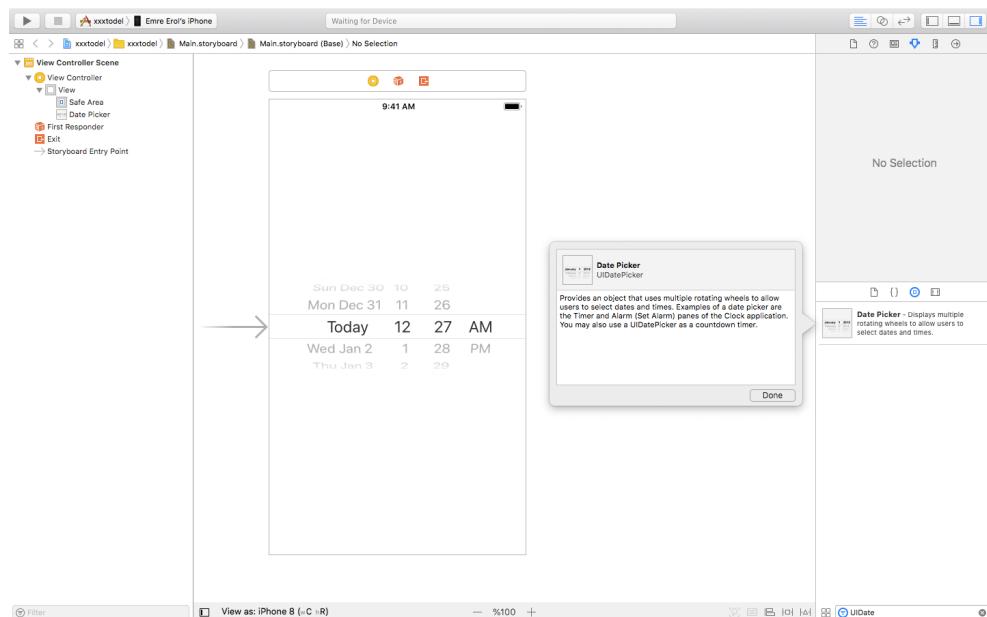


Resim 71 : Picker View Uygulama Üstünde Çalışırken

k. Date Picker

PickerView nesnesinden üretilmemiş olsa da kullanım biçimini pickerview e çok benzeyen datepicker isminden de anlaşılacağı üzere kullanıcının tarih seçimi yapması için geliştirilmiş bir elementtir.

Pickerview' de olduğu gibi yine genellikle textview' ler ile kullanılır ve seçim textview' e ya da görüntülenmek üzere bir label elementine yazdırılabilir.

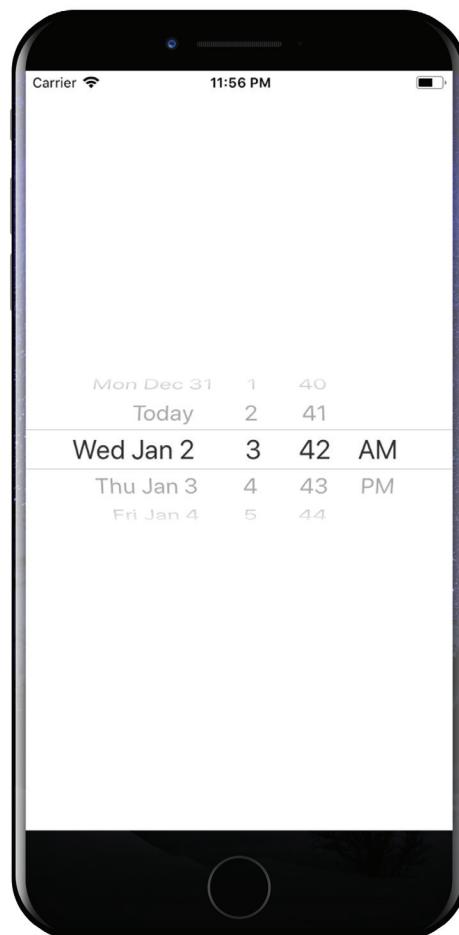


Resim 72 : DatePicker

Storyboard üzerine eklenen datepicker IBOutlet bağlantısı ile swift koduna dahil edilir. Ardından viewcontroller sınıfı içerisinde yer alan *viewDidLoad* fonksiyonunun kod bloğu içerisinde datepicker için zaman işlemleri belirlenir. Uygulama çalıştırıldığında datepicker tarih seçenekleri ile dolu biçimde kullanılabilir vaziyettedir.

```
8 import UIKit
9
10 class ViewController: UIViewController {
11
12
13     @IBOutlet weak var datepicker: UIDatePicker!
14
15
16     override func viewDidLoad() {
17         super.viewDidLoad()
18         // Do any additional setup after loading the view.
19
20         datepicker.date = Date.init() as Date
21
22     }
23
24 }
25 }
```

Resim 73 : DatePicker Elementinin Çalıştırılmasına İlişkin Swift Kodu

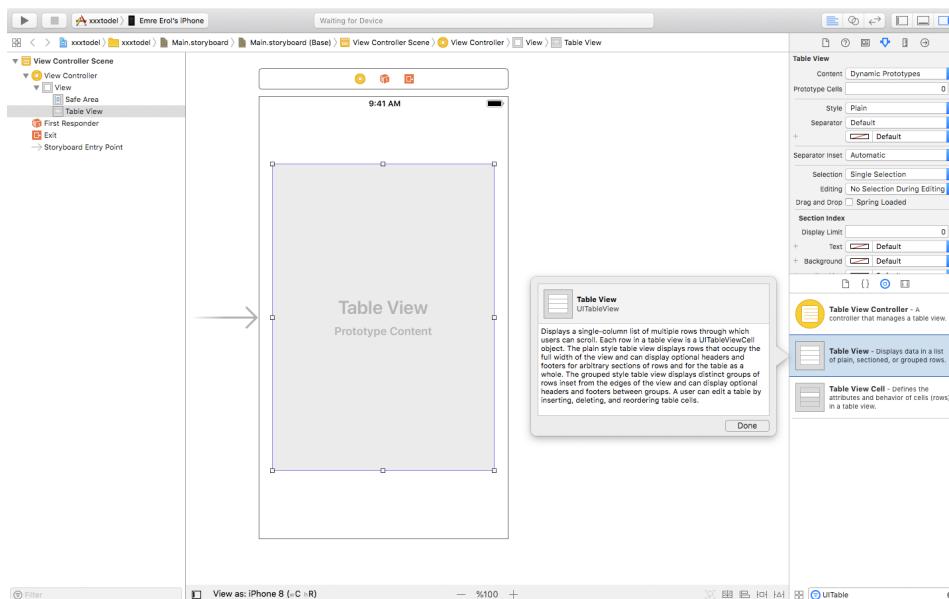


Resim 74 : DatePicker Elementi Uygulama Üstünde Çalışırken

1. TableView

En çok kullanılan elementlerden bir başkasıda tableview' dir. Tableview belli bir veri kaynağındaki tüm veriyi tek bir kolonda hücreler halinde göstermek için kullanılır. Hücrelerin hepsinin tasarımları birbirinden farklı olabileceği gibi, tek bir tasarım veri kaynağının içindeki veri sayısına ne kadar çok olursa olsun kullanılabilir.

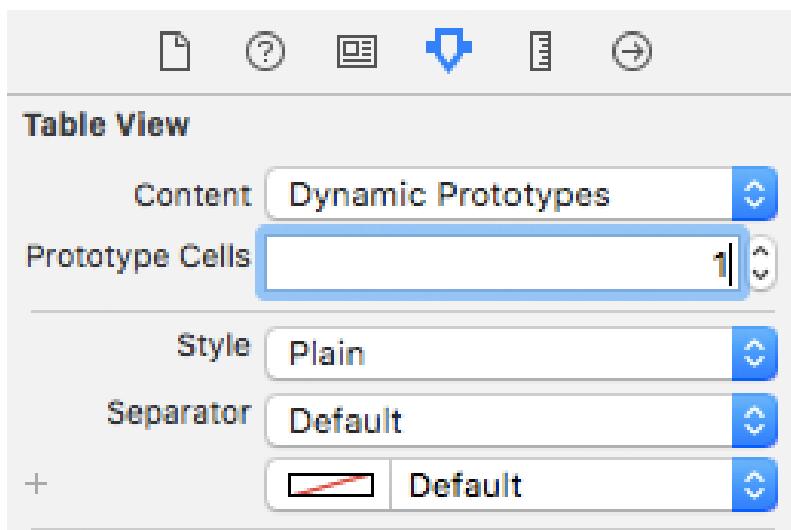
Tableview scrollview elementinden kalıtım yoluyla üretilmiştir. Bu yüzden kaydırma gibi özellikleri varsayılan olarak bulunmaktadır.



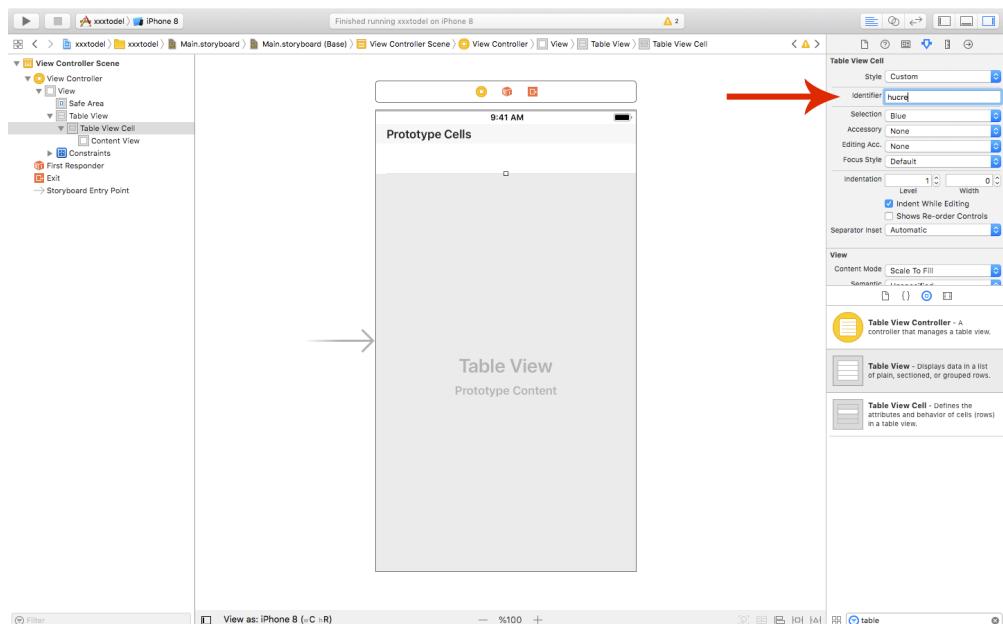
Resim 75 : TableView

TableView in kullanımına ilişkin yönergeler adım adım aşağıda ifade edilmiştir.

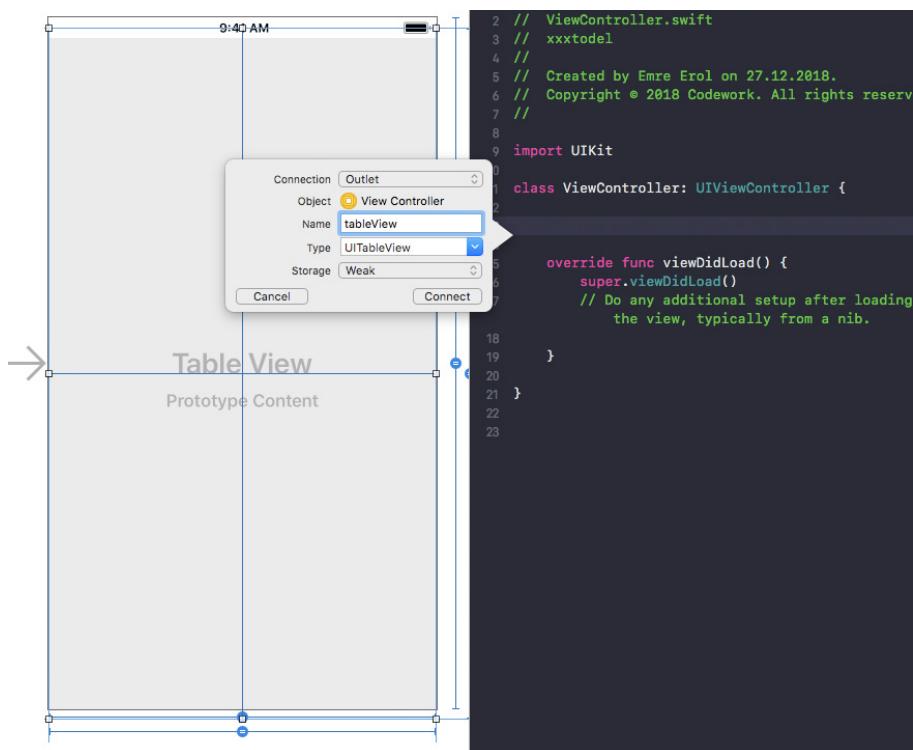
Storyboard üzerinde viewcontroller ekranına bir tableview eklenir. Sağ bölümde yer alan attributes panelinden bu tableview için bir prototip hücre eklenmelidir. Bunun için *Prototype Cells* özelliğinin 1 yapılması yeterlidir. Prototip hücre eklendikten sonra seçili hale getirilir ve yine attributes panelinden kimliklendirilerek swift kodları içerisinde erişimi sağlanmalıdır. Hücre seçildikten sonra attributes panelinde *identifier* özelliğine bir anahtar kelime girilerek kimliklendirme yapılabilir. Eklenen tableview IBOutlet ile viewcontroller sınıfına bağlanır.



Resim 76 : TableView Üzerinde Prototip Hücre Oluşturulması



Resim 77 : TableView Elementine Eklenmiş Prototip Hücrenin Kimliklendirilmesi



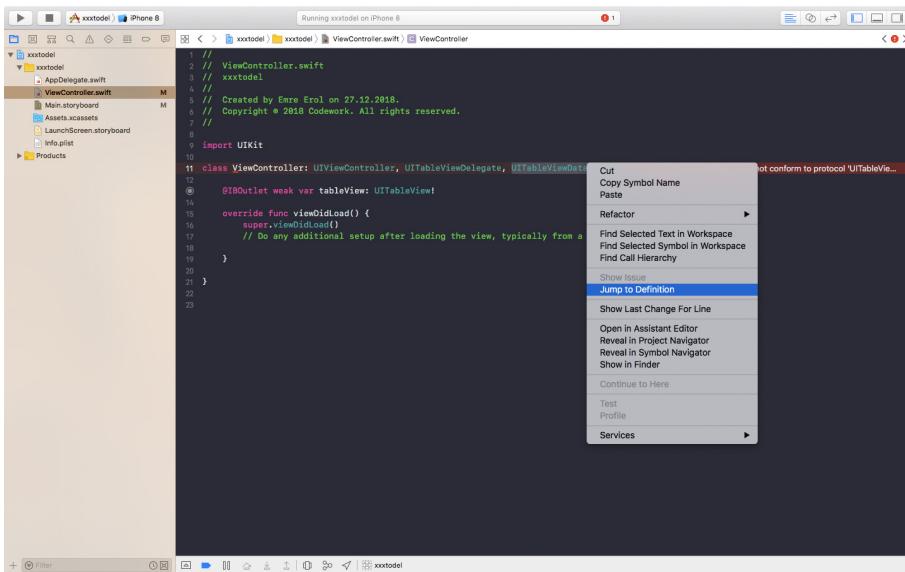
Resim 78 : TableView Elementinin IBOulet Bağlantısı

TableView üzerinde görüntülenecek olan veriler bir dizi içerisinde tutulmalıdır. Viewcontroller sınıfı içerisinde tanımlanan bu dizi elemanları sonrasında tableview satırlarında görüntülenecektir.

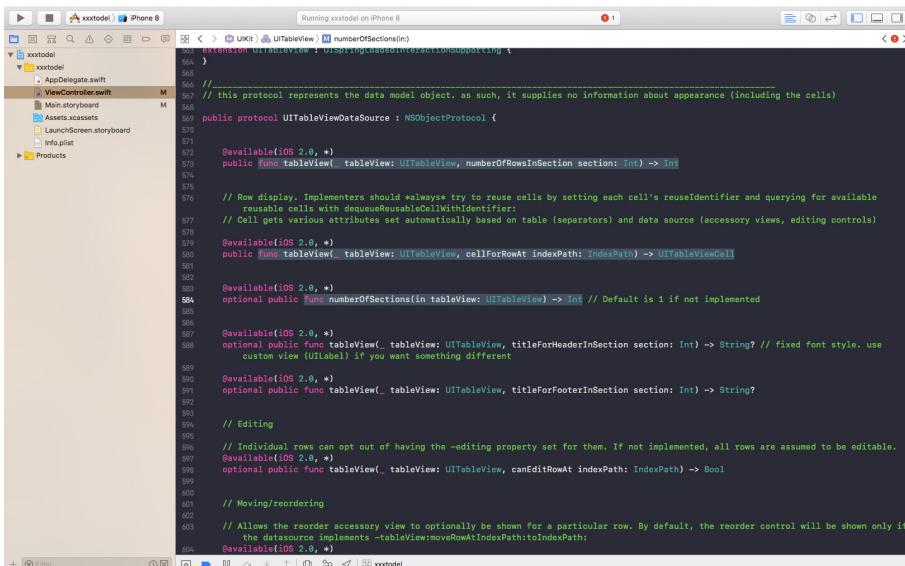
Viewcontroller'ın sınıf tanımlamasının yapıldığı satırda *UITableViewDelegate* ve *UITableViewDataSource* protokollerinden kalıtım alınır. Bu işlem yapıldığında *UITableViewDelegate* ve *UITableViewDataSource* protokollerinden dahil edilmesi gereken fonksiyonlar olduğuna ilişkin hata mesajı alınır.

Daha önce pickerview' de bahsedilen durum gibi burada da *UITableViewDataSource* üzerinde mouse ile sağ tıklanarak açılan popup menü üzerinden “Jump To Definition” butonuna basılarak Tableview sınıf dosyasına geçilebilir. Bu sınıf dosyası içerisinde yer alan protokollerde bulunan bazı fonksiyonların tableview için viewcontroller swift dosyasına dahil edilmesi gerekmektedir.

Fonksiyonlar “func” anahtar kelimesinden itibaren seçilerek viewcontroller ekranına bağlı olan swift dosyası içerisine, sınıf kodbloğu bölümüne dahil edilmelidir.



Resim 79 : TableView Sınıf Dosyasına Geçiş



Resim 80 : UITableViewDataSource Protokolü İçerisinden ViewControllera Bağlı Olan Swift Dosyasına Dahil Edilecek Fonksiyonlar

Dahil edilen fonksiyonların kullanımı pickerview elementindeki kullanımına çok benzerdir. *numberOfSections* kolon sayısını belirlemek için; *numberOfRowsInSection* tableview içerisinde görüntülenecek olan eleman sayısını belirtmek için kullanılır. Pickerview' den farklı olarak *cellForRowAt* fonksiyonu bulunur. Bu fonksiyonun dönüş tipine bakıldığında *UITableViewCell* olduğu göze çarpar. Bunun sebebi her hücrenin eğer istenirse farklı biçimde tasarılanıp tableview içerisinde gösterilebilmesi için esneklik sağlanmasıdır. Şu anda bahsedilen örnek için farklı bir tasarım durumu söz konusu değildir ve sadece dizi içerisindeki veriler metin olarak tableview elementi içerisinde sırasıyla gösterilebilir.

numberOfSections fonksiyonu kod bloğu içerisinde return anahtar kelimesi ile tam sayı (Int) cinsinden 1 rakamı ile tableview in tek kolonlu bir yapıda olacağı belirtilir.

numberOfRowsInSection fonksiyonu kod bloğunda return anahtar kelimesi ile tableview içerisinde gösterilmek istenen dizi değişkenin eleman sayısı döndürülür.

cellForRowAt fonksiyonunda ise geri döndürülecek olan verinin tipi *UITableViewCell* olduğundan bu sınıfın bir nesne türetilir. Bu nesnenin tasarımı storyboard ekranında tableview elementi için eklediğimiz prototip hücre üzerinden alınmaktadır. Bu yüzden bu nokta da prototip hücrenin identifier' ı üzerinden tekrar kullanılabilir bir hücre tasarımlına erişilir ve değişkene atanır. Sonrasında bu nesne içerisinde bir textview "cast" edilerek textview in text özelliği dizi içerisinde alınan veriler doğrultusunda değiştirilir. Dizi içerisinde verilerin çekilebilmesi için indeks numarası fonksiyon parametrelerinden *indexPath.row* aracılığı ile alınabilir. *IndexPath.row* burada henüz eklenecek olan tableview' in tableview üzerindeki satır numarasını tutan değişkendir. Böylelikle son olarak return anahtar kelimesi ile hücre fonksiyonda geri döndürülür.

```
func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}

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

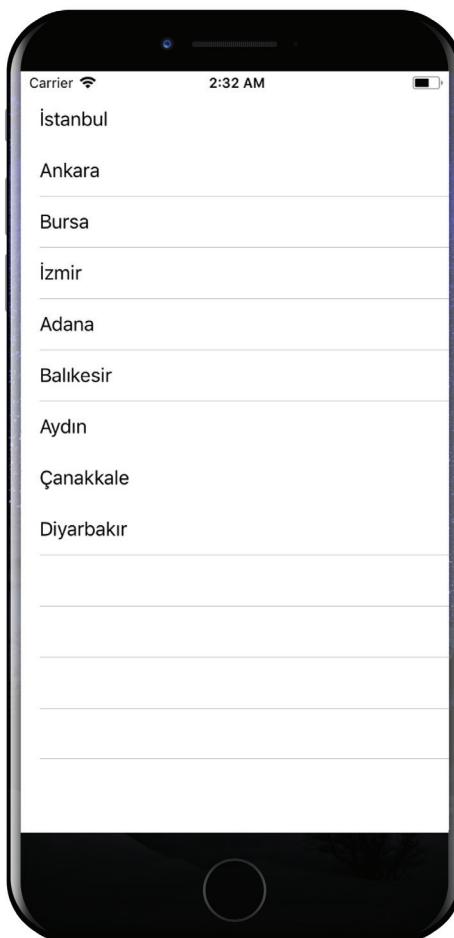
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell{
    let hucre = tableView.dequeueReusableCell(withIdentifier: "cell") as! UITableViewCell
    hucre.textLabel?.text = sehirler[indexPath.row]
    return hucre
}
```

Resim 81 : ViewController Sınıf Dosyasına Dahil Edilmiş TableView Protokol Fonksiyonları

Son olarak viewcontroller sınıfı içerisinde override edilerek ilk yaratıldığı anda bulunan *viewDidLoad* fonksiyonu kod bloğu içerisinde IBOutlet bağlantısı oluşturduğumuz tableview elementinin *delegate* ve *datasource* kaynakları belirlenir. Bu aşamadan itibaren uygulama çalıştırıldığında tableview çalışır vaziyette olacaktır.

```
31     override func viewDidLoad() {  
32         super.viewDidLoad()  
33         // Do any additional setup after loading the view,  
34         tableView.delegate = self  
35         tableView.dataSource = self  
36     }
```

Resim 82 : *ViewDidLoad* Fonksiyonunda TableView Elementinin Kaynaklarının Belirlenmesi



Resim 83 : TableView Elementi Uygulama Üzerinde Çalışırken



Kare Kod 10 : <https://github.com/ios-kitap/allBasicViews>

View lere ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

22. Extensionlar

Bir diğer adı ile uzantılar, swift programlama dilinin esnek yapısını oluşturan en önemli kavramlardan biridir. Extension' lar küçük bir özellik olmasına karşın büyük bir esneklik yaratmaktadır. Extension' lar ile mevcut bir sınıfın, struct' in, enumeration' in veya protokolün içeriği değiştirilmeden yeni bir fonksiyon eklenebilir. Sonrasında bu fonksiyonlara sınıfın içerisinde yazılmış gibi düşünülerek erişim sağlanabilir.

Bir uzantı tanımlamak için extension anahtar kelimesi kullanılır. Ardından uzantı tanımlaması yapılacak olan tip girilmelidir. Sonrasında küme parantezlerinin kullanımıyla kod bloğu belirlenir ve istenen tüm işlemler fonksiyon biçiminde bu kod bloklarının arasına yazılır.

```
extension SınıfAdı {  
    // Yeni fonksiyonlar buraya yazılır.  
}
```

Yazım Kuralı 15 : Extension Yapısı

Aşağıda bir textfield elementinin boş olup olmadığını kontrol eden extension tanımlamasının ekran görüntüsü bulunmaktadır.

```
180  extension UITextField {
181      func isEmpty () ->Bool {
182          if self.text == "" {
183              return true
184          }else{
185              return false
186          }
187      }
188  }
```

Resim 84 : TextField Elementinin Boş Olup Olmadığını Kontrol Eden Bir Extension



Kare Kod 11 : <https://github.com/ios-kitap/Extensions>

Extension' lara ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

23. View Animasyonları

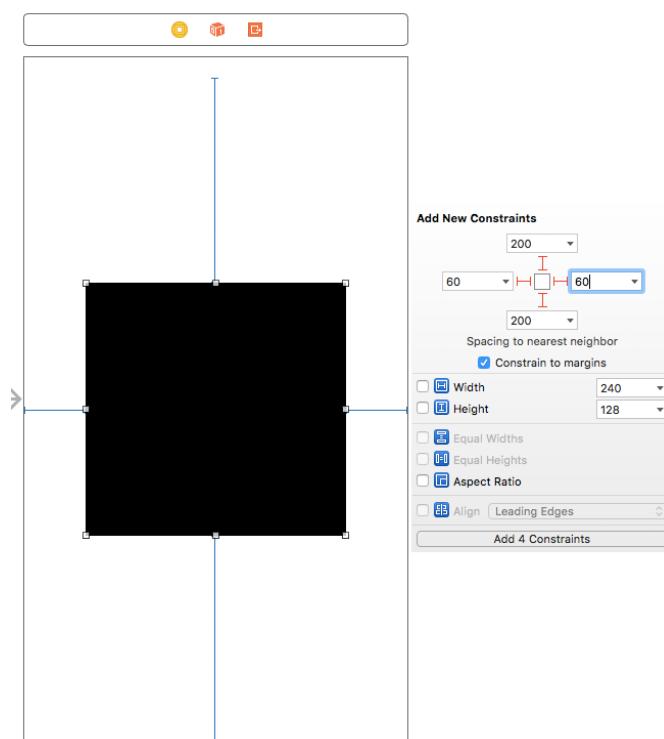
Bir mobil uygulama geliştirirken küçük animasyonlar ile uygulama arayüzünün daha tatmin edici bir yapıya kavuşturulması kuşkusuz uygulama kullanıcıları için daha iyi kullanım tecrübeleri sağlamaktadır. IOS üzerinde tüm arayüz elementleri view nesnesinden türetildiğinden, basit anlamda animasyon konusu da view bağlamında incelenmelidir. Animasyon üretmek için kullanılacak olan metod da

UIView sınıfı altında bulunan *animate* fonksiyonudur.

