



T.C.

**BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ**

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

HAYVAN SAHİPLENME VE KÖPEK GEZDIRME IOS UYGULAMASI

(AdoptingHay)

Engin GÜLEK

Bilgisayar Mühendisliği Tasarım Çalışması II

DANIŞMANI : Dr.Öğr.Üyesi Salim CEYHAN

BİLECİK

9 Ocak 2022



T.C.
BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

HAYVAN SAHİPLENME VE KÖPEK GEZDIRME IOS UYGULAMASI
(AdoptingHay)

Engin GÜLEK

Bilgisayar Mühendisliği Tasarım Çalışması II

DANIŞMANI : Dr.Öğr.Üyesi Salim CEYHAN

BİLECİK
9 Ocak 2022

ÖZET

Projenin Amacı

Çoğu insan bazı zamanlar hayvan sahiplenmek istemektedir. Bunun için etrafındaki kişilere sorarak sahiplene bilecekleri hayvanı aramaktadırlar. Ayrıca köpek sahibi olan kişiler köpeklerini gezdiremeyecekleri zamanları olmadıkları zamanlarda sıkıntı çekmektedirler. Bu proje ile bu iki sorunuda çözmeyi amaçlamaktayım

Projenin Kapsamı

Süreçleri,müşterileri , hayvanlar hakkındaki bilgileri ve internet yoluyla iletilecek servisleri tanımlayan bir iOS uygulaması stratejisi geliştirmek

Sonuçlar

Projenin sonunda projenin başında gerçekleştirilmek istenen planlara ulaşılmıştır. Kullanıcılar her iki bölüm için ilan ekleye bilemektedirler. Köpek gezidirme bölümünde istek gönderebilmektedirler. İlanları inceleyebilmektedirler. Her iki bölüm içinde kullanıcılar birbirleri ile mesajlaşabilmektedirler. İlanları favorilerine ekleyebilemektedirler

ABSTRACT

Project Objective

Most people want to own a pet sometime. For this, they look for the animal they can adopt by asking the people around them. In addition, when people who own dogs cannot walk their dogs, they have trouble when they do not have time. With this project, I aim to solve both these problems.

Scope of Project

Develop an iOS application strategy that defines processes, customers, information about animals, and services to be delivered via the Internet

Results

At the end of the project, the plans that were intended to be realized at the beginning of the project were reached. Users can add advertisements for both sections. They can send requests in the dog walking section. They can view the advertisements. In both sections, users can message each other. They can add ads to their favourites.

TEŞEKKÜR

Bu projenin başından sonuna kadar hazırlanmasında emeği bulunan ve beni bu konuya yönlendiren saygıdeğer hocam ve danışmanım Sayın Dr. Öğr. Üyesi Salim CEYHAN'a tüm katkılarından ve hiç eksiltmediği desteginden dolayı teşekkür ederim.

Engin GÜLEK

9 Ocak 2022

İÇİNDEKİLER

ÖZET	ii
ABSTRACT	iii
TEŞEKKÜR	iv
ŞEKİL LİSTESİ	x
1 GİRİŞ	1
1.1 Uygulamanın Kullanıcılar İçin Önemi	1
1.2 Uygulamanın Hayvanlar İçin Önemi	1
1.3 Benzer Uygulamalar	1
2 MATERİYAL VE YÖNTEMLER	2
2.1 iOS Hakkında Genel Bilgi	2
2.2 Swift Nedir?	2
2.3 Xcode	3
2.4 Uygulamayı Çalıştırma	3
2.4.1 Simülatör	3
2.5 Fiziksel Cihazda Test Yapmak	4
2.6 Neden iOS	5
2.7 MVVM Nedir?	6
2.7.1 MVVM Katmanları	6
2.7.2 MVVM Kullanılma Nedeni?	6
3 AdoptingHay	7
3.1 Uygulama Açıılış Ekranı	7
3.2 Kullanıcı Giriş Sayfası	8
3.2.1 Eposta ile Devam Et	8
3.3 İstenilen Hizmetin Seçilmesi	10
3.4 Hayvan Sahiplenme	11
3.4.1 İlan Ana Sayfa	11

3.4.2	Hayvan Cinsleri ve İlanlar	12
3.4.3	İlan Detay	14
3.4.4	Tıklanılan İlannın Detayların Alınması	15
3.4.5	Bildirimler	17
3.4.6	İlan Ekleme	19
3.4.7	ImageView Tıklanılma Özelliği Eklenmesi	19
3.4.8	Galeriden Fotoğraf Seçmen İşlemi	20
3.4.9	Mesaj Sayfası	22
3.4.10	Kullancıya Bildirim Gösterilmesi	25
3.5	Köpek Gezdirmeye	30
3.5.1	İlan Ana Sayfa	30
3.5.2	İlan Ekleme	31
3.5.3	Hesabım	31
3.5.4	İlan detayları	32
3.5.5	Köpek sahibi ile mesajlaşma	33
3.5.6	Gezdirmeye işlemini takip sayfaları	34
3.5.7	Gezdirmeye işleminin başlatılması	34
3.5.8	Gezdirmeye işleminin bitmesi	35
3.5.9	Kullanıcıya gelen yorumlar sayfası	35
3.5.10	Favorilerine ekleyen ve istek gönderen kullanıcı profili	35
3.6	Hesap Doğrulama(Authentication)	36
3.6.1	Eposta ile Yapılabilecek İşlemler	36
3.6.2	Google ile Yapılabilecek İşlem	41
3.6.3	Girişin Hatırlanması	42
3.6.4	Çıkış İşlemi	43
3.7	Veritabanı	44
3.7.1	Neden Firestore Database?	44
3.7.2	MVVM Modeli İçin Ortamın Hazırlanması	45
3.7.3	İlanların Veritabanından Alınması	47
3.7.4	İlan Arama Yapma	51
3.7.5	Hayvan Türleri Arasında Filtreleme İşlemi	53

3.7.6	Yaşa Göre Sıralame ve Hastalık Filtresi	55
3.7.7	İlan Ekleme	57
3.7.8	Kullanıcıların Mesajlanması	60
3.7.9	Köpek Gezdirmeye İlanı Eklenmesi	62
3.7.10	Köpek Gezdirmeye İsteğinin Yapılması	63
3.7.11	İsteğin Kabul Edimesi Ve Edilmemesi	64
3.7.12	Gezme İşlemin Başlatma Ve Bitirmei	65
4	SONUÇLAR VE ÖNERİLER	67
4.1	Sonuçlar	67
4.2	Öneriler	68
5	EKLER	69
	KAYNAKLAR	70
	ÖZGEÇMİŞ	72

ŞEKİL LİSTESİ

1	iOS	2
2	Xcode	3
3	Xcode Hesabım	4
4	BuidRunDebug	5
5	MVVM	6
6	Açılış ekranı	7
7	iOS	8
8	Eposta ile devam et	9
9	Segmented control kod yapısı	9
10	Hizmetin Seçilmesi	10
11	İlan Ana Sayfa	11
12	Collection view cell kod yapısı	12
13	Cell sayısı	13
14	Cell'e verilerin yerleştirilmesi	13
15	İlan detay	14
16	İlan detay başlık	15
17	İlan id alınması ve gönderilmesi	15
18	İlan id kullanılacağı sayfaya aktarılması	16
19	İdye göre verilerin alınıp alanlara yerleştirilmesi	16
20	Bildirimler	17
21	TableViewCell	18
22	Tableviewcell silme	18
23	Imagepicker tanımlanması	19
24	Hastalık kısmının aktif hale getirilmesi	19
25	ImageView tıklanılma özelliği	20
26	Galeriden fotoğraf seçme işlemi kod yapısı	20
27	Seçilen resmi imageView ekleme	21
28	Galeri	21
29	Mesaj yazılan kısım	23

30	Mesajların ayırt edilmesi	23
31	Mesaj sayısı	23
32	Kullanıcıların resimlerinin tanımlanması	24
33	Mesajlaşma	24
34	Apple developer hesabı	25
35	Apple developer hesabı	26
36	AppDelegate	26
37	İzin alınması	27
38	Token	28
39	Knuff	29
40	Köpek gezdirme ilan ana sayfa	30
41	Köpek gezdirme ilanı ekleme	31
42	Köpek gezdirme hesabım	32
43	Köpek gezdirme ilanı detayları	32
44	Köpek gezdirme mesajlaşma	33
45	Gezdime Takip Sayfası	34
46	Yorum ve teşekkür sayfası	35
47	Kullanıcıya gelen yorumlar	36
48	Kullanıcıya gelen yorumlar	37
49	Kullanıcı girişi	38
52	Eposta üye olma	38
50	Kullanıcı Üye Olma	39
54	Uyarı mesajı görüntüsü	39
51	Podfile	40
53	Uyarı mesajı	40
56	Eposta ile giriş yapma	40
55	Kullanıcı eposta bilgileri	41
57	ClientId	41
58	GoogleAuthProvider	42
59	Goolgle ile giriş yap	42
60	Giriş işleminin hatırlanması	43

61	Çıkış işlemi	43
62	Firestore database	44
63	Örnek service fonksiyonu	46
64	İlanlar model	47
65	İlanlar.viewmodel struct 1	48
66	İlanlar.viewmodel struct 2	48
67	İlanlar sorgu	49
68	İlanların service çağrılmaması	50
69	İlanların veri sayısının alınması	50
70	İlanların verinlerin yerlerine	51
71	Service arama isteği yapılması	52
72	Veritabanına aram isteği yapılması	53
73	Hayvan türleri	53
74	Service havyan türü filtreleme isteği yapılması	54
75	Havyan türü filtreleme sorgusu yapılması	54
76	Filtreleme ve sıralama	55
77	Servide hayvan hastalık yok filtre isteği	56
78	Resim stroge	57
79	Resmin storage eklenmesi örneği	58
80	İlan ekleme sorgusu	59
81	Mesaj veritabanı	60
82	Mesajın veritabanına eklenmesi	61
83	Mesajın silinmesi	61
84	DogService ilan ekleme isteği	62
85	Veritabanına gezdirme isteğini kaydetme	63
86	Köpek gezdirme isteğinin kabul edilmemesi	64
87	Köpek gezdirme isteğinin kabul edilmemesi bildirim	64
88	Köpek gezdirme isteğinin kabul edilmesi	65
89	Gezdirme işleminin başlatılıp bitirilmesi	66

1 GİRİŞ

1.1 Uygulamanın Kullanıcılar İçin Önemi

Bu uygulama ile hayvan sahiplenmek isteyen kişiler uygulama üzerinde yer alan sahiblenebilecekleri hayvanları inceleyebilmektedirler. Sahiplenmek istedikleri hayvanın sahibi ile mesaj yoluyla görüşerek hayvan hakkında daha fazla bilgi alabilmektedirler.

Ayrıca köpeğini gezdirmeye vakit bulamayan kişiler artık köpeğini gezdirebilecekleri kişiler ile temasa geçerek saatlik ücretler ile köpeklerini artık gezdirmeye çıkartabilmektedirler. Köpeği gezdiren kişi köpek sahibine bir fotoğraf atmak zorundadır. Böylelikle köpek sahibi endişe içinde bırakılmamaktadır.

İş sahibi olmayan ve hayvanları seven kişiler için para kazanma yolu açılmaktadır.

1.2 Uygulamanın Hayvanlar İçin Önemi

Bu uygulama ile artık sahip olduğu hayvana bakmaya gücü yetmeyen kişiler uygulama üzerinden ilan açarak hayvanına yeni bir sahip bulma imkanı sağlamaktadır.

Sahibinin yoğun işleri yüzünden gezmeye çıkartılamayan köpekler artık evde kalmaktan kurtularak dışarıda gezebilmektedirler.

1.3 Benzer Uygulamalar

Internet üzerinde araştırıldığından aşağıda bulunan benzer uygulamalara rastlanmaktadır

DogGo: Evcil hayvan sahiplerine güvenli ve profesyonel köpek gezdirme ve bakım hizmeti sağlayan bir uygulamadır.

2 MATERİYAL VE YÖNTEMLER

2.1 iOS Hakkında Genel Bilgi

iOS (eski adıyla iPhone/iPad OS) Apple'ın başlangıçta iPhone için geliştirdiği ancak daha sonra iPod touch ve iPad de kullanılan mobil işletim sistemidir. 2007-2010 yılları arasında iPhone OS isminde idi [1] .



Şekil 1: iOS
[3]

2.2 Swift Nedir?

Swift herkesin muhteşem uygulamalar geliştirebilmesi için güçlü ve açık kaynaklı bir dildir.

Swift; iOS, Mac, Apple TV ve Apple Watch uygulamaları geliştirebilmek için Apple tarafından oluşturulmuş güçlü ve kullanıcı dostu bir programlama dilidir. Geliştiricilere, her zamankinden daha fazla özgürlük sağlamak için tasarlandı. Swift, kullanımı kolay ve açık kaynaklı bir dildir [2]

Swift, gerçek zamanlı geri bildirim sağlayan ve mevcut Objective-C koduna sorunsuz bir şekilde entegre edilebilen hızlı ve verimli bir dil. Böylece, geliştiriciler sadece daha güvenli ve güvenilir kodlar yazmakla kalmıyor, aynı zamanda zamandan tasarruf edebiliyor ve daha zengin bir uygulama deneyimi yaratıyorlar. Objective-C göre 2,6 kata kadar daha hızlı ve Python 2.7 göre 8,4 kata kadar daha hızlı çalışabilmektedir[2].

Swifti derslerine dahil eden bazı üniversiteler

- Stanford University.
- Plymouth University.
- California Polytechnic State University.
- Technical University of Munich [2].

2.3 Xcode

Diğer tüm Mac uygulamalarını ve aynı zamanda tüm iOS uygulamalarını geliştirmek için kullanılan Mac uygulamasıdır. Muhteşem bir deneyim sağlamak için ihtiyaç duyduğunuz tüm araçlara sahiptir. Mac App Store'dan ücretsiz olarak indirilebilir[2]



Şekil 2: Xcode
[4]

2.4 Uygulamayı Çalıştırma

Uygulamayı çalıştırırmak için iki yöneteme bulmaktadır. Bunlar **Simülatör** ve **Fiziksel cihazdır**

2.4.1 Simülatör

Bu bölüm, uygulamanızın hangi ortamda çalıştıracağınızı belirlemenizi sağlıyor. Üstüne tıkladığınızda ise Xcode ortamınızda yüklü olan bütün simülatörleri görebilirsiniz. Bu

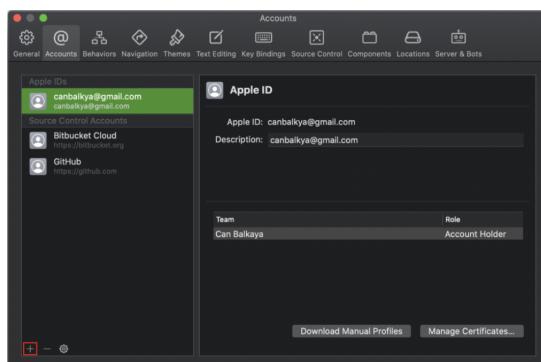
simülatörlerden birini seçmeniz durumunda, Xcode projenizi çalıştırığınızda o simülatör uygulamanızı çalıştıracaktır

2.5 Fiziksel Cihazda Test Yapmak

Geliştirdiğiniz uygulamaları kendi cihazınıza yüklemek için Apple Developer hesabına ihtiyacınız vardır. Apple Developer ve AppleID hesaplarında da uygulamalarınızı kişisel cihazlarınıza kurabilirsiniz ama bu işlemi belli kısıtlamalara göre yapmanız gerekmektedir.

Bir Apple ID oluşturduğunuzu düşünürsek, artık Xcode ortamına bu Apple IDyi tanımlayabiliriz. Bunun için üst taraftaki Xcode sekmesinden Preferences sekmesine tıklayalım.

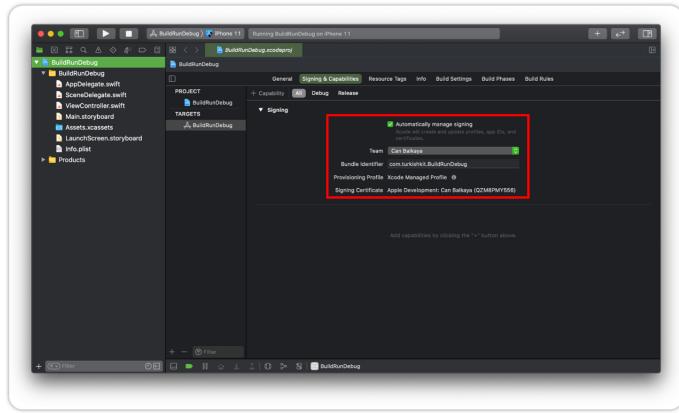
Önümüze çıkan yeni sayfanın altında bulunan + tuşuna basarak hangi tipte hesap tanımlayacağımızı seçebiliriz.



Şekil 3: Xcode Hesabım

Simülatörü bağlamadan önce yapılması gereken son şey, Signing Capabilities kısmına gidip az önce eklenen Apple IDyi Team olarak seçmek olacaktır.