Herhangi *view* arayüz elementine ilişkin bir animasyon hedeflendiğinde yapılması gerekenler basittir. Bu nokta da zor olan kısmı animasyonun meydana gelmesi için hangi unsurların hangi sıra ile değiştirileceğinin iyi planlanmasıdır. Eğer bunlar iyi biçimde planlanabilirse geriye kalan işlemler ios tarafından halledilmektedir.

a. Belirlenen Sürede Değişen Sabitler

Bir arayüz tasarımları hazırlanırken arayüz elementlerinin konumlarını ve boyutlarını verilen sabitler belirlemektedir. Bu sabit değerler, eğer belli bir zaman içerisinde değiştirilirse ve bu değişim canlı olarak arayüze aktarılabilirse bu durumda arayüz elementlerinin konumu ve boyutlarına ilişkin bir animasyon elde edilebilir.



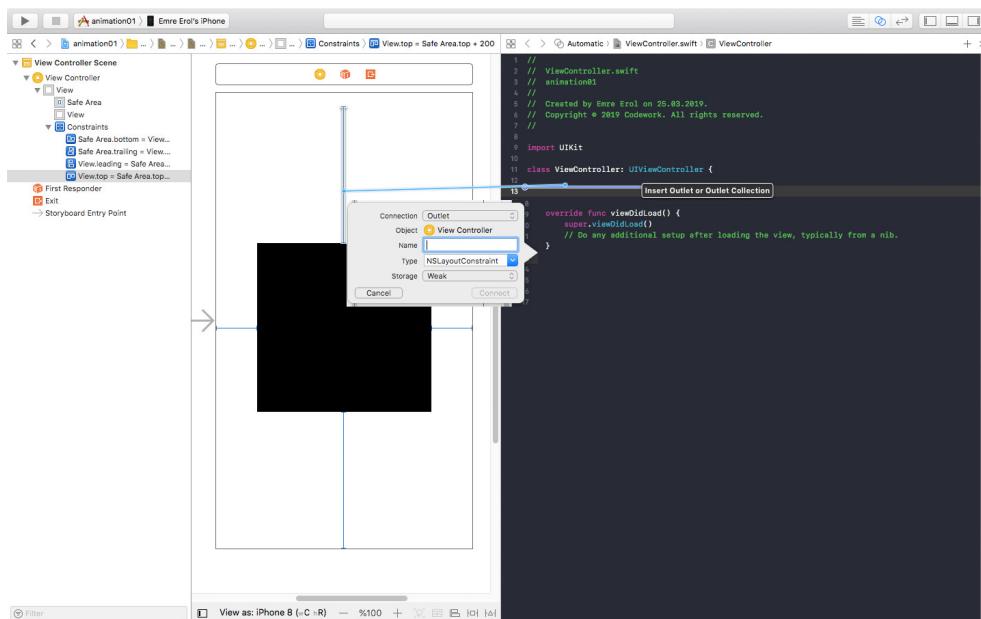
Resim 85 : View Nesnesi ve Sabitler

Storyboard üzerinde mavi çizgiler ile gösterilen sabitler, ilgili viewcontroller için oluşturulmuş olan swift dosyasına IBOulet bağlantısı ile bir değişken olarak bağlanabilmektedirler. Sonrasında bu değişkenlerin taşıdığı değerler değiştirilerek animasyonlar gerçekleştirilebilmektedir.

Aşağıda ekranın tam ortasına yerleştirilmiş bir *view* elementinin, 4 saniye içerisinde ekranı kaplaması anime edilerek gösterimi adım adım anlatılmaya çalışılmıştır.

Storyboard bölümüne geçiş yapıldıktan sonra mevcut viewcontroller ekranı üzerine bir *view* elementi eklenir ve sabitler verilir.

Asistant editörü'ne geçiş yapılarak ekranın sol tarafında storyboard ve sağ tarafında viewcontroller swift dosyasının görüntülenmesi sağlanır. Ardından storyboard arayüzünde mavi çizgi biçiminde görüntülenen sabitler IBOulet bağlantı kurmak için mouse'un sağ tuşu ile sürükle bırak işlemi uygulanır. Sürükle bırak işlemi ekranın sağ kısmında yer alan viewcontroller swift dosyası üzerinde sona erdirilmelidir. İşlem tamamlandığında IBOulet bağlantı kurulacağına ilişkin bir açılır pencere ile karşılaşılır ve buraya bu sabite geliştirme sırasında erişilebilmek için bir anahtar kelime girilir.



Resim 86 : Sabit İçin Oluşturulan IBOulet Bağlantısı

```

9 import UIKit
10
11 class ViewController: UIViewController {
12
13     @IBOutlet weak var ustBosluk: NSLayoutConstraint!
14     @IBOutlet weak var sagBosluk: NSLayoutConstraint!
15     @IBOutlet weak var altBosluk: NSLayoutConstraint!
16     @IBOutlet weak var solBosluk: NSLayoutConstraint!
17
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         // Do any additional setup after loading the view, typically from a nib.
22     }
23
24 }
```

Resim 87 : Tüm Sabitlerin Swift Dosyasında Bulunan IBOutlet Bağlantıları

Arayüzde bulunan diğer tüm sabitler aynı işlem tekrar edilerek swift dosyasına bağlanmalıdır. (Bknz. Resim 85)

Tüm bu işlemler yapıldıktan sonra, artık sabitlerin değerlerinin değişim işlemi gerçekleştirilebilir haldedir. Sabitlerin değerleri değiştirildiğinde, işlem sonucu olarak sabitlerin verildiği *view* elementinde konum ve boyut ile ilgili değişiklikler gözlenmelidir.

Bu aşamada karar verilmesi gereken nokta, animasyonun hangi kısımda başlayacağıdır. Bir butona basıldığında, ya da ekran yüklenliğinde bu animasyonu tetikleyip animasyonun çalışması sağlanabilir. Anlatılmaya çalışılan örnekte *viewDidAppear* fonksiyonu içerisinde animasyon tetiklenmiştir.

Animasyonu tetiklemeden önce sabit değerlerine animasyon gerçekleştirikten sonra varılacak olan değerler atanır. Sonrasında *UIView* sınıfının *animate* isimli alt fonksiyonu çağrılır.

Animate fonksiyonu *withDuration* isminde bir parametre almaktadır. Bu parametre “float” türünden bir değeri taşıyabilir. Parametre animasyonun ne kadar sürede tamamlanacağıının verisidir. Fonksiyonun kod bloğunda ise *view self* anahtar kelimesi ile erişilen *viewcontroller* sınıfının alt özelliğinde bulunan *layoutIfNeeded* fonksiyonu ile sabit değişikliklerin uygulanması sağlanır. *layoutIfNeeded* fonksiyonu ekrandaki tüm *view*’lerin tekrar çizilmesine ilişkin yönergeleri çalıştırır. Bu noktada artık animasyon çalışır vaziyettedir.

```
24     override func viewDidAppear(_ animated: Bool) {  
25         super.viewDidAppear(animated)  
26  
27         self.ustBosluk.constant = 0  
28         self.sagBosluk.constant = 0  
29         self.altBosluk.constant = 0  
30         self.solBosluk.constant = 0  
31  
32         UIView.animate(withDuration: 4) {  
33             self.view.layoutIfNeeded()  
34         }  
35     }  
36 }
```

Resim 88 : UIView Animate Fonksiyonunun Kullanımı

b. Belirlenen Sürede Değişen Alpha Değeri ve Zincirleme Animasyon

```
9 import UIKit  
10  
11 class ViewController: UIViewController {  
12  
13     @IBOutlet weak var imageView: UIImageView!  
14  
15     override func viewDidLoad() {  
16         super.viewDidLoad()  
17     }  
18  
19     override func viewDidAppear(_ animated: Bool) {  
20         super.viewDidAppear(animated)  
21         UIView.animate(withDuration: 2.5, animations: {  
22             self.imageView.alpha = 1.0  
23         }) { (finished) in  
24             self.secondAnimation()  
25         }  
26     }  
27  
28     func secondAnimation(){  
29         UIView.animate(withDuration: 2.5, animations: {  
30             self.imageView.alpha = 0  
31         }) { (finished) in  
32             print("İkinci Animasyon Tamamlandı")  
33         }  
34     }  
35 }  
36 }
```

Resim 89 : UIView Animate Fonksiyonu ve Zincirleme Animasyon Örneği

Bir animasyon *view* ve türevlerinin çizimine yönelik değilde özelliklerinin değişmesine ilişkinde hazırlanabilir. Örneğin bir *view*'in "alpha" değeri değiştirilerek yavaş yavaş görünür ya da kaybolur hale getirilebilir.

Ayrıca tetiklenmiş bir animasyonun bitiş anı *animate* fonksiyonu içerisinde tanımlanan bir *completion handler* ile yakalanabilir. Bir animasyonun bitişinin ardından başka bir animasyonun çalıştırılması ile de zincirleme animasyon yapısı hazırlanabilir ve karmaşık ve farklı durumlara göre farklı çalışan animasyonlar üretilebilir.

Aşağıda konuya ilişkin örnek anlatılmaya çalışılmıştır.

Viewcontroller üzerine eklenen bir *ImageView* elementinin *IBOutlet* bağlantısı ile swift koduna bağlanması sonrası *ImageView* elementinin "alpha" özelliğine ulaşılabilir.

Resim 86 üzerinde dikkatle incelenirse görülecek olan en büyük farklılık Resim 87 daki örnekte kullanılan *layoutIfNeeded* fonksiyonunun kullanılmamış olmasıdır. Bu defa *animate* fonksiyonun kod沼guna, değiştirilmek istenen özelliğe animasyon sonucunda varılacak olan sonuç değeri ataması yapılır. Bunun sebebi, *view* ve türevlerinin animasyon çalıştırılırken tekrar çizilmemesi, yalnızca özelliklerinin değiştirilmesidir.

Yine incelendiğinde göze çarpan diğer bir bölüm ise *completion handler* kısmıdır. Bu kısım animasyon tamamlandıktan sonra çalışan kod沼unu içerir. *Completion handler* içerisinde tetiklenecek diğer animasyonlar, zincirleme animasyonlar yapısını ortaya çıkararak daha farklı ve algoritmik bir animasyon yapısının kurulmasına olanak tanır.



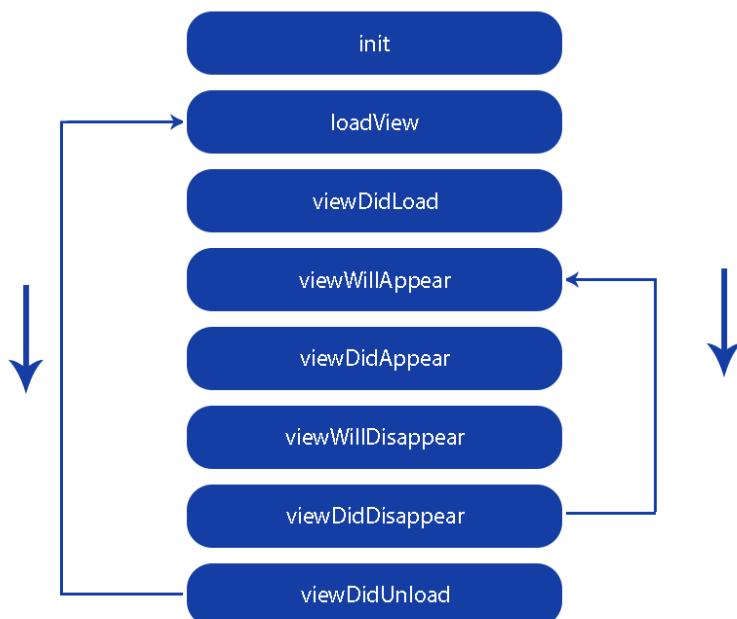
Kare Kod 12 : <https://github.com/ios-kitap/Animasyonlar>

Animasyonlara ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

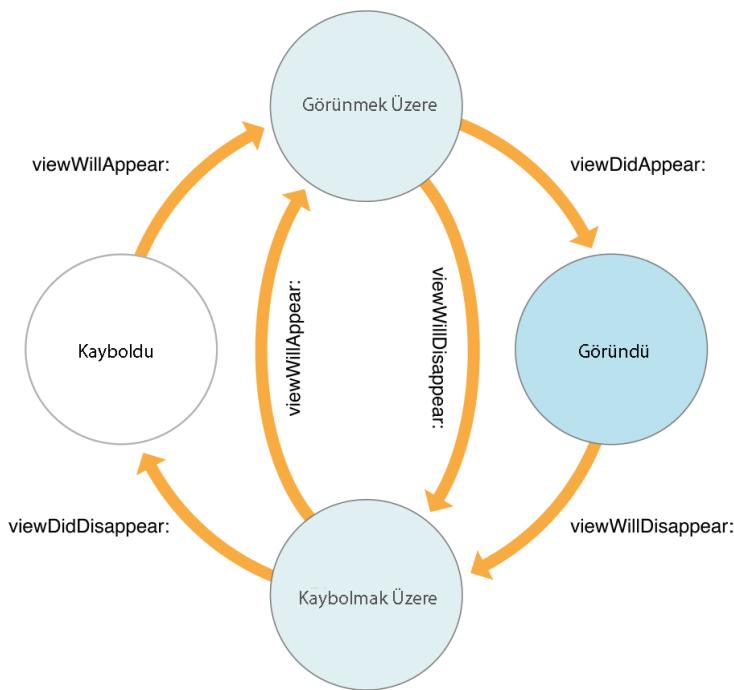
24. ViewController Yaşam Döngüsü

ViewController'lar ios uygulama geliştirme sürecinde kullanıcı arayüzleri geliştirmek için geliştirilmiş olan ve en fazla kullanılan yapılardan birisidir. Genellikle her uygulama birden fazla viewController'ler ile çalışmaktadır. ViewController'lar kendi bünyelerinde diğer *view*'leri yaşırlar ve uygulama içeriği kullanıcıya gösterilmiş olur.

ViewController'ın uygulama çalışma döngüsü içerisinde bir yaşam hiyerarşisi yada döngüsü vardır. Burada yaşam döngüsünden kasıt, viewController'ın yaratılışından (ilk yüklenme), yok oluşuna kadar (uygulamanın kapanması) olan süreçtir ve bu süreç olaylardan (event) oluşur. Aşağıda viewController'ın yaşam döngüsünü ifade eden bir görsel mevcuttur.



Resim 90 : ViewController Yaşam Döngüsü



Resim 91 : ViewController Görünüm İlişkileri

a. init Fonksiyonu

init fonksiyonu yaratılacak olan viewController için ilk çalışan fonksiyondur. Bu fonksiyon ile tasarıma ilişkin yapıtlarda değişiklikler yapılabilir ve viewController için bağımlılıklar belirlenebilir. Bir viewController yaşam döngüsü içerisinde yalnızca bir defa çalıştırılabilir.

b. loadView Fonksiyonu

loadView fonksiyonu viewController yazılımsal açıdan hazır olduğunda çalıştırılır. Bu fonksiyon override edilerek yazılımsal olarak başka viewler viewController'a dahil edilebilir. ViewController içerisindeki arayüzü arayüz dosyasından yükler ve kullanıma hazır hale getirir. Ancak storyboard ekranı genel olarak kullanıldığı için bu fonksiyon görmezden gelinebilir.

c. viewDidLoad Fonksiyonu

viewDidLoad fonksiyonu bir viewcontroller için muhtemelen en fazla önem arz eden olaydır. Viewcontroller artık yüklenmiş ve kullanılabilir vaziyettedir. Arayüz objelerinin ayarlanması ve arkaplan işlemlerinin gerçekleştirilmesi için kullanılabilir. Örneğin bazı network işlemleri bu fonksiyon altında gerçekleştirilebilir.

d. viewWillAppear Fonksiyonu

Bazı durumlar vardırkı kullanıcılar uygulamayı kapatmadan başka uygulamalara geçebilirler ve tekrar asıl uygulamaya geri dönenbilirler. Bu durumda yürürlükteki viewcontroller ram de tutulur ve oradan çağrılarak tekrar görünür hale getirilir. İşte tamda görünür hale gelmeden önce bu fonksiyon çalıştırılır. Dolayısı ile bu fonksiyon birden fazla kez çalışabilir. Bu özelliğinden dolayı genellikle kullanıcı arayüzünün güncellenmesi için kullanılabilir.

e. viewDidAppear Fonksiyonu

viewWillAppear fonksiyonunda bahsedilen ve henüz görünür hale gelmemiş ancak görünmek üzere olan viewcontroller'ın görünür hale geldiği anda çalışacak olan fonksiyondur. Bu fonksiyon override edilerek, başlayacak olan animasyonlar, videolar, sesler başlatılabilir ya da bazı network işlemleri gerçekleştirilebilir.

f. viewWillDisappear Fonksiyonu

Bir sonraki viewcontroller' a geçilmeden önce ve yürürlükteki viewcontroller ekranından kaldırılmadan hemen önce çalışan fonksiyondur. Nadiren override edilebilir.

g. viewDidDisappear Fonksiyonu

Viewcontroller ekranдан kaldırıldığında bu fonksiyon çağrırlır. Dolayısı ile viewcontroller ekranda değilken yapılmaması gereken işlemler bu fonksiyon override edilerek durdurulur.

b. viewDidUnload Fonksiyonu

Aslında *viewDidUnload* fonksiyonu iOS 6 güncellemesi ile geçersiz kılınmış bir fonksiyondur ve kullanımı artık yoktur ancak birçok kaynakta hala bahsedilmektedir.

i. didReceiveMemoryWarning Fonksiyonu

iOS cihazları sınırlı bir hafızaya sahiptir. Viewcontroller' lar ile ilgili işlemler yaparken bu sınırlı hafızada aşırı kullanım durumu söz konusu olursa bu durum bu fonksiyon aracılığı ile takip edilebilir. Böylelikle bu fonksiyon çalıştığı anda hafıza temizliği gerçekleştirilebilir.

```
32     override func loadView() {
33         print("ViewController Yükleniyor")
34     }
35
36     override func viewWillAppear(_ animated: Bool) {
37         print("ViewController Görünmek Üzere")
38     }
39
40     override func viewDidAppear(_ animated: Bool) {
41         print("ViewController Göründü")
42     }
43
44     override func viewDidLoad() {
45         super.viewDidLoad()
46         // Do any additional setup after loading the view, typically from a nib.
47     }
48
49     override func viewWillDisappear(_ animated: Bool) {
50         print("ViewController Kaybolmak Üzere")
51     }
52
53     override func viewDidDisappear(_ animated: Bool) {
54         print("ViewController Kayboldu")
55     }
56
57     override func didReceiveMemoryWarning() {
58         print("Hafıza Aşımı!")
59     }
```

Resim 92 : ViewController Yaşam Döngüsü Fonksiyonları Kullanımı



Kare Kod 13 : <https://github.com/ios-kitap/ViewControllerLifecycle>

ViewController yaşam döngüsüne ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

25. UIAlertController İle Açıılır Pencereler

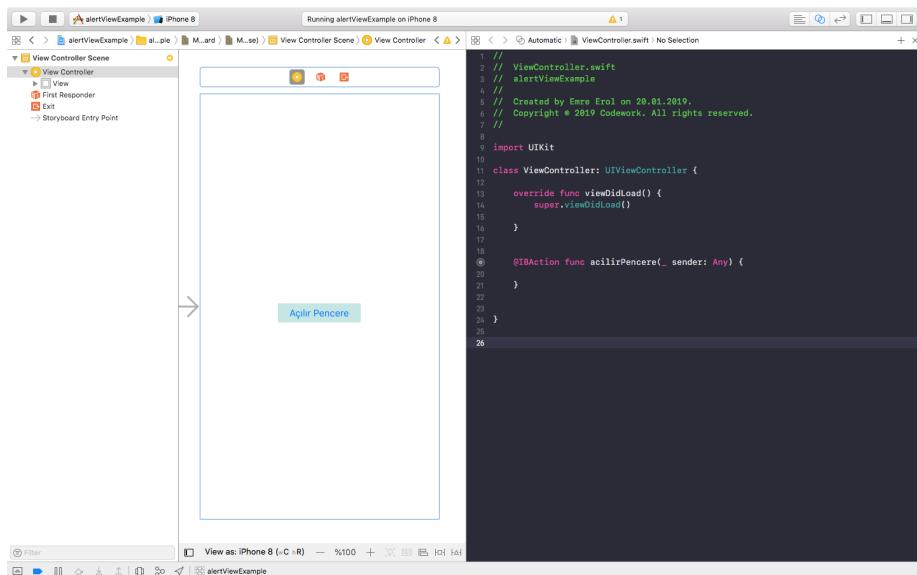
Bazı durumlarda kullanıcıya açılır pencere (PopUp) ile uyarılar, mesajlar gösterilmesi gerekebilir. Web sitelerinde özellikle reklamlar için sıkça kullanılan bu yapılar aynı şekilde iOS işletim sisteminde uygulamalar içerisinde de kullanılmaktadır. Bunun için iOS8 den önce *UIalertView*, iOS8 ve sonrasında *UIAlertController* sınıfları kullanılmaktadır.

Bu sınıflar açılır pencerenin başlık içerip içermeyeceğinin, nasıl bir mesaj barındıracığının ve hangi buttonların açılır pencere içerisinde mevcut olacağının belirlenmesinde geliştiriciye yardımcı olmaktadır. Bunların dışında tamamen farklı özelleştirilmiş bir tasarıma sahip açılır pencere hazırlamak ve uygulama içerisinde kullanmakta mümkündür.

Tüm açılır pencereler sadece bilgi vermek ya da uyarmak için değil, bilgi almak içinde kullanılabilir. Örneğin SMS kontrolünün yapıldığı bir uygulamada gelen SMS içerisinde yer alan kodun girilmesi için bir popup penceresi kullanılabilir. Bunun için açılır pencere içerisine textfields'ların yerleştirilip takip edilmesi yeterli olacaktır.

Bir buttona basıldığında basit bir açılır pencerenin gösterilmesine ilişkin örnek adım adım aşağıda verilmiştir.

Storyboard üzerinde bulunan ve IBAction bağlantısı ile swift dosyamıza eklenmiş olan bir buton aracılığı ile bir açılır pencere oluşturulabilir.



Resim 93 : Storyboard ve Class Görünümü

Öncelikle swift dosyası içerisinde buttonun IBAction fonksiyonuna *alertViewController* sınıfından bir nesne türetilmelidir. Bu sınıfın yapıcı metodları bulunduğuundan, türetilmek istediğiinde XCode geliştiriciye uyarıda bulunacak ve bazı parametrelerin girilmesine ilişkin uyarı metni gösterecektir. Bu parametreler açılır pencere başlığı, açılır pencere mesajı ve stilidir.

```

@IBAction func acilirPencere(_ sender: Any) {
20     let alertDialog = UIAlertController(title: "Dikkat !!", message: "Bu bir
        popup penceresidir!", preferredStyle: .alert)
21 }

```

Resim 94 : AlertController Sınıfının Türetilmesi

Ardından oluşturduğumuz *alertViewController* nesnesi viewcontroller sınıfının sunucu metodu olan *self.present* ile mevcut viewcontroller'in üzerinde görüntülenmek üzere *present* fonksiyonuna verilir. Bu fonksiyon oluşturulan açılır pencereyi uygulama üzerine yükler ve görünür hale getirir.

```
18
19 @IBAction func acilirPencere(_ sender: Any) {
20
21     let alertDialog = UIAlertController(title: "Dikkat !!",
22                                         message: "Bu bir popup penceresidir!",
23                                         preferredStyle: .alert)
24
25     self.present(alertDialog, animated: true) {
26         print("Alert Dialog Gösterildi!")
27     }
28 }
```

Resim 95 : Türetilen UIAlertController Sınıfının Ekranda Gösterimi



Resim 96 : PopUp Penceresi Uygulama Üzerinde Çalışırken

Ekran görüntüsünde de görüldüğü gibi uygulama çalıştırıldığında açılır pencere başarılı bir şekilde viewController üzerine yüklenmiştir. Ancak Herhangi bir button mevcudiyeti bulunmamaktadır.

AlertController ile üretilmiş bir açılır pencereye button eklemek ve butonların aksiyonlarını takip etmek ihtiyacı hissedilebilir. Bunun için *UIAlertAction* isimli sınıfın yeni bir nesne üretilmelidir. Bu nesnede yapıcı fonksiyon barındırmaktadır. Aynı zamanda *completion handler* ismi verilen bir yapı barındırmaktadır. Bu yapı sayesinde kullanıcıların açılır pencere içerisinde bulunan butonlara basıp basmadıkları takip edilebilir.

*UIAlertAction*lar daha önce oluşturulan *AlertDialog* nesnesine *addAction* metodu ile eklenmelidir.

```
①    @IBAction func acilirPencere(_ sender: Any) {
20        let alertDialog = UIAlertController(title: "Dikkat !!",
21                                         message: "Bu bir popup penceresidir!",
22                                         preferredStyle: .alert)
23
24        alertDialog.addAction(UIAlertAction(title: "Tamam",
25                                         style: .default, handler: { (resp) in
26            print("Tamam a basıldı!")
27        }))
28
29        alertDialog.addAction(UIAlertAction(title: "İptal",
30                                         style: .cancel, handler: { (resp) in
31            print("İptal e basıldı!")
32        }))
33
34        self.present(alertDialog, animated: true) {
35            print("Alert Dialog Gösterildi!")
36        }
37    }
```

Resim 97 : *AlertController* Sınıfı İle Üretilmiş PopUp Penceresine *Action* lar İle Buttonların Eklenmesi ve Kontrolü

Uygulama çalıştırıldığında, açılır pencere üzerinde iki button artık görünür vaziyette ve kullanılabilir durumdadır.



Kare Kod 14 : <https://github.com/ios-kitap/UIAlertController>

UIAlertController sınıfı ile açılır pencerele ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

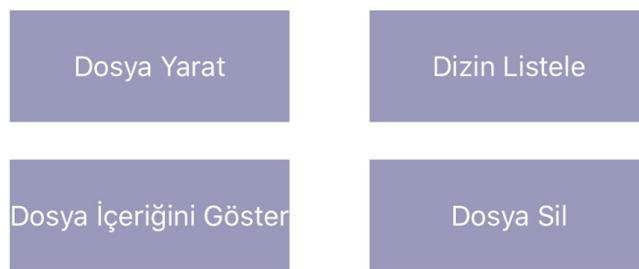


Resim 98 : Buttonlar Eklenmiş Bir PopUp Penceresinin Uygulama Üzerinde Görünümü

26. Dosya İşlemleri

Bazen geliştirilen uygulamalarda işletim sistemi içerisindeki bir konuma dosya yazmak gerekebilir veya daha önceden yazılmış bir dosyayı okumamız gerekebilir. Örneğin; internetten indirilen bir dosya önce okunup, sonra kullanıcının

tarafından değiştirilmesi sağlanıp tekrar internete yüklenmesi gerekebilir. Bunun için swift ile dosya yazma ve okuma işlemleri yapılmalıdır.