Artık cihazımızı bilgisayarımıza bağlantı vakti geldi. Bunun için bir lightning kablosu ile cihazınızı Macinize bağlamanız gerekmektedir. Bağladığınız zaman Xcodeun üst tarafında simülatörlerin bulunduğu yerde cihazınızda gözükecektir. Artık uygulamanızı cihazınızda kurmak için yapmanız gereken tek şey cmd+R tuşlarına veya sol üstteki play tuşuna basıp projeyi çalıştırmaktır. (İlk defa cihazınıza Xcode üzerinden bir uygulama ku-



Şekil 4: BuidRunDebug

ruyorsanız kurulum biraz uzun sürebilir.) Kablo ile bir cihazı Mace bağlayarak test yapmanın yanı sıra bu işlemi kablosuz bir şekilde yapmak da mümkündür. Sonučta, bir AR uygulamasını test ederken veya Apple TV için bir uygulama geliştirdiğinizde kablolarla uğraşmak sizin için baya kötü olabilir. Bunu yapmak için ilk önce cihazınızın kablo ile bilgisayarınıza bağlı olması gerekiyor. Simülatörünüzü ulaşmak için yukarıdaki Window sekmesinin içerisindeki Devices and Simulators sekmesine tıklayalım.[16]

2.6 Neden iOS

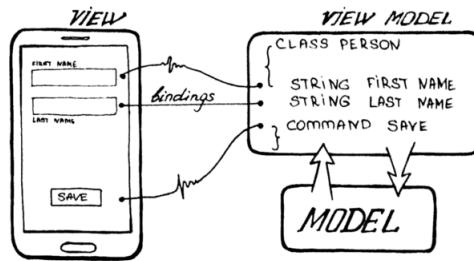
Dünya çapında en çok kullanılan telefon markası Apple'dır. Ayrıca en çok kullanılan akıllı saat markası da Apple'dır. Bunların hepsi Swift ile geliştirilmektedir. Ülkemizde de en çok ihtiyaç duyulan mobil programlamacıdan biri olan iOS geliştiriciler olduğu için Swift kullanmayı tercih etmekteyim.

2.7 MVVM Nedir?

Yazılım mimarileri geliştirdiğimiz projelerin kod kalitesini ve verimliliğini arttırmamızı sağlamaktadır. Bunlardan MVVM, bugün özellikle mobil(android, iOS), için uygulamalar geliştirirken tercih edilen yazılım mimarilerinden bir tanesi. MVVM, 2005 yılında Microsoftta çalışan iki yazılım mimarı Ken Cooper ve Ted Peters tarafından ortaya konmuştur [5].

2.7.1 MVVM Katmanları

View: Kullanıcı ile etkileşime geçen katmandır. **ViewModel:** View ile model arasında işgören bir ara katmandır. View üzerindeki bilgilerin sunumunu kapsar. **Model:** Servisler, lokal veritabanı veya diğer veri kaynaklarından gelen verileri temsil eder.



Şekil 5: MVVM

[5]

2.7.2 MVVM Kullanılma Nedeni?

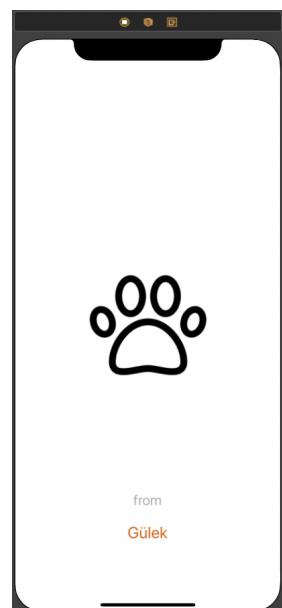
MVVM modeli içerisinde bulunan yapıları birbirinden ayırtırarak az bağlı bir yapı görsel arayüz ile arka planda yer alan business (faliyet) yapıları, bileşenlerin birbirinden ayrışmasını sağlamaktadır. Bu sayede yazılım geliştiriciler ve arayüz tasarımcıları aynı proje üzerinde birbirinden bağımsız bir şekilde eş zamanlı olarak çalışabilme imkanını yakalamaktadır. Görsel arayüzün ve kodun birbirinden ayrışması uygulamanın test edilebilirliğini artırmaktadır. Aynı zamanda bu ayrışımlar uygulamamızın bakımını da kolaylaştırmış oluyor. Model tarafında bir değişiklik yapılmak istendiğinde UI tarafında hiçbir değişiklik yapılmasına gerek kalmadan geliştirmeler rahatlıkla yapılabilir[9].

3 AdoptingHay

Uygulama tasarılanırken **UIKit** uygulama çerçevesi kullanılmaktadır. İçerisinde **UIButton**, **UILabel**, **UISwitch** gibi iOS uygulamanızda kullanabileceğiniz temel elemanları bulundurmaktadır.

3.1 Uygulama Açılış Ekranı

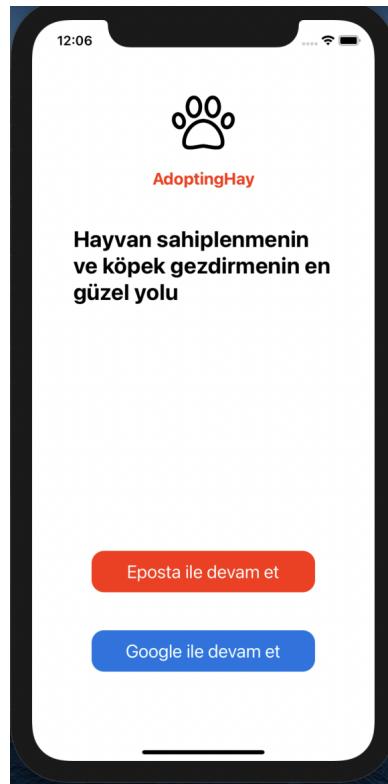
Uygulama açılincaya kadar beyaz ekran yerine aşağıdaki alan gösterilmektedir. Bu alan **LaunchScreen**'de tasarılmaktadır.



Şekil 6: Açılış ekranı

3.2 Kullanıcı Giriş Sayfası

Kullanıcı giriş sayfasında kullancını iki farklı yöntemle üye olabileceği veya giriş yapabileceği iki buton bulunmaktadır.



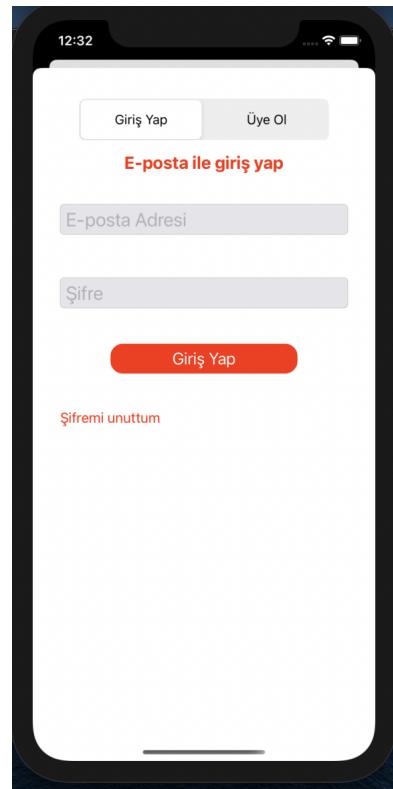
Şekil 7: iOS

Kullanıcı eğer **Eposta ile devam et** tıklar ise kendisini giriş yapabileceği ya da üye olabileceği bir alana yönlendirecektir.

3.2.1 Eposta ile Devam Et

Bu sayfada iki seçenek bulunmaktadır. Bu alanlar **Giriş Yap** ve **Üye Ol** kısımlarıdır. Her biri ayrı bir **UIView** (tasarım) alanı ile oluşturulmuştur. Sayfa üzerine yerleştirildikten sonra control tuşuna basılıp sürekli kullanılabileceği **ViewController** bırakılması gerekmektedir. Bu tasarım alanları arasında geçiş yapılabilmesi için bir görsel nesne olan **UISegmentedControl** kullanılmaktadır. Aynı şekilde bu görsel nesnenin kullanılabileceği **ViewController** yerleştirilmesi gerekmektedir.

Yerleştirilen UISegmentedControl sayesinde oluşturulan fonksiyon ile gerekli UIView tasarım alanlarının gözükmesi sağlanmaktadır.



Şekil 8: Eposta ile devam et

```
1 @IBAction func segmentedControlAction
2   (_ sender: UISegmentedControl) {
3     switch sender.selectedSegmentIndex {
4       case 0:
5         singInContainer.isHidden = true
6         singUpContainer.isHidden = false
7         break
8       case 1:
9         singInContainer.isHidden = false
10        singUpContainer.isHidden = true
11        break
12      default:
13        break
14    }
15 }
```

Şekil 9: Segmented control kod yapısı

3.3 İstenilen Hizmetin Seçilmesi

Uygulama içerisinde hem hayvan sahiplenme hemde köpek gezdirme hizmetinin olduğu belirtilmiştir. Bu sayfada istenilen hizmeti seçerek uygulamayı kullanmaya devam edilebilinmektedir. Her iki hizmet içinde aynı hesap giriş işlemi gerçekleşmektedir.



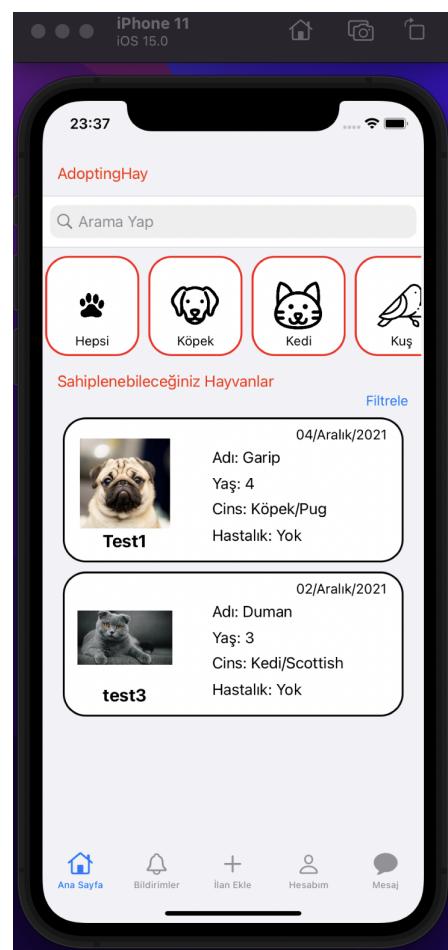
Şekil 10: Hizmetin Seçilmesi

3.4 Hayvan Sahiplenme

Kullanıcı bu hizmet ile hayvan sahiplenebilmektedir. Ayrıca hayvanın bakamayacağı bir durum gerçekleme ihtimali var ise buradan ilan açabilmektedir. Kullanıcılar ile mesajlaşarak ilan hakkında daha fazla bilgi alabilmektedir

3.4.1 İlan Ana Sayfa

İlan ana sayfasında kullanılan görsel nesneler **Searchbar** ve **CollectionView** kullanılmıştır. Sayfalar arasında geçiş için ise **TabBar Controller** kullanılmaktadır.



Şekil 11: İlan Ana Sayfa

3.4.2 Hayvan Cinsleri ve İlanlar

İlan ana sayfası üzerinde hayvan türleri arasında filtreleme yapılabilmesi için bir **collectionView** kullanılarak sistem üzerinde bulunan hayvan türleri ve ilanlar için Collectionview görsel nesnesi eklendikten sonra **collectionViewName.delegate = self** ile metodların kullanılması sağlanmaktadır ve verilerin yerleştirilmesi için **collectionViewName.dataSource = self** kullanılmaktadır. Collectionview içerisinde görsel nesne eklenmesi gerekmektedir. Bunun için **CollectionViewCell** kısmının oluşturulması gerekmektedir ve eklenen görsel nesneler buraya eklenmesi gerekmektedir.

```
1 import UIKit
2 class AnimalAdvertCVC: UICollectionViewCell {
3     @IBOutlet weak var animalAdvertImage: UIImageView!
4     @IBOutlet weak var animalAdvertNameLabel: UILabel!
5     @IBOutlet weak var animalAdvertAgeLabel: UILabel!
6     @IBOutlet weak var animalAdvertKindsLabel: UILabel!
7     @IBOutlet weak var animalAdvertSickLabel: UILabel!
8 }
```

Şekil 12: Collection view cell kod yapısı

Bu dosya oluşturulduktan sonra **CollectionViewCell** için özel bir **identifier** gerekmektedir. Hangi **cell** (hücreye) verilerin yerleştirileceği belirlenmektedir.

Verilerin yerleştirilmesi için **UICollectionViewDelegate** ve **UICollectionViewDataSource** kullanımı gerekmektedir. **Extension** yönetimi kullanılmaktadır.

Verilerin sayısına göre cell (hücre) oluşturulacaktır. Bunun içi bulunduğu listeden sayısının alınması gerekmektedir. İlanlar içinde collectionView kullanıldığı için hangisini kullanılacağını belirlemek için if ile kontrol yapılmaktadır. Geriye **Int** değer döndürülmesi gerekmektedir.

```

1 func collectionView(_ collectionView: UICollectionView,
2                     numberOfItemsInSection section: Int) -> Int {
3     if collectionView == self.animalAdvertCollectionView {
4         return veriSayisi
5     }
6     return animalKinds.count
7 }
```

Şekil 13: Cell sayısı

Verilerin yerleştirilmesi ve cell kısmına yerleştirilen görsel nesnelerin kullanılması gerekmektedir. İki adet collectionView kullanıldığı için if else ile kontrolü yapılmaktadır. Cell (hücre) ulaşmak için **let cell = collecitonViewName.dequeueReusableCellReusableCell (withReuseIdentifier "", for: indexPath) as! CellFile** özel olarak verilen cell identifirerini (kimliği) withReuseIdentifier kısmına yazılması gerekmektedir. Hücrenin sırasında **indexPath** ile alınması sağlanmaktadır. Aşağıdaki kod bloğunda hücrelerde kullanılan görsel nesnelere ulaşarak listeler içerisinde **indexPath.row** ile sırasına göre veriler yerleştirilme örneği yer almaktadır.

```

1 func collectionView(_ collectionView: UICollectionView, cellForItemAt
2                     indexPath: IndexPath) -> UICollectionViewCell {
3
4     if collectionView == self.animalAdvertCollectionView {
5         let cell = animalAdvertCollectionView.dequeueReusableCellReusableCell(
6             withReuseIdentifier: "animalAdvertCell", for: indexPath) as!
7             AnimalAdvertCVC
8
9         cell.animalAdvertImage.image = UIImage(named: "")
10
11        cell.animalAdvertNameLabel.text = "Ad: "
12        cell.animalAdvertKindsLabel.text = "Cins: "
13        cell.animalAdvertAgeLabel.text = "Yaş: "
14        cell.animalAdvertSickLabel.text = "Hastalık : "
15
16        return cell
17    }
18    else {
19        let cell = animalKindsCollectionView.dequeueReusableCellReusableCell(
20             withReuseIdentifier: "animalKindsCell", for: indexPath) as!
21             AnimalKindsCVC
22
23        cell.animaKindsIcon.image = UIImage(named: "")
24        cell.animalKindsTitle.text = ""
25
26        return cell
27    }
28 }
```

Şekil 14: Cell'e verilerin yerleştirilmesi

3.4.3 İlan Detay

Kullanıcı ilana bastığında açılacak sayfadır. Buradan kullanıcı ilan detaylarına, ilan açan kişinin notuna ve ayrıca burdan mesaj gönderme işlemide yapabilmektedir



Şekil 15: İlan detay

Burada iki adet CollectionView kullanılmaktadır. Birincisi resimler için kullanılan **animalDetailsImagesCollectionView** ikinci ise hazır mesajları göstermek için kullanılan **animalAdvertDetailMesajCollectionView**dir.

İlan detay başlığının kullanılması için aşağıdaki kod bloğu kullanılmaktadır. Burada **UILabel** ile genişlik ve uzunluğu belirlenmektedir. Merkezde konumlanması için ise **myNameLabel.textAlignment** kullanılmaktadır. Son olarak ise boyut ve kalınlık için **UIFont.systemFont()** kullanılmaktadır.

```

1 let myNameLabel = UILabel(frame: CGRect(x: 0, y: 0, width: 122
height: 30))
2     myNameLabel.text = "İlan Detay"
3     myNameLabel.textAlignment = .center
4     myNameLabel.center.x = self.view.center.x
5     myNameLabel.center.y = self.view.center.y/8
6     myNameLabel.font = UIFont.systemFont(ofSize: 25, weight: .bold)
7     myNameLabel.textColor = .red
8     self.view.addSubview(myNameLabel)

```

Şekil 16: İlan detay başlık

İlanı açılan hayvana ait olan resimlerin gösterilmesi için Collectionview kullandığı söylenmiştir. Burada ilan ana sayfasında kullanılması ile hiç bir fark bulunmamaktadır.

3.4.4 Tıklanılan İlannın Detayların Alınması

İlana tıklanıldığında aşağıdaki kod bloğu çalışarak ilan için tanımlanan class içerisinde id'sini almaktadır ve kullanılacağı sayfaya geçiş yapmasını sağlayan **performSegue** ile gönderilmektedir

```

1 func collectionView(_ collectionView: UICollectionView,
2 didSelectItemAt indexPath: IndexPath) {
3     if collectionView == self.animalAdvertCollectionView {
4         let animalAdvertUid = animalAdvertLists[indexPath.row].
animalUid
5         performSegue(withIdentifier: "homePageToAdvertDetails",
6 sender: animalAdvertUid) } }

```

Şekil 17: İlan id alınması ve gönderilmesi

İd kullanılacağı sayfada tanımlanan değişkene ulaşılması gerekmektedir. Bunun için **prepare** kullanılmaktadır. İçerisinde her geçiş işlemi için özel olarak tanımlanan id kontrolü yapılmaktadır. Bunun nedeni ilerde birden fazla geçiş olduğunda hata oluşmasını engellemektir.

İd opsiyonel olarak geldiği için Stringe çevrilir. Son olarak downCasting kullanılarak kullanılacağı sayfada tanımlanan değişkene ulaşılır ve id o değişkene aktarılmaktadır.

```

1     override func prepare(for segue: UIStoryboardSegue, sender: Any
2     ?) {
3         if segue.identifier == "homePageToAdvertDetails" {
4             if let dataUid = sender as? String {
5                 let toAnimalAdvertDetailVC = segue.destination as!
AnimalAdvertDetailsVC
6                     toAnimalAdvertDetailVC.getAnimalAdvertUid = dataUid } }
7     }

```

Şekil 18: İlan id kullanılacağı sayfaya aktarılması

Gelen id alındıktan sonra veritabanında bu idnin verileri alınması gerekmektedir. Bu-nun için **db.collection("animalAdvert").document(gelenId)** ile verilere ulaşılıp gerekli olan alanlara yerleştirilmesi sağlanmaktadır.

```

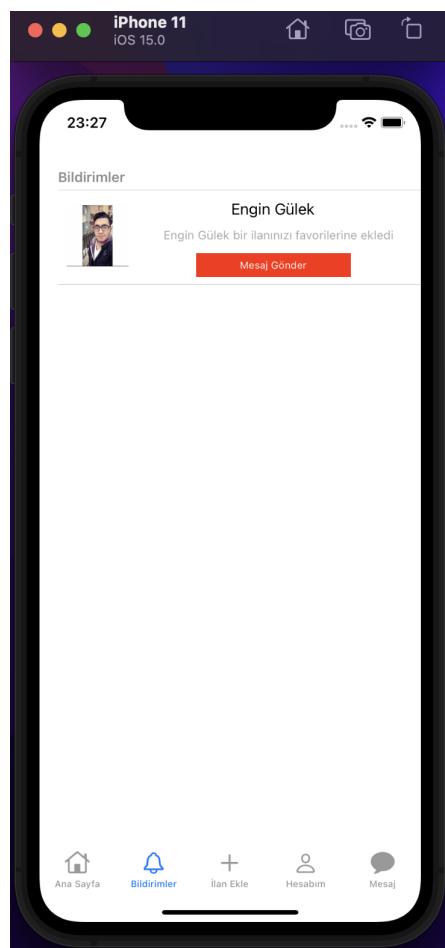
1 func getAnimalAdvertDetailNot() {
2     let db = Firestore.firestore()
3     if let documentId = getAnimalAdvertUid {
4 db.collection("animalAdvert").document(documentId).getDocument{
    snapshot, error in
5 if error != nil {
6 print(error?.localizedDescription ?? "Bir hata oluştu") }
7 else {
8 if let getAnimalName = snapshot?.get("animalName") as? String {
9     self.animalName.text = "Adı: \(getAnimalName)" }
10 if let getAnimalGenus = snapshot?.get("animalGenus") as? String {
11     if let getAnimalKinds = snapshot?.get("animalKinds") as? String {
12         self.animalGenus.text = "Cinsi: \(getAnimalGenus)-\(getAnimalKinds)" }
13 if let getAnimalAge = snapshot?.get("animalAge") as? Int {
14     self.animalAge.text = "Yaş: \(getAnimalAge) aylik" }
15 if let getAnimalSick = snapshot?.get("animalSick") as? String {
16     self.animalSick.text = "Hastalık: \(getAnimalSick)" }
17 if let getAnimalOwnerNote = snapshot?.get("animalOwnerNot") as? String {
18     self.animalOwnerNote.text = getAnimalOwnerNote } } } } }

```

Şekil 19: İdye göre verilerin alınıp alanlara yerleştirilmesi

3.4.5 Bildirimler

Kullanıcının eklediği ilanı başka bir kullanıcı favorilerine eklediğinde kullancıya bildirmeler kısmında favorilerine ekleyen kullanıcı gösterileceği yerdir. Dillerse kullanıcı bildirimi sağa çekip bildirimi silme işlemi uygulaya bilmektedir.



Şekil 20: Bildirimler

Bildirimlerin gösterilmesinde kullanılan görsel nesne olan **UITableView** eklenmesi gerekmektedir. Ekleme işlemi tamamladıktan sonra gerekli metodların kullanılması için **notificationTableView.delegate** ve **notificationTableView.dataSource** eklenmesi gerekmektedir.

Her bir verinin görsel nesneler içerisinde gösterilmesi için cell (hücre) oluşturulup cell yerleştirilmesi ve özel bir **identifier** verilmesi gerekmektedir. Ayrıca cell için ayrı bir **TableViewCellCell** oluşturulup özel nesnelerin buraya tanımlanması gerekmektedir

```
1 class NotficationTVC: UITableViewCell {
2
3     @IBOutlet weak var notficationImageView: UIImageView!
4     @IBOutlet weak var notficationNameLabel: UILabel!
5     @IBOutlet weak var notficationNotfiLabel: UILabel!
6
7     override func awakeFromNib() {
8         super.awakeFromNib()
9     }
10    override func setSelected(_ selected: Bool, animated: Bool) {
11        super.setSelected(selected, animated: animated)
12    }
13    @IBAction func notficationtoProfilButton(_ sender: Any) {
14        print("Mesaj Gönder ")
15    }
16 }
```

Şekil 21: TableViewCell

Silme işlemi için oluşturulan kısma tıklanlığında yapılacak işlemin tanımladığı method bulunmaktadır. Burada işlemin ismi **.destructive** ve **.normal** görünüm türleri belirlenmektedir.

```
1 func tableView(_ tableView: UITableView,
2                 trailingSwipeActionsConfigurationForRowAt indexPath: IndexPath) ->
3                 UISwipeActionsConfiguration? {
4             let notificationDelete = UIContextualAction(style: .destructive,
5                 title: "Sil") { (contextualAction, view, boolValue) in
6                 print("\(self.notificationLists[indexPath.row].id ) silindi")
7             }
8             return UISwipeActionsConfiguration(actions: [notificationDelete,
9                 notificationBlock]) }
```

Şekil 22: Tableviewcell silme

3.4.6 İlan Ekleme

Bu bölümde kullanıcı ilan açabilmektedir. Burada imageView tanımlanmıştır. Bunlardan üç tanesi detay kısmında gösterilen diğeri ise ilan sayfasında gösterilen resimlerdir. Her bir resim için ayrı bir **UIImagePickerController()** tanımlanması gerekmektedir. Eğer tanımlanmaz ise diğer resimlerde de seçilen resimdeki değişikler görülecektir.

```
1 let imagepicker = UIImagePickerController()
2 let imagepickerOne = UIImagePickerController()
3 let imagepickerTwo = UIImagePickerController()
4 let imagepickerThird = UIImagePickerController()
```

Şekil 23: Imagepicker tanımlanması

Kullanıcının hayvan hakkında bilgiler girebileceği alanlar tanımlanmıştır. Bu alanlardan sadece **Hastalık** kısmı boş bırakılabilir. Yoksa kendisine uyarı mesajı verilecektir. Kullanıcının bu kısma yazı yazabilmesi için **Switch** ile bunu aktif hale getirmesi gerekmektedir. Aktif hale gelirse hastalık var diye işlenerek hastalık hakkında bilgi girebileceği alan aktif hale getirilecektir. Bu olay switch ile gelen fonksiyona yazılmaktadır

```
1 @IBAction func valueChanged(_ sender: Any) {
2     if sickSwitch.isOn {
3         addSickInfo.isUserInteractionEnabled = true
4         sickBool.text = "Var"
5     } else {
6         addSickInfo.isUserInteractionEnabled = false
7         sickBool.text = "Yok" }
```

Şekil 24: Hastalık kısmının aktif hale getirilmesi

3.4.7 ImageView Tıklanılma Özelliği Eklenmesi

Her bir imageView için tıklanılma özelliği yapılması gerekmektedir. Ayrıca tıklanılma işlemi yapıduğunda gerçekleşcek kendimizin tanımladığı fonksiyonunun tanımlanması gerekmektedir. Daha sonra bu imageView eklenmesi gerekmektedir .

```

1 // tıklanılma özelliğinin eklenmesi
2     addImageAdvert.isUserInteractionEnabled = true
3     let gestureRecognizer = UITapGestureRecognizer(target: self,
4         action: #selector(chooseAddAdvertImage))
5         addImageAdvert.addGestureRecognizer(gestureRecognizer)
6 // tıklanılma özelliğinin eklenmesi
7     addAdvertImageOne.isUserInteractionEnabled = true
8 // tıklanınca çalışacak fonsiyon
9     let gestureRecognizerImageOne = UITapGestureRecognizer(target:
10 self, action: #selector(chooseaddAdvertImageOne))
11     addAdvertImageOne.addGestureRecognizer(
12 gestureRecognizerImageOne)
13 //addAdvertImageTwo ve addAdvertImageThird içinde aynı işlemler
14     gerçekleştirilecek

```

Şekil 25: ImageView tıklanılma özelliği

3.4.8 Galeriden Fotoğraf Seçmen İşlemi

Bu olay tıklanıldığı zaman yapılması istenen fonksiyonun içeresine yazılacaktır. İlk önce **.delegate = self** yazılarak kendisine ait parametrelere erişebilinmektedir. **imagepickerOne.sourceType = .photoLibrary** ile galeriden fotoğraf alınacağını tanımlanmaktadır. En son olarak bu alınma isteğini view eklenmesi gerekmektedir.

```

1 @objc func chooseAddAdvertImage() {
2     imagepicker.delegate = self
3     imagepicker.sourceType = .photoLibrary
4     self.present(imagepicker, animated: true, completion: nil)
5 }
6 @objc func chooseaddAdvertImageOne() {
7     imagepickerOne.delegate = self
8     imagepickerOne.sourceType = .photoLibrary
9     self.present(imagepickerOne, animated: true, completion: nil)
10 //addAdvertImageTwo ve addAdvertImageThird içinde aynı işlemler
11     gerçekleştirilecek

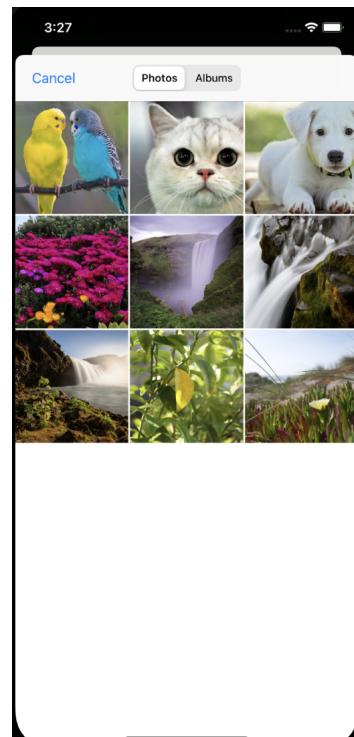
```

Şekil 26: Galeriden fotoğraf seçme işlemi kod yapsı

En başta her bir imageView için UIImagePickerController tanımlanması gerektiğini söylemiştir. Resim seçme işlemi tanımlandığında resim alınması için imageView tıklanılma özelliği ile gelen fonksiyon kullanılır. Burada resmin orjineli alınarak her bir imageView eklenmektedir.

```
1 func imagePickerController(_ picker: UIImagePickerController,
2 didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
3     if picker == self.imagepicker {
4         addImageAdvert.image = info[.originalImage] as? UIImage }
5     if picker == self.imagepickerOne {
6         addAdvertImageOne.image = info[.originalImage] as? UIImage
7     }
8 //addAdvertImageTwo ve addAdvertImageThird içinde aynı
9 işlemler gerçekleştirilecek
```

Şekil 27: Seçilen resmi imageView ekleme



Şekil 28: Galeri

3.4.9 Mesaj Sayfası

Kullanıcı beğendiği ilan hakkında daha fazla bilgi alınması için ilan sahibine mesaj gönderebilmektedir.

Mesaj Gönderilen Kullanıcı Listesi

İlk önce mesaj gönderilen kullanıcıların listesini göstermek için bir **tableView** oluşturulması gerekmektedir. TableView **MessageUserListViewController**'da oluşturulması gerekmektedir. **Bildirimler** kısmında kullanılan yöntem ile tableView oluşturulmaktadır. Kullanıcıya basınca mesajlaşma kısmına geçilmektedir.

Mesaj Gönderilecek Kısım

Burada kullanıcı ilan sahibine mesaj gönderebilmektedir. Bunu için hazır bir kütüphane kullanılmaktadır. Bu **Podfile** içerisinde **MessageKit** eklenerek **pod install** denilerek indirilmesi gerekmektedir.

Mesaj sayfası **ChatVC** adında **ViewController** şeklinde oluşturulması gerekmektedir. Daha sonra içerisinde **import MessageKit** denilmesi gerekmektedir. Ekleme işlemi yapıldıktan sonra ChatVC içerisinde bulunan **class ChatVC: ViewController** kısmında ViewController silinerek yerine **MessagesViewController** yapılması gerekmektedir.

Mesajı gönderen kişinin bilgilerinin tanımlanması gerekmektedir. Bunun için **struct Sender: SenderType** yazıldığındá birlikte gelen id ve displayName tanımlanması gerekmektedir.

Gönderilecek mesajın bilgilerin tanımlanması gerekmektedir. Bunun için **struct Message : MessageType** içerisinde mesaj idsi , gönderilme zamanı ve mesaj türü belirlenir. Mesaj türleri ise normal yazı , resim, lokasyon ve benzerleri şeklindedir. Mesajların tutulması için **Message** model tipinde bir liste oluşturulması gerekmektedir. Daha sonra mesajlaşan kullanıcıların bilgilerin tanımlama şekli **private var selfSender = Sender()** şeklinde tanımlanmalıdır.

`der(senderId: "1", displayName: "Engin Gülek")` şeklindedir.

Mesajlaşma için gerekli fonksiyonlara `messagesCollectionView.messagesDataSource = self`, `messagesCollectionView.messagesLayoutDelegate = self`, `messagesCollectionView.messagesDisplayDelegate = self` ve `messageInputBar.delegate = self` ile ulaşmaktadır.

Mesaj yazılan kısımda ki yazınlara aşağıdaki fonksiyon ile ulaşılmaktadır.

```
1 func inputBar(_ inputBar: InputBarAccessoryView,  
2 didPressSendButtonWith text: String) {  
    print("Mesaj : \(text)") }
```

Şekil 29: Mesaj yazılan kısım

Mesajlaşmadaki mesajların kime ait olduğunu ayırt edilmesi için aşağıdaki fonksiyon kullanılmaktadır. Geri dönüş değeri olarak hesap sahibinin bilgilerinin yer aldığı **sender**'ın döndürülmesi gerekmektedir.

```
1 func currentSender() -> SenderType {  
2     return selfSender  
3 }
```

Şekil 30: Mesajların ayırt edilmesi

Mesajların kaç adet olduğunu bilinerek o kadar konuşma baloncuğu oluşturulacaktır. Bunun için aşağıdaki fonksiyon kullanılmaktadır.

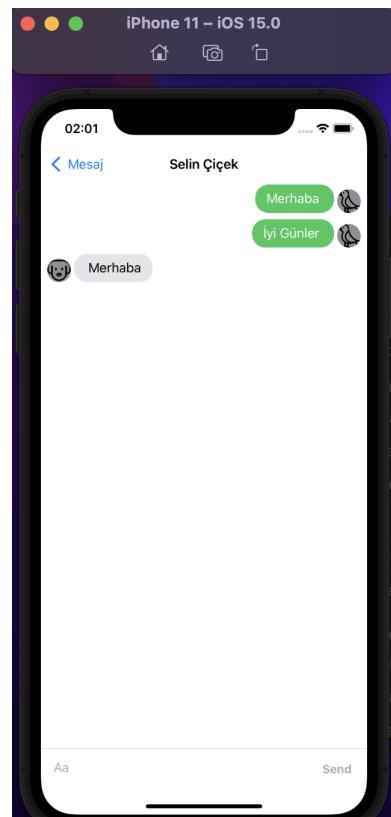
```
1 func messageForItem(at indexPath: IndexPath, in  
2 messagesCollectionView: MessagesCollectionView) -> MessageType {  
3     return "\\mesaj sayısı  
4 }
```

Şekil 31: Mesaj sayısı

Mesajlaşan kullanıcıların resimlerinin gösterilmesi için aşağıdaki fonksiyon kullanılmaktadır. Burada **sender** kullanılarak kullanıcılarla ulaşılması sağlanmaktadır.

```
1 func configureAvatarView(_ avatarView: AvatarView, for message:  
2   MessageType, at indexPath: IndexPath, in messagesCollectionView:  
3   MessagesCollectionView) {  
4   let sender = message.sender  
5   if sender.senderId == selfSender.senderId {  
6     // our image  
7     avatarView.image = UIImage(named: "bird")  
8   }  
9   else {  
10     avatarView.image = UIImage(named: "dog")  
11   }  
12 }
```

Şekil 32: Kullanıcıların resimlerinin tanımlanması



Şekil 33: Mesajlaşma

3.4.10 Kullancıya Bildirim Gösterilmesi

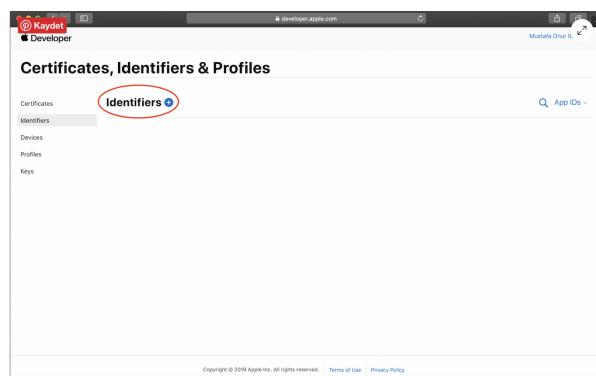
Kullancının uygulamadan uzak kalmaması için yazılımcılar bildirimleri kullanmaktadır. iOS üzerinde bildirim yapabilmek içi **Apple Developer Program Üyeliği** gerekmektedir. Bunun şuanki ücreti **99 dolardır**. Onun için bu projede bildirimleri kullanmak için farklı yöntemler denenmesine rağmen sonuç alınamamıştır. Ancak bildirimlerin Apple Developer Program Üyeliği olduğunda nasıl yapılacağı anlatılmaktadır.