Resim 99 : Dosya Yazma ve Okuma İşlemleri İçin Oluşturulmuş Örnek Arayüz

Dosya yazma, okuma, silme işlemleri için *FileManager* sınıfı kullanılmaktadır. *FileManager* sınıfının alt metodları ile hem dosya yazma, okuma gibi işlemler hemde dizin listeleme ve silme gibi işlemler yapılmaktadır.

a. Dosya Yazma İşlemi

Filemanager sınıfı ile öncelikle dosya谱写mak istenen konum belirlenir. Ardından谱写mak istenen dosyanın adı ile beraber dizin oluşturularak değişkene verilir. Sonrasında bir *try catch* bloğu arasına string değişkeninin *write* metodu kullanılır. Bu metodun iki önemli parametresi olan *to* ile谱写acak olan dosya adı ve dizin, *encoding* ile karakter kodlaması belirlenir. Bu metod çalıştırıldığında hata döndürme ihtimaline karşı *try catch* bloğu ile kontrol edilir.

```

 1  @IBAction func createfile(_ sender: Any) {
 2
 3      let metin = "Dosyaya yazılacak olan metin."
 4      if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
 5          let dosyaYolu = dir.appendingPathComponent(file)
 6          do {
 7              try metin.write(to: dosyaYolu, atomically: false, encoding: .utf8)
 8          }
 9          catch {
10              print("Hata meydana geldi.")
11          }
12      }
13  }

```

Resim 100 : Dosya Yazma İşlemi Kod Bloğu

b. Dosya Okuma İşlemi

Dosya okuma işlemi dosya yazma işlemi ile oldukça benzerdir. Yine *filemanager* sınıfı ile dizin belirlenir ve yine dosya ismide kullanılarak dizin bir değişkene atanır. Ardından *try catch* bloğu ile hata yakalama ve kontrol kısmı hazırlanarak String sınıfından bir nesne türetilir. Nesne türetilirken bir yapıçı metot parametresi olarak *contentsOf* parametresine değişkende tutulan dizin verilir ve *encoding* parametresine de karakter kodlaması tanımlanır. Bu işlem bir string değişkenine atama operatörü ile eşitlenir. Eğer *try catch* bloğu hata yakalamazsa bu durumda string değişkeni dosya içeriğini artık taşımaktadır.

```

55    @IBAction func showFileContent(_ sender: Any) {
56        if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
57            let dosyaYolu = dir.appendingPathComponent(file)
58            do {
59                let text2 = try String(contentsOf: dosyaYolu, encoding: .utf8)
60                print(text2)
61            } catch {
62                print("Hata meydana geldi.")
63            }
64        }
65    }

```

Resim 101 : Dosya Okuma İşlemi Kod Bloğu

c. Dizin Listeleme İşlemi

Bir dizinin içerisindeki tüm dosya ve klasörlerin listelenmesi her işletim sisteminde sıkça başvurulan bir yöntemdir. Dizin içerisindeki dosya ve klasörler listelenerek hepsi tek tek kontrol edilebileceği gibi, kullanıcının seçimine bırakılarak işlem yapılması da sağlanabilir.

```

64    func enumerateDirectory() {
65        let documentsURL = fileManager.urls(for: .documentDirectory, in: .userDomainMask)[0]
66        do {
67            let fileURLs = try fileManager.contentsOfDirectory(at: documentsURL, includingPropertiesForKeys: nil)
68            print(fileURLs)
69        } catch {
70            print("Error while enumerating files \(documentsURL.path): \(error.localizedDescription)")
71        }
72    }
73}

```

Resim 102 : Dizin Listeleme İşlemi Kod Bloğu

Filemanager sınıfı ile listelenmek istenen ana dizin seçildikten sonra yine *filemanager* sınıfının *contentsOfDirectory* metodu ile dizin içerisindeki tüm içerik bir dizi değişken olarak alınabilir.

d. Dosya Silme İşlemi

Üretilen, okunan ve değiştirilen dosyalar sonunda zamanı geldiğinde silinmek istenebilir. Bu durumda yine *filemanager* sınıfı kullanılarak dosya silinebilir. *Filemanager* sınıfı ile dosyanın bulunduğu dizin belirlenir ve ardından dosya ismi ile beraber bir değişkene alınır. *Try catch* bloğu hazırlanarak *filemanager* sınıfının *removeItem* metoduna değişkene alınan dosya ve dosya yolu parametre olarak verilir. *Try catch* bloğunda hata yakalanmadıysa eğer bu metot çalıştığında dosya artık silinmiş vaziyettedir.

```
51    @IBAction func deleteFile(_ sender: Any) {
52        if let dir = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first {
53            let dosyaYolu = dir.appendingPathComponent(file)
54            do {
55                try fileManager.removeItem(at: dosyaYolu)
56                print("Dosya Silindi!")
57            } catch {
58                print("Hata meydana geldi")
59            }
60        }
61    }
62}
63}
```

Resim 103 : Dosya Silme İşlemi Kod Bloğu



Kare Kod 15 : <https://github.com/ios-kitap/filereadwrite>

Dosya yazma ve silme işlemlerine ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

27. Network İşlemleri

Bir sunucudan verilerin alınması, bazı bilgilerin uygulama kullanıcıları tarafından güncellenmesi veya internetteki bir dosyanın uygulamanın bulunduğu diske yazılması gerekebilir. Bu işlemlerin yapılabilmesi, canlı ve dinamik bir uygulamanın en önemli kısmını oluşturur. Apple tarafından sunulan *URLSession* sınıfı bu işlemlerin tamamı için geliştirilmiş ve kullanıma sunulmuştur.

URLSession sınıfı “GET” ve “POST” http istek yöntemlerini temel alan yapıdadır.

“GET” metodu ile bir sunucuya istek yapmak görece kolaydır. “GET” metodunda tüm istek bilgisi, web adresleri (link) üzerinden sunucuya yollanmaktadır. İşlemlerde sunucu tarafında bu isteğe göre yapılmaktadır. Sınırlı sayıda veri gönderilebilmektedir.

“POST” metodunda ise istek bilgisi header bilgisi içerisinde sunucuya gönderilir. Sınırsız sayıda bilgi gönderilebilmektedir. Ayrıca web adresleri üzerinden gönderilmediği için görece güvenli yapıdadır denilebilir.

a. *Http Get İsteği*

Http get isteği web adresi üzerinden gönderilen bir istektir. Aşağıda bir butona basıldığında, bir sunucu üzerindeki web servise get isteğinde bulunan kod incelenebilir. Örnek kod butonun IBAction bağlantısı ile oluşturulmuş fonksiyonuna eklenmiş vaziyettedir.

```

O @IBAction func start(_ sender: Any) {
20
21     if let url = URL(string: "http://codeworkagency.com/ioskitap/example02.php?veri=kullanici") {
22         let session = URLSession.shared
23         let request = URLRequest(url: url)
24         let task = session.dataTask(with: request) { (data,response,error) in
25             if error == nil {
26                 guard let cevap = data else {return}
27                 DispatchQueue.main.async {
28                     self.sonuc.text = String(data:cevap,encoding: .utf8)!
29                 }
30             }else{
31                 DispatchQueue.main.async {
32                     self.sonuc.text = error?.localizedDescription
33                 }
34             }
35         }
36         task.resume()
37     }
38 }
39 }
```

Resim 104 : Http GET İsteği Kod Bloğu

Birinci satırda GET metodu parametrelerini taşıyan web adresi *URL* nesnesinden türetiliyor ve bir sabite aktarılıyor. Sonrasında *URLSession* nesnesi *shared* metodu ile türetiliyor. Bu noktada *shared* metodu genel kullanım için *URLSession* nesnesine gereken varsayılan konfigürasyonu sağlıyor. Ardından *URLRequest* protokolü sonrasında *URLSession* görevinde kullanılmak üzere oluşturuluyor.

Task sabetine, *session.dataTask* metodu ile oluşturulan görev atanıyor. *dataTask* metodu parametre olarak *URLRequest* protokolünü *request* sabiti ile içerisinde alıyor ve bir completion handler yapısı barındırıyor.

Completion handler bize 3 veri sağlıyor; *data*, *response* ve *error*. Bu noktada *error*, http isteği yapılmırken bir hata olup olmadığına dair bilgiyi içermektedir. Eğer *nil* barındırıiyorsa, herhangi bir hata meydana gelmediği dikkate alınmalıdır. *Response* ise sunucunun istek sonrasında döndürdüğü cevap kodunu içerisinde barındırmaktadır ve genellikle başarılı isteklerde sunucular 200 veya 201 kodunu döndürürler. Son olarak *data* ise sunucunun cevabını içerisinde barındırmaktadır. Aslında bu kısım sunucunun ürettiği cevaptır. url değişkenindeki adresi tarayıcıya kopyalayıp o adresle gidilmek istediğiinde, tarayıcıda görüntülenen içeriktedir. Dolayısı ile alacağımız veride *data* değişkeninde barınmaktadır.

Guard kelimesi ile başlayan satırda *data* verisinin alınıp alınmadığı kontrol edilir ve hemen ardından, *String* veri tipine *utf8* kodlaması ile dönüştürülerek, sonuc ismindeki *UILabel* elementine ve dolayısı ile ekrana yazdırılır.

Bu kısımda farklı olarak göze çarpan *DispatchQueue.main.async* kod blogudur.

Bu kod bloğu ios işletim sisteminde *thread* denilen yapıları oluşturmak için kullanılmaktadır. *Thread'ler* birden fazla işlemi aynı anda yapabilmeyi sağlayan işlem parçacıklarıdır. *Completion handler* barındıran fonksiyonlar arkaplanda asenkron çalışan yapılar olduğundan bu fonksiyonlarda arayüz elementlerinin güncellenmesi için *thread* ler kullanılmalıdır.

Son satırda *task* değişkeninin *resume* metodu ile görev çalıştırılır.

b. Http Post İsteği

Http Get isteğinde olduğu gibi yine bir butona basıldığında sunucuya http post yöntemi ile *edittext* alanına girilen veriyi gönderen örnek kod aşağıda incelenebilir. Yine buradaki kodda buttonun IBAction bağlantısı ile oluşturulmuş fonksiyonuna yazılmıştır.

```

18    @IBAction func sendData(_ sender: Any) {
19        if let url = URL(string: "http://codeworkagency.com/ioskitap/example03.php") {
20            let session = URLSession(configuration: .default)
21            var dataTask: URLSessionDataTask?
22            var request = URLRequest(url: url)
23            request.httpMethod = "POST"
24            let postString = "veri="+field.text!;
25            request.httpBody = postString.data(using: String.Encoding.utf8);
26
27            dataTask = session.dataTask(with: request, completionHandler: { (data, response, error) in
28                if error == nil {
29                    let cevap = String(data:data!,encoding: .utf8)
30                    DispatchQueue.main.async {
31                        self.cevap.text = cevap
32                    }
33                } else {
34                    DispatchQueue.main.async {
35                        self.cevap.text = error?.localizedDescription
36                    }
37                }
38            })
39            dataTask?.resume()
40        }
41    }
42}

```

Resim 105 : Http POST İsteği Kod Bloğu

Birinci satırda POST metodu ile veri gönderilecek olan sunucunun web adresi *URL* nesnesinden türetilmiştir. Hemen sonrasında *URLSession* sınıfı varsayılan konfigürasyon ile türetilmekte ve bir *URLSessionDataTask* tipinde değişken oluşturulmuştur.

Sonrasında kullanılmak üzere *URLRequest* protokolü url değişkeni ile *request* değişkenine atanmıştır. *URLRequest* protokolünde bulunan *httpMethod* özelliği “POST” ifadesi ile değiştirilerek istek işleminin POST yapısında olduğu belirlenir.

Sunucuya gönderilecek olan veri *postString* ismindeki sabite string tipinde atanır. Arayüzde bulunan *edittext* metin ifade girişinin içeriğide bu stringe eklenmiş vaziyettedir. Dolayısı ile sunucuya gönderilecek olan veriyi kullanıcı kendisi belirleyebilmektedir. *URLRequest* protokolünde bulunan *httpBody* özelliği *postString* sabitinin *data* metodu ile byte dizisine çevrilerek eşitlenir.

Sonrasında daha önce oluşturulan *dataTask* değişkeni daha önce oluşturulan *session* nesnesinin *dataTask* görevine eşitlenir. *dataTask* *URLRequest* protokolünü parametre olarak almaktadır. Görev *resume* metodu çalıştırıldığında sunucunun ürettiği cevap arayüzde bulunan *UILabel* elementine yazdırılmaktadır.

c. Bir İmaj Dosyasını Okuma

Webdeki bir dosyayı okuma işlemi ile imaj dosyasını okuma işlemi aynı yönergelere sahiptir. Aşağıda incelenebilecek olan örnekte bir web adresindeki imaj dosyası http GET metodu ile okunarak *imageView* üzerinde gösterilmektedir.

```

1    @IBAction func start(_ sender: Any) {
2        if let url = URL(string:"http://codeworkagency.com/ioskitap/logoaydin.jpg")
3        {
4            let session = URLSession.shared
5            let request = URLRequest(url: url)
6            let dataTask = session.dataTask(with: request) { (data, response, error) in
7                if error == nil {
8                    if let data = data {
9                        if let image = UIImage(data: data) {
10                            DispatchQueue.main.async {
11                                self.imgview.image = image
12                            }
13                        }
14                    }
15                }
16            }
17            dataTask.resume()
18        }
19    }

```

*Resim 106 : Bir Http İsteği Oluşturarak Web Adresinde Yer Alan
İmaj Dosyasının Uygulamaya Alınması*

Birinci satırda sunucudaki imaj dosyasını gösteren web adresi *URL* nesnesinden türetiliyor ve bir sabite aktarılıyor. Sonrasında *URLSession* nesnesi *shared* metodu ile türetiliyor. Bu noktada *shared* metodu genel kullanım için *URLSession* nesnesine gereken varsayılan konfigürasyonu sağlıyor. Ardından *URLRequest* protokolü sonrasında *URLSession* görevinde kullanılmak üzere oluşturuluyor.

Task sabetine, *session.dataTask* metodu ile oluşturulan görev atanıyor. *dataTask* metodu parametre olarak *URLRequest* protokolünü *request* sabiti ile içerisinde alıyor ve bir *completion handler* yapısı barındırıyor.

Completion handler ‘ın sağladığı data içerisinde byte dizisi olarak web adresinin gösterdiği imajı taşımaktadır. *UIImage* nesnesinin yapıçı metodu *data* parametresi ile türetilerek, sunucudan alınan imaj; *imageView* elementini barındıran değişkenin *image* özelliğine atanmakta ve uygulama ekranında görüntülenmesi sağlanmaktadır.

Son satırda *task* değişkeni aracılığı ile *resume* metodu kullanılarak *dataTask* görevinin kod bloğunun çalıştırılması sağlanmaktadır.



Kare Kod 16 : <https://github.com/ios-kitap/NetworkIslemleri>

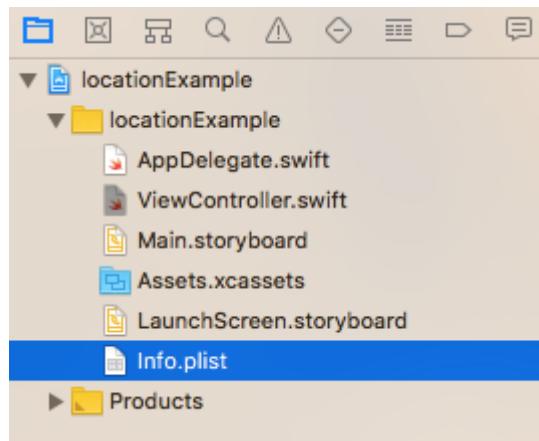
Network işlemlerine ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

28. Konum Tabanlı İşlemler

Kullanıcının konumlarına ilişkin yazılımlar hazırlamak, uygulamaların kullanışlığını ve popülerliğini artırdığı gibi aynı zamanda harita sistemleri ile entegre edilmeleriyle de başarılı sonuçlara sebebiyet verebilmektedir.

CoreLocation kütüphanesi aracılığı ile swift programlama dili ile cihaz gps modülü üzerinden uygulama üzerinde konum tespiti yapılmaktadır ve koordinatlara erişim sağlanabilmektedir.

iOS, kullanıcıların güvenliği için bazı özelliklerin kullanımını varsayılan olarak kısıtlamıştır. Bir kullanıcının konumunu talep etmek o kullanıcının güvenlik kısıtına takılır. Bu aşamada yapılması gereken ise kullanıcıyı konumunun alınmasına dair bilgilendirmek ve bu konum işlemi için izin istemektir. Kullanıcı eğer izin verirse ancak o halde konum verilerine ulaşılabilir. Bu tarz durumlar için proje içerisinde *Info.plist* isminde bir dosya mevcuttur. Bu dosya anahtar ve değer biçiminde verileri taşımakta ve uygulama içerisinde uygulama ile ilgili birçok konfigürasyonu barındırmaktadır.



Resim 107 : Proje Navigator 'ünde Info.plist Dosyasının Görünümü

Info.plist dosyasına tıklandığında aşağıdaki gibi bir ekran ile karşılaşılmaktadır.

Key	Type	Value
Information Property List		
Localization native development re...	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Privacy - Location Always Usage D...	String	We need your location always..
Privacy - Location When In Use Us...	String	We need your location..
Privacy - Location Always an...	String	We need your location always. Sorry..
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
► Required device capabilities	Array	(1 item)
► Supported interface orientations	Array	(3 items)
► Supported Interface Orientations (...)	Array	(4 items)

Resim 108 : Info.plist Dosyası

Sayfada Key, Type ve Value isminde 3 kolonlu bir yapı bulunmaktadır. Key kolonunun içeriği satırların üzerinde mouse ile gezinti yapıldığında, küçük bir artı işaretinin belirdiği görülür. Bu artı işaretine tıklanarak yeni değerler info.plist dosyasına girilebilmektedir.

Konum tabanlı işlemler uygulamak ve kullanıcının konumuna erişebilmek için info.plist dosyasına 3 farklı değer girilebilir;

- Privacy – Location Always Usage Description : Bu değer uygulama kapalı durumdayken konum istendiğinde kullanılmaktadır.
- Privacy – Location When In Use Usage Description : Bu değer uygulama açık durumdayken konum istendiğinde kullanılmaktadır.
- Privacy – Location Always and When In Use Usage Description : Bu değer yukarıda bahsedilen iki durumunda geçerli olduğu durumlarda kullanılmaktadır.

Key	Type	Value
Information Property List		
Localization native development re...	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Always and When In Use Usage D...	String	
↳ Privacy - Location Always and...	String	Main
↳ Privacy - Location Always Usag...	Array	(1 item)
↳ Privacy - Location Usage Descr...	Array	(3 items)
↳ Privacy - Location When In Use...	Array	(4 items)
↳ Privacy - Media Library Usage...		
↳ Privacy - Microphone Usage De...		
↳ Privacy - Motion Usage Descri...		
↳ Privacy - Music Usage Descri...		
↳ Privacy - NFC Scan Usage Des...		

Resim 109 : Info.plist Dosyasına Yeni Parametre Eklenmesi

Info.plist dosyasına girilen değerlerin ardından sağ kısımdaki value kolonuna metinsel ifade olarak kullanıcıyı bilgilendiren ve konumunun kullanılacağına dair bilgilendirme mesajı girilmelidir.

Privacy - Location Always Usage D...	String	We need your location always..
Privacy - Location When In Use Us...	String	We need your location..
Privacy - Location Always an...	String	We need your location always. Sorry..

Resim 110 : Info.plist Dosyasına Eklenmiş Konum Servisleri İzin Parametreleri

Bu aşamaların tamamından sonra artık swift kodları ile konumun alınmasına ilişkin işlemler gerçekleştirilebilir. Aşağıda swift kodlarına ilişkin ekran görüntüleri bir butona basıldığında kullanıcıdan konum alarak, ekranda yer alan label' a yazdıracak biçimde hazırlanmıştır.

```
8
9 import UIKit
10 import CoreLocation
11
12 class ViewController: UIViewController, CLLocationManagerDelegate {
13
14    @IBOutlet weak var lbl: UILabel!
15    let locationManager = CLLocationManager()
16
17    override func viewDidLoad() {
18        super.viewDidLoad()
19    }
}
```

Resim 111 : CoreLocation Kütüphanesinin Uygulamaya Dahil Edilmesi ve CLLocationManagerDelegate Kalitiminin Sınıfa Alınması

Öncelikle *CoreLocation* kütüphanesi projeye *import* anahtar kelimesi ile dahil edilmelidir. Ardından bu api üzerinden viewcontroller sınıfına *CLLocationManagerDelegate* kalıtım olarak alınmalıdır ve sınıf içerisinde konum işlemlerinin yapılmasına olanak sağlayan *CLLocationManager* sınıfı türetilecek bir sabite yada değişkene atanmalıdır.

```

34 func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
35     if let location = locations.first {
36         print(location.coordinate)
37         DispatchQueue.main.async {
38             self.lbl.text = "Lat: \(location.coordinate.latitude) & Lon: \
39                         \(location.coordinate.longitude)"
40         }
41     }

```

Resim 112 : Kalıtım Alınan locationManager Fonksiyonu

Kalıtım alınan *CLLocationManagerDelegate*, *didUpdateLocations* parametresini taşıyan *locationManager* metodu barındırmaktadır. Bu metot viewcontroller sınıfına dahil edilerek kullanılmaktadır ve konum koordinatları bu metot aracılığı ile alınabilmektedir.

didUpdateLocations parametresini taşıyan *locationManager* metodu *locations* isminde bir parametre daha taşımaktadır. Bu parametrenin *longitude* özelliği alınan konumun enlemi, *latitude* özelliği ise alınan konumun boylamını içerisinde barındırmaktadır.

```

20
21 @IBAction func locate(_ sender: Any) {
22
23     locationManager.requestAlwaysAuthorization()
24     if CLLocationManager.locationServicesEnabled(){
25         locationManager.delegate = self
26         locationManager.desiredAccuracy = kCLLocationAccuracyBest
27         locationManager.startUpdatingLocation()
28     }
29
30 }

```

Resim 113 : IBAction Bağlantısı İle Oluşturulmuş Fonksiyon İle Konum Servislerinin Başlatılması

Daha önce oluşturduğumuz *CLLocationManager* sınıfından türettiğimiz *locationManager* sabiti konum tabanlı işlemleri başlatmak ve takip etmek için kullanıldığı gibi, bazı hassasiyet konfigürasyonlarının belirlememiz için yardımcı olmaktadır. Buttonun IBAction bağlantısı ile oluşturulmuş *locate* ismindeki metodunun kod bloğuna *locationManager* sabitini kullanarak *requestAlwaysAuthorization* metodu ile kullanıcıya daha önce *info.plist* üzerinde oluşturulan bilgilendirme ve izin mesajı gösterilerek izin alınması sağlanmalıdır.

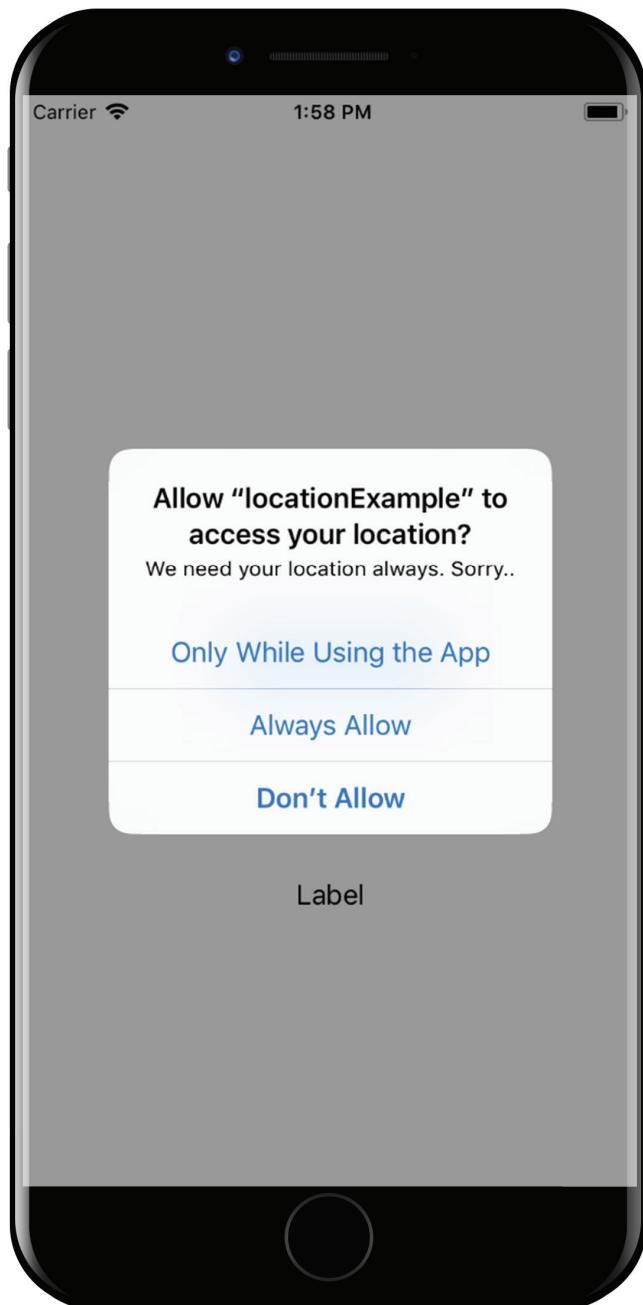
Ardından bir *if* kontrol bloğu ile konum servislerinin aktif olup olmadığı kontrol edilerek, *CLLocationManager* sınıfını taşıyan *locationManager* sabiti *delegate* edilmelidir. Yine aynı sabit aracılığı ile desired *Accuracy* özelliğine *kCLLocationAccuracyBest* sabetine eşitlenmelidir. Bu özellik uygulamanın kullanıldığı cihazın tüm mümkün donanımını kullanarak konum tespiti yapılmasını sağlayarak konuma ilişkin daha iyi sonuçlar elde edilmesini sağlamaktadır. Son olarak *locationManager* nesnesinin *startUpdatingLocation* metodu çağrılarak, *CLLocationManagerDelegate* sınıfından kalıtım alarak sınıfı dahil edilen *locationManager* ismindeki *didUpdateLocations* parametresini taşıyan fonksiyonun kodbloğu çalıştırılır. Bu noktada artık uygulama konum servislerini kullanarak konumu enlem ve boylam bazında ekrana yazdırmaktadır.

```

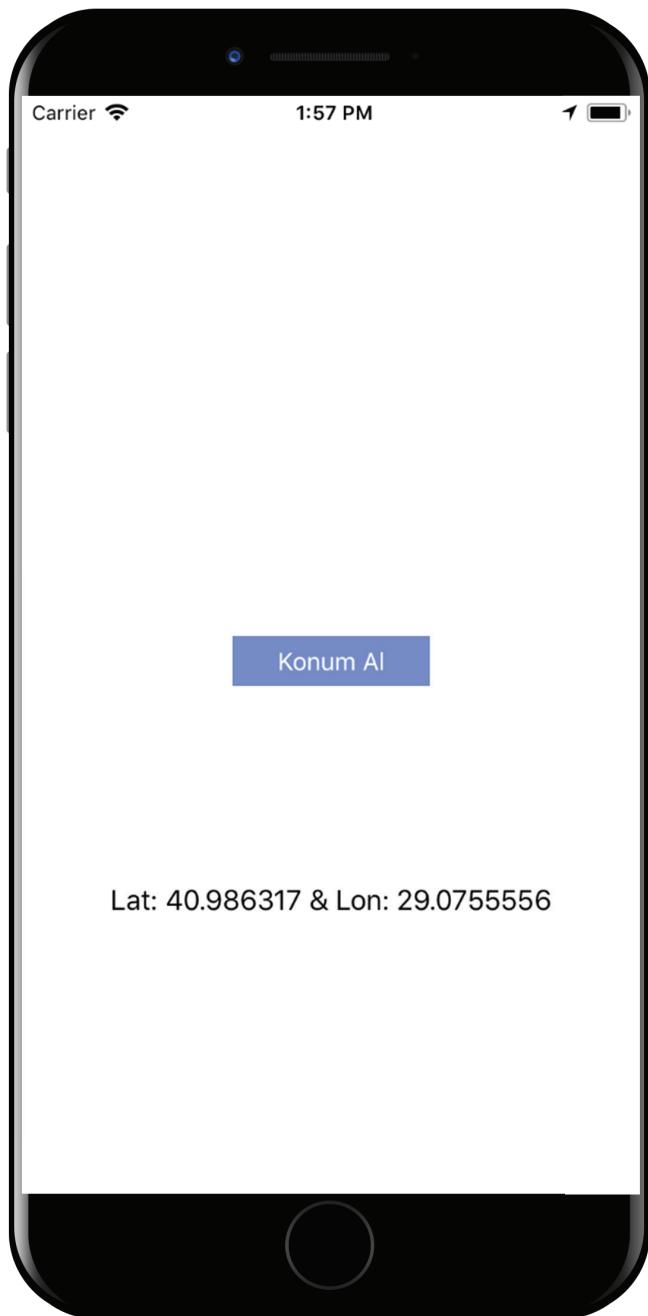
6
7 import UIKit
8 import CoreLocation
9
10
11 class ViewController: UIViewController, CLLocationManagerDelegate {
12
13
14    @IBOutlet weak var lbl: UILabel!
15    let locationManager = CLLocationManager()
16
17    override func viewDidLoad() {
18        super.viewDidLoad()
19    }
20
21
22    @IBAction func locate(_ sender: Any) {
23        locationManager.requestAlwaysAuthorization()
24        if CLLocationManager.locationServicesEnabled(){
25            locationManager.delegate = self
26            locationManager.desiredAccuracy = kCLLocationAccuracyBest
27            locationManager.startUpdatingLocation()
28        }
29
30    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
31        if let location = locations.first {
32            print(location.coordinate)
33            DispatchQueue.main.async {
34                self.lbl.text = "Lat: \(location.coordinate.latitude) & Lon: \(location.coordinate.longitude)"
35            }
36        }
37    }
38 }
39

```

Resim 114 : Konum Servislerine İlişkin ViewController Sınıfı Tam Görünümü



Resim 115 : Konum Servislerinin Kullanımına İlişkin Kullanıcı Bilgilendirme ve İzin PopUp Penceresi



Resim 116 : Konum Servisleri Çalıştırılarak Alınmış Koordinatların Ekranda Gösterimi



Kare Kod 17 : <https://github.com/ios-kitap/KonumServisleri>

Konum servislerine ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

29. Cihaz Donanım İşlemleri

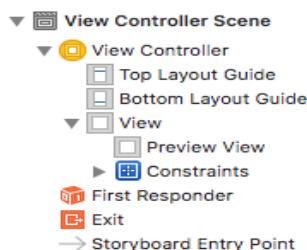
Mobil cihazlar daha avantajlı bir kullanım tecrübesi sunmak için çeşitli sensör donanımlarını barındırabilirler. Bu sensör donanımlarını kullanarak daha etkileşimli, daha kullanıcı tatmini sağlayan yazılımlar geliştirilebilir.

a. Kamera Erişimi ve Kullanımı

iOS donanım erişimlerinde kamera kullanımı en çok önem arz eden ve en çok kullanılan yöntemlerdendir. Günümüzde birçok trend mobil uygulama kullanıcılarına, cihazlarının kamerası aracılığı ile etkileşime geçerek onlardan görsel içerik oluşturmalarını ve daha iyi bir kullanım tecrübesi yaşatmayı hedeflemektedir.