Başlarken:

- **iOS işletim sistemine sahip bir adet cihaz:** Bildirimler simülatörde çalışmıyor. Bu yüzden gerçek bir cihaza ihtiyacımız var.
- **Apple Developer Program üyeliği:** Bildirim gönderebilmek için uygulamamız adına üretilmiş bir adet sertifikaya ihtiyacımız var. Sertifika üretimi sadece bu üyelik ile yapılabiliyor.
- **Knuff:** Bildirim gönderebilmek ve alabilmek için bu programa ihtiyacımız var.

Part1:

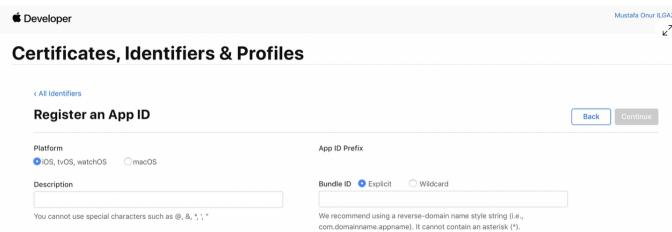
Apple Developer hesabımıza giriş yaparak başlıyoruz. Giriş yaptıktan sonra menüden Certificates, Identifiers Profiles kısmına tıklıyoruz, ardından sol menüden Identifiers’ı seçip üstteki + butonuna tıklıyoruz.



Şekil 34: Apple developer hesabı

Register a New Identifier ekranındaki seçeneklerden AppIDs'i seçiyor ve Continue diyoruz.

Platform olarak iOS seçiyoruz. Description bölümüne genel bir isim veriyoruz. Proje bu-
raya yazdığımız isim ile görünecek. Daha sonra "Bundle ID" kısmından Explicit seçe-
ceği seçiyoruz ve uygulamamız için bir bundle id belirliyoruz. Burası önemli çünkü
Xcode ile projemizi oluştururken burada yarattığımız Bundle IDyi kullanacağız.



Şekil 35: Apple developer hesabı

App Services altından Push Notificationsı seçiyoruz ve Continue butonuna tıklıyoruz.
Girdiğimiz bilgileri kontrol ediyor ve App ID oluşturma adımını tamamlıyoruz.

Bölüm:2

Xcode ile yeni bir proje yaratıyor. Yarattıktan sonra project navigator altında projemize tıklıyoruz. Targetlar altından uygulamamızı seçip General tabı altında bundle identifier kısmını ilk adımda App ID oluştururken belirlediğimiz Bundle ID ile değiştiriyoruz. Daha sonra Capabilities kısmından Push Notifications ayarını açıyoruz. AppDelegate.swifti açı-
yoruz. En üste import UserNotifications yazıyoruz. Aşağıdaki metodu AppDelegate'in so-
nuna ekliyoruz.

```
1 func registerForPushNotifications() {
2     UNUserNotificationCenter.current()
3     .requestAuthorization(options: [.alert, .badge, .sound]) {
4         (granted, error) in
5             print("Permission granted: \(granted)") } }
```

Şekil 36: AppDelegate

Burada kullandığımız keywordleri biraz açalım.

1-) UNUserNotificationCenter: Uygulama içinde bildirim ile ilgili işleri kontrol ediyor.

2-) requestAuthorization(options: completionHandler:): Kullanıcıdan uygulamanın bildirim alabilmesi için izin istiyor. Options parametreleri uygulama içinde kullanmak istediğimiz bildirim özelliğini iletmemize yarıyor. 7 çeşit bildirim özelliği var. Bunlar:

- **badge:** Uygulamanın sağ üst köşesinde sayı gösteriyor.
- **sound:** Bildirim ile birlikte ses çıkarıyor.
- **alert:** Bildirime yazı eklenmesini sağlıyor.

3-) Granted, error: Bu kısım requestAuthorization metodunun completionHandlerları. requestAuthorization metodu tamamlandığında dönüyor. Granted ile izin alınıp alınmadığını, error ile metod koşulurken herhangi bir hata oluşup olmadığını dönüyor.

```
1 func application(_ application: UIApplication,  
2     didFinishLaunchingWithOptions launchOptions: [UIApplication.  
3         LaunchOptionsKey: Any]?) -> Bool {  
4     registerForPushNotifications()  
5     return true  
6 }
```

Şekil 37: İzin alınması

Bölüm 3:

Kullanıcıdan izni aldık ve artık bildirim göndermeye hazırız. Şimdi APNS(Apple Push Notification Service)e uygulamamız ile register olmamız gerekiyor. Yazdığımız registerForPushNotifications() metodu APNSe register olma işlemini başlatıyor. Register olma işlemi tamamlandığında iOS, işlem başarılı tamamlandıysa application(:didRegisterForRemoteNotificationsWithDeviceToken:) metodunu, eğer hata alındıysa application(:didFailToRegisterForRemoteNotificationsWithError:) metodunu call ediyor. application(:didRegisterForRemoteNotificationsWithDeviceToken:) metodu içinden deviceToken alarak işlemimize devam edeceğiz. Bu deviceToken her uygulama için o devicee özel olarak APNS

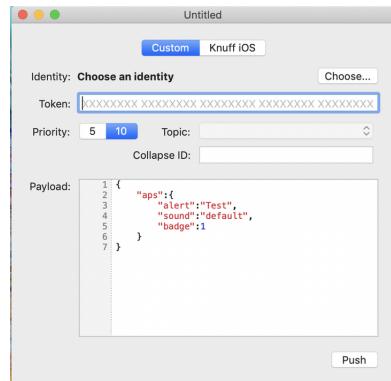
tarafından oluşturuluyor. Bir nevi adres. Bu token ile bildirimin nereye gideceği belirleniyor. Metodumuzu aşağıdaki şekilde düzenliyoruz ve consoledan deviceTokenımızı alıyoruz. Test ederken bu tokena ihtiyacımız olacak.

```
1 func application(_ application: UIApplication,  
2 didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data)  
3 {  
4     let token = deviceToken.map { String(format: "%02.2hhx", $0) }.joined  
5     ()  
6     print("Device token: \(token) ")  
7 }
```

Şekil 38: Token

Bölüm 4:

Bu adımda uygulamamıza bildirim gönderebilmek için APNS sertifikası oluşturacağız. Apple Developer hesabımıza giriş yapıyoruz. Giriş yaptıktan sonra menüden Certificates, Identifiers Profiles kısmına tıklıyoruz, ardından sol menüden Certificates'i seçip üstteki + butonuna tıklıyoruz. Bir sonraki ekranda Apple Push Notification service SSL (Sandbox Production)ı seçiyoruz. Devamında önceden oluşturduğumuz app idyi seçiyoruz. Daha sonraki adımda bizden bir adet CSR dosyası isteyecek. Bunun için bilgisayarımızdan Keychain Access (Anahtar Zincir Erişimi) uygulamasını açıyoruz. Resimdeki gibi Sertifika Otoritesinden Sertifika İste diyoruz. Mail adresimizi girdikten sonra maile veya diske kaydedip devam ediyoruz. Daha sonra tekrar apple developer sayfasına geri dönüp oluşturduğumuz CSR dosyasını yükliyoruz. Daha sonra oluşturduğumuz sertifikayı bilgisayarımıza indiriyoruz.



Şekil 39: Knuff

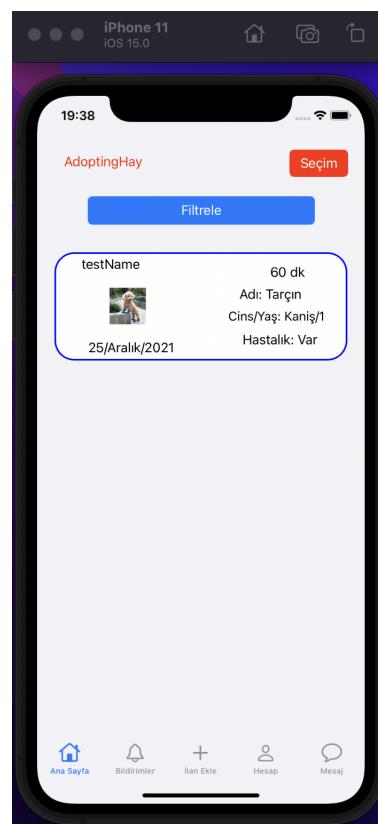
Sertifikayı da oluşturduğumuza göre artık bildirimleri gönderebiliriz. İndirdiğimiz Knuff programını açıyoruz. Choose identity yazan yerden bilgisayarımıza indirdiğimiz APNS sertifikasını seçiyoruz. Token yazan yere uygulamamızın console kısmından alıp kaydettiğimiz deviceTokenı yapıştırıyoruz. Priority kısmını değiştirmiyoruz ve Payload kısmına geliyoruz. Bu kısımda bilmemiz gereken birkaç şey var. APNS server'ına göndereceğimiz istek için bu alanda gördüğünüz gibi bir body oluşturuyor. Bu kısmı istediğimiz şekilde değiştirerek bildirimimizin içeriğiyle oynayabiliyoruz. [14]

3.5 Köpek Gezdirmeye

Kullanıcı bu hizmet ile köpeğini gezdimeye vakit bulmadığı zama bu hizmet üzerinden kullanıcı ilan açarak köpeğini köpeğini gezdirebilecek kişiler ile temasa geçebilmektedir. Ayrıca köpek gezdirmek istiyen kişiler burada ilan sahibi ile temasa geçerek para kazanma fırsatı bulabilmektedir.

3.5.1 İlan Ana Sayfa

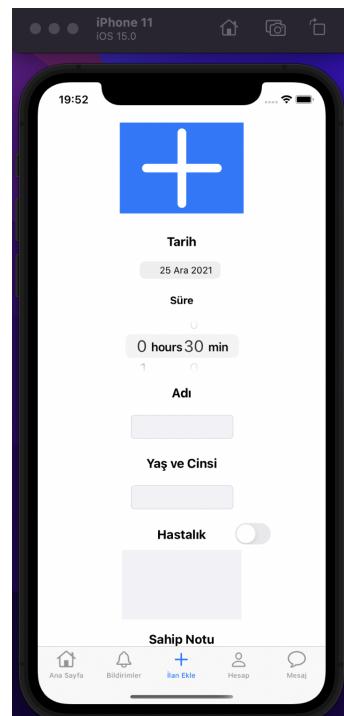
Köpek gezdirmek isteyen kullanıcı burada ilanlara göz atarak istediği ilanın detaylarına geçerek ilan hakkında dah fazla bilgi alabilmektedir. Eğer seçim ekranına geri dönmek isterse **Seçim** buttonuna tıklayarak seçim ekranına geri dönebilmektedir.



Şekil 40: Köpek gezdirmeye ilan ana sayfa

3.5.2 İlan Ekleme

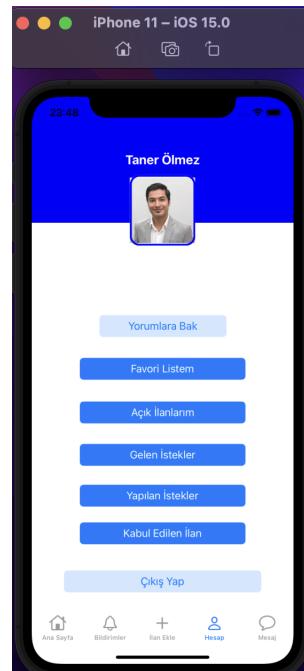
Köpeğini gezdirmeye çıkartmak isteyen kullanıcı köpeğinin resmini, ismini, yaşını,gibi bilgileri ve hangi tarihte ne kadar süre ile gezdiriceğini belirleyerek ilanı ekleyebilmektedir.



Şekil 41: Köpek gezdirmeye ilanı ekleme

3.5.3 Hesabım

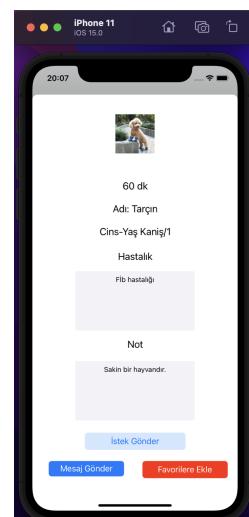
Hesabım kımında kullanıcı aktif ilanlarına , favori ilanlarına, gezdirmek için yaptığı isteklere ve ilanına gelen istekleri,kendisine yapılan yorumlara ve gezdirmede olan ilanına bakmak için gerekli buttonları seçerek görüntüleye bilmektedir. Ayrıca bir gezdirmeye işlemi yapmış ise verilen puanları burada görebilmektedir.



Şekil 42: Köpek gezdirme hesabım

3.5.4 İlan detayları

Kullanıcı ilanın detaylarına basıldığında ilan hakkında daha fazla bilgiye ulaşabilmektedir. Eğer isterse ilan sahibine mesaj gönderebilmektedir. İlanı favorilerine ekleyebilmektedir. Ayrıca köpeği gezdirmek için istekte bulunabilmektedir.



Şekil 43: Köpek gezdirme ilanı detayları

3.5.5 Köpek sahibi ile mesajlaşma

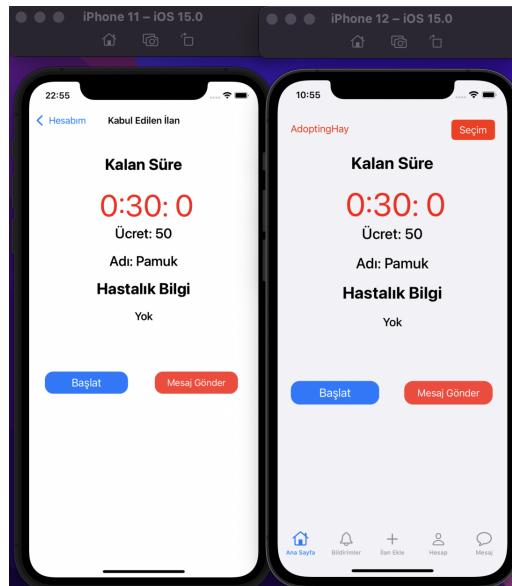
Köpeğini gezdirmeye çekartmış kişi istediği zaman köpeğini gezdirmeye çıkaran kişiye mesaj atabilmektedir. Dilerse karşı taraftan köpeğinin resmini isteye bilmektedir. Köpeği gezdirmeye çıkaran kişi mesaj yazılan kısımdan fotoğrafı seçerek fotoğrafı gönderebilmektedir.



Şekil 44: Köpek gezdirme mesajlaşma

3.5.6 Gezdirmeye işlemini takip sayfaları

Köpek gezdirmek isteyen kullanıcının köpek gezdirmeye işlemini kabul ettiği taktirde her iki kullanıcıya gösterilecek sayfa aşağıda yer almaktadır.



Şekil 45: Gezdime Takip Sayfası

Her iki kullanıcısında bu sayfaları farklı yerlerde gösterilmektedir. Köpek gezdirenen kişinin takip sayfası ilanların bulunduğu kısımda yer almaktadır. Bunun nedeni köpek gezdirenen kişinin sadece bir köpek gezdirmeye işlemini yapması sağlanmaktadır. Böylelikle köpek gezdirenen kişi farklı ilanlara istek gönderip farklı köpekleri aynı anda gezdirebilmesi engellenmektedir.

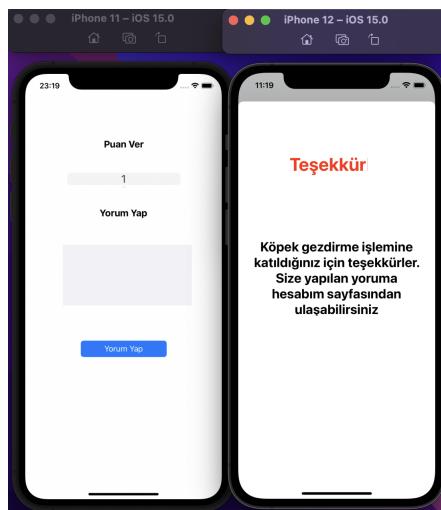
Köpek sahibinin takip sayfası ise hesabın bölümündeki gelen istekler kısmından bunu görüntüleyebilmektedir.

3.5.7 Gezdirmeye işleminin başlatılması

Gezdirmeye işleminin başlaması için ilk önce köpek sahibin gezdirmeye işlemini başlatması gerekmektedir. Süre sıfır olduğunda **Başlat** buttonu **Bitir** buttonuna dönmektedir. Burada da başlat buttonunda ki gibi ilk önce köpek sahibinin bitir buttonuna basması gerekmektedir.

3.5.8 Gezdirmeye işleminin bitmesi

Gezdirmeye işlemi bittiğinde köpeği gezdiren kullanıcıya bir **Teşekkür** ekranı gelmektedir. Teşekkür ekranından çıktığında ise tekrardan ana sayfa yönlendirilecektir. Köpek sahibi ise gezdirmeye işlemi bittiğinde köpeği gezidiren kullanıcıya yorum ve puan vermesi için bir ekran karşısına çıkmaktadır. Buradan yorum yapabilmektedir. Ayrıca 5 puana kadar puan verebilme imkanı bulunmaktadır.



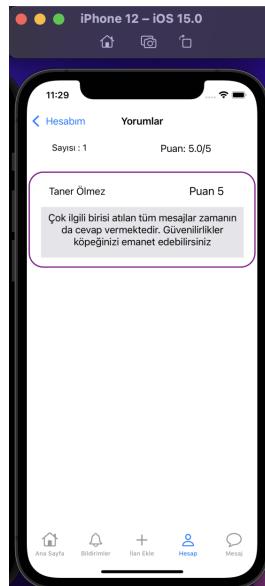
Şekil 46: Yorum ve teşekkür sayfası

3.5.9 Kullanıcıya gelen yorumlar sayfası

Gezdirmeye işlemi yapan kullanıcının gezdime işlemi sonunda kendisine yapılan yorumaları hesabım kısmında bunları görüntüleye bilmektedir.

3.5.10 Favorilerine ekleyen ve istek gönderen kullanıcı profili

Eğer bir kullanıcının ilanını başka bir kullanıcı favorilerine ekler ise kullanıcıya gelen bildirim **Bildirimlerim** sayfasında yer almaktadır. Buradan bildireme tıklayarak kullanıcıya ait yorumları ve puanları görüntülebilinmektedir.



Şekil 47: Kullanıcıya gelen yorumlar

3.6 Hesap Doğrulama(Authentication)

Bu bölümde uygulamaya giriş yapılmakta kullanılan yöntemler hakkında bilgi verilmektedir.

3.6.1 Eposta ile Yapılabilen İşlemler

Eposta ile yapılabilecek işlemler şu şekildedir

Eposta ile Üye Ol

Bu kısımda kullanıcıların eposta ile üye olabileceği bir alan yer almaktadır. Kullanıcı adı, soyad, eposta ve şifre bilgilerini girerek üyelik oluşturulabilmektedir.

Üyelik oluşturmada Google tarafından yönetilen **Firebase** hizmeti kullanılmaktadır. Proje dahil edilmesi için **CocoaPods** hizmetinin bilgisayarda kurulu olması gerekmektedir. Bunun içi terminal üzerinde **sudo gem install cocoapods** çalıştırılması gerekmektedir.



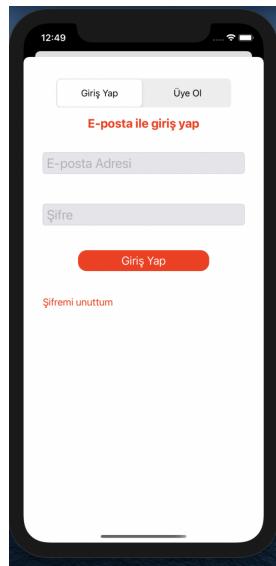
Şekil 48: Kullanıcıya gelen yorumlar

CocoaPods: CocoaPods, Objective-C, Swift ve harici kitaplıklarını yönetmek için standart bir format sağlayan RubyMotion gibi Objective-C çalışma zamanında çalışan diğer diller için bir uygulama düzeyinde bağımlılık yöneticisidir.

Kurulum tamamlandıktan sonra projenin dosyalarının bulunduğu yeri terminal üzerinde açılması gerekmektedir. Terminal üzerinde **pod init** komutu çalıştırılarak oluşturulan **Podfile** içeresine üyelik için gerekli olan **pod 'Firebase/Auth'** eklenir ve dosyanın kaydedilmesi gerekmektedir. En son olarak proje dosyalarının bulunduğu alan terminal üzerinde açılarak **pod install** komutu çalıştırılarak gerekli olan kitaplığın kurulumu gerçekleştirilmektedir.

Kurulumdan sonra artık **AdoptingHay.xcodeproj** ile değil **AdoptingHay.xcworkspace** ile devam edilmesi gerekmektedir. Proje **import Firebase** dahil edilmesi gerekmektedir. Firebase üzerinde oluşturulduğunda proje eklenmesi için verilen **GoogleService-Info.plist** ekledikten sonra onun içerisinde yer alan **REVERSECLIENTID** alınarak proje ayarlanlarında yer alan **Info** kısmında yer alan **URL Types** kısmına eklenmesi gerekmektedir.

Aşağıda ki kod bloğunda yer alan kodda kullanıcının girmiş olduğu eposta adresi ve şifre alınarak kullancının üyeliği **Firebase Authentication** üzerine kaydedilmektedir. Kaydedilme işlemi başarılı bir şekilde gerçekleştiriliyor ise **performs Segue** ile kendisine özel bir



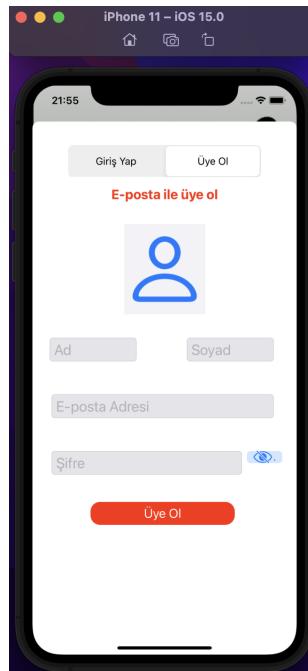
Şekil 49: Kullanıcı girişi

id vermiş olan bağlantı ile **Ana Sayfaya** geçiş sağlanmaktadır

```
1 @IBAction func singUpButtonAction(_ sender: Any) {
2     if textFieldName.text == "" || textFieldSurname.text == "" ||
3     textFieldEmail.text == "" || textFieldPassword.text == "" {
4         self.alertMessage(title: "Hata", text: "Boş kalan yerleri
5         lütfen doldurunuz")
6     } else {
7         Auth.auth().createUser(withEmail: textFieldEmail.text!,
8             password: textFieldPassword.text!)
9         { (authData, error) in
10             if error != nil {
11                 self.alertMessage(title: "Hata", text: error!.localizedDescription)
12             } else{
13                 self.performSegue(withIdentifier: "singUptoHomePage", sender: nil)
14             }}}}
```

Şekil 52: Eposta üye olma

Eğer işlemde bir sorunla karşılaşılursa aşağıda bulunan **alertMessage** fonksiyonu çalıştırılarak bir uyarı verilmektedir. Uyarı mesajı **let alertMessage = UIAlertController()** ile oluşturulmaktadır. Uyarı mesajı için bir button oluşturulmak istendiğinde ise **let alertButton = UIAlertAction()** kullanılmaktadır. Bu button uyarı mesajına **UIAlertAction.addAction(alertButton)** ile eklenmektedir. Uygulama üzerinde **self.present(alertMessage, animated: true)** ile eklenmektedir. Yanlış bir işlem yapıldığında karşılaşacak uyarı mesajı aşağıda yer almaktadır.



Şekil 50: Kullanıcı Üye Olma

```
1 func alertMessage(title:String, text:String) {
2     let alertMessage = UIAlertController(title: title, message: text
3 , preferredStyle: .alert)
4     let alertButton = UIAlertAction(title: "Okey", style: .cancel)
5     alertMessage.addAction(alertButton)
6     self.present(alertMessage, animated: true)
7 }
```

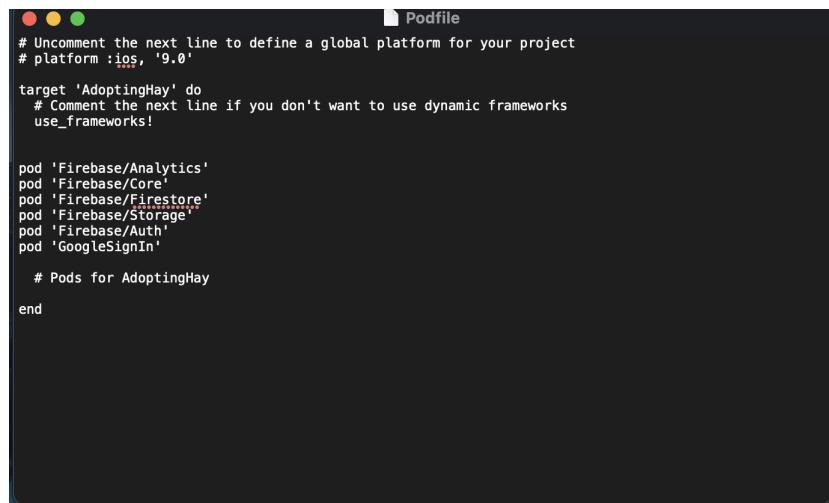
Şekil 54: Uyarı mesajı görüntüsü

Kullanıcı eposta bilgilerinin yer aldığı alan aşağıda bulunmaktadır.

Eposta ile Giriş Yapma

Bu kısımda kullanıcıların eposta ve şifre ile giriş yapabileceği bir alan yer almaktadır.

Aşağıda kullanıcının **Giriş Yap** buttonuna basıldığında kullancının girdiği eposta adresini ve şifresini alarak giriş yapabilmesini sağlayan kod bloğu yer almaktadır. Giriş yapma işlemi başarılı bir şekilde gerçekleştirir ise **performs Segue** ile kendisine özel bir id vermiş olan bağlantı ile **Ana Sayfaya** geçiş sağlanmaktadır. Eğer işlemde bir sorunla karşıla-



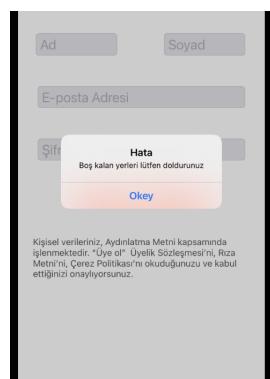
```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

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

  pod 'Firebase/Analytics'
  pod 'Firebase/Core'
  pod 'Firebase/Firestore'
  pod 'Firebase/Storage'
  pod 'Firebase/Auth'
  pod 'GoogleSignIn'

  # Pods for AdoptingHay
end
```

Şekil 51: Podfile



Şekil 53: Uyarı mesajı

Şıllırsa aşağıda bulunan **alertMessage** fonksiyonu çalıştırılarak biz bir uyarı verilmektedir.

```
1 @IBAction func signInButtonAction(_ sender: Any) {
2     if textFieldEmail.text == "" || textFiledPassword.text == "" {
3         self.alertMessage(title: "Hata", text: "Boş kalan yerleri
doldurunuz")
4     } else {
5         Auth.auth().signIn(withEmail: textFieldEmail.text!, password:
textFiledPassword.text!) { (authResult, error) in
6             if error != nil {
7                 self.alertMessage(title: "Hata", text: error!.localizedDescription)
8             } else {
9                 self.performSegue(withIdentifier: "signInToHomePage", sender: nil)
} }}
```

Şekil 56: Eposta ile giriş yapma

Identifier	Providers	Created ↓	Signed In	User UID
gulekengin0@gmail.com	G	Oct 24, 2021	Oct 24, 2021	hdw7EsT0ySvmcMw3HB059RuBJ...
enggilk@gmail.com	✉	Oct 24, 2021	Oct 24, 2021	iKcirAz0giPwgREHnK06jxolw0i2

Rows per page: 50 1 – 2 of 2

Şekil 55: Kullanıcı eposta bilgileri

3.6.2 Google ile Yapılabilecek İşlem

Kullanıcı eposta ile üye olmak veya giriş yapmak istemeyebilmektedir. Bu durumda uygulamaya kullanıcıların **Google** hesaplarıyla giriş yapabileceği bir alan bulunmaktadır. Google giriş kullanılabilmesi için **pod GoogleSignIn** kitaplığının **Podfile** dosyasına eklenmesi gerekmektedir. Daha sonra kullanılacağı alana **import GoogleSignIn** eklenmesi gerekmektedir.

Giriş yapılan bilinmesi için **ClientId** alınması gerekmektedir. **ClientId** **GoogleService-Info.plist** içerisinde alınabilinmektedir.

ClientId **let config = GIDConfiguration(clientID:)** ile alındıktan sonra aşağıdaki kod bloğu ile kullanım için ayarlanmaktadır.

```

1 GIDSIGNIN.sharedInstance.signIn(with: config, presenting: self) { user
  , error in
2     if error != nil {
3         print(error?.localizedDescription ?? "Hata")
4     }

```

Şekil 57: ClientId

Google ile devam etmek için **authentication** ile **accessToken** ve **idToken** bilgilerine ihtiyaç duyulmaktadır. Bu bilgiler **GoogleAuthProvider.credential** kısmı eklenmesi gerekmektedir.

```

1   let authentication = user?.authentication
2     let idToken = authentication?.idToken
3     let credential = GoogleAuthProvider
4       .credential(withIDToken: idToken!,
5                     accessToken: authentication!.accessToken)

```

Şekil 58: GoogleAuthProvider

Tüm gerekli bilgiler alındıktan sonra **Auth.auth().signIn()** fonksiyonu ile giriş işlemi gerçekleştirilecektir. Aşağıdaki kod bloğunda gerekli kod bloğunun tamamı yer almaktadır. Eğer işlem başarılı bir şekilde gerçekleştirilir ise **performs Segue** ile kendisine özel bir id vermiş olan bağlantı ile **Ana Sayfaya** geçiş sağlanmaktadır.

```

1 @IBAction func goGoogleButtonActions(_ sender: Any) {
2
3   let config = GIDConfiguration(clientID: "765162810048-
4     vekuds3fj6p66tkeij2p38cip6tv40gm.apps.googleusercontent.com")
5     GIDSignIn.sharedInstance.signIn(with: config, presenting: self) {
6       user, error in
7         if error != nil {
8           print(error?.localizedDescription ?? "Hata")
9         }
10        else {
11          let authentication = user?.authentication
12            let idToken = authentication?.idToken
13            let credential = GoogleAuthProvider
14              .credential(withIDToken: idToken!,
15                            accessToken: authentication!.accessToken)
16
17            Auth.auth().signIn(with: credential) { (result, error) in
18              self.performSegue(withIdentifier: "googleSignInToHomePage",
19                sender: nil)
20            }
21        }
22    }

```

Şekil 59: Goolgle ile giriş yap

3.6.3 Girişin Hatırlanması

Eğer bir kez giriş yapılır ise **SceneDelegate** içerisinde tutulmaktadır. Storybord da **let board = UIStoryboard(name: "Main", bundle: nil)** ulaşılmaktadır. Başlangıç sayfasının Ana Sayfa olması sağlanmaktadır.

```
1 func scene(_ scene: UIScene, willConnectTo session: UISceneSession,
2             options connectionOptions: UIScene.ConnectionOptions) {
3     let currentUser = Auth.auth().currentUser
4     if currentUser != nil {
5         let board = UIStoryboard(name: "Main", bundle: nil)
6         let tabBar = board.instantiateViewController(withIdentifier:
7             "tabBar") as! UITabBarController
8         window?.rootViewController = tabBar }
9     guard let _ = (scene as? UIWindowScene) else { return }
10 }
```

Şekil 60: Giriş işleminin hatırlanması

3.6.4 Çıkış İşlemi

Hesabım sayfasından çıkış buttonuna tıklandığında çıkış yapılp performSegue ile giriş sayfasına yönlendirilmektedir.

```
1 @IBAction func exitAccountButton(_ sender: Any) {
2     let firebaseAuth = Auth.auth()
3     do {
4         try firebaseAuth.signOut()
5         performSegue(withIdentifier: "exitToSingPage", sender: nil)
6     } catch let signOutError as NSError {
7         print("Error signing out: %@", signOutError)
8     } }
```

Şekil 61: Çıkış işlemi

3.7 Veritabanı

Hayvan sahiplenme ilan verileri ve köpek gezdirme verileri sürekli sürekli değişimle müsait olan verilerdir. Bu türlerin bir veritabanı içerisinde tutulması gerekmektedir. Bu veritabanında uygulamaya bağlanılarak uygulamanın güncel tutulması sağlanmaktadır.

3.7.1 Neden Firestore Database?

Bir çok veritabanı bulunmaktadır. Bunlardan bazıları **Mysql** ve **Microsoft SQL Server** gibileridir. Ayrıca bir sunucu kiralanması gerekmektedir. Bu işlemler hem masraflı hemde uğraştırıcı işlemlerdir.