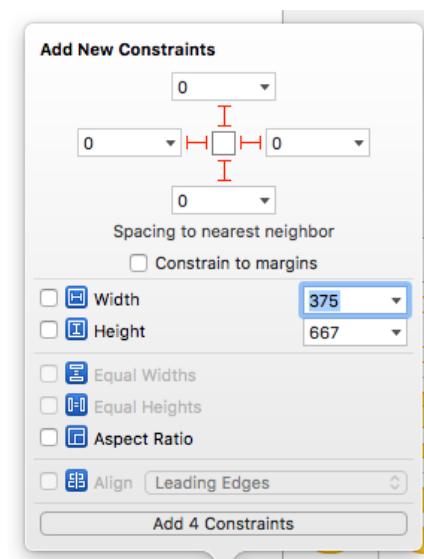
Swift kullanarak kamera erişimi ile fotoğraf çekimi ve video kayıt etme işlemlerini başlatma konusunda *AVFoundation* isimli framework incelenmelidir. Şüphesiz *AVFoundation* framework ü fazlasıyla büyük bir yapıdadır ancak bu konuda yalnızca kamera işlemlerine ilişkin özelliklerine degenilmiştir.

Bilindiği üzere, ios uygulama ortamında her şey view üzerine kuruludur. Dolayısı ile kamera erişimi ile sağlanan medya içeriğide bir view üzerinde görüntülenebilir. Yeni bir proje yarattığımızda oluşan viewcontroller üzerine isteneilen yerlere yerleştirilen *UIView* nesnesi, uygulama içerisinde kameradan sağlanacak olan medya içeriğinin görüntülenmesini sağlayacak olan view olacaktır.



Resim 117 : *UIView* Komponenti Preview View İsmi İle

Sonrasında gerekli sabitlerin verilmesi ile birlikte view yerleşimi sağlanır.



Resim 118 : *UIView* Komponenti Sabitleri

Kullanıcının cihazının kamerasına erişmek bir mahremiyet ve gizlilik konusudur. Dolayısı ile apple bu noktada bizi kullanıcından izin almaya ve kullanıcıyı bilgilendirmeye zorlar. Bu işlem için daha önce konum servislerinin kullanımında bahsedilen *info.plist* dosyası aracılığı ile izin talebinde bulunacağımıza ilişkin ayarlamaların yapılması gerekmektedir. *Privacy – Camera Usage Description* anahtar kelimesi ile bir bilgilendirme notu *info.plist* konfigürasyon dosyasına eklenmelidir.

Key	Type	Value
▼Information Property List		
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Launch screen interface file base name	String	LaunchScreen
Main storyboard file base name	String	Main
Privacy - Camera Usage Description	String	☐ Kameranızı kullanmamıza izin verir misiniz?
► Required device capabilities	Array	(1 item)
► Supported interface orientations	Array	(3 items)
► Supported interface orientations (iPad)	Array	(4 items)

Resim 119 : *info.plist* Konfigürasyon Dosyası Kamera İzni

Tasarım ve yerleşim konusu tamamlandıktan sonra, *uiview* elementine swift kodları ile erişim sağlamak ve kamera aracı ile sağlanacak olan medya içeriğini bu *uiview* üzerinde görüntülemek için *IBOutlet* bağlantısı kurulmalıdır.

Bu aşamadan itibaren swift kodlarına geçiş yapılabilir ve kamera görüntüsünün uygulama içerisinde görüntülenmesine ilişkin kod yazımı başlayabilir.

Öncelikle *AVFoundation* frameworkü projeye dahil edilmeli ve proje içerisinde kullanılacak olan değişkenler oluşturulmalıdır.

```

9 import UIKit
10 import AVFoundation
11
12 class ViewController: UIViewController {
13
14     @IBOutlet weak var previewView: UIView!
15
16     var captureSession: AVCaptureSession?
17     var videoPreviewLayer: AVCaptureVideoPreviewLayer?
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21     }
22
23 }
```

Resim 120 : *AVFoundation Framework' ünün Dahil Edilmesi*

Tüm bu bahsedilen işlemlerden sonra artık istenilen anda ya da istenilen şekilde kamera erişimi ile sağlanan medya içeriğine ulaşılabilir. Aşağıdaki ekran görüntülerinde viewcontroller yaşam döngüsünde anlatılan *viewDidLoad* metodu üzerinde erişim sağlanmıştır ve dolayısı ile uygulama ilk çalıştığı anda kamera görüntüsü *UIView* elementine verilmektedir.

viewDidLoad metodu içerisinde oluşturulan *captureDevice* sabitine *AVCaptureDevice* sınıfından hangi cihazın kullanılacağı ataması yapılır. Bu noktada istenilen cihaz seçimi yapılmamıştır. Örneğin dual kamera ya da standart kamera gibi seçenekler yapılmamıştır. *Default* kamera olarak tabir edilen kamera ise cihaz arkasında yer alan arka kamerasıdır.

```

 9 import UIKit
10 import AVFoundation
11
12 class ViewController: UIViewController {
13
14     @IBOutlet weak var previewView: UIView!
15
16     var captureSession: AVCaptureSession?
17     var videoPreviewLayer: AVCaptureVideoPreviewLayer?
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         let captureDevice = AVCaptureDevice.default(for: .video)
22     }
23
24 }
```

Resim 121 : Medya İçeriği Alınacak Cihazın Seçilmesi

Bu aşamadan sonra kameranın kullanılabilir olmaması ya da öyle bir kamera olmaması gibi durumlardan kaçınmak için *do-catch* bloğu ile hata yakalama işlemi gerçekleştirilir ve *do* bloğu içerisinde daha önce kamera seçimi yapılan *captureDevice* sabiti bir input isimli sabite atanmaya çalışılır. Bu noktada *try* anahtar kelimesi ile birlikte *AVCaptureDeviceInput* sınıfı, device parametresinin yer aldığı yapıçı metot ile türetilir ve yapıçı metot parametresine seçim yapılan *captureDevice* sabiti verilir.

```

9 import UIKit
10 import AVFoundation
11
12 class ViewController: UIViewController {
13
14     @IBOutlet weak var previewView: UIView!
15
16     var captureSession:AVCaptureSession?
17     var videoPreviewLayer:AVCaptureVideoPreviewLayer?
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         let captureDevice = AVCaptureDevice.default(for: .video)
22         do{
23             let input = try AVCaptureDeviceInput(device: captureDevice!)
24         }catch{
25             print(error)
26         }
27     }
28 }
29
30 }
```

Resim 122 : Seçimi Yapılan Cihazdan Giriş Nesni Oluşturulması

Hemen ardından ilk başta oluşturduğumuz değişkenlerden olan *captureSession* *AVCaptureSession* sınıfından türetilerek *addInput* metodu aracılığı ile input sabiti bu sınıfı atanır.

```

9 import UIKit
10 import AVFoundation
11
12 class ViewController: UIViewController {
13
14     @IBOutlet weak var previewView: UIView!
15
16     var captureSession:AVCaptureSession?
17     var videoPreviewLayer:AVCaptureVideoPreviewLayer?
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         let captureDevice = AVCaptureDevice.default(for: .video)
22         do{
23             let input = try AVCaptureDeviceInput(device: captureDevice!)
24             captureSession = AVCaptureSession()
25             captureSession?.addInput(input)!
26         }catch{
27             print(error)
28         }
29     }
30 }
31
32 }
```

Resim 123 : Cihaz Kamerasından Alınan Görüntüye Erişim

UIView nesnesinde kamera görüntüsünün yerleştirilmesi için kullanılacak olan *videoPreviewLayer* değişkeni *AVCaptureVideoPreviewLayer* sınıfından türetilir ve session parametresine *captureSession* değişkeni verilir.

videoPreviewLayer sınıfında yer alan *videoGravity* metodu ile görüntünün en boy oranı gibi özellikleri istenilen şekilde ayarlanır.

IBOutlet bağlantısı ile erişim sağlanabilecek olan tasarıma eklenilen *previewView* ismindeki *UIView* nesnesinin alt özelliği olan layer içerisinde yer alan *addSubLayer* metodu ile *videoPreviewLayer* değişkeni, *UIView* elementine bir alt katman olarak yerleştirilir. Bu alt katman görüntüyü *UIView* üzerinde görüntüleyecek olan yapıdır.

```

9 import UIKit
10 import AVFoundation
11
12 class ViewController: UIViewController {
13
14     @IBOutlet weak var previewView: UIView!
15
16     var captureSession: AVCaptureSession?
17     var videoPreviewLayer: AVCaptureVideoPreviewLayer?
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         let captureDevice = AVCaptureDevice.default(for: .video)
22         do{
23             let input = try AVCaptureDeviceInput(device: captureDevice!)
24             captureSession = AVCaptureSession()
25             captureSession?.addInput(input)
26             videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession!)
27             videoPreviewLayer?.videoGravity = AVLayerVideoGravity.resizeAspectFill
28             videoPreviewLayer?.frame = view.layer.bounds
29             self.previewView.layer.addSublayer(videoPreviewLayer!)
30             captureSession?.startRunning()
31         }catch{
32             print(error)
33         }
34     }
35 }
36 }
```

Resim 124 : iOS İşletim Sistemi Üzerinde Swift Programlama Dili İle Kamera Erişimi

Son olarak *captureSession* sınıfının metodу olan *startRunning* fonksiyonunun çağırılması ile kamera erişimi ile sağlanan medya içeriği *UIView* elementine aktarılmasına başlanır.

XCode üzerindeki ios simülatörleri kamera erişimi gibi özellikleri barındıran uygulamalarda destek sunmamaktadır. Bu sebeple, gerçek bir iphone bilgisayara

bağlanarak xcode üzerinde çalıştırılırsa uygulama test edilebilir. Uygulama çalıştırıldığında canlı kamera görüntüsü *info.plist* konfigürasyon dosyasında hazırlanan bilgilendirme ve izin açılır penceresinin ardından görünür vaziyette olmalıdır.



Kare Kod 18 : <https://github.com/ios-kitap/KameraErisimi>

Kamera erişimine ve kullanımına ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

b. Mikrofon Erişimi ve Kullanımı

Mikrofona erişip, kayıt yaparak bir dosya oluşturma ve oluşturulan dosyanın tekrar kullanılmasına ilişkin işlemler için *AVFoundation* frameworkü bize gerekli kabiliyeti sağlamaktadır. Bunun için özel olarak tasarlanan *AVAudioRecorder* isimli sınıf, yapılacak olan işlemler için gerekli tüm metotları içerisinde barındırarak geliştiriciye kolaylık sağlamaktadır.

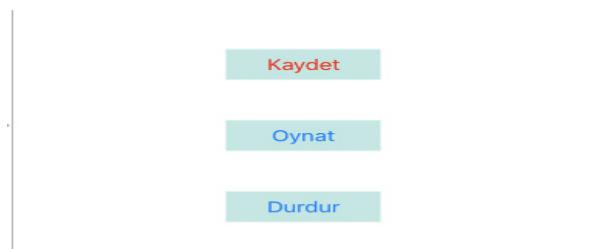
Ayrıca yine yaratılan ses dosyasının kullanımı da *AVAudioPlayer* sınıfı ile gerçekleştirilebilmektedir.

Bir iOS uygulamasında mikrofon donanımına erişerek ses kaydı gerçekleştirmek kullanıcı mahremiyetine ve gizliliğine girdiğinden, daha önce kamera erişimi ve konum tabanlı işlemlerde gerçekleştirildiği gibi *Info.plist* isimli konfigürasyon dosyası üzerinden, kullanıcıyı bilgilendiren ve kullanıcından izin alan ilgili parametreleri girilmesi gerekmektedir.

Key	Type	Value
▼ Information Property List	Dictionary	(15 items)
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Launch screen interface file base name	String	LaunchScreen
Main storyboard file base name	String	Main
Privacy - Microphone Usage Description	String	Bu uygulama mikrofonunuza erişmek istiyor.
► Required device capabilities	Array	(1 item)
► Supported interface orientations	Array	(3 items)
► Supported interface orientations (iPad)	Array	(4 items)

Resim 125 : Info.plist Dosyası Mikrofon Erişimi

Bu bölümde yer alan swift kodlarına ilişkin tüm ekran görüntülerini, kaydet, oynat ve durdur isimli 3 adet butonun bulunduğu ve IBAction bağlantısı ile swift koduna dahil edildiği bir senaryo çerçevesinde hazırlanmıştır.



Resim 126 : Mikrofon Erişimi Örnek Arayüz Tasarımı

En baştada belirtildiği gibi, mikrofon erişimi ve kullanımına ilişkin tüm gerekli işlemler *AVFoundation* frameworkü ile kullanılabilirliktedir. Bunun için projeye *AVFoundation* framework dahil edilmelidir.

AVFoundation framework ünün projeye dahil edilmesinden sonra, *AVAudioPlayerDelegate* ve *AVAudioRecorderDelegate* protokollerinden, arayüz tasarımının oluşturulduğu viewcontroller sınıfına kalıtım alınmalıdır ve sonrasında kullanılmak üzere *AVAudioPlayer* tipinde *audioPlayer* isminde ve *AVAudioRecorder* tipinde ve *audioRecorder* isminde iki değişken tanımlanmalıdır.

```

9 import UIKit
10 import AVFoundation
11
12 class ViewController: UIViewController, AVAudioPlayerDelegate, AVAudioRecorderDelegate {
13
14     var audioPlayer: AVAudioPlayer?
15     var audioRecorder: AVAudioRecorder?
16
17     override func viewDidLoad() {
18         super.viewDidLoad()
19         // Do any additional setup after loading the view, typically from a nib.
20     }
21
22 }

```

Resim 127 : AVFoundation Framework i̇nün Dahil Edilmesi ve Değişkenler

Uygulama ilk çalıştırıldığında *AVAudioRecorder* sınıfı yaratılmalıdır. *AVAudio* sınıfı ilk oluşturulduğunda kayıt edilecek ses dosyasının dosya yolu ile birlikte yaratılabilmektektir. Ayrıca ses kalitesi ve bit rate gibi bazı özellikleride içerisinde barındırır. Dolayısı ile *viewDidLoad* metodu içerisinde dosya yolu ve sınıfların türetilmesine ilişkin işlemlerin gerçekleştirilmesi gerekmektedir.

```

17     override func viewDidLoad() {
18         super.viewDidLoad()
19         let fileMgr = FileManager.default
20         let dirPaths = fileMgr.urls(for: .documentDirectory, in: .userDomainMask)
21         let soundFileURL = dirPaths[0].appendingPathComponent("ses.caf")
22         let recordSettings = [AVEncoderAudioQualityKey: AVAudioQuality.min.rawValue,
23                               AVEncoderBitRateKey: 16,
24                               AVNumberOfChannelsKey: 2,
25                               AVSampleRateKey: 44100.0] as
26                               [String:Any]
27         let audioSession = AVAudioSession.sharedInstance()
28         do {
29             try audioSession.setCategory(AVAudioSessionCategoryPlayAndRecord)
30         }catch let error as NSError {
31             print("Audio Session Error : \(error.localizedDescription)")
32         }
33
34         do{
35             try audioRecorder = AVAudioRecorder(url: soundFileURL, settings: recordSettings as [String:AnyObject])
36             audioRecorder?.prepareToRecord()
37         }catch{
38             print("Audio Session Error : \(error.localizedDescription)")
39         }
40     }

```

Resim 128 : Ses Kaydı Öncesi ViewDidLoad Üzerinde Hazırlık

Resim 125 üzerinde 19, 20 ve 21 numaralı satırlarda kayıt edilecek olan ses dosyasının dosya yolu belirlenmiştir. Ardından kayıt edilecek olan ses dosyasının kayıt kalitesine ilişkin veriler *recordSettings* isimli sabitte dizi biçiminde tanımlanmıştır.

audioSession isimli değişken *AVAudioSession* sınıfından *sharedInstance* metodu ile türetilmiştir. *AVAudioSession* birazdan yapılacak olan ses işlemleri ile ilgili kullanım amacının işletim sistemine bildirildiği ara bir sınıftır. Bu sabitin tanımlanmasının hemen ardından hata yakalama işlemi içerisinde *AVAudioSession* sınıfının alt metodu olan *setCategory*, *AVAudioSessionCategoryPlayAndRecord* parametresi ile çağrılmış ve sisteme yapılacak olan ses işlemlerine ilişkin niyet bildirilmiştir.

Daha önce belirlenen *audioRecorder* değişkeni *AVAudioRecorder* sınıfından; *url* ve *settings* parametreleri ile türetilmiştir. Bu parametrelere yine daha önce ses dosyası yolu ve ses kaydına ilişkin hazırlanan *recordSettings* ismindeki kayıt kalitesine ilişkin verileri taşıyan dizi verilmiştir. Son olarak *audioRecorder* değişkeni, *prepareToRecord* metodu ile ses kaydı yapmaya hazır vaziyete getirilmiştir.

Bu aşamadan itibaren artık butonların IBAction bağlantılarına ilişkin işlemler ve bu bağlantılar ile oluşturulan fonksiyonların içerisinde barınacak işlemler hazırlanabilir.

```

42
43     @IBAction func recordAudio(_ sender: Any) {
44         if audioRecorder?.isRecording == false {
45             audioRecorder?.record()
46         }
47     }
48
49     @IBAction func stopAudio(_ sender: Any) {
50         if audioRecorder?.isRecording == true {
51             audioRecorder?.stop()
52         }else{
53             audioPlayer?.stop()
54         }
55     }
56
57     @IBAction func playAudio(_ sender: Any) {
58         if audioRecorder?.isRecording == false{
59             do{
60                 try audioPlayer = AVAudioPlayer(contentsOf: (audioRecorder?.url)!)
61                 audioPlayer!.delegate = self
62                 audioPlayer!.prepareToPlay()
63                 audioPlayer!.play()
64             }catch let error as NSError{
65                 print("audioPlayer error : \(error.localizedDescription)")
66             }
67         }
68     }
69 }
```

Resim 129 : Kaydet, Oynat ve Durdur Buttonları IBAction Bağlantıları

Bu örnekte hazırlanan senaryo gereği arayüz tasarımda bulunan 3 butonun IBAction bağlantıları resim 126 üzerinde görülmektedir. *recordAudio* isimli IBAction bağlantısı kaydı başlatabilcek olan, *stopAudio* isimli IBAction bağlantısı devam eden kayıt varsa kaydı durduracak

olan yada devam eden bir kayıt oynatma işlemi varsa durduracak olan ve *playAudio* isimli IBAction bağlantısı da kaydı tamamlanmış ve bitmiş olan ses dosyasını oynatacak olan fonksiyonlardır.

İlk fonksiyon olan *recordAudio* üzerinde daha önce oluşturulan *audioRecorder* değişkeni kullanılarak *AVAudioRecorder* sınıfının özelliği olan *isRecording* ile devam eden bir kayıt olup olmadığı kontrol edilmektedir. Eğer devam eden bir kayıt yok ise bu durumda *record* metodu ile kayıt başlanmaktadır.

stopAudio isimli fonksiyonda ise yine devam eden bir kayıt olup olmadığı kontrol edilmekte ve eğer devam eden bir kayıt varsa stop metodu ile kayıt durdurulmaktadır. Eğer devam eden bir kayıt yoksa bu durumda *audioPlayer* değişkeni kullanılarak stop metodu ile devam eden bir kayıtta yürütme işleminin durdurulması işlemi gerçekleştirilmiştir.

Kayıttan yürütme işleminin gerçekleştirileceği *playAudio* isimli fonksiyonda öncelikle devam eden bir kaydın olup olmadığı kontrol edilmiş ve ardından daha önce tanımladığımız *audioPlayer* isimli değişken *AVAudioPlayer* sınıfından *contentsOf* parametresi ile türetilmiştir. *ContentsOf* parametresine *audioRecorder* değişkeninde yer alan kayıt dosyası url' si verilmiştir.

audioPlayer delegate ve prepare işlemleri ile kayıttan yürütme öncesi duruma hazırlanmıştır. Son olarak *audioPlayer*, *play* metodu kullanılarak kayıttan yürütme işlemine başlatılmıştır.

Viewcontroller sınıfına daha önce kalıtım alınarak eklenen *AVAudioPlayerDelegate* ve *AVAudioRecorderDelegate* protokollerini içerisinde yer alan fonksiyonlar viewcontroller sınıfına dahil edilerek kayıt işlemi ve kayıttan yürütme işlemleri hakkında sistem bilgisi alınabilir. Kayıttan yürütme devam ediyor mu ya da kayıt sırasında bir hata ile karşılaşılıp karşılaşılmasına ilişkin sistem bildirimleri takip edilebilir.

```

67
68    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool) {
69        print("Kayıttan Yürütme İşlemi Durdu")
70    }
71
72
73    func audioPlayerDecodeErrorDidOccur(_ player: AVAudioPlayer, error: Error?) {
74        print("Kayıttan yürütme hatası")
75    }
76
77
78    func audioRecorderDidFinishRecording(_ recorder: AVAudioRecorder, successfully flag: Bool) {
79        print("Kayıt işlemi sonlandırdı")
80    }
81
82    func audioRecorderEncodeErrorDidOccur(_ recorder: AVAudioRecorder, error: Error?) {
83        print("Kayıt işlemi hatası")
84    }

```

Resim 130 : Kayıt ve Kayıttan Yürütme İşlemlerinin Sistem Bildirimi

AudioPlayerDidFinishPlaying isimli fonksiyon delegate edilmiş player nesnesinin kayıttan yürütme işleminin bittiğini bildirir. Dolayısı ile kayıttan yürütme işlemi bittiginde çalışan bir fonksiyondur.

AudioPlayerDecodeErrorDidOccur isimli fonksiyon ise kayıttan yürütme ile ilgili bir çözüm hatası meydana geldiğinde çalışarak sistem içerisinde hatanın yakalanmasına olanak sağlar.

AudioRecorderDidFinishRecording isimli fonksiyon kayıt işleminin başarılı bir şekilde tamamlandığını bildiren fonksiyondur.

AudioRecorderEncodeErrorDidOccur kayıt işlemi sırasında veya başına bir kodlama hatası meydana geldiğinde çalışarak sistem içerisinde hatanın yakalanmasına olanak sağlar.

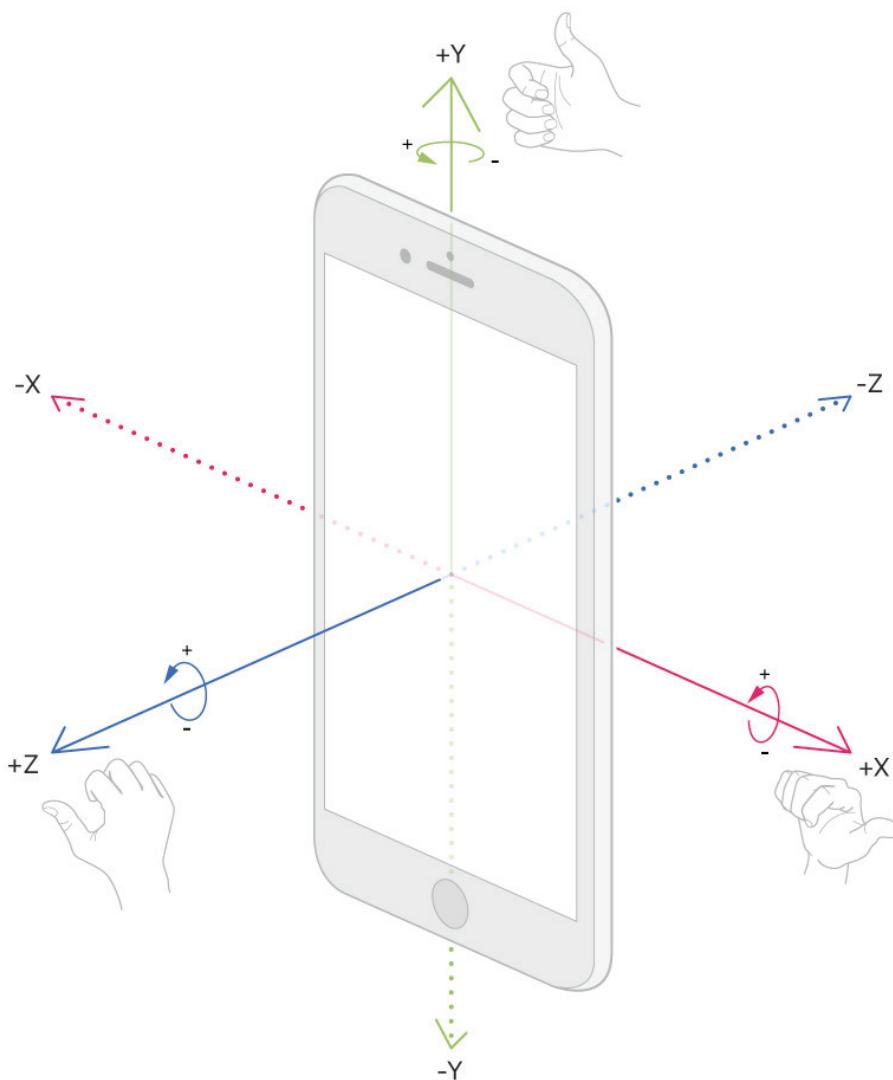


Kare Kod 19 : <https://github.com/ios-kitap/MikrofonSesKayit>

Kamera erişimine ve kullanımına ilişkin tüm örnekleri, karekodu kullanarak erișeceğiniz linkten bilgisayarınıza indirebilirsiniz.

c. Gyroscope Erişimi ve Kullanımı

iOS cihazlar ile erişilebilip kullanılabilen bir diğer donanımda jiroskopdur. Jiroskop ile cihazın uzamsal bir eksen etrafında dönme hızı ölçülebilir. iOS işletim sistemine sahip cihazların bir çoğu x,y ve z eksenleri için ölçüm yapabilen üç eksenli jiroskopa sahiptir. Bu jiroskop üzerinden ölçümler eksen çevresinde saniye başına radyan olarak ölçülebilir. Değerler dönüş yönüne bağlı olarak pozitif veya negatif olabilir.



Resim 131 : Jiroskop Eksenleri

Jiroskop ve değerlerine ulaşabilmek için *Core Motion* isimli framework yararlanılır ve basit bir biçimde jiroskop değerlerini okumak için *CMMotionManager* isimli sınıfından bir örnek türetilir.

Türetilen bu sınıf'a ait bir özellik olan *gyroUpdateInterval*'a verilen değer ile jiroskop değerlerinin ne kadar sürede bir alınacağı belirlenir. Yine aynı sınıf'a bağlı *startGyroUpdates* metodu ile güncel jiroskop değerleri alınır.

```
8 import UIKit
9 import CoreMotion
10
11 class ViewController: UIViewController {
12
13     @IBOutlet weak var xField: UITextField!
14     @IBOutlet weak var yField: UITextField!
15     @IBOutlet weak var zField: UITextField!
16
17     var motion = CMMotionManager()
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         self.gyro()
22     }
23
24     func gyro() {
25         motion.gyroUpdateInterval = 0.5
26         motion.startGyroUpdates(to: OperationQueue.current!) { (data, error) in
27             print(data as Any)
28             if let trueData = data {
29                 self.view.reloadInputViews()
30                 let x = trueData.rotationRate.x
31                 let y = trueData.rotationRate.y
32                 let z = trueData.rotationRate.z
33                 self.xField.text = "x: \(Double(x).rounded(toPlaces: 3))"
34                 self.yField.text = "y: \(Double(y).rounded(toPlaces: 3))"
35                 self.zField.text = "z: \(Double(z).rounded(toPlaces: 3))"
36             }
37         }
38     }
39 }
40 }
41
42 }
43
44 extension Double{
45     func rounded(toPlaces places:Int) -> Double {
46         let divs = pow(10.0,Double(places))
47         return (self * divs).rounded() / divs
48     }
49 }
50
51
```

Resim 132 : Gyroscope Kullanım Örneği



Kare Kod 20 : <https://github.com/ios-kitap/Jiroskop>

Jiroskop erişimine ve kullanımına ilişkin tüm örnekleri, karekodu kullanarak erişeceğiniz linkten bilgisayarınıza indirebilirsiniz.

d. Manyetometre Değerlerinin Ölçümü

Apple geliştirdiği mobil cihazlarında manyetometre de barındırmaktadır. Manyetometre kısaca bulunduğu ortamda manyetik gücü ölçmeye çalışan bir sensördür. Bir metal detektöründür de denilebilir. Etrafta yoğun güç sergileyen bir manyetik gürültü olmadığında, manyetometre dünyanın manyetik gürültüsünü ölçer.

```
9 import UIKit
10 import CoreMotion
11
12 class ViewController: UIViewController {
13
14     let manager = CMMotionManager()
15
16     override func viewDidLoad() {
17         super.viewDidLoad()
18
19         if manager.isMagnetometerAvailable {
20             manager.magnetometerUpdateInterval = 0.1
21             manager.startMagnetometerUpdates(to: OperationQueue.main) { (data, error) in
22                 if error == nil {
23                     print(data!)
24                 }
25             }
26         }
27     }
28 }
29
30 }
```

Resim 133 : Manyetometre Kullanımı

Manyetometre kullanımı için *CoreMotion* kütüphanesinin sağladığı sınıflar kullanılmaktadır. Bu yüzden projeye import edilmesi gereklidir. *CMMotionManager* sınıfı türetilerek bu sınıf aracılığı ile manyetometre kullanılabilirliği kontrol edilir ve yine bu sınıfa bağlı *startMagnetometerUpdates* metodunun belirlenen *interval* kapsamında çağrılmaması ile magnetometre verilerine ulaşılır.

e. Barometre İle Rakım (Yükseklik) Tespiti

iPhone 6 ve 6 Plus ile birlikte artık apple mobil cihazlarında barometrede (Basınç Sensörü) bulunmaya başlandı. Barometre ile kilopaskal cinsinden cihazınızın bulunduğu ortamdaki basınç değişiklikleri ölçülebilir ve aynı zamanda buna bağlı olarak, göreceli yükseklik tespiti metre cinsinden tahmin edilebilir.

Barometre kullanımına ilişkin sınıflarda yine *CoreMotion* frameworkü altındadır. Dolayısı ile barometreye ve onunla ilgili sınıflara erişim sağlamak için *CoreMotion* projenize import edilmelidir.

```
9 import UIKit
10 import CoreMotion
11
12 class ViewController: UIViewController {
13
14     let altimeter = CMAltimeter()
15
16     @IBOutlet weak var altitu: UILabel!
17     @IBOutlet weak var press: UILabel!
18
19
20     override func viewDidLoad() {
21         super.viewDidLoad()
22     }
23
24 }
```

Resim 134 : *CoreMotion Framework ve CMAltimeter Sınıfı*

CoreMotion import edildikten sonra bu framework içerisinde *CMAltimeter* sınıfının türetilmesi ile barometreye erişilebilir. Barometreye erişim sağlanmadan önce yapılması gereken donanımın kullanılabilir olup olmadığını kontrol etmektedir. *CMAltimeter* sınıfı altında yer alan *isRelativeAltitudeAvailable* isimli metod barometre nin kullanılabilir olup olmadığını yazılım içinkontrol ederek true yada false sonucunu üretir.

```

28     func startM(){
29         if CMAltimeter.isRelativeAltitudeAvailable() {
30
31             }else{
32                 print("Basınç Sensörü yok yada erişilemiyor..")
33             }
34     }

```

Resim 135 : Barometrenin Erişilebilirliğinin Kontrolü

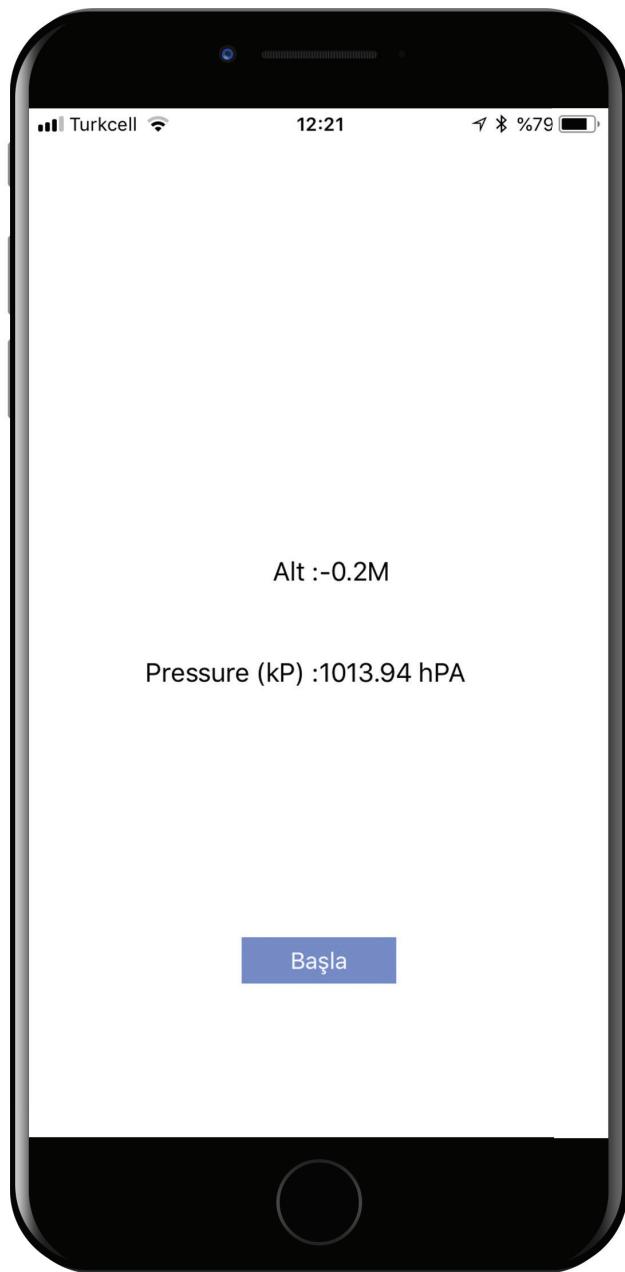
Türettiğimiz *CMAltimeter* sınıfı metodlarından *startRelativeAltitudeUpdates* metodu ile sınıfın ölçüm değerleri *CMAltitudeData* sınıfından alınmaya başlanır. *CMAltitudeData* sınıfının iki alt bileşeni olan “pressure” ile kilopascal cinsinden basıncı, “relativeAltitude” ile göreceli yüksekliğin metre cinsinden ölçümü alınır.

```

9 import UIKit
10 import CoreMotion
11
12 class ViewController: UIViewController {
13
14     let altimeter = CMAltimeter()
15
16     @IBOutlet weak var altitu: UILabel!
17     @IBOutlet weak var press: UILabel!
18
19
20     override func viewDidLoad() {
21         super.viewDidLoad()
22     }
23
24     @IBAction func startPRes(_ sender: Any) {
25         startM()
26     }
27
28     func startM(){
29         if CMAltimeter.isRelativeAltitudeAvailable() {
30             altimeter.startRelativeAltitudeUpdates(to: OperationQueue.main) { (data, error) in
31                 if error == nil {
32                     DispatchQueue.main.async {
33                         self.altitu.text = String.init(format: "%.1fM", (data?.relativeAltitude.floatValue)!)
34                         self.press.text = String.init(format: "%.2f hPa", (data?.pressure.floatValue)!*10)
35                     }
36
37                 }else{
38                     print(error?.localizedDescription)
39                 }
40             }
41         }else{
42             print("Basınç Sensörü yok yada erişilemiyor..")
43         }
44     }
45
46
47 }

```

Resim 136 : *startRelativeAltitudeUpdates()* Fonksionunun Kullanım Örneği



Resim 137 : Barometre İle Yükseklik ve Basınç Değeri Ölçümü



Kare Kod 21 : <https://github.com/ios-kitap/Barometre>

Barometre erişimine ve kullanımına ilişkin tüm örnekleri, karekodu kullanarak erişecğiniz linkten bilgisayarınıza indirebilirsiniz.

30. Kullanıcı Bildirimleri

iOS işletim sistemi içerisinde meydana gelen olaylar hakkında kullanıcıyı bilgilendirmek için kullanılan bir bildirim paneli bulunmaktadır. Bildirim sistemi sayesinde kullanıcı o esnada cihazını kullanmıyor olsa da, bildirim paneline mesaj bırakılabilir. Kullanıcı bildirim paneline bırakılan bu mesaj ile etkileşime geçtiğinde istenen işlemler uygulama üzerinden yapılabilir.

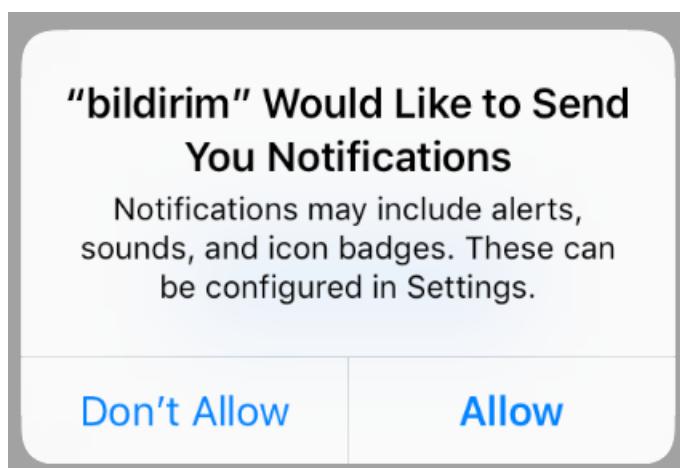
Sunucu tarafından kullanıcıya bildirim gönderileceği gibi ki genellikle bu yapı “push notifications” olarak isimlendirilir ve kullanılır, uygulamanın içinden lokal ortamda da bildirimler gönderilebilmektedir.

```
14     override func viewDidLoad() {
15         super.viewDidLoad()
16         UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .badge, .sound], completionHandler:
17             {didAllow,error in
18                 if error == nil {
19                     }
20                 })
21     }
22 }
```

Resim 138 : Bildirim İzinlerinin İstenmesi

Kullanıcı bildirimlerinin bırakılabilmesi için *UserNotifications* isimli framework bize gerekli sınıfları sağlamaktadır. Bir bildirim mesajı bırakmadan önce uygulamanın kullanılacak olan bildirimlere ilişkin kullanıcıdan izin alması gerekmektedir. Dolayısıyla ilk adım frameworkün dahil edilmesi ve izin işleminin gerçekleştirilmesidir.

UserNotificationsCenter sınıfının alt metodu olan *requestAuthorization* metodu ile gerekli izinle kullanıcıdan alınabilir. Burada dikkat edilmesi gereken nokta ise izin fonksiyonu çağrılrken, parametre olarak options isminde bir diziyi almasıdır. Bu dizi içerisinde alert, sound ve badge isminde değerler taşırlar. Bu değerler bildirim nasıl bırakılacağına ilişkin parametrelerdir. Örneğin bildirim düşüğünde uyarı sesi çalınacak mı veya bildirim sayısı uygulama ikonunun üzerinde artırılacak mı bu kısımda belirlenen parametreler ile ayarlanabilir.



Resim 139 : Bildirim İzinlerinin Talebi

İzin alma işlemi gerçekleştirildikten sonra *UNMutableNotificationContent* sınıfı ile bir bildirim içeriği oluşturulur. Varsayılan olarak bir başlık, altbaşlık ve içerik metin ifadesi olarak belirlenir. Sınıfın "badge" isimli özelliğine 1 sayısı atanarak, uygulama ikonunun üzerinde kaç sayısının yazacağı tespit edilir. Bu noktada artık bildirim gösterilmeye hazırız ancak bildirim paneline düşürülmemiştir.

Hazırlanan panelin bildirim paneline düşürülmesi için

UNTimeIntervalNotificationTrigger sınıfı kullanılarak bir tetikleme mekanizması oluşturulur. Tetikleyici parametre olarak bir *timeInterval* ve *repeat* isminde değerler almaktadır. *timeInterval*, tetikleyicinin aktif hale geldikten sonra ne zaman çalışacağını ve *repeat* parametreside tetikleyicinin kendisini verilen süre içerisinde devamlı tekrar edip etmeyeceğini belirlemektedir.

Tetikleyicinin ve bildirim içeriğinin tamamı bir *UNNotificationRequest* sınıfında bir kimliklendirme ile yeni bir değişkene türetilir.

```

1 @IBAction func showNotif(_ sender: Any) {
2     let content = UNMutableNotificationContent()
3     content.title = "Bildirim Başlığı"
4     content.subtitle = "Bildirim alt başlığı"
5     content.body = "Bildirimin asıl içeriği"
6     content.badge = 1
7     let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)
8     let request = UNNotificationRequest(identifier: "ZamanDoldu", content: content, trigger: trigger)
9     UNUserNotificationCenter.current().add(request, withCompletionHandler: nil)
10 }

```

Resim 140 : Bildirim İçerığının Oluşturulması ve Bildirim Paneline Düşürülmesi



Resim 141 : Oluşturulan Bildirimin Bildirim Paneline Düşürülmüş Hali

Son olarak *UNUserNotificationCenter* sınıfının altında bulunan add metodu çalıştırılarak *UNNotificationRequest* sınıfından türetilen değişken parametre olarak verilir. Diğer bir parametre ise *completion handlers* diye bildiğimiz, kullanıcı bildiriminin, bildirim paneline düştükten sonra yapılacak olan işlemleri hazırlamamıza olanak sağlayan kısımdır.

31. WebServis Kavramı ve WebServis Kullanım Alanları

Bilişim toplumunun en önemli unsuru, gelişmiş iletişim altyapısı oluşturmaktadır. Bununla beraber bilişim toplumu varlığını sahip olduğu iletişim teknolojilerine ve bu teknoloji alt yapılarının kararlılığına borçludur.

İlk yapılan “Machine-to-Machine (M2M) iletişiminin üzerinden çok zaman geçmiş olmamasına rağmen, internetin yaygınlaşması ve son olarak “nesnelerin interneti” (iot – internet of things) yada bulut teknolojileri gibi anlayışların ortaya çıkıştı makineler arası iletişimi farklı bir boyuta taşıdı.

Mobil uygulamalar hayatımıza girdiği günden beri internet altyapısını kullanmakta ve bu sayede popüler ve kullanışlı olma niteliğini kazanmaktadır. Dolayısı ile bir mobil uygulamanın internette bulunan bir sunucuya ya da benzer bi yapı ile iletişime geçerek işlemleri gerçekleştirmesi en önemli konulardan biridir. Tam da bu noktada web servisler karşımıza çıkmaktadır.

a. WebServis Nedir

Büyük ya da küçük olsun internet altyapısını kullanan her yazılım sistemi genellikle bir veritabanı kullanmaktadır. Kullanıcıların iletişim ve kimlik bilgilerini tutan, uygulamanın spesifik kayıtlarını içerisinde barındıran bu veritabanları veriyi kategorize edilmiş biçimde kitlesel olarak içlerinde barındırırlar.

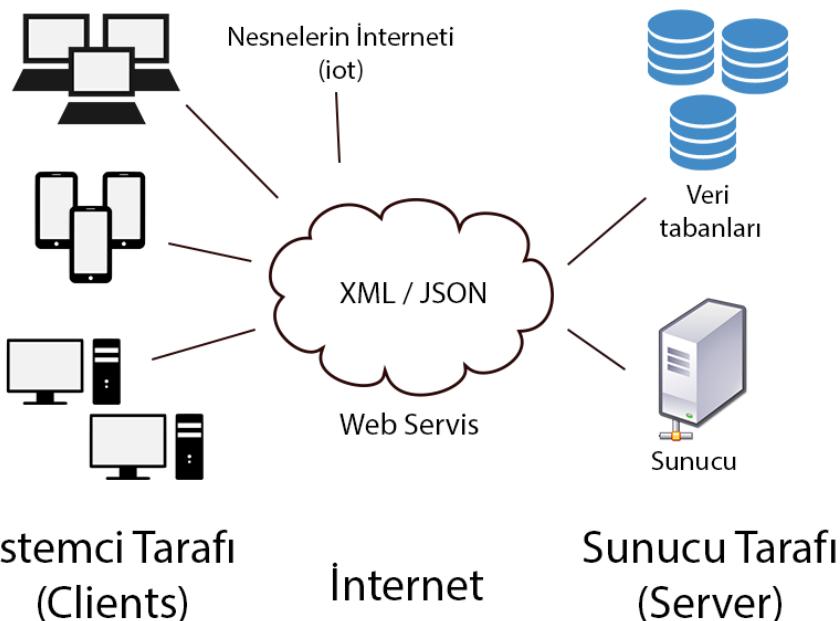
Ancak bu veritabanlarının içerisinde yer alan kitlesel verilerin istenilen zamanlarda mümkünse bilgiye dönüştürülerek alınması ve kullanılması gerekmektedir. Örneğin bir e-ticaret sisteminde bir kullanıcının satmak için eklediği ürünlerin listelenmesi gerektiğinde yazılım sistemi veritabanına erişerek, bu kullanıcının

daha önce eklediği bütün ürünleri arayıp bulması ve gerektiği şekilde ekran'a yansıtması gereklidir. Bu durum farklı platformlarda da aynı biçimde yapılmak durumunda kalınabilir. Web sitesi, iOS uygulaması ve android uygulaması şeklinde çalışan 3 farklı platforma verilerin gönderilmesi gerekebilir. İşte bu gibi durumlarda platformların farklılığına rağmen ortak bir iletişim modülü olarak web servisler karşımıza çıkmaktadır.

Tanım itibarı ile web servis; farklı platformların birbirleri ile iletişimine olanak sağlayan, makineler için anlamlı biçimde hazırlanmış yazılımlardır. Web servisler sunucu (Server) ile istemci (Client) arasında veri transferine olanak sağlayan yazılımlardır.

Genellikle tek yönlü olarak çalışan web servislerin çalışma mantığı basit kavramlar içermektedir. İstemci sunucudan bir istekte bulunur ve sunucu bu isteği alıp gereken cevabı istemciye iletir. Aslında tüm iletişim istekler üzerine kuruludur.

Web servisler istemci cevaplarını belli bir standartta verdiklerinden dolayı, birden çok platform bir web servis ile çalışılabilir.



Resim 142 : Web Servis İletişim Hiyerarşisi

Çeşitli web servis türleri bulunmasına rağmen burada önemli olan istemcinin sunucudan talebi sonrasında sunucunun verdiği cevabın yapısıdır. Web servisler XML (Extensible Markup Language) veri yapısı ile ya da JSON (Javascript Object Notation) veri yapısı ile cevap verebilirler.

JSON veri yapısı anahtar ve değer mantığı ile hazırlanan metinsel bir ifadedir. Basit ifadeler hazırlanabileceğ gibi içerisinde iç içe dizilerin bulunduğu karmaşık JSON yapılarında hazırlanabilir.

```
1  {
2      "name": "John",
3      "age": 30,
4      "cars": [ "Ford", "BMW", "Fiat" ]
5 }
```

Resim 143 : JSON Veri Yapısı

b. RestFull Api ve İletişim Protokollerı

Birçok web servis türü olmasına ve birçok iletişim protokoli olmasına karşın burada son zamanlarda kullanımı artan *Restfull* web servis yada “api” (Applicatiopn Programming Interface) incelenmiştir. *Restfull* web servisler iletişim protokolü olarak http (Hyper Text Transfer Protocol) protokolünü kullanırlar. Dolayısı ile tüm iletişim http protokolünün çizdiği çerçeve içerisinde gerçekleşmektedir.

Http protokoli bir web protokolüdür ve internet üzerinde web sitelerinde gezinirken kullanılan en yaygın iletişim protokollerinden birisidir.

Daha önce http GET ve http POST metodlarından Network İşlemleri başlığı altında bahsedilmiştir. *RestFull* web servislerde bu metodları kullanarak platformlar arası iletişimini sürdürmektedirler.

32. Harici Kütüphaneler ve Cocoapods Kullanımı

a. Harici Kütüphane Nedir ?

Bazı durumlarda başka geliştiricilerin hazırladığı kod dosyalarını, projeleri kullanmak akıllıca olabilir. Bir projede yapılması gereken devasa işler daha önce

başka bir geliştirici tarafından yapılmış, bir firma tarafından yayınlanmış olabilir. Yayınlanan bu yapılar sayesinde iş yükü azalabilir, yapılan işin performans artışı sağlanabilir.

Modül olarakта isimlendirilen harici kütüphaneler bazen firmalar tarafından yayınlanırken bazen de geliştiricilerin kendileri yayinallyabilmektedir. Böyle bir ortamda dolayısı ile çok çeşitli ve farklı amaca hizmet eden yapılarla karşılaşmak mümkündür. Tabi bu zenginlik geliştiricilerin proje yönetimi safhasını kolaylaştırmaktadır. Örneğin imaj filtreleme işlemi için bir sürü kod yazıp, algoritmalar üzerinde uğraşmak yerine mevcut bir kütüphanenin projeye dahil edilerek kullanılması, proje süresini kısaltabilir.

b. Paket Yöneticisi ~ Bağımlılık Yöneticisi Nedir ?

Paket yöneticisi yada bağımlılık yönetici olarak nitelendirilen yapılar geliştiriciye kütüphane yönetimini kolaylıkla yapabilmesi için avantajlar sunan bir yazılımdır.

Paket yönetim sistemleri genellikle bir depo üzerinde mevcut olan kütüphanelerin projeye dahil edilmesi ve dahil edilen bu kütüphanelerin güncel tutulmasına yönelik çalışırlar.

Farklı platformlar için geliştirilmiş farklı paket yönetim sistemleri mevcuttur. Örneğin .Net framework 'ü için hazırlanmış NuGet, ya da Android sistemi için kullanılabilecek olan Gradle paket yönetim sistemlerine verilebilecek en iyi yapılardır.

Swift ve Objective-C projeleri için geliştirilmiş olan CocoaPods ta iOS projelerinde sıkılıkla kullanılan paket yönetici olmuştur.

c. Cocoapods Kurulumu

Cocoapods yalnızca Mac bilgisayarlar üzerinde çalışan ve XCode projeleri için paket yönetimi sağlayan ruby ile yazılmış bir yazılımdır.

Harici kütüphaneler geliştirici tarafından projeye manuel olarak eklenebilir. Ancak

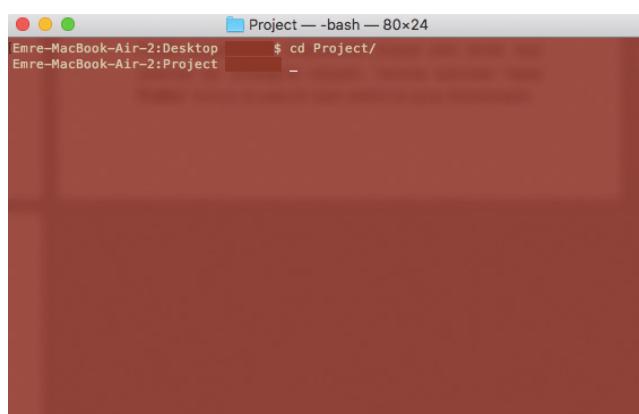
bu işlem bazı durumlarda, sürüm ve dil farklılıklarında sancılı olabilmektedir. Cocoapods bu sıkıntıların giderilmesi için kritik rol oynamaktadır. Cocoapods bir veya birden fazla harici kütüphanenin projeye otomatik olarak dahil edilip projenin gerekli kalibrasyonlarını geliştirici için yapmaktadır.

Cocoapods ‘un kurulumu terminal üzerinden gerçekleştirilmektedir. Mac üzerinde terminal uygulamasını açmak için “spotlight araması” başlatıp *terminal* yazıldıkten sonra klavyeden enter tuşuna basılması yeterlidir. Açılan pencere, mac bilgisayarın terminal konsuludur.

Terminal penceresi açıldıktan sonra, terminal üzerinden cocoapods kurulumu başlatılabilir. Kurulum için terminale “**sudo gem install cocoapods**” komutu yazıldıkten sonra enter a basılır. Bu aşamada kullanıcıdan mac için sistem parolası istenebilir. Terminal çıktıları sona erdikten sonra yine terminale “**pod setup**” komutu yazılıp enter tuşuna basılarak cocoapods kurulumu tamamlanır.

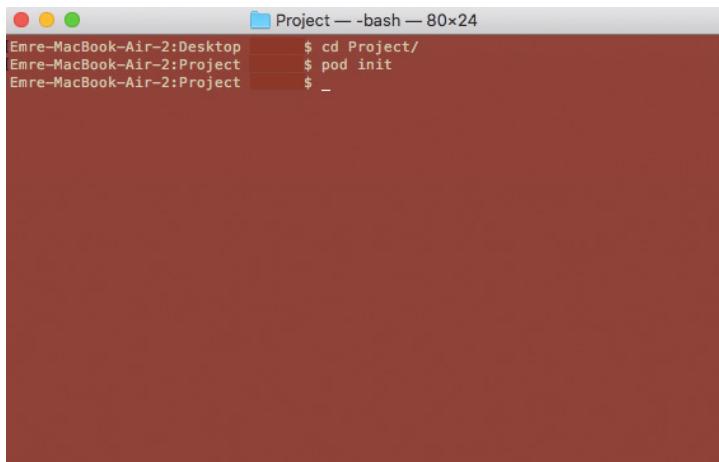
d. Cocoapods İle Projeye Harici Kütüphane Dahil Edilimi

Mevcut ve geliştirilmekte olan bir projeye harici kütüphanenin dahil edilmesi için de terminal uygulaması kullanılmaktadır. Yine “spotlight araması” ile terminal uygulaması açıldıktan sonra geliştirilmekte olan projenin dizinine geçiş yapılmalıdır. Örneğin masaüstünde proje isimli bir klasörün içerisindeyse geliştirilmekte olan proje, bu durumda terminale “**cd Desktop/proje**” komutu yazılarak geçiş yapılır.



Resim 144 : Terminal Üzerinden Proje Dizinine Geçiş

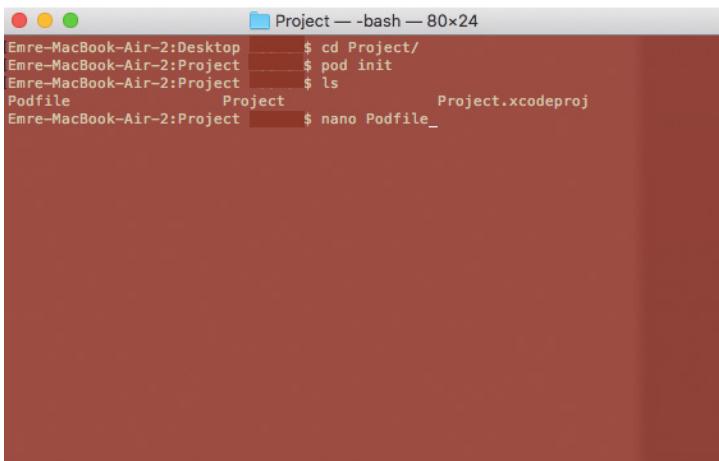
Proje dizini içinde terminale “**pod init**” komutu yazılıarak cocoapods ‘un projeyi harici kütüphanelerin eklenmesi için derleme işlemleri başlatılır. Bu işlem sonucunda proje dizini içerisinde “Podfile” isminde bir dosya yaratılır.



```
Project — bash — 80x24
Emre-MacBook-Air-2:Desktop $ cd Project/
Emre-MacBook-Air-2:Project $ pod init
Emre-MacBook-Air-2:Project $ _
```

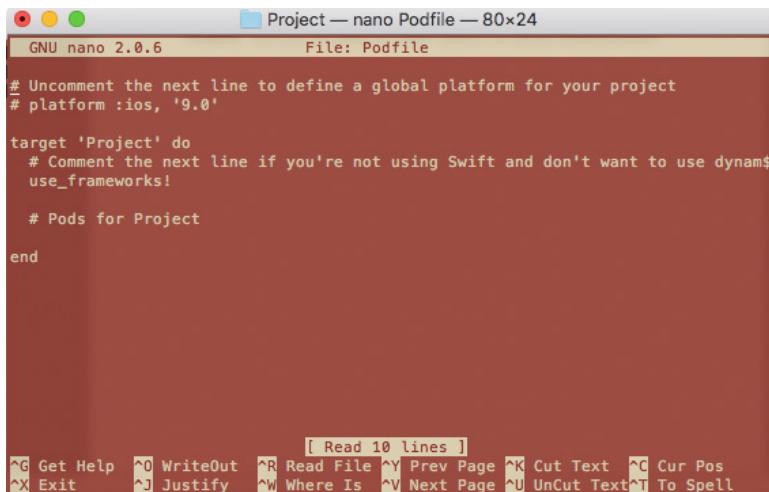
Resim 145 : Terminal Üzerinde pod init Komutunun Çalıştırılması

Podfile dosyası harici kütüphaneleri projeye dahil etmek veya çıkarmak için yönöttiğimiz dosyadır. Terminal üzerinden “**nano Podfile**” komutu ile ya da bir metin editörü ile açılıp düzenlenlenebilir.



```
Project — bash — 80x24
Emre-MacBook-Air-2:Desktop $ cd Project/
Emre-MacBook-Air-2:Project $ pod init
Emre-MacBook-Air-2:Project $ ls
Podfile          Project           Project.xcodeproj
Emre-MacBook-Air-2:Project $ nano Podfile_
```

Resim 146 : Terminal Üzerinde nano Podfile Komutunun Çalıştırılması



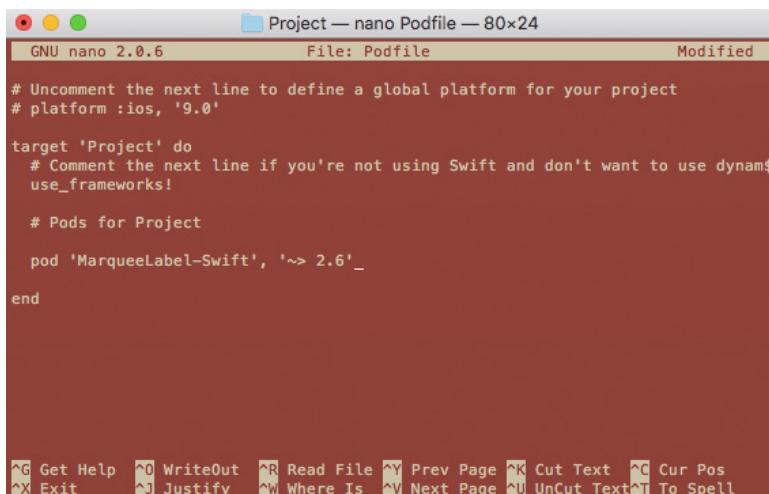
The screenshot shows a terminal window titled "Project — nano Podfile — 80x24". The title bar also displays "GNU nano 2.0.6" and "File: Podfile". The main area of the editor is currently empty, showing only the file name and the nano version. At the bottom of the screen, there is a menu bar with various keyboard shortcut commands.

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

target 'Project' do
  # Comment the next line if you're not using Swift and don't want to use dynamic frameworks!
  # Pods for Project
end
```

Resim 147 : Nano Uygulaması İle Podfile Dosyası İçeriği

Podfile açıldığında platform ve hedef uygulama gibi bazı parametrelerin yazdığı bir dosya olduğu görülür. Bu dosya içerisinde “do” ve “end” anahtar kelimelerinin yazıldığı satırlar arasına “pod” anahtar kelimesi ile harici kütüphanelerin yalnızca isimleri写ざらk dosya kaydedilir ve kapatılır.



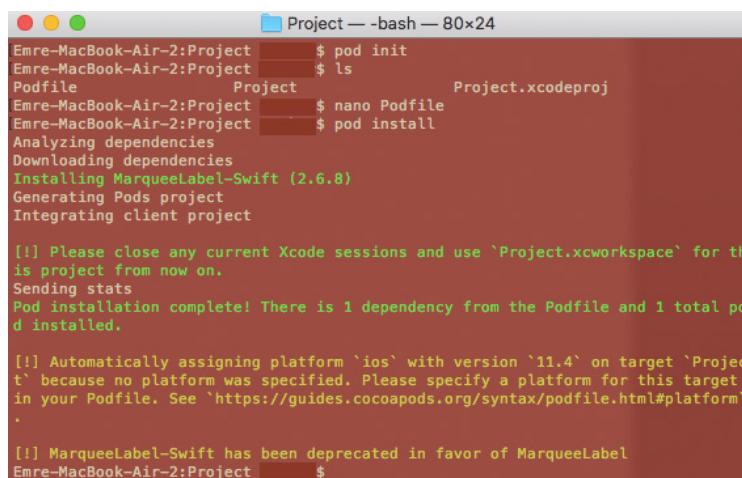
The screenshot shows a terminal window titled "Project — nano Podfile — 80x24". The title bar also displays "GNU nano 2.0.6" and "File: Podfile". The main area of the editor now contains code, indicating that changes have been made. The status bar at the top right shows "Modified". At the bottom of the screen, there is a menu bar with various keyboard shortcut commands.

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

target 'Project' do
  # Comment the next line if you're not using Swift and don't want to use dynamic frameworks!
  # Pods for Project
  pod 'MarqueeLabel-Swift', '~> 2.6'
end
```

Resim 148 : Harici Kütüphane Eklenmiş Bir Podfile Dosyası İçeriği

Projeye dahil edilmek istenen harici kütüphaneler podfile üzerinde belirlendikten sonra, XCode kapatılır ve yine terminal uygulaması üzerinde proje dizininde yer alırken “**pod install**” komutu çalıştırılarak podfile üzerinde belirlenen tüm harici kütüphanelerin projeye kurulumu sağlanır. Kurulum sırasında XCode kapalı olmalıdır.



```

Project — bash — 80x24
Emre-MacBook-Air-2:Project $ pod init
Emre-MacBook-Air-2:Project $ ls
Podfile Project Project.xcodeproj
Emre-MacBook-Air-2:Project $ nano Podfile
Emre-MacBook-Air-2:Project $ pod install
Analyzing dependencies
Downloading dependencies
Installing Marqueelabel-Swift (2.6.8)
Generating Pods project
Integrating client project

[!] Please close any current Xcode sessions and use `Project.xcworkspace` for th
is project from now on.
Sending stats
Pod installation complete! There is 1 dependency from the Podfile and 1 total po
d installed.

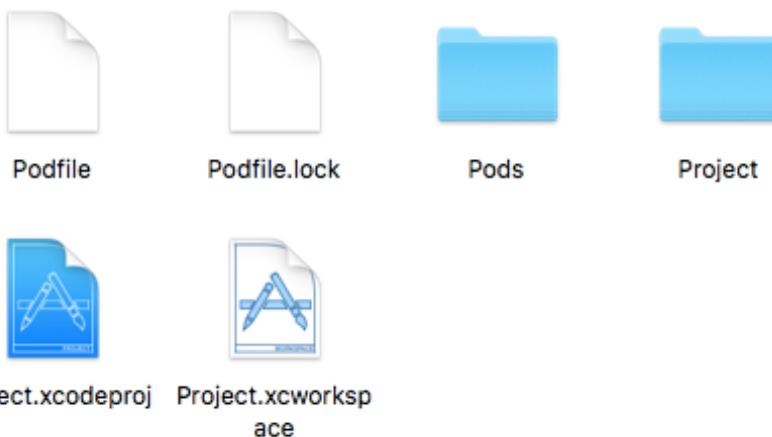
[!] Automatically assigning platform 'ios' with version '11.4' on target 'Projec
t' because no platform was specified. Please specify a platform for this target
in your Podfile. See 'https://guides.cocoapods.org/syntax/podfile.html#platform'.

[!] Marqueelabel-Swift has been deprecated in favor of MarqueeLabel
Emre-MacBook-Air-2:Project $ 

```

Resim 149 : Terminal Üzerinde pod install Komutunun Çalıştırılması

Tüm terminal çıktılarının sona ermesinden sonra, proje dizininde oluşan *workspace* uzantılı dosyanın xcode üzerinde açılması ile harici kütüphanelerin dahil edildiği proje açılabilir ve geliştirilmeye devam edilebilir.



Resim 150 : Yeni Proje Dizini ve Proje Workspace Dosyası

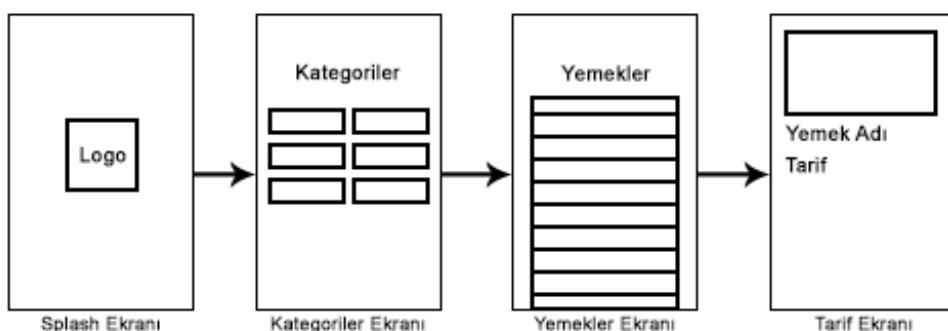
33. Örnek Uygulama – Yemek Tarifleri Uygulaması

a. Hazırlık

Her projede olduğu gibi bir uygulama geliştirme projesine de başlanmadan önce bazı temel hazırlık süreçleri geçirilmelidir. Bu hazırlık süreci projenin geliştirilme aşamasında geliştiricinin geliştirme dışında başka bir şeyi düşünmemesine ve berrak biçimde kod geliştirme sürecine adapte olmasına olanak sağlar.

Hazırlık süreci projenin geliştirme süresi boyunca tasarımsal, yazılımsal ya da algoritmik anlamda neye ihtiyaç olacağına ilişkin herşeyin ortaya konmasını sağlar. Ayrıca uygulama içeriğinin de planlanması için fırsat yaratır.

Ayrıca *paper prototyping* veya *digital prototyping* işlemlerinin uygulanarak arayüzlerin hazırlanmadan önce planlanması proje geliştirilmesine hız katmaktadır.



Resim 151 : Arayüz Prototipi

Hazırlanacak olan yemek tarifleri uygulaması ile seçilen kategoriler doğrultusunda bir yemek listesinin görüntülenmesi ve yine seçilen yemek doğrultusunda da o yemeğin tarifine erişilmesine ilişkin bir yapı hedeflenmiştir.

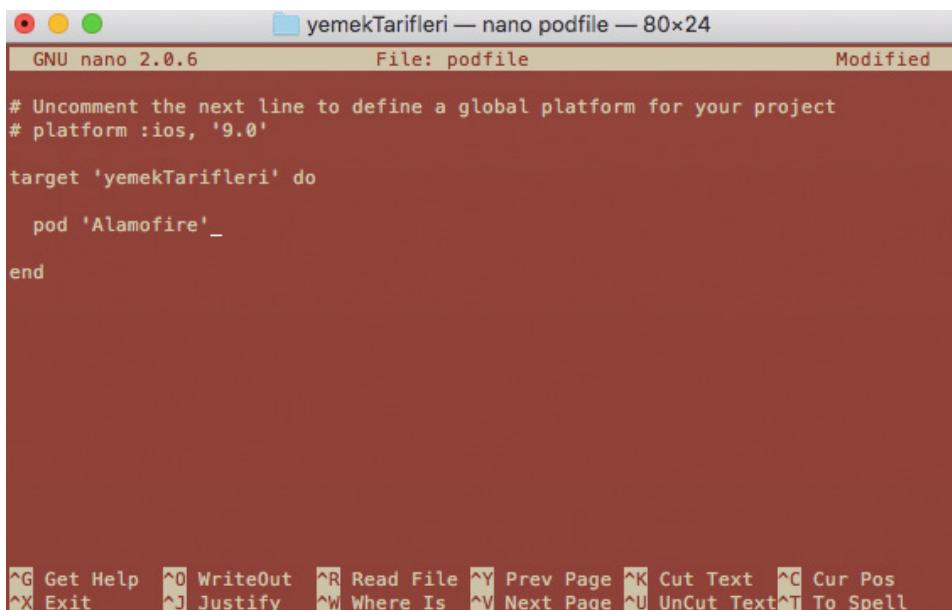
Tüm kategoriler ve tarifler bir url üzerinde bulunan web servis aracılığı ile alınacaktır. Web servis json veri yapısı ile isteklere cevap vermektedir ve GET metodu ile çalışmaktadır.

Web servisle iletişime geçecek bir uygulama olacağı için Cocoapods yardımı ile başarılı bir network kütüphanesi olan Alamofire kütüphanesinin kullanılması geliştirme süremizi kısaltacağı gibi performansımızda artıracaktır.

b. Projenin Oluşturulması ve Tasarımın Hazırlanması

XCode üzerinde normal bir “Single View Application” projesi oluşturulduktan sonra XCode kapatılarak cocoapods aracılığı ile *Alamofire* kütüphanesi projeye dahil edilir. *Alamofire* webservis isteklerinde kullanılacak olan swift kütüphanesidir.

Bunun için terminal üzerinde pod init komutu çalıştırıldıktan sonra, proje dizininde oluşan podfile içerisinde *pod “Alamofire”* satırının eklenmesi gerekmektedir.



```
yemekTarifleri — nano podfile — 80x24
GNU nano 2.0.6          File: podfile          Modified

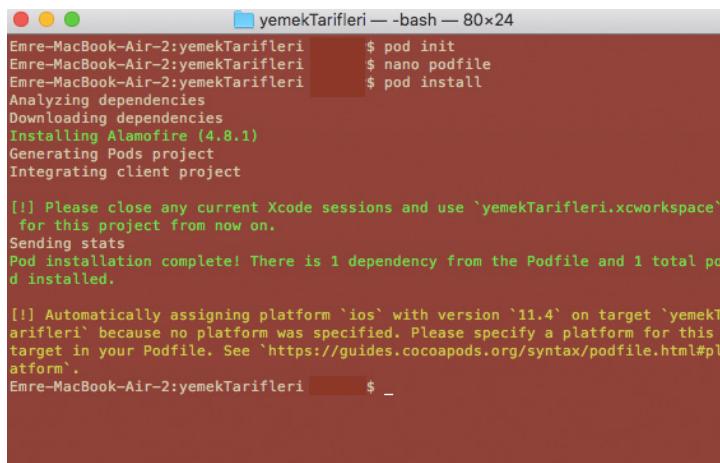
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'yemekTarifleri' do
  pod 'Alamofire'
end

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Resim 152 : Podfile Dosyasına Alamofire Kütüphanesinin Eklenmesi

Podfile dosyasına gerekli parametrelerin girilmesinin ardından yine terminal üzerinde proje dizini içerisindeyken “pod install” komutu çalıştırılır ve harici kütüphanelerin dahili sağlanır.



```

yemekTarifleri — -bash — 80x24
Emre-MacBook-Air-2:yemekTarifleri      $ pod init
Emre-MacBook-Air-2:yemekTarifleri      $ nano podfile
Emre-MacBook-Air-2:yemekTarifleri      $ pod install
Analyzing dependencies
Downloading dependencies
Installing Alamofire (4.8.1)
Generating Pods project
Integrating client project

[!] Please close any current Xcode sessions and use `yemekTarifleri.xcworkspace` for this project from now on.
Sending stats
Pod installation complete! There is 1 dependency from the Podfile and 1 total pod installed.

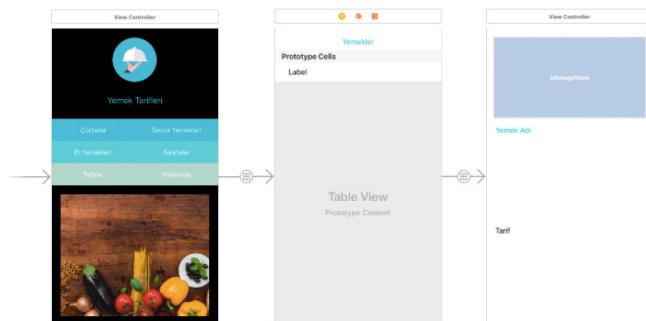
[!] Automatically assigning platform `ios` with version `11.4` on target `yemekTarifleri` because no platform was specified. Please specify a platform for this target in your Podfile. See `https://guides.cocoapods.org/syntax/podfile.html#platform`.
Emre-MacBook-Air-2:yemekTarifleri      $ _

```

Resim 153 : Pod Install Komutunun Çalıştırılması

Install komutunun işlemlerinin sona ermesi ile proje dizininde oluşan yeni dosyalardan workspace Xcode üzerinde açılarak tasarıma ilişkin çalışmalarla başlanabilir.

Proje navigatörü üzerinden ana dizin içerisindeki *main.storyboard*’a geçiş yapılarak daha önce prototipte belirlenen tasarıma ilişkin gerekli bütün elementler viewcontroller ekranlarına eklenmelidir.

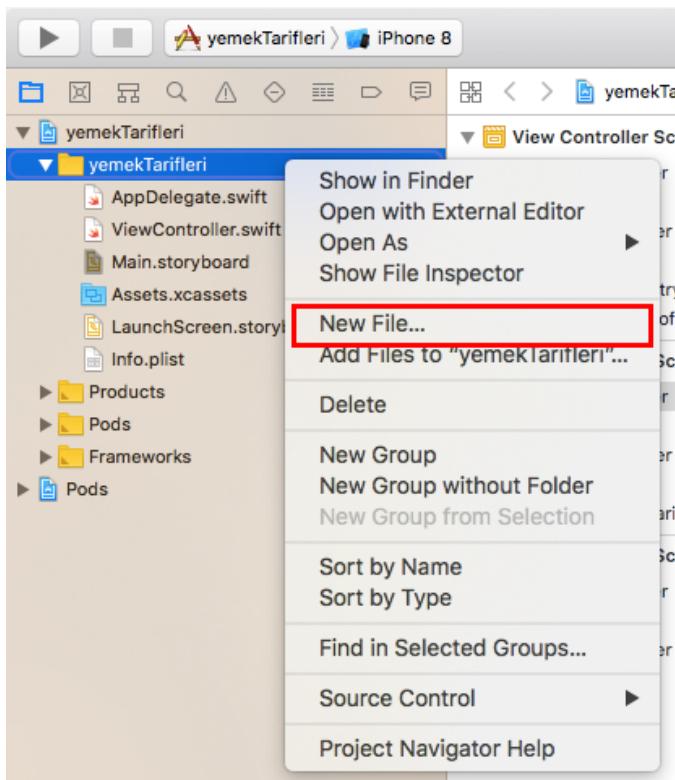


Resim 154 : Storyboard Görünümü

Eklenen viewcontroller'lar arası *segue* bağlantılarında kodlama sırasında kullanılmak üzere eklenmelidir. *Segue* bağlantıları oluşturulduktan sonra bu *segue* bağlantılarına erişim swift kodu aracılığı ile erişim sağlamak için kimliklendirmeleri yapılmalıdır.

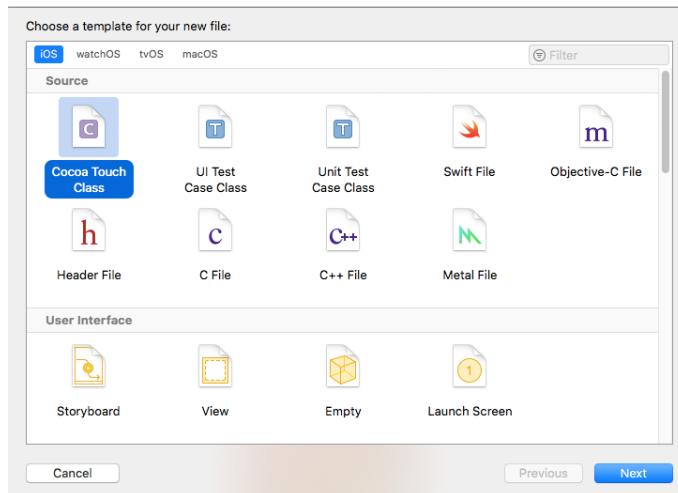
Kodlama sürecine geçmeden önce son olarak tüm viewcontroller ekranları için bir swift dosyası oluşturulmalıdır. Bu swift dosyaları oluştururken viewcontroller sınıfından kalıtım alınmalıdır.

Dosya oluşturmak için proje navigatörü üzerinden proje dizinine mouse üzerinden sağ tıklanarak açılan menüden “New File” butonu tıklanmalıdır.



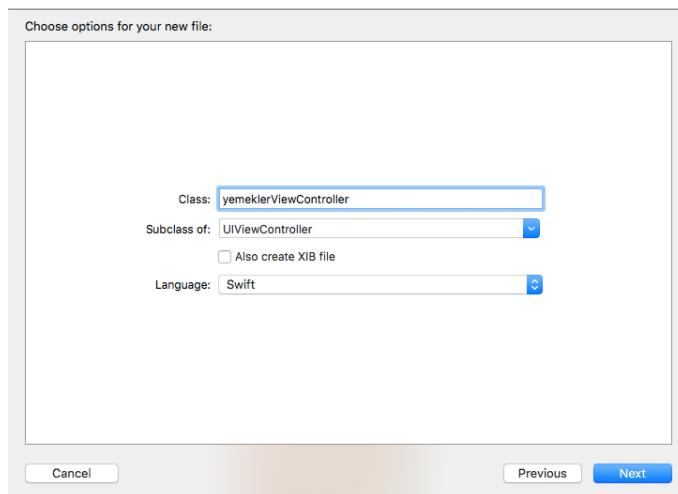
Resim 155 : Yeni Swift Dosyası Oluşturmak İçin New File Butonu

Yeni dosya butonu tıklandıktan sonra proje oluşturulurken karşılaşılan, tema penceresi ile karşılaşılır. Bu pencere üzerinde oluşturulacak olan dosya için “Cocoa Touch Class” isimli seçenek ile ileri butonu tıklanmalıdır.



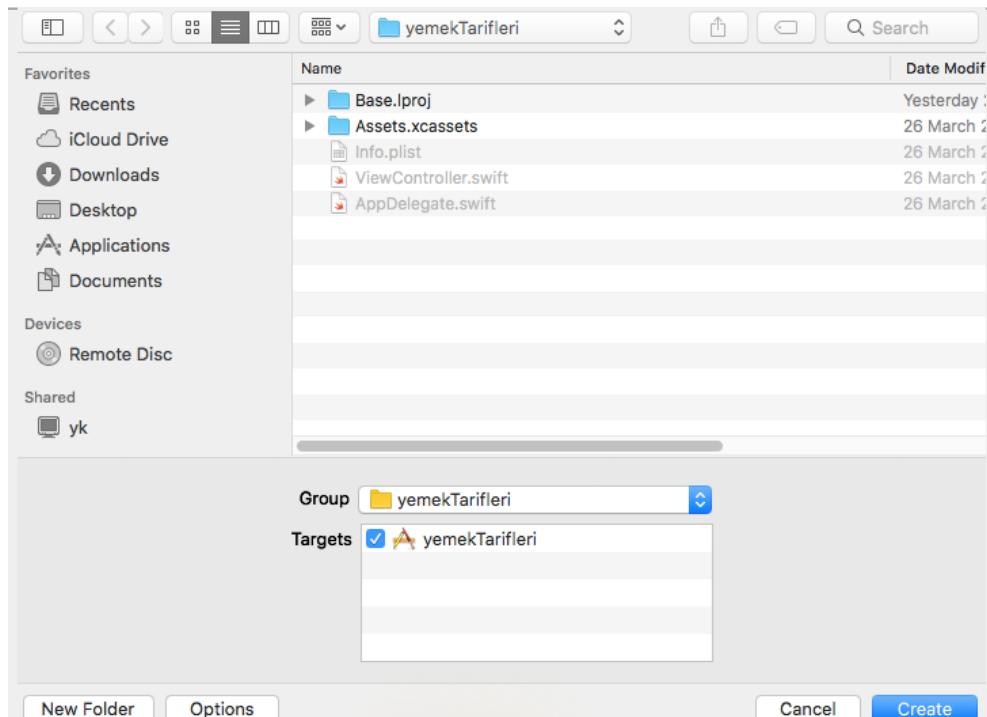
Resim 156 : Cocoa Touch Class Temalı Bir Swift Dosyası Oluşturma

Oluşturulacak olan swift dosyasının tema seçimi yapıldıktan sonra, oluşturulacak olan dosyaaya ilişkin bazı ayarlamaların yapıldığı bir pencereye geçiş yapılır. Bu pencere aracılığı ile oluşturulacak olan dosyanın içerisinde yer alacak olan sınıfın ismi belirlenir. Ancak daha da önemlisi “subclass of” seçeneği altında bu sınıfın hangi sınıfından kalıtım alacağıının belirtilmesidir. Viewcontroller için swift dosyası oluşturulduğundan, alınacak olan kalıtımında viewcontroller olmalıdır.



Resim 157 : Cocoa Touch Class Özelliklerinin Belirlenmesi

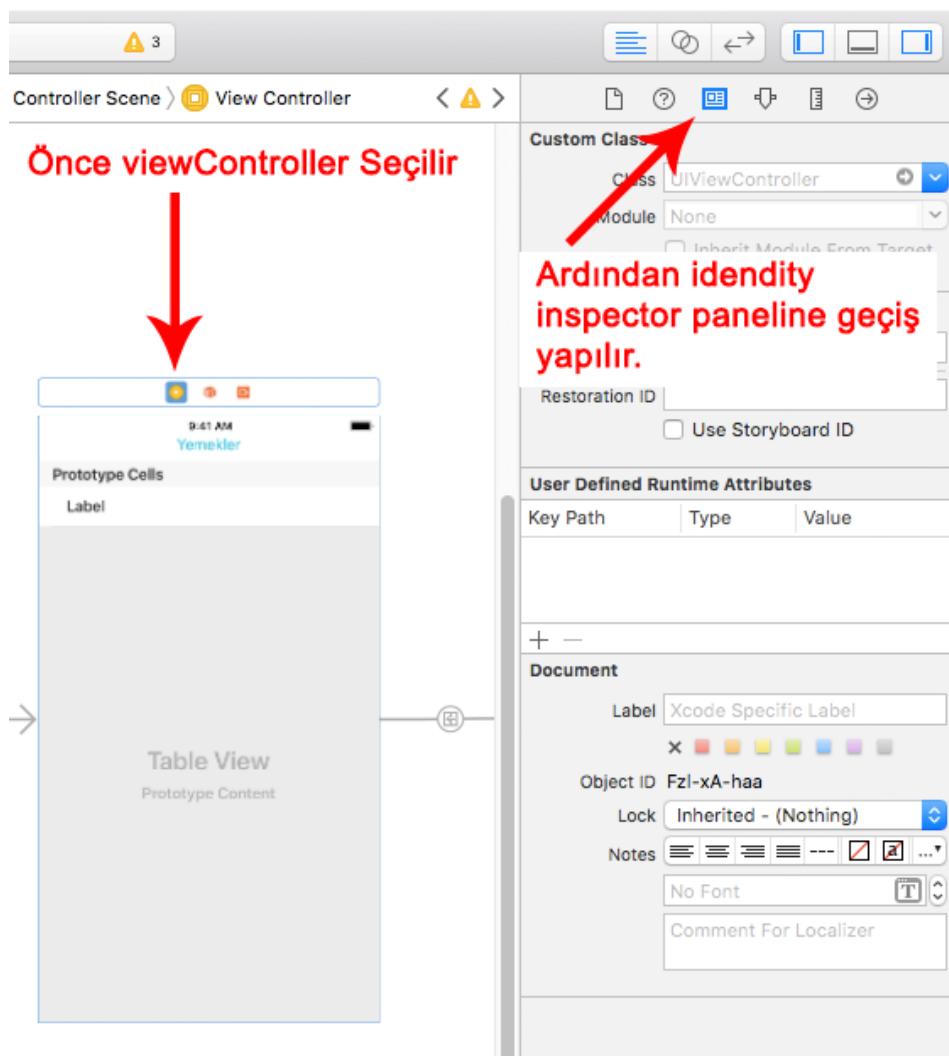
Tüm seçenekler tamamlandıktan sonra, ileri butonu ile devam edilir ve bu defa oluşturulacak olan dosyanın kayıt edileceği yerin belirlendiği ekran ile karşılaşılır. Bu bölümde “create” butonu aracılığı ile dosya oluşturma işlemi tamamlanır.



Resim 158 : Swift Dosyası Kayıt Ekranı

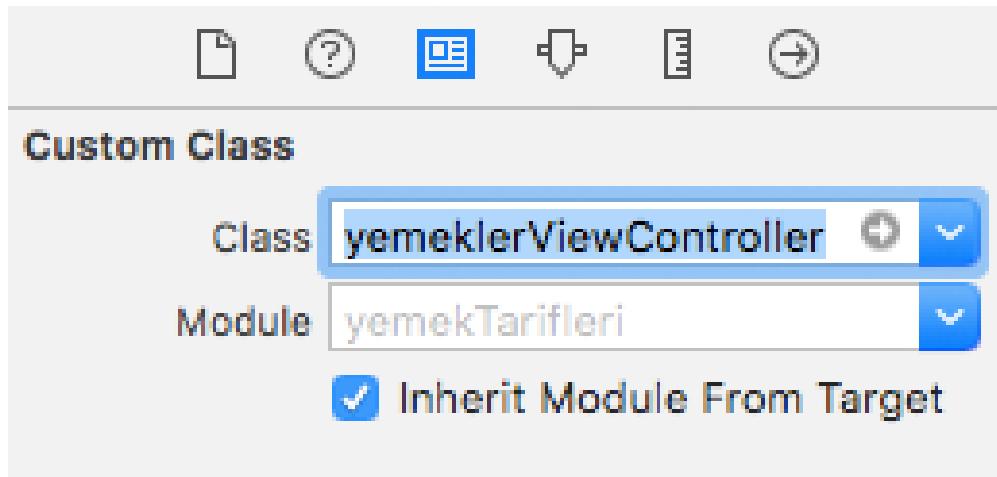
Aynı işlemler, diğer eklenen storyboarddaki viewcontroller ekranları içinde tekrar edilir. Böylelikle her storyboard ekranı için ayrı ayrı swift dosyaları oluşturulur.

Ayrı ayrı oluşturulan bu swift dosyaları bu aşamadan sonra, storyboard üzerindeki viewcontroller ekranlarına bağlanmalıdır. Bunun için *main.Storyboard*'a geçiş yapılır ve ilgili viewcontroller seçilir, sağ kısımda yer alan özellikler paneli üzerinde yer alan “identity inspector” ikonuna tıklanarak ilgili panele geçiş yapılır.



Resim 159 : Identity Inspector Paneli

Identity Inspector paneli storyboard da yer alan viewcontroller ekranlarına swift dosyaları bağlamak için kullanılır. Bu panelin üst kısmında yer alan “Custom Class” başlığı altında yer alan “Class” özelliğinin karşısındaki metin düzenleme alanı yeni storyboard’ a eklenen yeni viewcontroller ekranları için boş olmalıdır. Bu alana oluşturulan swift dosyalarının isimleri yazılarak storyboard da yer alan viewcontroller lara oluşturulan swift dosyaları bağlanmalıdır.



*Resim 160 : Yeni Oluşturulan Swift Dosyalarının StoryBoard'a
Eklenen ViewController lara Bağlanması*

Aynı bağlama işlemleri yeni eklenen tüm storyboard viewcontroller ekranları için yapılmalıdır. Burada dikkat edilmesi gereken konu, eğer swift dosyası oluşturulurken “subclass of” seçeneği *ViewController* olarak belirlenmediyse bağlama işleminin başaramayacağıdır.

Bu aşamaya kadar gelindiğten sonra artık kaba hatları ile tasarım süreci tamamlanmış ve kodlama için gerekli tüm swift dosyaları oluşturulmuştur ve kodlama sürecine geçiş yapılmabilir.

c. Kodlama Süreci

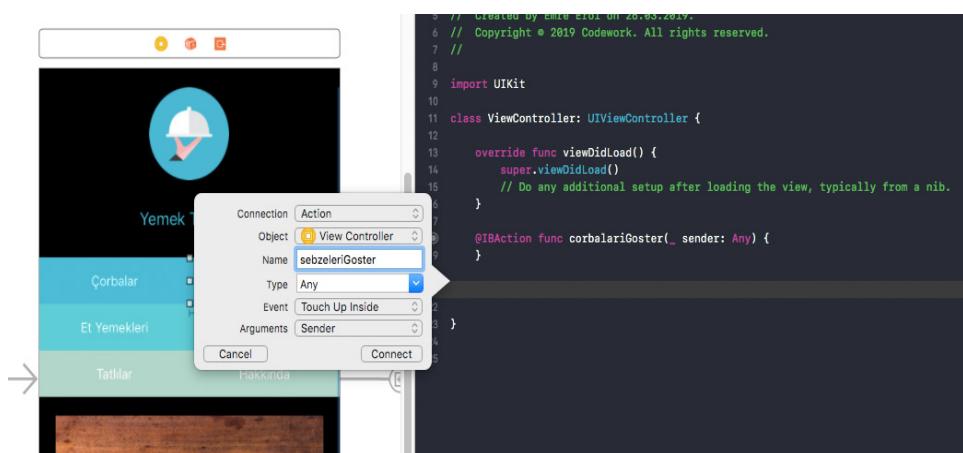
Proje de kullanılan web servis “GET” metodu ile çalışmaktadır. Yapılan isteklere ve bu isteklerde yer alan parametrelere göre JSON veri yapısı olarak cevap vermektedir. Bu isteklerde yer alması gereken parametreler aşağıdaki tabloda gösterilmiştir.

Parametreler	Dönecek Cevaplar
sec=yemekler ID={YEMEK TÜRÜ ID}	Yemek türüne göre o türde kayıtlı olan tüm yemeklerin listesi JSON veri yapısı ile döndürülür. JSON veri yapısında yemeklerin adı ve ID değerleri bulunur. Eğer ID değeri 1 olarak gönderilirse Çorbalar türündeki tüm yemekler, 2 olarak gönderilirse Sebze yemekleri türündeki tüm yemekler cevap olarak döndürülmektedir.
sec=yemek ID={YEMEK ID}	Bir yemeğin tarifine ulaşmak istendiğinde bu parametreler kullanılmalıdır. Parantezler arasında tarifine ulaşmak istenen yemeğin ID değeri yazılır. JSON veri yapısı ile yemeğe ilişkin tüm bilgiler cevap olarak döndürülür.

Tablo 5 : Web Servis İstek Parametreleri

Hazırlanan uygulamada ilk viewcontroller’da yer alan butonlar aracılığı ile yemek türü kullanıcı tarafından seçilmekte ve sonraki ekrana geçiş yapılmaktadır. Geçiş yapıldığında ise yapılan seçime göre yemek listesi gösterilmektedir. Bu yüzden kullanıcının hangi buton ile etkileşime geçtiği takip edilmeli ve diğer ekrana geçiş yapılırken seçimi diğer ekrana bildirilmelidir.

Bunun için öncelikle butonlar IBAction bağlantısı ile viewcontroller swift dosyasına bağlanmalıdır.



Resim 161 : Butonların IBAction Bağlantısı

Kullanıcı seçiminin hafızada tutulması için bir değişken oluşturulmalıdır. IBAction bağlantısı yaptığımız viewcontroller swift dosyası içerisinde yer almalı ve hangi butona basılırsa butonun durumuna göre taşıyacağı veri değişimelidir. Eğer kullanıcı çorbalar isimli butona basarsa değişken 1' e eşitlenmeli, sebze yemeklerine basarsa 2' ye eşitlenmelidir. Eşitlenme işleminin hemen arkasından *performSegue* metodu ile diğer sayfaya geçiş işlemi başlatılmalıdır.

```

 9 import UIKit
10
11 class ViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15         // Do any additional setup after loading the view, typically
16         // nib.
17     }
18
19     var yemekTürüSecimi = 0
20
21     @IBAction func corbalariGoster(_ sender: Any) {
22         yemekTürüSecimi = 1
23         performSegue(withIdentifier: "gofirst", sender: self)
24     }
25
26     @IBAction func sebzeleriGoster(_ sender: Any) {
27         yemekTürüSecimi = 2
28         performSegue(withIdentifier: "gofirst", sender: self)
29     }
30
31 }
```

Resim 162 : Kullanıcı Seçiminin Hafızaya Alınması ve Sonraki Sayfaya Geçiş İşlemi

Kullanıcı seçimini yapıp butona bastıktan sonra diğer sayfaya geçiş işlemi başlayacaktır, ancak diğer sayfaya kullanıcının hangi butona bastığı bildirilmelidir. Bunun için bir değişken oluşturulması, bu değişkenin kullanıcı seçimine göre değer taşıması ve bu değişkenin diğer sayfaya gönderilmesi gerekmektedir. Diğer sayfaya göndermek için *prepare* isimli bir metot override edilerek kullanılmalıdır. Bu metot aracılığı ile geçiş yapılmak istenen viewcontroller ekranının swift dosyası içerisinde yer alan değişkenlere erişim sağlanarak onlara değer ataması

yapılabilmektedir. Bu değer atamalarının yapılabilmesi için geçiş yapılmak istenen viewController ekranının swift dosyasına geçiş yapılmalı ve o dosya içerisinde değer ataması yapılacak değişken belirlenmelidir.

```

8
9 import UIKit
10
11 class yemeklerViewController: UIViewController {
12
13     var gelecekSecim:Int!
14
15     override func viewDidLoad() {
16         super.viewDidLoad()
17
18         // Do any additional setup after loading the view.
19     }
20
21 }
```

Resim 163 : Geçiş Yapılmak İstenen ViewController Ekranının Swift Dosyası Üzerinde Değişken Tanımlaması

Geçiş yapılmak istenen viewController ekranının swift dosyasında kullanıcı seçiminin tutulacağı değişken tanımlaması yapıldıktan sonra ilk swift dosyasına geri dönülerek *prepare* fonksiyonu üzerinden bu değişkene atama yapılmalıdır.

```

30     override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
31         let gidilecekViewController = segue.destination as! yemeklerViewController
32         gidilecekViewController.gelecekSecim = yemekTürüSecimi
33     }
34 }
```

Resim 164 : Prepare Fonksiyonu İle Geçiş Yapılmak İstenen ViewController Ekranının Swift Dosyası Üzerindeki Değişkene Veri Ataması Yapılması

prepare fonksiyonu parametrelerinden olan *segue*, yapılmak olası işleme dair tüm verileri içerisinde barındırmaktadır. Bu verilerden biriside geçiş yapılmak istenen viewController sınıfının kendisidir. Bu parametrenin alt özelliği olan *destination* ile bu viewController' a erişim sağlanabilir. Sonrasında ise erişim sağlanan viewController ile hedeflenen swift dosyasındaki değişkene atama yapılabilir.(Bknz. Resim : 161)

Yemeklerin listelenmesi için ikinci viewcontroller üzerinde artık işlem yapılabilir. Bunun için *tableview* barındıran viewcontroller ekranına ait swift dosyasına geçilmelidir.

Anlatılmaya çalışılan örnekte “Alamofire” kütüphanesi ile bir fonksiyon aracılığı ile “<http://codeworktest.com/iosKitap/api.php>” adresinde bulunan webservise istekte bulunularak yemek listesi alınmıştır.(Bnkz. Resim 163) Web servise istekte bulunabilmek için web servis adresinin alan adının *Info.plist* dosyasında bildirilmesi gerekmektedir. Bunun için sol bölümde yer alan proje navigatöründen *Info.plist* isimli dosyaya sağ tıklanarak “open as” isimli menü ile “Source Code” butonuna tıklanır. Böylelikle *Info.Plist* dosyasının XML düzenleme görünümüne ulaşılarak en sondan ikinci satırda yer alan “</dict>” etiketinden önce aşağıdaki parametreler eklenmelidir. (Resim : 162)

```

45      <key>NSAppTransportSecurity</key>
46      <dict>
47          <key>NSAllowsArbitraryLoads</key>
48          <true/>
49          <key>NSExceptionDomains</key>
50          <dict>
51              <key>codeworktest.com</key>
52              <dict>
53                  <key>NSExceptionAllowsInsecureHTTPLoads</key>
54                  <true/>
55                  <key>NSIncludesSubdomains</key>
56                  <true/>
57              </dict>
58          </dict>
59      </dict>
```

Resim 165 : Info.plist Dosyasına Api Adresinin Eklenmesi

Yemek listesinin görüntüleneceği viewcontroller ekranının swift dosyasına geçiş yapılarak öncelikle bu dosyaya “Alamofire” kütüphanesi import edilmelidir. Sonrasında sınıf içerisinde bir fonksiyon oluşturularak, bu fonksiyon içerisinde “Alamofire” isteği hazırlanabilir.

Bu sınıf içerisinde ayrıca, istek yapılacak web servis in adresi olan “<http://codeworktest.com/iosKitap/api.php?sec=yemekler&ID=>” bir değişkene alınarak, kullanıcının seçimi ile bu sınıf içerisinde ataması yapılan değişken ile birleştirilmelidir. Fonksiyonun *viewDidLoad* metodu içerisinde çağrılması ile web servis isteği tamamlanmış olur.

```

6
7 import UIKit
8 import Alamofire
9
10
11 class yemeklerViewController: UIViewController {
12
13     var gelecekSecim:Int!
14     var apiAdres = "http://codeworktest.com/iosKitap/api.php?sec=yemekler&ID="
15
16     |
17     override func viewDidLoad() {
18         super.viewDidLoad()
19         apiAdres += String(gelecekSecim)
20         yemekListesiAl()
21     }
22
23     func yemekListesiAl(){
24
25         Alamofire.request(apiAdres).responseJSON { (cevap) in
26             if let json = cevap.result.value as? [String:Any]
27             {
28
29             }
30         }
31     }
32 }
33
34 }
```

Resim 166 : Alamofire İle Webservis İsteği

Web servis isteğinden sonra dönen cevap yemekleri taşıyacağı için, bir dizi olduğu varsayılmalıdır. Dolayısı ile bu diziyi hafızada tutmak için boş bir dizi değişken tanımlanmalı ve dönen dizi cevabı bu hafızaya alınmalıdır.

```

23
24 var yemeklerDizisi = NSArray()
25
26 func yemekListesiAl(){
27
28     Alamofire.request(apiAdres).responseJSON { (cevap) in
29         if let json = cevap.result.value as? [String:Any]
30         {
31             if let error = json["error"] as? Bool {
32                 if error != true {
33                     if let yemekler = json["yemekler"] as? NSArray {
34                         self.yemeklerDizisi = yemekler
35                     }
36                 }
37             }
38         }
39     }
40 }
41
42 }
```

Resim 167 : Dönen Web Servis Cevabının Diziye Alınması

Artık dizi değişken tüm yemek isimlerini taşıdığından bu dizi değişkenin içindeki veriler *TableView* elementi içerisinde gösterilebilir. Bunun için *UITableViewDelegate* ve *UITableViewDataSource* protokollerinden kalıtım alınarak metodlar dahil edilmeli, storyboard üzerindeki *tableView* için bir *IBOutlet* bağlantısı oluşturulmalıdır.

```

63      @IBOutlet weak var tableView: UITableView!
64
65      func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
66          code
67      }
68
69      func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
70          code
71      }

```

Resim 168 : *TableView* *IBOutlet* Bağlantısı ve Protokol Metotları

Dahil edilen metodlardan *numberOfRowsInSection* kod bloğu içerisinde dizinin eleman sayısı return anahtar kelimesi ile döndürülmelidir. *cellForRowAt* fonksiyonunda ise bir sabit oluşturularak, *tableview* içerisinde daha önceden oluşturulan protatip hücreye anahtar kelimesi ile erişim sağlanır.

Hücre içerisinde bulunan *label* elementi de *viewWithTag* fonksiyonu ile bir değişkene atanmalıdır. Ardından, dizi değişkene alınan yemekler listesinden veriler bu label elementinin text özelliğine eşitlenmelidir. Dizi değişken içerisinde yer alan verilere *indexPath.row* parametresi ile erişim sağlanabilir.

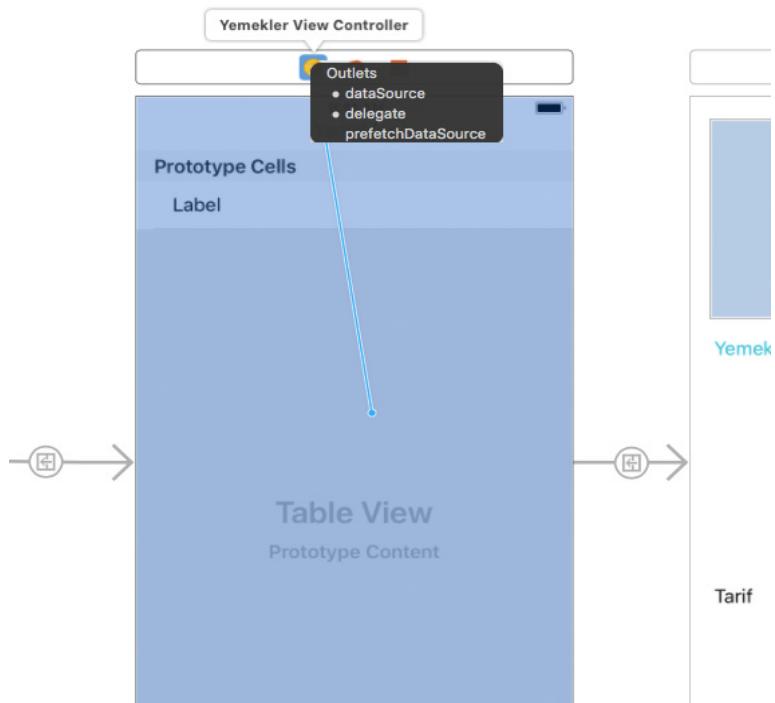
```

65      @IBOutlet weak var tableView: UITableView!
66
67      func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
68          return yemeklerDizisi.count
69      }
70
71      func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
72          let cell = tableView.dequeueReusableCell(withIdentifier: "hucre")
73          let label = cell?.viewWithTag(1) as! UILabel
74          let yemekJsonObj = yemeklerDizisi[indexPath.row] as! [String:Any]
75          label.text = yemekJsonObj["YEMEKADI"] as! String
76          // Treating a forced downcast to 'String' as optional
77          return cell!
78      }
79
80
81

```

Resim 169 : Protokol Metotları İle *TableView* Satırlarının Çoğaltılması

Son olarak *tableViewDelegate* ve *tableViewDatasource* işlemleri storyboard üzerinden yapılmalıdır. Ayrıca web servis isteğin gecikme ihtimaline karşı, "Alamofire" isteğin sona erdiği ve dizide eşitlendiği satırdan hemen sonra, *tableview* için *reloadData* fonksiyonu çalıştırılarak, *tableview* satırları çoğaltılır.



Resim 170 : TableView Nesnesinin DataSource ve Delegate İşlemi

Artık bu aşamada *tableview* çalışır vaziyettedir ve kullanıcının yemek seçimi ardından yemek tarifinin görüntüleneceği sayfaya geçiş yapılmalıdır. Tableview elementinin *didSelectRowAtIndexPath* isimli fonksiyonu kullanıcı bir *tableview* satırını seçtiğinde, seçilen *tableview* satırının satır numarasını döndürmektedir. Dolayısı ile bu satır numarası ile dizinin içerisinde yer alan yemeğin ID değerine erişim sağlanarak, tariflerin görüntüleneceği diğer sayfaya bu değer gönderilebilir.

Bu işlem için ID yi hafızada tutacak bir değişkene oluşturulmalı ve *didSelectRowAtIndexPath* fonksiyonunda bu değişkene atama yapılmalıdır. Ayrıca bu atamanın hemen sonrasında yeni viewcontroller ekranına geçiş için *performSegue* fonksiyonu çağrılmalıdır. Böylelikle kullanıcı bir yemeği seçtiğinde o yemeğin tarifinin görüntülenebilmesi için ilgili ekrana geçiş yapabilir.

```

58     var secilenYemekID = 0
59
60    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
61        let yemekJsonObj = yemeklerDizisi[indexPath.row] as! [String:Any]
62        secilenYemekID = yemekJsonObj["ID"] as! Int
63        performSegue(withIdentifier: "tarif", sender: self)
64    }
65

```

Resim 171 : Kullanıcının TableView Etkileşiminin Yakalanması

Kullanıcının yemek seçimine ilişkin yemek ID si bir değişkene alındıktan sonra, bu değişkenin taşıdığı veri tariflerin görüntülenmesiyle ilgili işlemlerin yapılabacağı viewcontroller ekranına *prepare* fonksiyonu ile gönderilmelidir.

```

66    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
67        let tariflerEkrani = segue.destination as! tarifViewController
68        tariflerEkrani.secilenYemek = secilenYemekID
69    }

```

Resim 172 : Seçilen Yemek ID sinin Tarifler Ekranına Gönderimi

Son viewcontroller ekranının swift dosyasına da “Alamofire” kütüphanesi import edilmeli ve ardından api adresi değişkene alınmalıdır. Bu swift dosyasına gönderilen yemek id si ile web servis üzerinde istek yapılarak, tarifler alınmalıdır. Bu kısımda bir önceki ekranın swift dosyasında yapıldığı gibi, bir fonksiyon içerisinde hazırlanabilir.

```

9 import UIKit
10 import Alamofire
11
12 class tarifViewController: UIViewController {
13
14     var secilenYemek:Int!
15     var apiAdres = "http://codeworktest.com/iosKitap/api.php?sec=yemekGetir&ID="
16
17     override func viewDidLoad() {
18         super.viewDidLoad()
19         apiAdres += String(secilenYemek)
20         yemekDetayGetir()
21     }
22
23     func yemekDetayGetir(){
24
25         Alamofire.request(apiAdres).responseJSON { (cevap) in
26
27         }
28
29     }
30 }

```

Resim 173 : Yemek Tariflerinin Web Servisden Çekilmesi

Tariflerin ve yemek adının görüntüleneceği *label* elementlerinin IBOOutlet bağlantısının yapılmasıının ardından, dönen web servis cevabının içerisinde alınan tarif ve yemek adı ilgili *label* elementlerinin *text* özelliğine eşitlenerek, arayüzde görüntülenmesi sağlanır. Böylelikle kullanıcı yemek tarifine erişmiş ve görüntüler pozisyonadır.

```

22
@IBOutlet weak var yemekadilbl: UILabel!
@IBOutlet weak var tariflbl: UILabel!
25
26 func yemekDetayGetir(){
27
28     Alamofire.request(apiAdres).responseJSON { (cevap) in
29         if let json = cevap.result.value as? [String:Any] {
30             if let error = json["error"] as? Bool {
31                 if !error {
32                     if let yemek = json["yemek"] as? [String:String] {
33                         self.yemekadilbl.text = yemek["YEMEKADI"]
34                         self.tariflbl.text = yemek["TARIF"]
35                     }
36                 }
37             }
38         }
39     }
40 }
41
42

```

Resim 174 : Seçilen Yemek Tarifinin Web Servisten Alınarak Arayüzde Gösterimi



Kare Kod 22 : <https://github.com/ios-kitap/yemek-tarifleri>

Örnek olarak hazırlanan uygulamaya ilişkin projeyi, karekodu kullanarak erișeceğiniz linkten bilgisayarınıza indirebilirsiniz.

34. Resimler Dizini

RESİM 1 : STORYBOARD GENEL GÖRÜNÜM	14
RESİM 2 : STORYBOARD SAYFA GEÇİŞLERİ	15
RESİM 3 : ASSET KATALOĞU	16
RESİM 4 : ASSET KATALOĞU MENÜSÜ	17
RESİM 5 : FARKLI ÖLÇEKLER	18
RESİM 6 : SEGUE BAĞLANTILARI	19
RESİM 7 : SEGUE BAĞLANTILARI 2	20
RESİM 8 : AUTOLAYOUT UYGULANMAMIŞ BİR TASARIMIN DİĞER EKRAN TIPLERİNDEKİ YERLEŞİMLERİ	21
RESİM 9 : ADDNEW CONSTRAINTS POPUP PENCERESİ	22
RESİM 10 : SABİT AÇIKLAMALARI	22
RESİM 11 : AUTOLAYOUT UYGULANMIŞ BİR TASARIMIN DİĞER EKRAN TIPLERİNDEKİ YERLEŞİMLERİ	23
RESİM 12 : İLİŞKİLENDİRİLMİŞ SABİTLERİN UYGULANMASI	24
RESİM 13 : İLİŞKİLENDİRİLMİŞ SABİTLER POPUP PENCERESİ	24
RESİM 14 : SABİT KAVRAMI İÇİN SABİTLENENECEK ÖZELLİKLERİN GÖSTERİMİ	25
RESİM 15 : DEĞİŞKEN TANIMLAMALARINA ÖRNEKLER	30
RESİM 16 : SAYISAL DEĞİŞKENİN METİNSEL DEĞİŞKENE DÖNÜŞÜMÜ	30
RESİM 17 : METİNSEL DEĞİŞKENİN SAYISAL DEĞİŞKENE DÖNÜŞÜMÜ	30
RESİM 18 : DİZİ TANIMLANMASI	33
RESİM 19 : APPEND VE INSERT METOTLARI	33
RESİM 20 : DİZİDEN ELEMAN ÇIKARMA İŞLEMİ	34
RESİM 21 : ÇOK BOYUTLU DİZİ	34
RESİM 22 : DICTIONARY DİZİ TANIMLANMASI	34
RESİM 23 : IF ELSE KONTROL YAPISI	39
RESİM 24 : SWITCH-CASE KONTROL YAPISI	41
RESİM 25 : FOR IN DÖNGÜ YAPISI	43
RESİM 26 : WHILE DÖNGÜSÜ	44
RESİM 27 : REPEAT - WHILE DÖNGÜSÜ	45
RESİM 28 : TEMEL FONKSİYON TANIMLANMASI VE KULLANIMI	47
RESİM 29 : PARAMETRELİ FONKSİYON TANIMLANMASI VE KULLANIMI	48

RESİM 30 : BİRDEN ÇOK PARAMETRE ALAN FONKSİYON TANIMLANMASI VE KULLANIMI	49
RESİM 31 : BİRDEN FAZLA RETURN ANAHTAR KELİMESİNİN KULLANIMI.....	51
RESİM 32 : BİRDEN FAZLA VERİYİ GERİ DÖNDÜREN FONKSİYONLAR.....	52
RESİM 33 : DEĞİŞKEN SAYIDA PARAMETRE ALAN FONKSİYONLARIN TANIMLANMASI VE KULLANIMI.....	53
RESİM 34 : NESTED FONKSİYONLARIN TANIMLANMASI VE KULLANIMI	54
RESİM 35 : COĞRAFI YÖNLERİN ENUMERATION BİÇİMİNDE TANIMLANMASI VE KULLANIMI	56
RESİM 36 : ENUMERATION İLE HATA TANIMLAMALARI.....	57
RESİM 37: FONKSİYON İÇERİSİNDE HATA FIRLATMA.....	58
RESİM 38 : DO-TRY-CATCH BLOĞU İLE FIRLATILAN HATANIN YAKALANMASI.....	58
RESİM 39 : SWIFT DİLİNDE SINIF YAPISI ÖZELLİKLER VE FONKSİYONLAR.....	62
RESİM 40 : BİR SINIF İÇERİSİNDE YAPICI METOT TANIMLANMASI VE KULLANIMI.....	63
RESİM 41 : BİR SINIF İÇERİSİNDE YIKICI METOT TANIMLANMASI VE KULLANIMI	64
RESİM 42 : SINIFLarda KALITIM ÖRNEĞİ	65
RESİM 43 : KAPSÜLLEME VE ERIŞİM KONTROL GRAFİĞİ	67
RESİM 44 : SOYUT SINIF TANIMLAMA ÖRNEĞİ	69
RESİM 45 : PROTOKOLLER	71
RESİM 46 : ÇOK BİÇİMLİLİK ÖRNEĞİ VE OVERRIDE EDİLEN BİR FONKSİYON.....	73
RESİM 47 : STRUCTURE TANIMLANMASI ÖRNEĞİ.....	75
RESİM 48 : STRUCTURE YAPISI TÜRETİMİ, ÖZELLİK VE FONKSİYONLARININ KULLANIMI	76
RESİM 49 : IBOUTLET ADIMLARI 1	79
RESİM 50 : IBOUTLET ADIMLARI 2	79
RESİM 51 : IBOUTLET VE IBACTION BAĞLANTILARI OLUŞTURULMUŞ BİR GÖRÜNÜM	80
RESİM 52 : UIVIEW	82
RESİM 53 : LABEL.....	83
RESİM 54 : BUTTON.....	85

RESİM 55 : TEXTFIELD KULLANIM ÖRNEĞİ.....	87
RESİM 56 : TEXTFIELD	87
RESİM 57 : IMAGEVIEW.....	88
RESİM 58 : SCROLLVIEW	89
RESİM 59 : SWITCH.....	90
RESİM 60 : PROGRESSVIEW	91
RESİM 61 : SEGMENTED CONTROL	92
RESİM 62 : PICKERVIEW	93
RESİM 63 : PICKERVIEW IBOUTLET BAĞLANTISI.....	94
RESİM 64 : UIPICKERVIEWDELEGATE VE UIPICKERVIEWDATASOURCE KALITIMI	95
RESİM 65 : PICKERVIEW SINIF DOSYASINA GEÇİŞ	96
RESİM 66 : UIPICKERVIEWDATASOURCE PROTOKOLÜ VE DAHİL EDİLECEK FONKSİYONLAR	97
RESİM 67 : UIPICKERVIEWDELEGATE PROTOKOLÜ VE DAHİL EDİLECEK FONKSİYONLAR	97
RESİM 68 : VIEWCONTROLLER SINIFINA DAHİL EDİLMİŞ PROTOKOL FONKSİYONLARI VE PICKERVIEW DİZİ DEĞİŞKENİ	98
RESİM 69 : DAHİL EDİLMİŞ FONKSİYONLARIN KULLANIMLARI	99
RESİM 70 : VIEWDIDLLOAD FONKSİYONUNDA PICKERVIEW ELEMENTİNİN KAYNAKLARININ LENMESİ	100
RESİM 71 : PICKERVIEW UYGULAMA ÜSTÜNDE ÇALIŞIRKEN	101
RESİM 72 : DATEPICKER	102
RESİM 73 : DATEPICKER ELEMENTİNİN ÇALIŞTIRILMASINA İLİŞKİN SWIFT KODU	103
RESİM 74 : DATEPICKER ELEMENTİ UYGULAMA ÜSTÜNDE ÇALIŞIRKEN	104
RESİM 75 : TABLEVIEW	105
RESİM 76 : TABLEVIEW ÜZERİNDE PROTATİP HÜCRE OLUŞTURULMASI.....	106
RESİM 77 : TABLEVIEW ELEMENTİNE EKLENMİŞ PROTATİP HÜCRENİN KİMLİKLENDİRİLMESİ	107
RESİM 78 : TABLEVIEW ELEMENTİNİN IBOUTLET BAĞLANTISI.....	108
RESİM 79 : TABLEVIEW SINIF DOSYASINA GEÇİŞ	109
RESİM 80 : UITABLEVIEWDATASOURCE PROTOKOLÜ İÇERİSİNDEN VIEWCONTROLLERA BAĞLI OLAN SWIFT DOSYASINA DAHİL EDİLECEK FONKSİYONLAR	110

RESİM 81 : VIEWCONTROLLER SINIF DOSYASINA DAHİL EDİLMİŞ TABLEVIEW PROTOKOL FONKSİYONLARI	112
RESİM 82 : VIEWDIDLLOAD FONKSİYONUNDA TABLEVIEW ELEMENTİNİN KAYNAKLARININ BELİRLENMESİ	112
RESİM 83 : TABLEVIEW ELEMENTİ UYGULAMA ÜZERİNDE ÇALIŞIRKEN	113
RESİM 84 : TEXTFIELD ELEMENTİNİN BOŞ OLUP OLMADIĞINI KONTROL EDEN BİR EXTENSION	115
RESİM 85 : VIEW NESNESİ VE SABİTLER	118
RESİM 86 : SABİT İÇİN OLUŞTURULAN IBOUTLET BAĞLANTISI.....	119
RESİM 87 : TÜM SABİTLERİN SWIFT DOSyasında BULUNAN IBOUTLET BAĞLANTILARI	119
RESİM 88 : UIVIEW ANIMATE FONKSİYONUNUN KULLANIMI	120
RESİM 89 : UIVIEW ANIMATE FONKSİYONU VE ZİNCİRLEME ANİMASYON ÖRNEĞİ.....	122
RESİM 90 : VIEWCONTROLLER YAŞAM DÖNGÜSÜ	124
RESİM 91 : VIEWCONTROLLER GÖRÜNÜM İLİŞKİLERİ	125
RESİM 92 : VIEWCONTROLLER YAŞAM DÖNGÜSÜ FONKSİYONLARI KULLANIMI	128
RESİM 93 : STORYBOARD VE CLASS GÖRÜNÜMÜ	130
RESİM 94 : ALERTCONTROLLER SINIFININ TÜRETİLMESİ	130
RESİM 95 : TÜRETİLEN ALERTCONTROLLER SINIFININ EKRANDA GÖSTERİMİ	131
RESİM 96 : POPUP PENCERESİ UYGULAMA ÜZERİNDE ÇALIŞIRKEN	132
RESİM 97 : ALERTCONTROLLER SINIFI İLE ÜRETİLMİŞ POPUP PENCERESİNE ALERTACTION LAR İLE BUTTONLARIN EKLENMESİ VE KONTROLÜ	134
RESİM 98 : BUTTONLAR EKLENMİŞ BİR POPUP PENCERİNİN UYGULAMA ÜZERİNDE GÖRÜNÜMÜ	135
RESİM 99 : DOSYA YAZMA VE OKUMA İŞLEMLERİ İÇİN OLUŞTURULMUŞ ÖRNEK ARAYÜZ.....	136
RESİM 100 : DOSYA YAZMA İŞLEMİ KOD BLOĞU.....	137
RESİM 101 : DOSYA OKUMA İŞLEMİ KOD BLOĞU	138
RESİM 102 : DİZİN LISTELEME İŞLEMİ KOD BLOĞU.....	138
RESİM 103 : DOSYA SILME İŞLEMİ KOD BLOĞU	139
RESİM 104 : HTTP GET İSTEĞİ KOD BLOĞU	141
RESİM 105 : HTTP POST İSTEĞİ KOD BLOĞU	143

RESİM 106 : BİR HTTP İSTEĞİ OLUŞTURARAK WEB ADRESİNDE YER ALAN İMAJ DOSYASININ UYGULAMAYA ALINMASI	145
RESİM 107 : PROJE NAVİGATOR 'ÜNDE INFO.PLİST DOSYASININ GÖRÜNÜMÜ	147
RESİM 108 : INFO.PLİST DOSYASI	148
RESİM 109 : INFO.PLİST DOSYASINA YENİ PARAMETRE EKLENMESİ	149
RESİM 110 : INFO.PLİST DOSYASINA EKLENMİŞ KONUM SERVİSLERİ İZİN PARAMETRELERİ	149
RESİM 111 : CORELOCATION KÜTÜPHANESİNİN UYGULAMAYA DAHİL EDİLMESİ VE CLLOCATIONMANAGERDELEGATE KALITIMININ SINIFI ALINMASI	150
RESİM 112 : KALITIM ALINAN LOCATIONMANAGER FONKSİYONU	150
RESİM 113 : IBACTION BAĞLANTISI İLE OLUŞTURULMUŞ FONKSİYON İLE KONUM SERVİSLERİNİN BAŞLATILMASI	151
RESİM 114 : KONUM SERVİSLERİNE İLİŞKİN VIEWCONTROLLER SINIFI TAM GÖRÜNÜMÜ	152
RESİM 115 : KONUM SERVİSLERİNİN KULLANIMINA İLİŞKİN KULLANICI BİLGİLENDİRME VE İZİN POPUP PENCERESİ	153
RESİM 116 : KONUM SERVİSLERİ ÇALIŞTIRILARAK ALINMIŞ KOORDİNATLARIN EKRANDA GöSTERİMİ	154
RESİM 117 : UIVIEW KOMPONENTİ PREVIEW VIEW İSMİ İLE	156
RESİM 118 : UIVIEW KOMPONENTİ SABİTLERİ	157
RESİM 119 : INFO.PLİST KONFIGÜRASYON DOSYASI KAMERA İZNİ	158
RESİM 120 : AVFOUNDATION FRAMEWORK' ÜNÜN DAHİL EDİLMESİ	159
RESİM 121 : MEDYA İÇERİĞİ ALINACAK CİHAZIN SEÇİLMESİ	160
RESİM 122 : SEÇİMİ YAPILAN CİHAZDAN GİRİŞ NESNESİ OLUŞTURULMASI	161
RESİM 123 : CİHAZ KAMERASINDAN ALINAN GÖRÜNTÜYE ERİŞİM	161
RESİM 124 : IOS İŞLETİM SİSTEMİ ÜZERİNDE SWIFT PROGRAMLAMA DİLİ İLE KAMERA ERİŞİMİ	163
RESİM 125 : INFO.PLİST DOSYASI MİKROFON ERİŞİMİ	165
RESİM 126 : MİKROFON ERİŞİMİ ÖRNEK ARAYÜZ TASARIMI	165
RESİM 127 : AVFOUNDATION FRAMEWORK ÜNÜN DAHİL EDİLMESİ VE DEĞİŞKENLER	166

RESİM 128 : SES KAYDI ÖNCESİ VIEWDIDLLOAD ÜZERİNDE HAZIRLIK.....	167
RESİM 129 : KAYDET, OYNAT VE DURDUR BUTTONLARI IBACTION BAĞLANTILARI.....	168
RESİM 130 : KAYIT VE KAYITTAN YÜRÜTME İŞLEMLERİNİN SİSTEM BİLDİRİMİ	170
RESİM 131 : JİROSKOP EKSENLERİ.....	172
RESİM 132 : GYROSCOPE KULLANIM ÖRNEĞİ.....	173
RESİM 133 : MANYETOMETRE KULLANIMI	175
RESİM 134 : COREMOTİON FRAMEWORK VE CMALTİMETER SINIFI	176
RESİM 135 : BAROMETRENİN ERİŞİLEBİLİRLİĞİNİN KONTROLÜ....	177
RESİM 136 : STARTRELATIVEALTIITUDEUPDATES() FONKSİONUNUN KULLANIM ÖRNEĞİ	178
RESİM 137 : BAROMETRE İLE YÜKSEKLİK VE BASINÇ DEĞERİ ÖLÇÜMÜ	179
RESİM 138 : BİLDİRİM İZİNLERİNİN İSTENMESİ	181
RESİM 139 : BİLDİRİM İZİNLERİNİN TALEBİ.....	182
RESİM 140 : BİLDİRİM İÇERİĞİNİN OLUŞTURULMASI VE BİLDİRİM PANELİNE DÜŞÜRÜLMESİ.....	183
RESİM 141 : OLUŞTURULAN BİLDİRİMİN BİLDİRİM PANELİNE DÜŞÜRÜLMÜŞ HALİ.....	184
RESİM 142 : WEB SERVİS İLETİŞİM HİYERARŞİSİ	187
RESİM 143 : JSON VERİ YAPISI.....	188
RESİM 144 : TERMİNAL ÜZERİNDEN PROJE DİZİNİNE GEÇİŞ	192
RESİM 145 : TERMİNAL ÜZERİNDE POD İNİT KOMUTUNUN ÇALIŞTIRILMASI	193
RESİM 146 : TERMİNAL ÜZERİNDE NANO PODFILE KOMUTUNUN ÇALIŞTIRILMASI	194
RESİM 147 : NANO UYGULAMASI İLE PODFILE DOSYASI İÇERİĞİ	194
RESİM 148 : HARİCİ KÜTÜPHANE EKLENMİŞ BİR PODFILE DOSYASI İÇERİĞİ.....	195
RESİM 149 : TERMİNAL ÜZERİNDE POD INSTALL KOMUTUNUN ÇALIŞTIRILMASI	196
RESİM 150 : YENİ PROJE DİZİNİ VE PROJE WORKSPACE DOSYASI	197
RESİM 151 : ARAYÜZ PROTATİPİ	198

RESİM 152 : PODFILE DOSYASINA ALAMOFİRE	
KÜTÜPHANESİNİN EKLENMESİ	199
RESİM 153 : POD INSTALL KOMUTUNUN ÇALIŞTIRILMASI	200
RESİM 154 : STORYBOARD GÖRÜNÜMÜ	201
RESİM 155 : YENİ SWIFT DOSYASI OLUŞTURMAK İÇİN NEW FILE BUTONU	202
RESİM 156 : COCOA TOUCH CLASS TEMALI BİR SWIFT DOSYASI OLUŞTURMA	203
RESİM 157 : COCOA TOUCH CLASS ÖZELLİKLERİNİN BELİRLENMESİ	204
RESİM 158 : SWIFT DOSYASI KAYIT EKRANI	205
RESİM 159 : IDENDİTY INSPECTOR PANELİ	206
RESİM 160 : YENİ OLUŞTURULAN SWIFT DOSYALARININ STORYBOARD A EKLENEN VIEWCONTROLLERLARA BAĞLANMASI	207
RESİM 161 : BUTONLARIN IBACTION BAĞLANTISI	209
RESİM 162 : KULLANICI SEÇİMİNİN HAFIZAYA ALINMASI VE SONRAKİ SAYFAYA GEÇİŞ İŞLEMİ.....	210
RESİM 163 : GEÇİŞ YAPILMAK İSTENEN VIEWCONTROLLER EKRANININ SWIFT DOSYASI ÜZERİNDE DEĞİŞKEN TANIMLAMASI.....	211
RESİM 164 : PREPARE FONKSİYONU İLE GEÇİŞ YAPILMAK İSTENEN VIEWCONTROLLER EKRANININ SWIFT DOSYASI ÜZERİNDEKİ DEĞİŞKENE VERİ ATAMASI YAPILMASI.....	211
RESİM 165 : INFO.PLİST DOSYASINA API ADRESİNİN EKLENMESİ	213
RESİM 166 : ALAMOFİRE İLE WEBSERVİS İSTEĞİ	214
RESİM 167 : DÖNEN WEB SERVİS CEVABININ DİZİYE ALINMASI.....	215
RESİM 168 : TABLEVIEW IBOUTLET BAĞLANTISI VE PROTOKOL METOTLARI	215
RESİM 169 : PROTOKOL METOTLARI İLE TABLEVIEW SATıRLARININ ÇOĞALTILMASI.....	216
RESİM 170 : TABLEVIEW NESNESİNİN DATASOURCE VE DELEGATE İŞLEMİ	217
RESİM 171 : KULLANICININ TABLEVIEW ETKİLEŞİMİNİN YAKALANMASI	218
RESİM 172 : SEÇİLEN YEMEK ID SİNİN TARİFLER EKRANINA	

GÖNDERİMİ	218
RESİM 173 : YEMEK TARİFLERİNİN WEB SERVİSDEN ÇEKİLMESİ....	219
RESİM 174 : SEÇİLEN YEMEK TARİFİNİN WEB SERVİSTEN ALINARAK ARAYÜZDE GÖSTERİMİ.....	220

35. Yazım Kuralları Dizini

YAZIM KURALI 1 : İF ELSE YAPISI	40
YAZIM KURALI 2 : SWITCH CASE YAPISI	42
YAZIM KURALI 3 : FOR İN DÖNGÜ YAPISI	43
YAZIM KURALI 4 : WHILE DÖNGÜ YAPISI.....	44
YAZIM KURALI 5 : REPEAT - WHILE DÖNGÜ YAPISI	45
YAZIM KURALI 6 : TEMEL FONKSİYON YAPISI.....	47
YAZIM KURALI 7 : PARAMETRELİ FONKSİYON YAPISI.....	48
YAZIM KURALI 8 : GERİYE VERİ DÖNEN FONKSİYON TANIMLANMASI VE KULLANIMI	50
YAZIM KURALI 9 : DEĞİŞKEN SAYIDA PARAMETRE ALAN FONKSİYONLARIN TANIMLANMASI VE KULLANIMI.....	53
YAZIM KURALI 10 : ENUMERATION TANIMLAMASI	55
YAZIM KURALI 11 : HATA YAKALAMA DO-TRY-CATCH YAZIM KURALI.....	59
YAZIM KURALI 12 : SINIF TANIMLANMASI	61
YAZIM KURALI 13 : KALITIM YAPISI	66
YAZIM KURALI 14 : STRUCTURE YAPISI	74
YAZIM KURALI 15 : EXTENSİON YAPISI	115

36. Kare Kodlar Dizini

KARE KOD 1 : HTTPS://GİTHUB.COM/İOS-KİTAP/DEĞİSKENLER....	32
KARE KOD 2 : HTTPS://GİTHUB.COM/İOS-KİTAP/DİZİLER	35
KARE KOD 3 : HTTPS://GİTHUB.COM/İOS-KİTAP/ OPERATORLER	38
KARE KOD 4 : HTTPS://GİTHUB.COM/İOS-KİTAP/ KONTROLYAPILARI	42
KARE KOD 5 : HTTPS://GİTHUB.COM/İOS-KİTAP/DONGULER	46
KARE KOD 6 : HTTPS://GİTHUB.COM/İOS-KİTAP/ FONKSİYONLAR.....	55
KARE KOD 7 : HTTPS://GİTHUB.COM/İOS-KİTAP/CLASSES	73

KARE KOD 8 : HTTPS://GITHUB.COM/IOS-KİTAP/STRUCTURES.....	76
KARE KOD 9 : HTTPS://GITHUB.COM/IOS-KİTAP/ IBOUTLETIBACTION	81
KARE KOD 10 : HTTPS://GITHUB.COM/IOS-KİTAP/ ALLBASICVIEWS	114
KARE KOD 11 : HTTPS://GITHUB.COM/IOS-KİTAP/ EXTENSIONS.....	115
KARE KOD 12 : HTTPS://GITHUB.COM/IOS-KİTAP/ ANİMASYONLAR	123
KARE KOD 13 : HTTPS://GITHUB.COM/IOS-KİTAP/ VIEWCONTROLLERLİFECYCLE.....	128
KARE KOD 14 : HTTPS://GITHUB.COM/IOS-KİTAP/ UIALERTCONTROLLER.....	134
KARE KOD 15 : HTTPS://GITHUB.COM/IOS-KİTAP/ FILEREADWRITE	139
KARE KOD 16 : HTTPS://GITHUB.COM/IOS-KİTAP/ NETWORKİŞLEMLERİ	146
KARE KOD 17 : HTTPS://GITHUB.COM/IOS-KİTAP/ KONUMSERVİSLERİ	155
KARE KOD 18 : HTTPS://GITHUB.COM/IOS-KİTAP/ KAMERAERİSİMİ	164
KARE KOD 19 : HTTPS://GITHUB.COM/IOS-KİTAP/ MİKROFONSESKAYIT	171
KARE KOD 20 : HTTPS://GITHUB.COM/IOS-KİTAP/JİROSKOP.....	174
KARE KOD 21 : HTTPS://GITHUB.COM/IOS-KİTAP/BAROMETRE....	180
KARE KOD 22 : HTTPS://GITHUB.COM/IOS-KİTAP/ YEMEK-TARİFLERİ	220
Kaynaklar	

1. <https://help.apple.com/xcode/mac/current/>
2. <https://developer.apple.com/swift/>
3. <https://developer.apple.com/documentation/>
4. <https://swift.org/>
5. <https://guides.cocoapods.org/>
6. [https://en.wikipedia.org/wiki/Swift_\(programming_language\)](https://en.wikipedia.org/wiki/Swift_(programming_language))
7. <https://en.wikipedia.org/wiki/Smalltalk>
8. https://en.wikipedia.org/wiki/Computer_programming