Ücretsiz olan bizlere sunduğu Firebase'in **Firestore Database** hizmetini uygulamada kullanmaktadır.

The screenshot shows the Firestore database interface. At the top, there's a navigation bar with a home icon, followed by 'animalAdvert > Nr2fbuTB33JX5...'. Below this, there are three main sections: 'adoptinghay' (with '+ Start collection'), 'animalAdvert' (with '+ Add document' and a document ID 'Nr2fbuTB33JX54Ps67uh' expanded), and 'Nr2fbuTB33JX54Ps67uh' (with '+ Start collection', '+ Add field', and a detailed view of its fields: animalAge: 3, animalGenus: "Pug", animalImage: "https://www.petihiyac.com/Data/Edi (string) bakimi.jpg", animalKinds: "Köpek", animalName: "Garip", animalOwnerNote: "Doğduğundan beri benimle birlikteyiz. Eşimin alerjisi olduğundan yeni bir ev aramaktayız", and animalSick: "Yok").

Şekil 62: Firestore database

3.7.2 MVVM Modeli İçin Ortamın Hazırlanması

MVVM modelini oluştururken bize 3 çeşit dosya tipi gerektiği söylemişti. Bunlar;

- Model
- ModelView
- View

Bunlara nazaran bir dosya tipimiz ise **Service**dir. Bu service **Android** uygulama gerçekleştirilirken modelview içerisinde yazılmaktadır. Ama **iOS** uygulama gerçekleştirilirken ayrı bir alanda yazılmaktadır.

Service

Veritabanı isteklerimizi yapacağımız alandır. Service veritabanına yapılacak istekleri fonksiyon yazarak yapılmaktadır. Burada farklı bir teknik kullanılmaktadır. Bu **completion** blok kullanmaktadır. Completion blok veritabanından bir veri indirildiğinde bir tamamlama bloğu bize verilmektedir. Ayrıca **@escaping** kullanılması gerekmektedir. **Escaping:** Bir tamamlama bloğunda kullandığımızda fonksiyon çağrıldıktan sonra arguman döndürülebilen bir fonksiyondur. Sonuç olarak bir fonksiyon çağrılıp internetten bir veri indiriliyorsa ve sonucunu döndürüleceği bir işlem yapılacaksa hata bunu kendimiz döndürmek istiyorsak **escaping** kullanılması çok mantıklı bir yöntemdir.

Escaping kullanılması için öncelikle ne verisi döndürüleceği yazılması gerekmektedir. Bunun için döndürülecek veriye uygun **Model**'in ismi yazılması gerekmektedir. Eğer birden fazla veri döndürülecek ise Model isminin parantez içine alınması gerekmektedir. Opsiyenel yapılması gerekmektedir. Opsiyonel yapılmaz ise kesinlikle geleceğini anlaşılımaktadır. Bu da hata çıkarabilemektedir. Son olarak **comletion()** içerisinde döndürülecek verinin yazılması gerekmektedir.

```

1 var veriListesi = [Modelİsmi]()
2 func someFunctionWithEscapingClosure(completion: @escaping ([Modelİsmi
   ]?) -> Void) {
3 //veritabanı sorugusu sonucunda dönen veriler listeye eklendi
4 veriListesi.append(veri)
5 completion(veriListesi)
6 }

```

Şekil 63: Örnek service fonksiyonu

ViewModel

Burada **Service** indirilen veriyi burada işleyerek **View** gösterilmeye hazır hale getirmektedir. Presiplere göre her view kendine ait **ViewModel** olmak zorundadır. Bunun nedeni yazılımcıların viewmodelleri basitleştirmek ve sadeleştirmek istemesinden kaynaklanmaktadır.

Öncelikle dönen veri listesi için bir struct oluşturulması gerekmektedir. İlk önce struct içerisinde service içerisinde döndürülen verilerin sayısını döndürmek için bir fonksiyon oluşturulması gerekmektedir. İkinci bir fonksiyon ise verilerin çağrılp çağrılan verilerin **collectionView** içerisinde gösterilmesi için de bir fonksiyon oluşturulması gerekmektedir. Bu fonksiyon ise ikinci structda oluşturulan struct tipinde verinin döndürülmESİ gerekmektedir. Bu struct veri tiplerini döndürülecek veriye uygun **Model**'den almaktadır. Bunların kod gösterimi İlanların ve hayvan türlerinin veritabanından alınmasında gösterilmektedir.

Verilerin Viewde Gösterimesi

Artık verilerin view içerisinde gösterilmesi gerekmektedir. Bunu içi ilk önce verinin gösterileceği **View** içerisinde **private var veriList : VeriViewList!** eklenmesi gerekmektedir. Böylelikle.viewmodel içerisinde tanımlanan viewModelListe ulaşılmasını sağlanmaktadır.

3.7.3 İlanların Veritabanından Alınması

Öncelikle veritabanındaki verilere uygun bir model tanımlanması gerekmektedir. Bunun için ise **AnimalAdvert** adında bir class oluşturulması gerekmektedir

```
1 class AnimalAdvert {
2     var animalUid:String?
3     var animalImage:Data?
4     var animalName:String?
5     var animalKinds:String?
6     var animalAge:Int?
7     var animalSick:String?
8     var animalGenus:String?
9     var animalOwnerNot: String?
10    var animalSickInfo: String?
11    var userId : String?
12    var userName : String?
13
14    init(userId:String,userName : String,animalUid:String,animalImage:
15        Data,animalName:String,animalKinds:String,animalAge:Int,animalSick:
16        String,animalGenus:String,animalOwnerNot: String,animalSickInfo:
17        String) {
18        self.animalUid = animalUid
19        self.animalImage = animalImage
20        self.animalName = animalName
21        self.animalKinds = animalKinds
22        self.animalSick = animalSick
23        self.animalAge = animalAge
24        self.animalGenus = animalGenus
25        self.animalOwnerNot = animalOwnerNot
26        self.userId = userId
27        self.userName = userName
28    }
29}
```

Şekil 64: İlanlar model

Verilerin işlenmesi için **AnimalAdvertViewModel** adında bir viewModel dosyası oluşturulması gerekmektedir. Bu dosya içerisinde verilerin sayısını ve hangi veriye tıklandığını anlamak için oluşturulan structür aşağıda gösterilmektedir. **let animalAdvertList : [AnimalAdvert]** ile verilerin liste biçiminde olduğunu gösterilmektedir.

```

1 struct AnimalAdvertListViewModel {
2     let animalAdvertList : [AnimalAdvert]
3
4 // Listenin veri sayısı döndürülmektedir
5     func numberOfRowsInSection() -> Int {
6         return self.animalAdvertList.count
7     }
8 // Verilerin sırayla döndürülmesi
9     func animalAdvertAtIndex(_ index:Int) ->
10    AnimalAdvertViewModel {
11         let advert = self.animalAdvertList[index]
12         return AnimalAdvertViewModel(animalAdvert: advert)
13     }
14 }
```

Şekil 65: İlanlar viewmodel struct 1

Model içerisinde tanımlanan veriler alınıp daha anlaşılır tanımlamalar haline getirilmektedir.

```

1 struct AnimalAdvertViewModel {
2     let animalAdvert : AnimalAdvert
3     var uuid : String {
4         return self.animalAdvert.animalUid!   }
5     var name : String {
6         return self.animalAdvert.animalName! }
7     var age : Int {
8         return self.animalAdvert.animalAge!   }
9     var image:Data {
10        return self.animalAdvert.animalImage!   }
11    var genus : String {
12        return self.animalAdvert.animalGenus! }
13    var kinds : String {
14        return self.animalAdvert.animalKinds! }
15    var ownerNot: String {
16        return self.animalAdvert.animalOwnerNot! }
17    var sick : String {
18        return self.animalAdvert.animalSick!   }
19    var userName : String {
20        return self.animalAdvert.userName!   }
21    var userId : String {
22        return self.animalAdvert.userId!   }
23 }
```

Şekil 66: İlanlar viewmodel struct 2

Service içerisinde veritabanı sorgularının yazılacağını söylemiştir. Onlardan bir tanesi ilanların veritabanında alınmasıdır. İlk önce **import Firebase** yapılarak Firebasesede bulunan işlevlerin kullanılması sağlamak için eklenmesi gerekmektedir.

Çağrılan veriler bir listeye eklenmesi gerekmektedir. Ama bu liste **AnimalAdvert** tipinde olması gerekmektedir. Oluşturulma şekli **var animalAdvertList = [AnimalAdvert]()** olması gerekmektedir. Sıra fonksiyonu yazmaktadır. **Service** kısmında anlatılanlara uygun olarak fonksiyon yazılmaktadır. Yazılan fonksiyon içerisinde firebase verilerin çekilmesi için firestore ulaşılması için **let db = Firestore.firestore()** tanımlamasının yapılması gerekmektedir.

Sorguyu yazmaya **collection** ulaşarak yapılması gerekmektedir. İstenilen collection ismi yazılarak yapılır. Daha sonra **getDocuments** ile veriler çağrılır. Bu bize iki sonuc döner bunlar **error** ve **snapshot** dır. Error ile hata kontrolü yapılır. Hata varsa nil döndürülür. Yoksa for döngüsü ile veriler üzerinde gezinerek veriler **if let veriİsmi = document.get("veritabanındakiveriismi") as? String** ile if kontrolü yapılarak veri tip kontrolü yapılmaktadır. Resimlerde farklı olarak resimler url olarak kaydedildiği için **data** tipine dönüştürme işlemi yapılmaktadır. Tüm veriler kontrol edildikten sonra bunlar **AnimalAdvert** tipinde alınmaktadır. Sonra AnimalAdvert tipinde tanımlanan liste eklenip **completion(self.animalKindsList)** şeklinde geri döndürülür.

```
1  var animalAdvertList = [AnimalAdvert]()
2  func dowlandAnimalKindsFromFirestore(completion: @escaping ([
3      AnimalKinds]?) -> ()) {
4      let db = Firestore.firestore()
5      db.collection("animalKinds").getDocuments { snapshot, error in
6          if error != nil { completion(nil) }
7          else {
8              for document in (snapshot?.documents)! {
9                  if let animalKindName = document.get("animalKindName") as? String {
10                     if let animalKindsImage = document.get("animalKindsImage") as? String {
11                         let imageUrl = URL(string: "\\\(animalKindsImage)")!
12                         if let imageData = try? Data(contentsOf: imageUrl) {
13                             let animalKinds = AnimalKinds(animalKindName: animalKindName,
14                                 animalKindsImage: imageData)
15                             self.animalKindsList.append(animalKinds)
16                             completion(self.animalKindsList) } }
17             } } } }
```

Şekil 67: İlanlar sorğu

Artık verilerin gösterilme işleminin yapılması gerekmektedir. Öncelikle ilanların gösterilmesi için kullanılan collectionView bulunduğu **HomePage**'e kendi **AdvertListView-Model**'ı tanımlamamız gerekmektedir. Böylelikle içerisinde bulunan iki fonksiyonuda ulaşabilmekteyiz. Tanımlama şekli **private var animalAdvertList : AnimalAdvertList!** şeklinde yapılmaktadır.

Service içerisinde bulunan ilanları veritabanından alınmasın sağlayan sorgunun bulunduğu fonksiyonun çağrılmaması gerekmektedir. Çağrıldığı zaman bize bir veri listesi dönmektedir. Bu veri opsiyonel olarak dönmesinden dolayı bir if kontrolü yapılarak tekrar tanımlanmaktadır. Kendi tanımladığımız viewModel kullandığımızda bizde bir **AnimalAdvert** türünde bir veri listesi istenmektedir. Buraya bize web serviceden gönderilen veri listesinin tanımlanması gerekmektedir. Son olarak **self.animalAdvertCollectionView.reloadData()** yapılarak verilerin güncel kalması sağlanmaktadır

```
1 func getAnimalAdvertData() {
2     Service().dowlandAnimalAdvertFromFirestore { (animalA) in
3         if let animalA = animalA {
4             self.animalAdvertListViewModel =
5                 AnimalAdvertListViewModel(animalAdvertList: animalA)
6             self.animalAdvertCollectionView.reloadData() }
7 }
```

Şekil 68: İlanların service çağrılmaması

```
1         if collectionView == self.animalAdvertCollectionView {  
2             return self.animalAdvertListViewModel == nil ? 0 : self.  
animalAdvertListViewModel.numberOfRowsInSection()  } }
```

Şekil 69: İlanların veri sayısının alınması

Veri listesi viewModel gönderildikten sonra verilerin collectionView içerisinde sırayla yazılması için **AnimalAdvertListViewModel** içerisinde bulunan **animalAdvertAtIndex**

fonksiyonu çağrılp kendisine bir **index** gönderilmesi gerekmektedir. Bu **index** **collectView** tanımlandığında hazır fonksiyon olarak gelen ve verilerin sırasıyla verilerin yazılmasını sağlayan fonksiyondan alınmaktadır. Yazıldıktan sonra bir **advertViewModel** döndürülecektir. Döndürülen **advertViewModel** ile verilere ulaşarak gerekli yerlere yazılması sağlanmaktadır. Birden fazla **collectionView** olduğunda kontrol yapılmaktadır.

```

1 func collectionView(_ collectionView: UICollectionView,
2 cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
3     if collectionView == self.animalAdvertCollectionView {
4         let cell = animalAdvertCollectionView.dequeueReusableCell(
5             withReuseIdentifier: "animalAdvertCell", for: indexPath) as!
6             AnimalAdvertCVC
7     let advertViewModel = self.animalAdvertListViewModel.
8         animalAdvertAtIndex(indexPath.row)
9         cell.animalAdvertImage.image = UIImage(data:
10            advertViewModel.image)
11         cell.animalAdvertNameLabel.text = "Adı: \(advertViewModel.name)"
12         cell.animalAdvertKindsLabel.text = "Cins: \(advertViewModel.kinds)"
13         cell.animalAdvertAgeLabel.text = "Yaş: \(String(advertViewModel.age)
14 )"
15         cell.animalAdvertSickLabel.text = "Hastalık: \((
16            advertViewModel.sick)"
17         // cell için görüntü düzenlemesi
18         cell.layer.cornerRadius = 25
19         cell.layer.borderWidth = 2
20         return cell    }
21 }
```

Şekil 70: İlanların verinlerin yerlerine

3.7.4 İlan Arama Yapma

Kullanıcı ilanlar arasında arama yapmak isteyebilmektedir. Bunun için Ana Sayfanın başına **Searchbar** eklendiği görülmektedir. Bu arama sadece hayvan cinsleri arasında yapılmasını sağlamaktadır. Bunun nedeni zaten diğer özellikler için bir filtreleme işlemi zaten yapılmıştır.

Searchbar eklendiğinde onunla birlikte gelen fonksiyon içerisinde bize arama kısmına yazıldığından her yazılma işlemini veren değeri kontrol ederek işe başlanmalıdır. Eğer arama kısmı boş ise **getAnimalAdvert()** fonksiyonu çağrılarak bütün verilerin gelmesi sağlanmaktadır. Bu fonksiyon service üzerinden veritabanına istek yapmaktadır. Ancak search kısmı dolu ise service üzerinden **dowlandAnimalAdvertSearchBarTextFromFi-**

restore fonksiyonuna yazılan değer gönderilerek bu fonksiyon filtreleme işlemini yaparak sonuçları döndürmektedir. Kendi tanımladığımız **viewModel** kullandığımızda bizden bir **AnimalAdvert** türünde bir veri listesi istenmektedir. Buraya bize web serviceden gönderilen veri listesinin tanımlanması gerekmektedir. Son olarak **self.animalAdvertCollection** **View.reloadData()** yapılarak verilerin güncel kalması sağlanmaktadır

```

1 extension HomePageVC: UISearchBarDelegate {
2     func searchBar(_ searchBar: UISearchBar, textDidChange searchText:
3     String) {
4         print("Arama \\" + searchText + ")")
5         if searchText == "" {
6             getAnimalAdvertData()
7         } else {
8             Service().dowlandAnimalAdvertSearchBarTextFromFirestore(
9             getSeacrhBarText: searchText) { animalA in
10                 if let animalA = animalA {
11                     self.animalAdvertListViewModel =
12                     AnimalAdvertListViewModel(animalAdvertList: animalA)
13                     self.animalAdvertCollectionView.reloadData()
14                 } } } } }
```

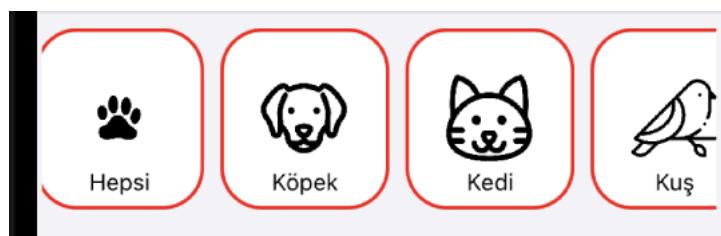
Şekil 71: Service arama isteği yapılması

Service içerisinde arama yapma fonksiyonu **dowlandAnimalAdvertSearchBarTextFromFirestore** bu şekilde tanımlanmıştır. Arama isteği yapılması kısmına gönderilen araması yapmak istenen veri alınır ve tüm verilerin çağrıılma kısmında kullanılan sorgu ve tüm verilerin kontrol edime işlemi yapılması gerekmektedir. Ancak **AnimalAdvert** tipinde alınmadan önce **animalGenus.lowercased().contains(getSeacrhBarText.lowercased())** if ile gelen arama verisinin hayvan cinsleri içerisinde bulunma sorusu yapılmaktadır. Burada **lowercased()** tüm yazılan verinin küçük harflere dönüştürülmesi ve **contains()** ise içine yazılan verinin yazıla veri içerisinde bulunup bulunmadığını **true** ya da **false** şeklinde döndürmektedir. Son olarak **AnimalAdvert** tipinde tanımlanan liste eklenip **completion(self.animalKindsList)** şeklinde geri döndürülmemektedir

Şekil 72: Veritabanına aram isteği yapılması

3.7.5 Hayvan Türleri Arasında Filtreleme İşlemi

Kullanıcı hayvan türleri arasında da filtreleme işlemi yapmak isteye bilmektedir. Bunun için İlan Ana Sayfada istediği türü seçerek bu işlemi gerçekleştirebilmektedir.. Türü seçtikten sonra tekrar diğer tüm ilanları göstermek için **Hepsi** kısmını basarak bu işlemi gerçekleştirebilmektedir.



Şekil 73: Hayvan türleri

Öncelikle hayvan türüne tıklandığının bilinmesi gerekmektedir. Bunun için hayvan tülerinin gösterilmesinde kullanılan collectionView ile gelen fonksiyon kullanılmaktadır. İki adet collectionview kullanıldığından dolayı if ile kontrol yapılması gerekmektedir. Kendi

oluşturduğumuz **AnimalKindsListViewModel** ile AdvertViewModeldan **animalKindsAtIndex(indexPath.row)** fonksiyonuna hangi hayvan türünü tıkladıysak onun sırasını gönderilmektedir. ViewModel bu verinin AnimalAdvert tipinde verisini göndermektedir. Bunun sonucunda da **kindsViewModel.name** gelen hayvan türünün ismini alınmaktadır. Eğer **Hepsi** gelir ise tüm veriler gösterilmektedir. Hepsi değil ise **Service** içerisinde bulunan **dowlandAnimalAdvertKindsFilterFromFirestore** istek yapılarak gelen hayvan türü gönderilmektedir. Kendi tanımladığımız viewModel kullandığımızda bizden bir **AnimalAdvert** türünde bir veri listesi istenmektedir. Buraya bize web serviceden gönderilen veri listesinin tanımlanması gerekmektedir. Son olarak **self.animalAdvertCollectionView.reloadData()** yapılarak verilerin güncel kalması sağlanmaktadır

```

1 func collectionView(_ collectionView: UICollectionView,
2 didSelectItemAt indexPath: IndexPath) {
3     if collectionView == self.animalKindsCollectionView {
4         let kindsViewModel = self.animalKindsListViewModel.
5         animalKindsAtIndex(indexPath.row)
6         let getAnimalKinds = kindsViewModel.name
7         if getAnimalKinds == "Hepsi" {
8             self.getAnimalAdvertData()
9             Service().dowlandAnimalAdvertKindsFilterFromFirestore(
10                getAnimalKinds: getAnimalKinds) { animalA in
11                    if let animalA = animalA {
12                        self.animalAdvertListViewModel =
13                            AnimalAdvertListViewModel(animalAdvertList: animalA)
14                        self.animalAdvertCollectionView.reloadData()
15                    } }
16    }
17 }
```

Şekil 74: Service havyan türü filtreleme isteği yapılması

Service üzerinde isteği yapılan **dowlandAnimalAdvertKindsFilterFromFirestore** fonksiyonu gelen hayvan türünü alarak **whreField** ile sorgulayarak bu hayvan türüne uygun olan ilanları geri döndürmektedir.

```

1 func dowlandAnimalAdvertKindsFilterFromFirestore(getAnimalKinds:
2 String, completion: @escaping ([AnimalAdvert]?) -> ()) {
3     let db = Firestore.firestore()
4     animalAdvertList = [AnimalAdvert]()
5     db.collection("animalAdvert").whereField("animalKinds",
6         isEqualTo: getAnimalKinds).getDocuments { snapshot, error in
7         if error != nil { print("Kinds filter hata var") }
8         else { // Veritabanından ilan sorğu ile aynı } }
```

Şekil 75: Havyan türü filtreleme sorgusu yapılması

3.7.6 Yaşa Göre Sıralame ve Hastalık Filtresi

İlan Ana Sayfa üzerinde **Filtrele** buttonuna basıldığında **UIAlertController** yardımı ile bir tane **actionSheet** oluşturulacaktır. Bu alertController ile actionSheet de başlık ve bir tane mesaj ekleye bilmektedir. ActionSheet üzerine **Hastalık Yok**, **Büyükten Küçüğe**, **Küçükten Büyüge** ve actionSheet çıkış işlemi için **İptal** buttonlarının eklenmesi gerekmektedir

İptal buttonun türü **cancel** olarak tanımlanarak tıklandığında actionSheet kapanması sağlanmaktadır.

En son olarak bu butonların `actionController.append ()` içersine sıraya yazılır `actionController eklenir`. ActionController çalışması için `view eklenmesi` gerekmektedir. Bu işlemde `self.present(actionController, animated: true, completion: nil)` işlemi ile gerçekleştirilmektedir



Şekil 76: Filtreleme ve sıralama

Hastalık yok滤镜

Kullanıcı sahipleneceği hayvanda bir hastalık istemeye bilmektedir. Kullanıcı hastalık yok kısmına tıklandığında actionSheet üzerine eklenen **Hastalık Yok** buttonunun fonksiyonu çalışmaktadır. Bu fonksiyon service içerisinde bulunan **dowlandAnimalAdvertSickBo-**

olFromFirestore fonksiyonuna istek yapmaktadır. Bu istek sonucunda geri dönen listeyi kendi tanımladığımız **animalAdvertListViewModel** gönderme işlemi yapmaktadır. Verilerin güncel kalması için **self.animalAdvertCollectionView.reloadData()** kullanılması gerekmektedir.

```
1 let sickBool = UIAlertAction(title: "Hastalık yok", style: .default) {
2     action in
3         Service().dowlandAnimalAdvertSickBoolFromFirestore { (
4             animalA) in
5                 if let animalA = animalA {
6                     self.animalAdvertListViewModel =
7                         AnimalAdvertListViewModel(animalAdvertList: animalA)
8                     self.animalAdvertCollectionView.reloadData()
9                 }
10            }
11        }
12    }
```

Şekil 77: Servide hayvan hastalık yok filtre isteği

Service tarafında hayvan türü filtreleme işleminin aynısı yapılmaktadır sadece **whereField** kısmı whereField("animalSick", isEqualTo:"Yok") bu şekilde yazılması gerekmektedir

Yaşlarını Büyüktен Küçüğe Sıralama

Kullanıcı isterse ilanda bulunan hayvanaların yaşlarını büyükte küçüğe sırala işlemi gerçekleştirebilmektedir. Kullanıcı büyükten küçüğe kısmına tıklandığında actionSheet üzerine eklenen **Büyükten Küçüğe** buttonunun fonksiyonu çalışmaktadır. Bu fonksiyon service içerisinde bulunan **dowlandAnimalAdvertMaxtoMinFromFirestore** fonksiyonuna istek yapmaktadır. Bu istek sonucunda geri dönen listeyi kendi tanımladığımız **animalAdvertListViewModel** gönderme işlemi yapmaktadır. Verilerin güncel kalması için **self.animalAdvertCollectionView.reloadData()** kullanılması gerekmektedir.

Service tarafında hayvan türü filtreleme işleminin aynısı yapılmaktadır sadece **whereField** kısmı kaldırılarak order(by: "animalAge", descending: true) bu şekilde yazılması gerekmektedir. Order fonksiyonu ile animalAge yaşı ulaşıp **descencing:true** yapılarak

azalma işlemi gerçekleştirilmektedir.

Yaşlarını Küçükten Büyüye Sıralama

Bu kısım neredeyse büyükten küçüğe kısmı ile aynıdır. Sadece service içerisinde bulunan **dowlandAnimalAdvertMintoMaxFromFirestore** istek yapılmaktadır. Service tarafından ise **descencing:false** azalma işlemini tersine çevirmektedir.

3.7.7 İlan Eklemeye

İlan eklem buttonuna basıldığında firebase bulunan Storge da **media** adında bir dosya alanı oluşturulacak ve buraya eklenmek istenen resimler bizim tanımladımız özel id lerel eklenecektir. Daha sonra buradan resimlerin url alınarak firestore ekleme işlemi gerçekleştirilecektir. Kimin hangi fotoğrafı eklediğini bilinmesi için Auth ile kullanıcını adı ve kullanıcıid sine ulaşarak bu verilerde veritabanına eklenecektir.

gs://adoptinghay.appspot.com > media			
<input type="checkbox"/>	Name	Size	Type
<input type="checkbox"/>	168813DE-B187-42E6-BF3F-6861AFD13516.jpeg	676.28 KB	application/octet-stream
<input type="checkbox"/>	2A3CD244-29D6-48B4-B778-D672702A8C30.jpeg	693.38 KB	application/octet-stream
<input type="checkbox"/>	2F5C596B-F5CD-4124-A952-67F47AA0ED68.jpeg	676.28 KB	application/octet-stream
<input type="checkbox"/>	3EA40BC8-5D74-4D86-88C2-57A8EED1B618.jpeg	816.13 KB	application/octet-stream
<input type="checkbox"/>	60ADC8F6-4595-43AC-9C81-120EB89EEF2E.jpeg	1.01 MB	application/octet-stream
<input type="checkbox"/>	74B74F9D-8339-425B-9617-8538BC785F9F.jpeg	816.13 KB	application/octet-stream
<input type="checkbox"/>	7700C729-E3DB-4F46-AA21-BCA59A430259.jpeg	816.13 KB	application/octet-stream

Şekil 78: Resim stroge

İlk önce boş alanların kontrolü yapılması gerekmektedir. Eğer gerekli yerler dolu ise veriler opsiyonellikten çıkarılması için if kontrolü yapılması gerekmektedir. Daha sonra detaylarda bulunan resimler stroge eklenir ve url alınarak sadece üçünün bulunduğu ve kendimizin tanımladığı bir liste eklenir. İlan sayfasında bulunacak resim ise sadece ayrı bir url olacak şekilde alınır . Tüm bu işlemler sorunsuz bir şekilde gerçekleşir ise tüm

veriler **AnimaAdvert** tipinde alır ve kullanıcı idsi ile birlite Service içerisinde bulunan **addAdvertToFirebase** fonksiyonuna gönderilir. En son olarak Ana Sayfaya geçmek için **self.tabBarController?. selectedIndex = 0** işlemi kullanılır.

```
1 if let imageOneData = addAdvertImageOne.image?.jpegData( compressionQuality: 0.5) {  
2 let imageOneDataUuid = UUID().uuidString  
3 let imageOneReference = medinaFolder.child("\(imageOneDataUuid).jpeg")  
4 imageOneReference.putData(imageOneData, metadata: nil) { metaData,  
    error in  
5 if error == nil {  
6 imageOneReference.downloadURL { url, error in  
7 let imageOneUrl = url?.absoluteString  
8 self.imageDetailId.append(imageOneUrl!) } }  
9 else {  
10 print("Resim hatası \(error?.localizedDescription)") } } }
```

Şekil 79: Resmin storage eklenmesi örneği

Service içerisinde tanımlanan **addAdvertToFirebase** fonksiyonuna gelen kullanıcı idsi ve eklenmek istenen ilan ait bilgiler veritabanına eklenmek üzere hazır hale getirilir. Her bir ilana özel olan bir id alınma işlemi gerçekleştirilir. Bu işlem **let advertId = UUID()** ile yapılır

Veriler **docData** adıyla tanımlanan bir veri liste tipinde tanımlanır. İlk önce veritabanındaki veri isimleri aynı olacak şekilde isimler yazılmaktadır. Uygun veri ismine uygun veri karşısına yazılır.

Bütün ilanların bulunduğu kısma eklemek için sorgu yazılır ve her ilan için özel bir id hazırlandığı belirtilmiştir. Bu id içerisinde bu veriler eklenmektedir

Sadece kullanıcının kendi ilanlarını görebilmesi için yeni bir alan açılmaktadır. Bu **userList** şeklinde olmaktadır. Daha sonra kullanıcını kendisine ait olan kullanıcı idsi ile bir alanda açılmaktadır. Burada kullanıcının ilanlarını eklemek için **advertList** adında bir alan oluşturularak buraya eklediği ilanlar eklenmektedir.

```

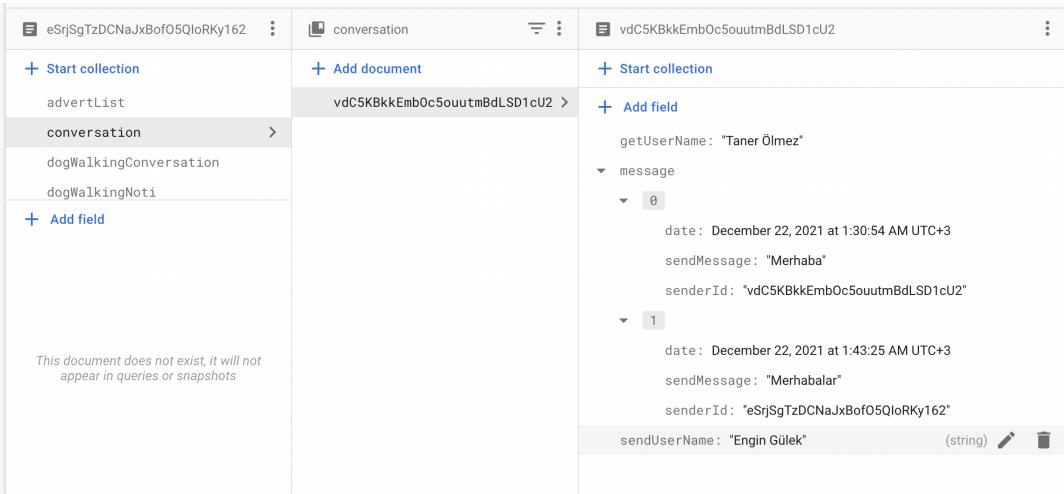
1 func addAdvertToFirebase(uuid:String,advert:Advert) {
2     let db = Firestore.firestore()
3     let advertId = UUID()
4     let docData : [String:Any] = [
5         "userId": uuid,
6         "userName" : advert.userName!,
7         "animalGenus" : advert.animalGenus!,
8         "animalAge" : advert.animalAge!,
9         "animalImage" : advert.animalImage!,
10        "animalImageDetails" : advert.animalImageDetails!,
11        "animalKinds" : advert.animalKinds!,
12        "animalName" : advert.animalName!,
13        "animalOwnerNot" : advert.animalOwnerNot!,
14        "animalSickInfo" : advert.animalSickInfo!,
15        "animalSick" : advert.animalSickBool ]
16 db.collection("animalAdvert").document("\(advertId)").setData(docData) {
17     err in if err != nil {
18         print("Ekleme Hata var \(err?.localizedDescription)")
19     } else { print("Ekleme işlemi başarılı advertList") } }
20     db.collection("userList").document("\(uuid)").collection("advertList")
21 .document("\(advertId)").setData(docData) {
22     err in
23     if err != nil {
24         print("Ekleme Hata var \(err?.localizedDescription)")
25     } else { print("Ekleme işlemi başarılı user list") } }

```

Şekil 80: İlan ekleme sorusu

3.7.8 Kullanıcıların Mesajlaşması

Bir kullanıcı ilan hakkında daha fazla bilgi almak için sahibne mesaj yolu ile ileyişime gecebileceğini belirtilmiştir. Mesajların kullanıcıları iletilmesi için mesajların veritabanına kaydedilmesi gerekmektedir. Veritabanında **userList** içerisinde kullanıcıyı **userId**'si ile oluşturulan döküman içerisindeki **conversation** koleksiyonu içerisinde bulunan mesajlaşan kullanıcıya ait olan userId ile oluşturulan doküman içerisinde bu mesaj verileri yerleştirilecektir.



Şekil 81: Mesaj veritabanı

Öncelikle kullanıcıların mesaj kısımlarına bağlantı kurulması gerekmektedir. Eğer boş ise hangi kullanıcı ile mesajlaşılıyor ise o kullanıcıye ve kendisine ait bilgilerin kaydedilmesi gerekmektedir.

Daha önceden bir mesajlaşma durumu var ise tekrardan kullanıcıların bilgilerinin gönderilmesine gerek yoktur. Sadece **message** verisinin gönderilmesi gerekmektedir. İlk önce gönderilen mesajın üzerine yeni gönderilen mesajı eklemek için **FieldValue.arrayUnion** (**[message]**) kullanılması gerekmektedir.

```

1 func sendMessage(sendUserId:String,sendUserName:String,sendMessage:
2     String,getUserName:String) { db.collection("userList").document(
3         authUserId).collection("conversation").document(sendUserId).
4         getDocument { snapshot, error in
5             if snapshot?.get("message") == nil
6                 let message : [String:Any] = [
7                     "date":Date(),
8                     "sendMessage":sendMessage,
9                     "senderId":authUserId]
10                if let getUserName = getUserName as? String{
11                    let sendMessageData : [String:Any] = [
12                        "sendUserName":sendUserName,
13                        "getUserName":getUserName,
14                        "message": [message]]
15                    let getMessageData : [String:Any] = [
16                        "sendUserName":getUserName,
17                        "getUserName":sendUserName,
18                        "message": [message]]
19                    db.collection("userList").document(authUserId).collection("conver
20                    sation").document(sendUserId).setData(sendMessageData)
21                    db.collection("userList").document(sendUserId).collection("conver
22                    sation").document(authUserId).setData(getMessageData) } }
23                else {
24                    let message : [String:Any] = [
25                        "date":Date(),
26                        "sendMessage":sendMessage,
27                        "senderId":authUserId]
28                    db.collection("userList").document(authUserId).collection("conver
29                    sation").document(sendUserId).updateData([
30                        "message": FieldValue.arrayUnion([message])])
31                    db.collection("userList").document(sendUserId).collection("conver
32                    sation").document(authUserId).updateData([
33                        "message": FieldValue.arrayUnion([message])) } } } }

```

Şekil 82: Mesajın veritabanına eklenmesi

Kullanıcı eğer mesajlaşmayı silmek ister ise mesajlaşılan kişilerin isimlerinin yazıldığı kullanıcı listesinden kullanıcı ismi sola doğru sürükleyerek sadece kendi veritabanında bulunan mesajlaşmaları silebilmektedir. Mesajlaşan kişi kendisi silmediği müddetçe kendi veritabanında bu mesajlar kalmaya devam etmektedir.

```

1 func deleteMessage(id:String){
2     let db = Firestore.firestore()
3     if let authUserId = Auth.auth().currentUser?.uid {
4         db.collection("userList").document(authUserId).collection("conversation")
5             .document(id).delete()
6     } }

```

Şekil 83: Mesajın silinmesi

3.7.9 Köpek Gezdirmeye İlanı Eklenmesi

Köpeğini gezdirmeye çıkartmak isteyen kullanıcı ilan ekleyebileceği söylenmiştir. Eklme işlemi ilk önce önemli verilerin girildiğinin kontrolü yapılması gerekmektedir. Eğer veriler doğru girilmiş ise ilk önce resim **stroge** eklenerken bir **url** adresi oluşturulmaktadır. Daha sonra girilen tarih ve süre bilgilerinin alınması gerekmektedir. Süre bilgisi **timePicker**'dan alındıktan sonra 1800 bölünmesi gerekmektedir. Çünkü süre 30 dakika eklenerken arttığı için her 30 dakika 1800 denk gelmektedir. 1800 bölündüğünde kaç adet 30 olduğu öğrenilecek ve daha sonra adet sayısı 30 ile çarpılarak gezdirilmek istenen dakikaya ulaşılmaktadır. Daha sonra bu tüm veriler bir model üzerinden Köpek gezdirme servisi olan **DogService**' iian ekleme isteği yapmaktadır. İstek doğru çalışır ise **self.tabBarController?.selectedIndex = 0** ilan ana sayfaya geçilecektir.

```
1 let date = self.datePicker.date
2 var time = self.timePicker.countDownDuration
3 time = (time/1800)*30
4 let advert = DogWalkAddAdvert(userId:authId,userName: userName,
      addImage: imagedUrl!, time: date, timeRange: Int(time) , animalName
      : name, ageAndGenus: ageGenus, sickBool: self.sickBoolText.text!,
      sickInfo: sickInfo, ownerNote: ownerNote)
5 DogWalkingService().addDogWalkingAdvertToFirebase(advert: advert)
6 self.tabBarController?.selectedIndex = 0
```

Şekil 84: DogService ilan ekleme isteği

3.7.10 Köpek Gezdirmeye İsteğinin Yapılması

Kullanıcı köpek gezdirmeye istediği zaman **DogService**'de bulanan **sendMessage** fonksiyonuna ilan bilgilerini gönderilmektedir. Burada bilgiler bir **dictionary** içerisinde tutulmaktadır. Öncelikle istediği yapan kişinin veritabanında **isentRequest** dökümanı içeresine yerleştirilmektedir. Ayrıca istek yapılan kişinin veritabanında **incomingRequest** dökümanı içeresine yerleştirilecektir.

```
1 func sendRequest(getAdvert:DogwalkingAdvert) {
2     let db = Firestore.firestore()
3     if let userId = Auth.auth().currentUser?.uid {
4         let docData : [String:Any] = [
5             "animalImage" : getAdvert.advertImage!,
6             "animalName" : getAdvert.advertAnimalName!,
7             "timeRange": getAdvert.advertRange!,
8             "sickBool" : getAdvert.advertAnimalsick!,
9             "ageAndGenus" : getAdvert.advertAnimalKindsandAge!,
10            "ownerNote" : getAdvert.ownerNote!,
11            "sickInfo":getAdvert.sickInfo!,
12            "userName" : getAdvert.userName!,
13            "userId" : getAdvert.userId!,
14            "situation" : getAdvert.situation!,
15            "date" : getAdvert.dateEvent!,
16            "sendRequestId" : userId ]
17        db.collection("userList").document(getAdvert.userId!).collection("incomingRequest").document(getAdvert.advertId!).setData(docData)
18        db.collection("userList").document(userId).collection("isentRequest") .
            document(getAdvert.advertId!).setData(docData) }
```

Şekil 85: Veritabanına gezdirmeye isteğini kaydetme

3.7.11 İsteğin Kabul Edimesi Ve Edilmemesi

Köpek sahibine yapılan gezdirme isteği köpek sahibinin gelen istekler kısmında yer alacaktır. Burada ilanı sola çekerek isteği kabul edebilmektedir ya da gelen isteği silebilmektedir

Köpek sahibi eğer isterse isteği kabul etmeyeilmektedir. Eğer kabul etmez ise hem köpek sahibinin hemde isteğli yapan kişinin veritabanında bu istek silinmektedir

```
1 func removeComingRequest(id:String, sendUserId:String) {
2     let db = Firestore.firestore()
3     if let authUserId = Auth.auth().currentUser?.uid {
4         db.collection("userList").document(authUserId).collection("incomingRequest").document(id).delete()
5         db.collection("userList").document(sendUserId).collection("isentRequest").document(id).delete()
6     } }
```

Şekil 86: Köpek gezdirme isteğinin kabul edilmemesi

Ayrıca isteğli yapan kişiye isteğinin kabul edilmediğine dair bir bildirim gönderilmesi için veritabanına bu bildirim eklenecektir.

```
1     let removeAction = UIContextualAction(style: .destructive, title:
2         "X") { contextuaAction, view, boolValue in
3     if let advertUid = self.incomingRequestViewModeList.
4         incomingRequestAtIndex(indexPath.row).id as? String{
5         if let sendUserId = self.incomingRequestViewModeList.
6             incomingRequestAtIndex(indexPath.row).sendId as?
7             String {DogWalkingService().removeComingRequest(id: advertUid,
8                 sendUserId:sendUserId) self.incomingRequestViewModeList.
9                 incomingList.remove(at: indexPath.row)
10 self.requestListTableView.deleteRows(at: [indexPath], with: .fade)
11 if let authUserName = Auth.auth().currentUser?.displayName {
12         let notiDate : [String:Any] = [
13             "sendUserName": authUserName, // gönderen kişi
14             "notiTitle": "AdoptingHay",
15             "notiSubtitle": "Gezdirme isteğiniz kabul edilmedi",
16             "notiMessage": "\u{202a}(authUserName) gezdirme isteğinizi
17             kabul etmedi",
18             "getUserName": "", // alan kişi
19             "getUserId" : sendUserId ]
20 DogWalkingService().addDogWalkingAdvertNoti(notiData: notiDate,
21             getUserId: sendUserId) } } } }
```

Şekil 87: Köpek gezdirme isteğinin kabul edilmemesi bildirim

Gezdirmeye isteği kabul eder ise gezdiren kişinin **sendAcceptAdvert** dökümanına ve hayvan sahibinin ise **comingAcceptAdvert** dökümanına eklenmektedir. Daha sonra istek yapan kişinin yapılan istek dökümandan, hayvan sahibinin gelen istekler dökümanında, isteği yapan kişinin eğer favori listesinde bu ilan var ise favoriler dökümanından ve bütün ilanların bulunduğu dökümandan bu ilan silinmektedir. İsteğin kabul edildiğine dair bildirim ise isteğin kabul edilmemesinde kullanılan bildirim ile aynı şekilde oluşturulmaktadır.

```

1 func acceptToRequestToFirebase(getAdvert:Request) {
2     if let authUserId = Auth.auth().currentUser?.uid {
3         if let sendId = getAdvert.sendId {
4             if let advertId = getAdvert.advertId {
5                 let docData : [String:Any] = [
6                     "animalImage" : getAdvert.advertImage!,
7                     "animalName" : getAdvert.advertAnimalName!,
8                     "timeRange": getAdvert.advertRange!,
9                     "sickBool" : getAdvert.advertAnimalSick!,
10                    "ageAndGenus" : getAdvert.
11                    advertAnimalKindsandAge!,
12                    "ownerNote" : getAdvert.ownerNote!,
13                    "sickInfo":getAdvert.sickInfo!,
14                    "userName" : getAdvert.userName!,
15                    "userId" : getAdvert.userId!,
16                    "date" : getAdvert.dateEvent!,
17                    "sendRequestId" : getAdvert.sendId! ]
18 db.collection("userList").document(authUserId).collection(""
19   comingAcceptAdvert").document(advertId).setData(docData)
20 db.collection("userList").document(sendId).collection("sendAcceptAdvert"
21   ).document(advertId).setData(docData)
22 db.collection("userList").document(sendId).collection(""
23   dogWalkingFavoriList").document(advertId).delete()
24 db.collection("userList").document(authUserId).collection(""
25   incomingRequest").document(advertId).delete()
26 db.collection("userList").document(sendId).collection("isentRequest")
27   .document(advertId).delete()
28 db.collection("dogWalkingAdvert").document(advertId).delete()
29 } } }

```

Şekil 88: Köpek gezdirmeye isteğinin kabul edilmesi

3.7.12 Gezme İşlemin Başlatma Ve Bitirme

Köpek sahibi isteği kabul ederinde veritabanında userList içerisinde kendi kullanıcı idsine ait alanda **acceptDogWalking** dosyası **ready** olarak işlenir. Eğer köpek sahibi gezdirmeyi başlatır ise **acceptDogWalking** dosyası **start** olarak işlenerek köpek gezdiren kişiye ge-

dirme işlemini başlatabilmektedir. Sayaç sıfırlandığında **acceptDogWalking** dosyası **finish** olarak işlenmektedir. Böylelikle köpek sahibi gezidirmeye bitirerek gezdireciyede bitirme izni vermektedir. Bunun örnek kodu aşağıda verilmektedir.

```
1 let walkingDocData : [String:Any] = [
2     "acceptDogWalkingBool" : "ready" ]
3     if let uuid = self.incomingRequestViewModelList.
4         incomingRequestAtIndex(indexPath.row).sendId as?
5             String {
6 if let authUserId = Auth.auth().currentUser?.uid {
7 db.collection("userList").document(authUserId).collection (
8     "acceptDogWalking").document(uuid).setData(walkingDocData) } }
```

Şekil 89: Gezdirme işleminin başlatılıp bitirilmesi

4 SONUÇLAR VE ÖNERİLER

4.1 Sonuçlar

Bu proje fikri sosyal medya üzerinde dolaşırken bir çok kişinin sosyal medya hikayelerinde hayvan sahiplendirmek için paylaşımlar yaptırdı. Ancak bu hikayelere sadece takipçileri ulaşabilmekteydi. Bu da az sayıda bir kişi anlamına gelmekteydi. Bu sorunlar ek olarak köpek sahipleri iş yoğunluklarından dolayı köpeklerini dışında gezdirmeye çökmek isteydi. Köpekler dışarıya çıkarılamadıkları için huysuzlaşmakta bu durumda sahiplerini üzmemektedir. İşte bu sorunları çözmek için **AdoptingHay** adında bir mobil uygulama yapmak akıma geldi. iOS kullanma nedenim dünyaca ünlü olan en değerli şirketi olan Apple ait olan **Swift** dilini kullanmasından dolayı iOS kullanmaya karar verdim. Buna ek olarak dünyada en çok kullanılan telefon Apple olması bu kararımı etkilemiştir.

Kullanıcılar uygulayı açtığında eğer giriş yapmadılar ise kendilerini giriş yapabilecekleri iki seçenek karşılamaktadır. Bunlar **Google** ve **Eposta** ile girişir. Bunlardan birini seçerek giriş işlemini yaptıktan sonra kendilerini iki seçenek karşılamaktadır. Bunlar **Hayvan Sahipleme** ve **Köpek Gezdirmedir**

Hayvan Sahiplenme seçilir ise bu seçeneğe ait alana yönlendirilir. Burada paylaşan ilanları görebilir. Bu ilanları hayvan türlerine göre filtreleye bilmektedir. Ayrıca başka filtreleme türlerinde kullanabilemektedir. Arama kısmında hayvan cinsi üzerinden arama yapabilmektedir. İlan detaylarına ulaşabilmektedir Kendisi ilan paylaşabilmektedir. Ayrıca favorilerine ilanları ekleyebilir. İlan sahibi ile mesajlaşa bilmektedir.

Seçim ekranına sağ üst köşeden tekrar dönüp köpek gezdirmeye seçeneği seçilir ise bu seçime ait alana yönlendirilecektir. Burada paylaşan ilanları görebilir. Bu ilanları filtreleye bilmektedir. Kendisi ilan paylaşabilmektedir. Ayrıca favorilerine ilanları ekleyebilir. İlan sahibi ile mesajlaşa bilmektedir. Gezdirmek için ilan sahibine istek gönderebilmektedir. Ayrıca istek kabul edilirse gezdirmeye işleminin takip edebilecekleri alanlar yer almaktadır. İlan detaylarına ulaşabilmektedir.

4.2 Öneriler

Bu proje hakkında önerilerim;

- Hayvan sahipleri dışarıya çıktığında hayvanların evde bakılması için ilanlar açılabılır.
- Veteriner konumları eklenebilir.
- Hayvanlar için bağış toplanabilir.
- Konumu mesaj olarak atılması sağlanabilir.

5 EKLER

Github Link: <https://github.com/engingulek/AdoptingHay>

KAYNAKLAR

- [1] <https://tr.wikipedia.org/wiki/iOS> [Ziyaret Tarihi: 23 Ekim 2021]
- [2] <https://www.apple.com/tr/swift/> [Ziyaret Tarihi: 22 Ekim 2021]
- [3] <https://startupteknoloji.com/ios-14-5-guncellemesi-yayinlandi/>
- [4] <https://developer.apple.com/xcode/> [Ziyaret Tarihi: 22 Ekim 2021]
- [5] <https://en.wikipedia.org/wiki/Model–view–viewmodel>
- [6] <https://en.wikipedia.org/wiki/CocoaPods> [Ziyaret Tarihi: 23 Ekim 2021]
- [7] <https://firebase.google.com/docs/auth/ios/start> [Ziyaret Tarihi: 24 Ekim 2021]
- [8] <https://www.udemy.com/course/ios-gelistirme-kursu/> [Ziyaret Tarihi: Sürekli]
- [9] <http://devnot.com/2015/mvvme-hizli-bir-giris/> [Ziyaret Tarihi: 27 Kasım 2021]
- [10] <https://docs.swift.org/swift-book/LanguageGuide/Closures.html> [Ziyaret Tarihi: 26 Kasım 2021]
- [11] https://developer.apple.com/documentation/uikit/view_controllers/showing_and_hiding_view_controllers [Ziyaret Tarihi: 1 Aralık 2021]
- [12] <https://stackoverflow.com/questions/28760541/programmatically-go-back-to-previous-viewcontroller-in-swift> [Ziyaret Tarihi: 1 Aralık 2021]
- [13] <https://ibjects.medium.com/simple-text-chat-app-using-firebase-in-swift-5-b9fa91730b6c> [Ziyaret Tarihi: 10 Aralık 2021]

[14] <https://www.mobiler.dev/post/ios-push-notification-entegrasyonu> [Ziyaret Tarihi: 5 Ocak 2021]

[15] <https://www.youtube.com/watch?v=tLVTlQvKsRY> [Ziyaret Tarihi: 2 Ocak 2021]

[16] <https://medium.com/turkishkit/swift-ile-uygulama-gelistirme-xcode-9f047e658435> [Ziyaret Tarihi: 2 Ocak 2021]

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı	:	Engin Gülek
Uyruğu	:	Türkiye Cumhuriyeti
Doğum Yeri ve Tarihi	:	İstanbul - 27:02:1999
Adres	:	Ferit Selim Paşa Cad. Sevinç Sok. Soğanlı Mah. No:27 D:6 B.Evler / İstanbul
Telefon	:	+905345658496
e-mail	:	engingulek0@gmail.com

EĞİTİM DURUMU

Lisans Öğrenimi	:	BŞEÜ Bilgisayar Mühendisliği Bölümü
Bitirme Yılı	:	2022
Lise	:	Bakırköy İmam Hatip Lisesi

İŞ DENEYİMLERİ

Yıl	:	-
Kurum	:	-
Stajlar	:	-

İLGİ ALANLARI: Kitap okuma ve yeni teknolojileri takip etmek

YABANCI DİLLER: İngilice

BELİRTMEK İSTEDİĞİNİZ DİĞER ÖZELLİKLER: