



**T.C.**

**SİVAS CUMHURİYET ÜNİVERSİTESİ  
MÜHENDİSLİK FAKÜLTESİ  
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**KOTLIN İLE  
ŞEHİR İÇİ GEZİ PLANLAMA  
MOBİL UYGULAMASI**

**YUSUF İSLAM KUTLUAY**

**LİSANS BİTİRME PROJESİ**

**HAZİRAN-2024  
SİVAS**

## **TEZ BİLDİRİMİ**

Bu tezdeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

İmza

YUSUF İSLAM KUTLUAY

01.07.2024

## ÖZET

### KOTLIN İLE ŞEHİR İÇİ GEZİ PLANLAMA MOBİL UYGULAMASI

**Yusuf İslam KUTLUAY**

**Danışman: Dr. Öğr. Üyesi Abdulkadir ŞEKER**

#### **Jüri**

**Danışmanın Unvanı Adı SOYADI**  
**Diğer Üyenin Unvanı Adı SOYADI**  
**Diğer Üyenin Unvanı Adı SOYADI**

Bu proje, Kotlin ile Android Studio üzerinde geliştirilen şehir içi gezi planlama uygulaması, günümüz mobil teknolojisinin sunduğu imkanlar doğrultusunda insanlar için pratik ve kişiselleştirilmiş bir gezi deneyimi sunmayı hedeflemektedir. Mobil uygulama, insanların kolayca şehir içinde gezilecek yerleri keşfetmelerini, rotalar oluşturmalarını ve gezilerini planlamalarını sağlamak amacıyla tasarlanmıştır.

Kullanıcı dostu arayüzü sayesinde, kullanıcılar gezilecek yerlerin isimlerini, bilgilerini ve görsellerini görebilirler. Bu bilgiler doğrultusunda gezmek istedikleri yerleri belirleyip harita üzerinde kendilerine en yakın mesafeden başlamak üzere rotalarını görebilirler. Kullanıcılar gezdikleri yerleri rota üzerinde işaretledikten sonra, bu bilgilere “Seyahatlerim” sayfasından erişebilirler.

Ayrıca kullanıcılar gezdikleri yerler hakkında duygu ve düşüncelerini paylaşabilecekleri, aynı zamanda gezdikleri yerlerle ilgili resimleri ekleyebilecekleri bir platform sunar. Bu şekilde, kullanıcılar seyahat deneyimlerini diğerleriyle paylaşabilir ve hatıralarını kolayca kaydedebilir.

**Anahtar Kelimeler:** Kotlin, Android Studio, Google MAP, Enlem, Boylam

# İÇİNDEKİLER

ÖZET.....	3
İÇİNDEKİLER.....	4
ŞEKİLLER LİSTESİ.....	5
1. GİRİŞ.....	6
1.1. Proje Tanıtımı .....	8
1.2. Projenin Kapsamı ve Hedefleri .....	8
1.3. Gereksinim Analizi .....	8
2. UYGULAMA .....	9
2.1. Platform / Ortam Kurulumu .....	9
2.1.1. Android Studio Kurulumu .....	9
2.1.2. Firebase .....	11
2.2. API Kullanımı.....	12
2.2.1. API Anahtarı.....	13
2.2.2. API Anahtarını Android Studio' ya Bağlama .....	14
3. TASARIM .....	15
3.1. Uygulama Giriş Ekranı .....	16
3.2. Uygulama Kayıt Ekranı .....	19
3.3. Uygulama Anasayfası .....	21
3.3.1. Şehirdeki Rotalar .....	25
3.3.2. Bilgi Ekranı.....	29
3.4. Uygulama Paylaş Ekranı .....	32
3.4.1. Gönderi Ekleme Ekranı .....	33
3.5. Uygulama Rota Ekranı .....	35
3.5.1. Harita Ekranı.....	37
3.6. Uygulama Profil Ekranı .....	42
3.6.1. Gönderiler .....	44
3.6.2. Seyahatlerim .....	45
4. SONUÇLAR VE ÖNERİLER.....	48
KAYNAKÇA.....	50

## ŞEKİLLER LİSTESİ

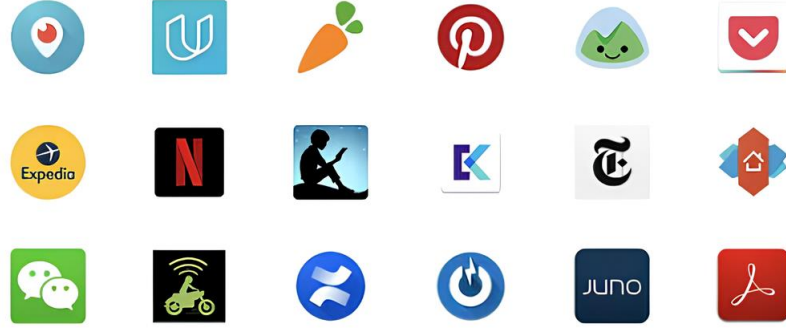
Şekil 1 - Kotlin dilini kullanan bazı uygulamalar .....	7
Şekil 2 - Android Studio genel pencere yapısı.....	8
Şekil 3 - Android Studio indirme ekranı .....	9
Şekil 4 - Android Studio proje türü belirleme ekranı .....	10
Şekil 5 - Android Studio proje bilgileri ekranı.....	10
Şekil 6 - Firestore bağlama ekranı -1 .....	12
Şekil 7 - Firestore bağlama ekranı -2 .....	12
Şekil 8 - APIs & Services.....	13
Şekil 9 – API Create Project.....	14
Şekil 10 - Uygulama ilk açılış ekranı .....	15
Şekil 11 - Uygulama giriş ekranı .....	16
Şekil 12 - Veritabanı Authentication ile kullanıcı gösterimi.....	18
Şekil 13 - Uygulama kayıt ekranı .....	19
Şekil 14 - Kullanıcı veritabanında gösterimi .....	20
Şekil 15 - Uygulama anasayfası.....	22
Şekil 16 - Şehirdeki gezilecek yerler.....	26
Şekil 17 - Şehirlerin verilerinin veritabanında gösterilmesi.....	27
Şekil 18 - Gezilecek yerlere ait bilgilerin veritabanında gösterimi.....	27
Şekil 19 - RecyclerView gösterimi .....	28
Şekil 20 - Gezilecek yere ait bilgi ekranı.....	29
Şekil 21- Uygulama Paylaş Ekranı .....	32
Şekil 22 - Yeni gönderi ekleme ekranı .....	34
Şekil 23 - Kullanıcı gönderilerinin veritabanında gösterilmesi.....	35
Şekil 24 - Rota menüsü .....	36
Şekil 25 - Kullanıcın seçtiği yerlerin veritabanında gösterimi .....	37
Şekil 26- Harita sayfası .....	38
Şekil 27 - Haritada gezilen yerler listesi.....	41
Şekil 28 - Profil ekranı .....	42
Şekil 29 - Kullanıcı Gönderileri.....	44
Şekil 30 - Kullanıcının gezdiği yerler .....	45
Şekil 31 - Seyahatlerim veritabanı gösterimi .....	46
Şekil 32 - Çıkış ekranı.....	47

## 1. GİRİŞ

Günümüzde mobil cihazlar çok yaygın bir kullanım alanına sahiptir. Bu cihazlarda farklı işlemler için geliştirilmiş mobil uygulama olarak adlandırılan programlar kullanılmaktadır. Bugün yazılım sektöründe amatör ya da profesyonel mobil uygulamalar geliştirmek amacıyla kullanılabilecek çok sayıda platform mevcut olup söz konusu platformlarının seçiminde çeşitli ölçütlere dikkat edilmesi gerekmektedir. Bunlar; mobil cihazların işletim sistemi (Android, IOS, Microsoft vb.), platformda kullanılan yazılım geliştirme dili (C, Swift, Java, Kotlin vb.), platformun çalışma şekli (çevrimiçi veya çevrimdışı) ve platformun ücretli ya da ücretsiz olması gibi sıralanabilmektedir.

Kotlin, JetBrains tarafından geliştirilmiş modern, statik olarak yazılmış bir programlama dilidir ve özellikle Android geliştirme için hızla popülerlik kazanmıştır. 2017 yılında Google tarafından resmi olarak desteklenmeye başlamasıyla birlikte Android uygulama geliştirme sürecinde geniş bir kullanım alanı bulmuştur. Kotlin, Java ile tamamen uyumlu olup, mevcut Java koduyla birlikte çalışabilme yeteneği sayesinde geçiş süreçlerini kolaylaştırır. Güçlü tip sistemi, null güvenliği, genişletilebilirliği ve daha az kodla daha fazla iş yapabilme özelliği ile Kotlin hem yeni başlayanlar hem de deneyimli geliştiriciler için cazip bir seçenek sunar. Modern yazılım geliştirme ihtiyaçlarına uygun olarak tasarlanmış olan Kotlin, sadece mobil uygulama geliştirmede değil, aynı zamanda sunucu tarafı, masaüstü ve web uygulamaları gibi geniş bir yelpazede de kullanılabilmektedir. Uygulamamızda da kullandığımız bu dil, Android Studio üzerinden uygulamaları kolaylıkla geliştirebilmemizi sağlar.

Kotlin halihazırda birçok büyük şirket(Netflix, Uber, Trello, Pinterest, Corda vb.) tarafından kullanılmaktadır. Bu kullanım ağı günlük bazda milyonlarca insan tarafından kullanılan aplikasyonlardır ve Kotlin, çok sayıda başarılı Android uygulamasını da (Pinterest, Twitter, Airbnb, Basecamp, Keepsafe App Kilit uygulaması) içerir.



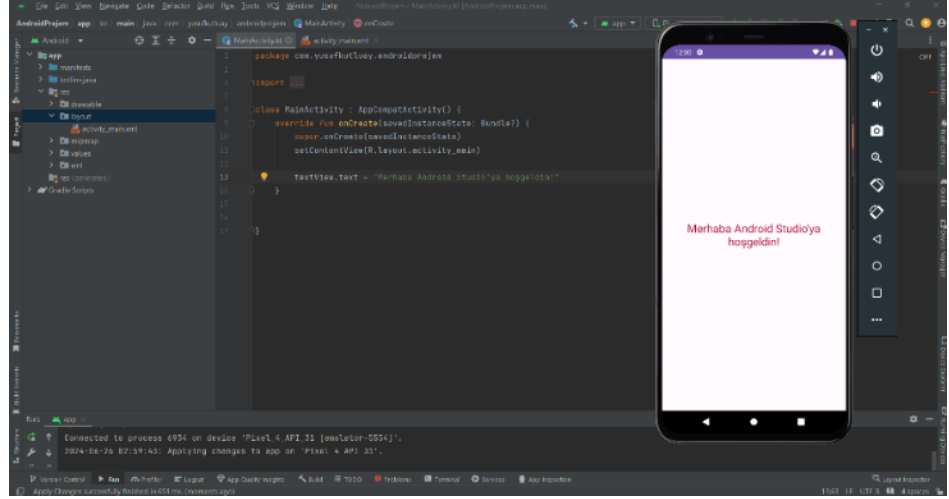
Şekil 1 - Kotlin dilini kullanan bazı uygulamalar

Uygulama marketlerinde bulunan kategoriler incelendiğinde insanlar kişiselleştirilmiş uygulamalara daha çok talepte bulunuyorlar. Kişiselleştirilmiş uygulamalar, kullanıcıların bireysel ihtiyaçlarını ve tercihlerine göre özelleştirilebilmesi nedeniyle kullanıcı deneyimini artırmakta ve sadakati güçlendirmektedir. Kullanıcıların ilgi alanlarına ve davranışlarına göre uyarlanmış içerik ve hizmetler sunan bu uygulamalar hem işlevsellik hem de kullanıcı memnuniyeti açısından önemli avantajlar sağlar. Bu doğrultuda, "Haydi Gezek" uygulaması da kullanıcıların şehir içi gezi planlarını kişiselleştirmelerine olanak tanıyan özellikleri ile öne çıkmaktadır. Kullanıcılar, gezilecek yerleri kendi tercihlerine göre seçip rotalarını oluşturabilir, deneyimlerini ve hatıralarını diğer kullanıcılarla paylaşabilirler. Bu kişiselleştirme, kullanıcıların uygulamayı daha fazla benimsemelerine ve düzenli olarak kullanmalarına katkıda bulunur.

Uygulamada kullanılan rotalama işlemleri için Google API hizmetlerini aktif olarak kullandık. Yol mesafe hesaplamaları için Google Maps Distance Matrix API ve detaylı rota yönlendirmeleri için Google Directions API'ini entegre ettik. Distance Matrix API, kullanıcıların belirledikleri başlangıç ve bitiş noktaları arasındaki mesafeyi ve tahmini seyahat süresini hesaplamak için kullanılırken, Directions API detaylı rota yönlendirmeleri ve navigasyon sağlamak için kullanılmaktadır. Bu entegrasyonlar sayesinde, kullanıcılar uygulamamız aracılığıyla şehir içinde gezilecek yerleri daha verimli bir şekilde keşfedebilir ve gezi planlarını daha etkili bir biçimde yönetebilirler.

## 1.1.Proje Tanıtımı

Şehir içi gezi planlama uygulaması Kotlin dilinde Android Studio ortamında geliştirilmiştir. Android platformuyla uyumludur.



Şekil 2 - Android Studio genel pencere yapısı

## 1.2. Projenin Kapsamı ve Hedefleri

Uygulama, kullanıcıların mobil cihazları üzerinden şehir içindeki turistik noktaları keşfetmelerini, gezilecek yerleri planlamalarını ve rotalarını oluşturmalarını sağlar. Temel amacı, kullanıcıların seyahat deneyimlerini kişiselleştirmelerine olanak tanımak ve şehir içindeki aktivitelerini daha verimli bir şekilde yönetmelerine yardımcı olmaktır.

## 1.3. Gereksinim Analizi

- **Kullanıcı Dostu Arayüz:** Kullanıcıların kolayca gezilecek yerleri keşfetmelerini, detaylı bilgilerini görmelerini ve rotalarını oluşturmalarını sağlayacak basit ve anlaşılır bir arayüz.
- **Gezi Planlama ve Rota Oluşturma:** Kullanıcıların seyahat etmek istedikleri noktalar arasında en kısa ve en hızlı rotaları hesaplayabilmesi için Google API'leri ile entegre bir rota planlama özelliği.



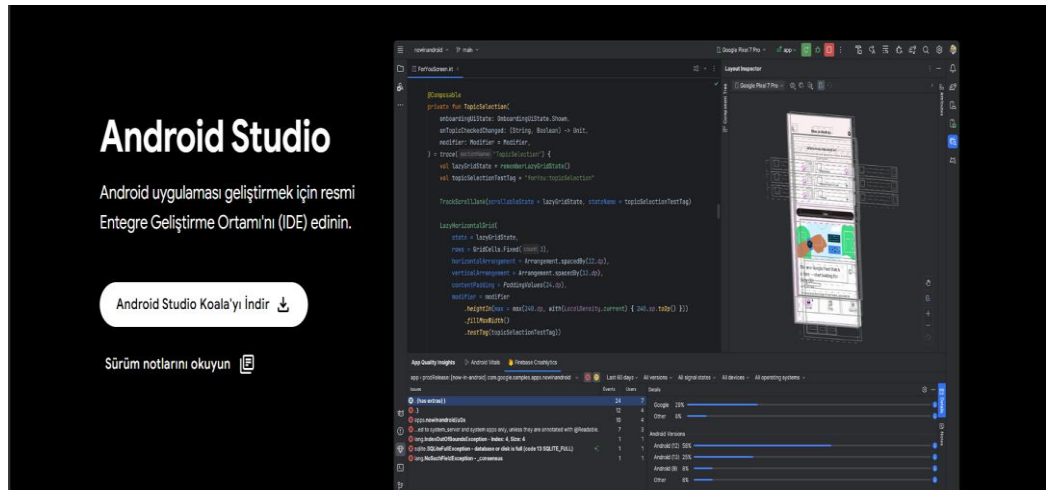
- **Özelleştirilmiş Kullanıcı Profili:** Kullanıcıların kişisel tercihlerine göre özelleştirilmiş gezi rotaları oluşturabilmeleri ve gezdikleri yerleri kaydedip paylaşabilmeleri için kullanıcı profili yönetimi.

## 2. UYGULAMA

### 2.1. Platform / Ortam Kurulumu

#### 2.1.1. Android Studio Kurulumu

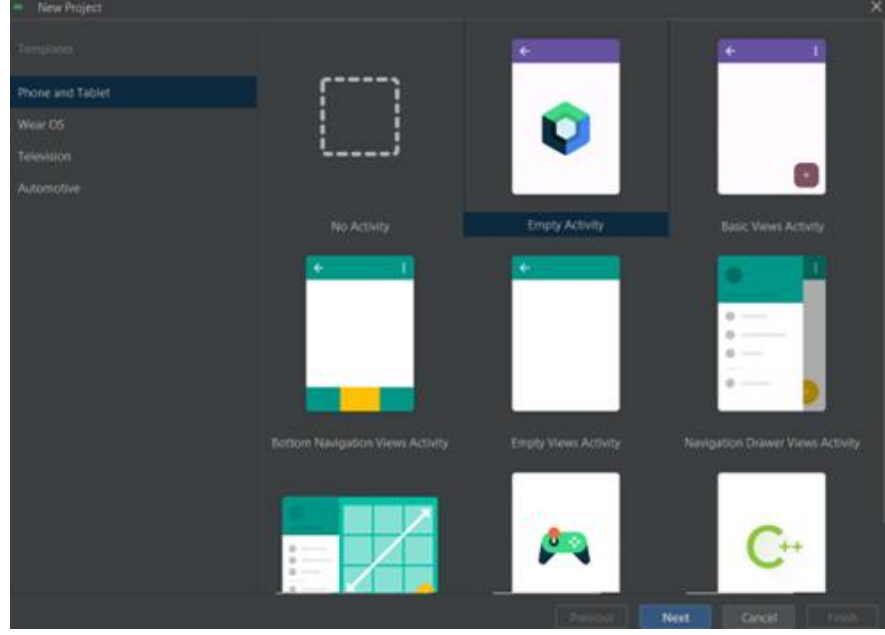
Kotlin ile Gezi Planlaması uygulamasında geliştirme ortamı olarak Android Studio kullanılmıştır.



Şekil 3 - Android Studio indirme ekranı

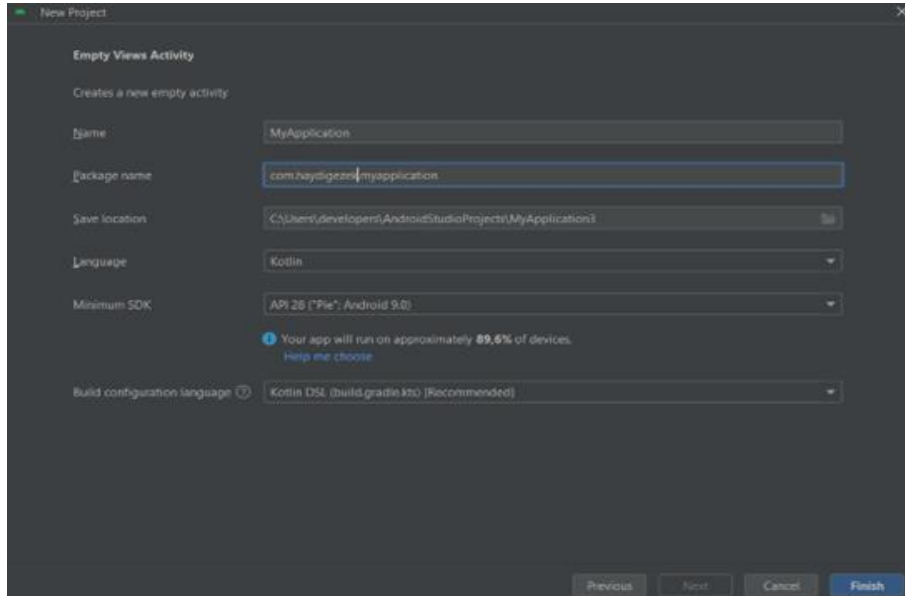
Android Studio, Android uygulamaları geliştirmek için Google tarafından sunulan resmi bir entegre geliştirme ortamıdır (IDE). IntelliJ IDEA tabanlı olan bu ortam, geliştiricilere güçlü ve kullanışlı araçlar sunar. Ayrıca güçlü bir layout editor'a sahiptir. Bu editör sayesinde sürükle ve bırak arayüzü ile kullanıcı arayüzlerini kolayca tasarlamaya olanak tanır ve gerçek zamanlı değişiklikleri önizlemenizi sağlar.

Bunun yanı sıra Android Studio emülatör desteği de sağlar. Bu emülatör, Android cihazlarının simülasyonunu yaparak uygulamalarınızı farklı cihazlarda test etme olanağı tanır. Ayrıca Firebase ve Google Cloud gibi çeşitli hizmetlerle entegrasyonu kolaylaştırır.



Şekil 4 - Android Studio proje türü belirleme ekranı

Android Studio’yu açtığınızda karşınıza başlangıç ekranı gelir. Burada “Create New Project” seçeneğine tıklayarak proje oluşturabilirsiniz. Tıklandıktan sonra projenin türünü seçmeniz gerekecektir. Burada (Şekil 4) solda ‘Templates’ kısmında hangi cihaz için geliştirme yapacağınızı seçebilirsiniz. Sağ tarafta ise belli başlı aktiviteler bulunmaktadır. Genelde ‘Empty Views Activity’ seçilir. ‘Next’ tuşuna basıldıktan sonra proje bilgilerini gireceğimiz ekran gelir.



Şekil 5 - Android Studio proje bilgileri ekranı

Karşınıza çıkan ekranda ([Şekil 5](#)) projenizin adını, proje dosyasının kaydedileceği konumu, projenin paket ismini, hangi dilde yazacağınızı (Kotlin veya Java) ve hangi SDK'yı kullanacağınızı seçersiniz. Proje adınızı belirlerken ilk harfin büyük olmasına, birden fazla kelimedenden oluşuyorsa her kelimenin baş harfinin büyük olmasına ve Türkçe karakter kullanılmamasına dikkat edilmelidir.

### 2.1.2. Firebase

Firebase, Google tarafından sağlanan ve mobil ile web uygulamaları geliştirmeye olanak tanıyan bir bulut bilişim hizmetidir. Uygulama içerisinde kullanıcı kimlik doğrulama (Authentication), veritabanı (Firestore Database) ve depolama (Storage) özellikleri kullanılmıştır.

Firebase kurmak için;

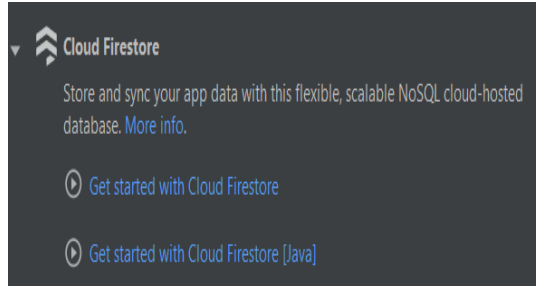
Tarayıcınızın arama çubuğuna Firebase yazarak veya "<https://firebase.google.com/>" adresine gidilerek oturum açılabilir veya yeni hesap oluşturarak kullanıma başlayabilirsiniz.

Daha sonrasında *Get Started > Proje Ekle (Add project)* seçeneklerine tıklayın ve gelecek ekranda projeniz için bir isim giriniz. İşlem tamamlandıktan sonra 'Continue' seçeneğine tıklanır. Firebase projeniz oluşturulduktan sonra, proje kontrol paneline yönlendirileceksiniz. Burada, "Uygulama ekle" (Add app) kısmından Android simgesine tıklayınız. Uygulamanızın paket adını giriniz. Daha sonra "Uygulamayı kaydet" (Register app) düğmesine tıklayın.

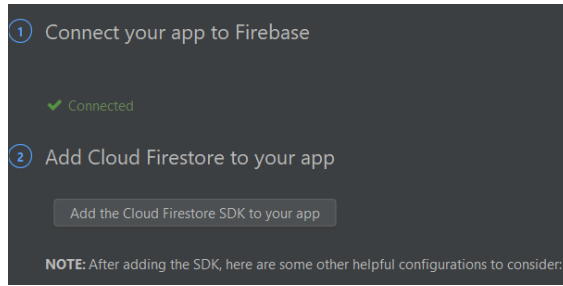
Firebase size bir 'google-services.json' dosyası sağlayacaktır. Bu dosyayı indirin ve Android projenizin 'app' dizinine ekleyin. Projenize gerekli bağımlılıkları kurarak kullanmaya başlayabilirsiniz.

Firestore Database'i Android Studio'ya entegre etmek için öncelikle Android Studio'yu açarak üst menüden 'Tools' sekmesine tıklanır ve buradan 'Firebase' seçeneği seçilir. Sağda açılan menüden 'Cloud Firestore' ([Şekil 6](#)) ifadesi tıklanarak 'Get Started with Cloud Firestore' seçeneği tercih edilir. Bu adımı takiben, 'Add the Cloud Firestore

SDK to your app' ([Şekil 7](#)) ifadesine tıklanarak projeye gerekli bağımlılıklar otomatik olarak eklenir. Sonrasında, aynı menüde bulunan öğretici yöntemle Firestore'un nasıl kullanılacağına dair bilgi edinilerek, uygulamada kullanıma başlanabilir. Bu süreç, geliştiricilerin veritabanı işlemlerini daha hızlı ve etkili bir şekilde gerçekleştirmelerine olanak tanır.



Şekil 6 - Firestore bağlama ekranı -1



Şekil 7 - Firestore bağlama ekranı -2

## 2.2. API Kullanımı

Modern mobil uygulamalarda kullanıcı deneyimini geliştirmek ve kullanıcıya konum tabanlı hizmetler sunmak için harita entegrasyonu büyük önem taşır. Harita API'leri, geliştiricilere uygulamalarına interaktif haritalar ekleme, konum belirleme, rota planlama ve yer işaretleri gibi çeşitli harita özelliklerini kullanma imkanı sağlar. Google Maps API, Mapbox, OpenStreetMap gibi popüler harita servisleri, zengin özellik setleri ve güçlü entegrasyon yetenekleri ile öne çıkar. Bu API'ler, kullanıcının bulunduğu konumu tespit etmekten, belirli bir noktaya en kısa rotayı hesaplamaya kadar geniş bir yelpazede işlevsellik sunar. Harita API'lerinin etkin kullanımı, uygulamanızın kullanıcılarına daha fazla değer katmak ve onlara daha iyi bir deneyim sunmak için kritik bir bileşendir.

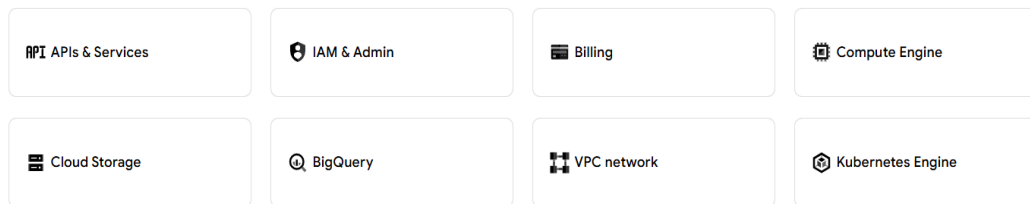
Uygulamamızda, Google API'den aldığımız anahtar ile kullanıcılara rotalarını oluşturma ve bu rotalarda en kısa mesafeyi belirleme olanağı sağladık. Google API'den aldığımız tek bir anahtar sayesinde birden fazla API'yi kullanma imkânına sahip olduk. Bu kapsamda, Directions API, Routes API ve diğer birçok API'yi etkin bir şekilde kullanabildik. Google, belirli bir kullanım kotasına kadar kullanıcılara ücretsiz destek sunmaktadır. Bu sayede, uygulamamızın testlerini gerçekleştirme imkânı bulduk ve bu ücretsiz destek, geliştirme sürecimiz için yeterli oldu. Bu API'ler sayesinde, kullanıcılara daha işlevsel ve kullanıcı dostu uygulamalar sunabildik. Kullanıcıların ihtiyaçlarına uygun rotalar oluşturma, seyahat sürelerini kısaltma ve daha etkin bir navigasyon deneyimi sağlama gibi avantajlar, uygulamamızın kullanılabilirliğini arttırdı. Bu entegrasyon hem teknik hem de kullanıcı deneyimi açısından uygulamamızın başarısına önemli katkılar sağladı.

### 2.2.1. API Anahtarı

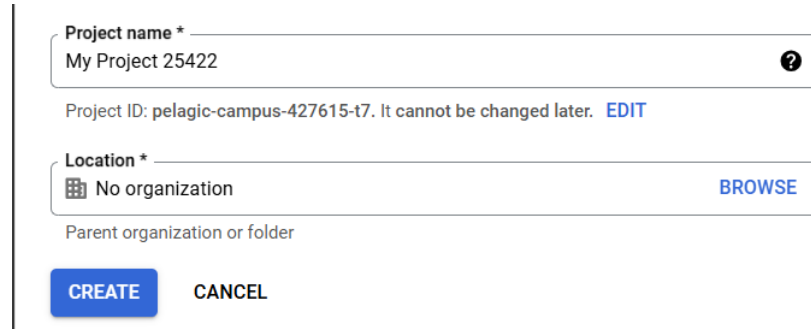
Google API'lerini kullanabilmek için öncelikle bir API anahtarı edinmek gerekmektedir. API anahtarı, Google Cloud Platform (GCP) üzerinden alınır ve uygulamanın Google servisleriyle güvenli bir şekilde iletişim kurmasını sağlar.

Google API'lerini kullanabilmek için öncelikle Google Cloud Platform'a kaydolmanız gerekmektedir. Bunun için tarayıcınızdan <https://cloud.google.com/> adresine giderek sağ üst köşede bulunan 'Console' seçeneğine tıklayın. Google Cloud Console'a giriş yaptıktan sonra, menüden 'APIs & Services' bölümüne gidin. Buradan 'Dashboard' altında yer alan 'New Project' (Yeni Proje) butonuna tıklayarak yeni bir proje oluşturun. Proje adını belirleyip 'Create' (Oluştur) butonuna basarak projenizi oluşturun.

Quick access



Şekil 8 - APIs & Services



Project name \*  
My Project 25422

Project ID: pelagic-campus-427615-t7. It cannot be changed later. [EDIT](#)

Location \*  
No organization [BROWSE](#)

Parent organization or folder

[CREATE](#) [CANCEL](#)

Şekil 9 – API Create Project

Proje oluşturulduktan sonra, sol taraftaki menüden 'Credentials' (Kimlik Bilgileri) bölümüne gidin. Burada 'Create Credentials' (Kimlik Bilgileri Oluştur) düğmesine tıklayın ve açılan menüden 'API Key' (API Anahtarı) seçeneğini seçerek API anahtarınızı oluşturun. Anahtar oluşturulduktan sonra, 'Show Key' seçeneğine tıklayarak anahtarınızı görüntüleyebilir ve kopyalayabilirsiniz. Son olarak, bu anahtarı Android Studio veya projenizin ilgili dosyalarına ekleyerek Google API'lerini entegre edebilir ve uygulamanızda kullanabilirsiniz.

### 2.2.2. API Anahtarını Android Studio'ya Bağlama

Android Studio'da API anahtarını projeye entegre etmek için öncelikle AndroidManifest.xml dosyasını düzenlemeniz gerekmektedir. Bu dosyayı proje ağacında *app > manifest > AndroidManifest.xml* yolunu takip ederek açabilirsiniz. Dosyayı açtıktan sonra, `<application>` etiketi içine aşağıdaki `<meta-data>` etiketini eklemeniz gerekmektedir:

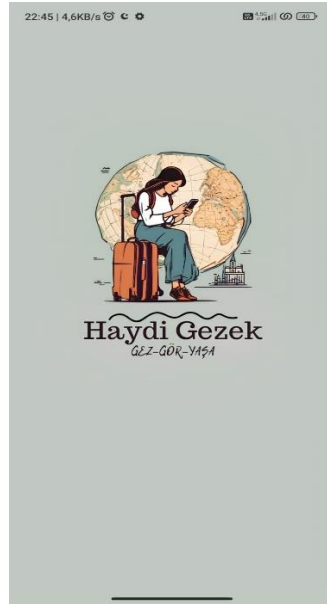
```
<meta-data  
    android:name="com.google.android.geo.API_KEY"  
    android:value="API_KEY" />
```

Burada, *android:value* özelliğine kendi Google API anahtarınızı eklemelisiniz. Bu anahtar, uygulamanızın Google harita veya diğer servislerle iletişim kurmasını sağlar. API anahtarını ekledikten sonra, değişiklikleri kaydedip Android Studio'da projenizi yeniden derleyerek entegrasyonun doğru çalıştığını kontrol edebilirsiniz.

### 3. TASARIM

Bir mobil uygulamanın başarısında kritik bir rol oynayan tasarım, kullanıcı deneyimini ve etkileşimini doğrudan etkileyen temel unsurlardan biridir. İyi bir tasarım, kullanıcıların uygulamayı kolayca anlamasını, rahatlıkla kullanmasını ve keyifli bir deneyim yaşamasını sağlar.

Uygulamanın tasarımında büyük bir özveri ile çalışılmış ve tamamen kullanıcı odaklı bir yaklaşım benimsenmiştir. Uygulamanın adı 'Haydi Gezek' olarak belirlenmiş ve sloganı “Hayat bir yolculuktur, keyfini çıkarın!” olarak ifade edilmiştir. 'Haydi Gezek' uygulaması, kullanıcıların kolayca erişebileceği ve gezintilerini keyifli hale getirecek bir arayüzle tasarlanmıştır. Uygulama, anasayfa, paylaş, rota ve profil menülerinden oluşan birden çok sayfa içermektedir. Her bir menü, kullanıcıların farklı ihtiyaçlarına cevap verecek şekilde ayrı sayfalar barındırmaktadır. Bu tasarım yaklaşımı, kullanıcıların uygulamayı rahatça kullanabilmesini ve en iyi deneyimi yaşayabilmesini sağlamayı hedeflemektedir.



Şekil 10 - Uygulama ilk açılış ekranı

Uygulamaya girmek için öncelikle kaydolmanız zorunludur. Bu sayede kişiselleştirilmiş bir deneyim sunulmakta ve kullanıcıların tercihleri doğrultusunda içerik ve özellikler sağlanmaktadır. Kayıt süreci, kullanıcıların kendi profillerini oluşturarak

uygulamayı daha etkin bir şekilde kullanmalarını mümkün kılmaktadır. Bu kişiselleştirme, uygulamanın kullanıcılarla daha güçlü bir bağ kurmasını ve onların ihtiyaçlarına daha iyi cevap verebilmesini sağlamaktadır.

### 3.1. Uygulama Giriş Ekranı

Kullanıcı uygulamayı ilk açtığında giriş ekranı ile karşılaşacaktır. Giriş ekranında kullanıcıdan e-posta adresi ve şifre istenmektedir. Eğer üye değilse ‘Hemen bize katıl’ butonuyla kayıt sayfasına yönlendirilecektir.



Şekil 11 - Uygulama giriş ekranı

```
<Button
    android:id="@+id/girisButton"
    android:layout_width="216dp"
    android:layout_height="49dp"
    android:layout_gravity="center"
    android:layout_marginTop="30dp"
    android:background="@drawable/sekiltext"
    android:onClick="girisBas"
    android:text="HAYDİ GEZEK"
    android:textSize="20sp" />
```

```
<EditText
    android:id="@+id/epostaKullaniciText"
```



```

        android:layout_width="310dp"
        android:layout_height="47dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentEnd="true"
        android:layout_marginStart="45dp"
        android:layout_marginEnd="45dp"
        android:background="@drawable/edit_text_background"
        android:clickable="true"
        android:ems="10"
        android:focusable="true"
        android:hint="E-postanızı giriniz"
        android:inputType="textEmailAddress"
        android:padding="14dp"
        android:textSize="15dp"
        android:typeface="monospace" />

<EditText
    android:id="@+id/sifreText"
    android:layout_width="310dp"
    android:layout_height="47dp"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_alignParentEnd="true"
    android:layout_marginStart="45dp"
    android:layout_marginTop="0dp"
    android:layout_marginEnd="45dp"
    android:background="@drawable/edit_text_background"
    android:clickable="true"
    android:ems="10"
    android:focusable="true"
    android:hint="$şifrenizi giriniz"
    android:inputType="textPassword"
    android:padding="14dp"
    android:textSize="15dp"
    android:typeface="monospace">

</EditText>

```

Yukarıdaki XML kod parçası, bir Android uygulamasının kullanıcı arayüzünde bir düğme (*Button*) ve iki metin girişi (*EditText*) alanı tanımlamaktadır.

Düğme (*Button*), kullanıcı arayüzünde görsel ve işlevsel özelliklere sahip bir bileşendir. "HAYDİ GEZEK" metni üzerine tıklanabilir ve belirli bir işlevi çağırarak için *onClick* özelliği (*girisBas* metodunu) kullanılır.

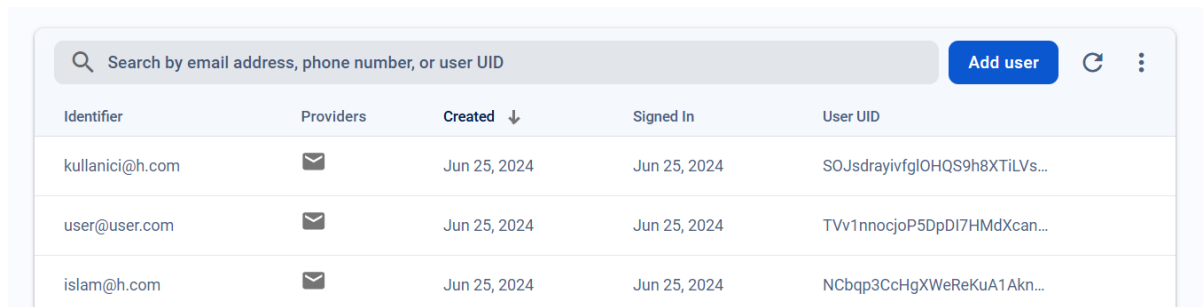
Metin girişi alanları (*EditText*), kullanıcıdan e-posta ve şifre bilgilerini girmesini bekler. Bu alanlar, kullanıcının veri girişini almak ve işlemek için kullanılır. E-posta

alanı, *textEmailAddress* giriş türü ile ayarlanmıştır ve kullanıcının doğru biçimde e-posta adresi girmesini sağlar.

Firebase'den veri çekerken sınıfta öncelikle Firebase tanımlanması gerekmektedir. Ve `getInstance()` ile başlatılmalıdır.

```
private val database = Firebase.firestore
private lateinit var auth : FirebaseAuth
auth = FirebaseAuth.getInstance()
```

Kullanıcı girişinde doğrulama Firebase Authentication ile gerçekleştirilmektedir. Kullanıcı ilk kaydolduğunda e-posta adresi ve şifresi Authentication üzerinde kaydolmaktadır. Kullanıcı giriş yaptığında kontrol edilmektedir.



The screenshot shows the Firebase Authentication console. At the top, there is a search bar with the text "Search by email address, phone number, or user UID" and an "Add user" button. Below the search bar is a table with the following columns: Identifier, Providers, Created, Signed In, and User UID. The table contains three rows of user data.

Identifier	Providers	Created	Signed In	User UID
kullanici@h.com	📧	Jun 25, 2024	Jun 25, 2024	SOJsdrayivfglOHQS9h8XTILVs...
user@user.com	📧	Jun 25, 2024	Jun 25, 2024	TVv1nnocjoP5DpDI7HMDxcan...
islam@h.com	📧	Jun 25, 2024	Jun 25, 2024	NCbqp3CcHgXWeReKuA1Akn...

Şekil 12 - Veritabanı Authentication ile kullanıcı gösterimi

```
val email = binding.epostaKullaniciText.text.toString()
val password = binding.sifreText.text.toString()

auth.signInWithEmailAndPassword(email,password)
    .addOnCompleteListener {
        if (it.isSuccessful){
            val intent =
            Intent(applicationContext, FragmentFirstActivity::class.java)
            Toast.makeText(this, "Hoşgeldiniz", Toast.LENGTH_LONG).show()
            startActivity(intent)
            finish()
        }
        .addOnFailureListener {
            Toast.makeText(applicationContext, "Hatalı giriş yaptınız", Toast.LENGTH_LONG).show()
        }
    }
```

Yukarıdaki kod parçasında, kullanıcının girdiği e-posta (*eposta*) ve şifre (*sifre*) bilgileri *binding* nesnesi üzerinden alınır. Daha sonra Firebase Authentication kullanılarak doğrulanır. *auth.signInWithEmailAndPassword(email, password)* fonksiyonu, Firebase Authentication hizmetine bir kimlik doğrulama isteği gönderir. Kullanıcının kimlik bilgileri doğru ise, *addOnCompleteListener* bloğu tetiklenir ve başarılı bir kimlik doğrulama işlemi gerçekleştirilir. Bu durumda, kullanıcı yeni bir aktiviteye (*FragmentFirstActivity*) yönlendirilir ve hoş geldiniz mesajı gösterilir.

### 3.2. Uygulama Kayıt Ekranı

Uygulama kayıt ekranına girmek için kullanıcı, "Hemen Bize Katıl" butonuna basar ve kayıt ekranına yönlendirilir. Kayıt ekranında kullanıcıdan isim, kullanıcı adı, e-posta ve şifre bilgileri istenir. Bu bilgiler, Firebase Authentication ile kimlik doğrulama sürecinde kullanılır ve başarılı bir kayıt işlemi gerçekleştirildikten sonra kullanıcı bilgileri Firestore Database'e eklenir.



23:15 | 3,2KB/s

HEMEN ÜYE OL

Adınızı ve soyadınızı giriniz

Kullanıcı adınızı oluşturunuz

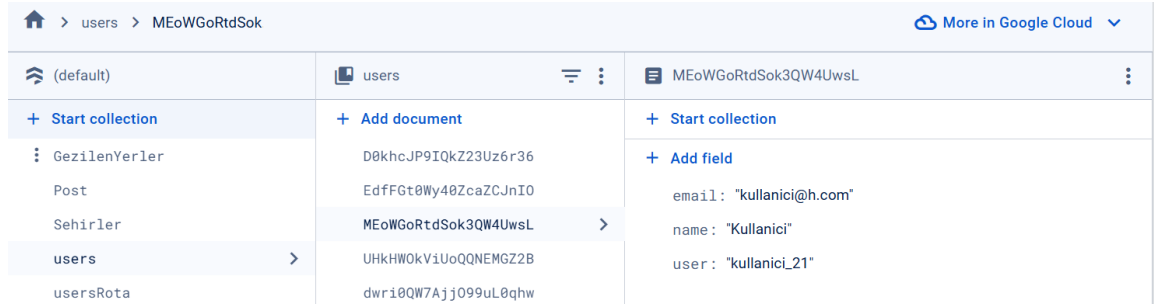
E-posta adresinizi giriniz

Şifrenizi oluşturunuz

HESAP OLUŞTUR

Şekil 13 - Uygulama kayıt ekranı

Firestore Database'e eklenmesinin amacı, kullanıcıların uygulamayı daha kişiselleştirilebilir bir şekilde kullanmalarını sağlamaktır. Bu veritabanı, kullanıcı profillerinin saklanması ve daha sonra erişilmesine olanak tanır. Böylece kullanıcılar, uygulamaya her giriş yaptıklarında kişisel ayarlarını ve verilerini görebilirler, bu da kullanıcı deneyimini önemli ölçüde geliştirir.



Şekil 14 - Kullanıcı veritabanında gösterimi

```
val uyeAd = binding.uyeAd.text.toString()
val uyeUserName = binding.uyeKullaniciAd.text.toString()
val uyeEmail = binding.uyeEposta.text.toString()
val uyeSifre = binding.uyeSifre.text.toString()

auth.createUserWithEmailAndPassword(uyeEmail, uyeSifre)
    .addOnCompleteListener {
        if (it.isSuccessful){
            db.signUp(uyeAd, uyeUserName, uyeEmail, uyeSifre)
            Toast.makeText(applicationContext, "Kayıt
başarılı", Toast.LENGTH_LONG).show()
            val intent = Intent(this, FragmentFirstActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
    .addOnFailureListener {
        Toast.makeText(applicationContext, "Lütfen eksiksiz
doldurun", Toast.LENGTH_LONG).show()
    }
}
```

Yukarıdaki kod parçasında, kullanıcının kaydolma işlemi gerçekleştirilir. İlk olarak, kullanıcının girdiği isim (uyeAd), kullanıcı adı (uyeUserName), e-posta (uyeEmail) ve şifre (uyeSifre) bilgileri binding nesnesi üzerinden alınır. Bu bilgiler, kullanıcının kayıt işlemi için gereklidir.

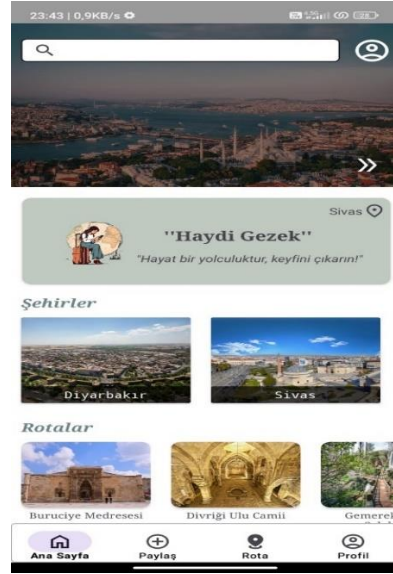
Daha sonra `auth.createUserWithEmailAndPassword(uyeEmail, uyeSifre)` fonksiyonu kullanılarak Firebase Authentication üzerinden kullanıcı kaydı yapılır. Bu fonksiyon, kullanıcı tarafından sağlanan e-posta ve şifre bilgilerini kullanarak yeni bir kullanıcı hesabı oluşturur.

`addOnCompleteListener` bloğu, kayıt işlemi tamamlandığında çalışır ve işlem başarılıysa (`it.isSuccessful`), `db.signUp` fonksiyonu çağrılarak Firebase Database'e kullanıcı bilgileri (`uyeAd`, `uyeUserName`, `uyeEmail`, `uyeSifre`) eklenir. Kod karışıklığını önlemek için veritabanı işlemleri ayrı Class'ta tutulur ve gerektiğinde çağrılır.

```
fun signUp(userName : String, user : String, email : Any, password : Any){  
    val signUpHashMap = hashMapOf(  
        "name" to userName,  
        "user" to user,  
        "email" to email,  
        "password" to password  
    )  
  
    db.collection("users")  
        .add(signUpHashMap)  
        .addOnSuccessListener {  
            toastMessage = true  
        }  
        .addOnFailureListener {  
            toastMessage = false  
        }  
}
```

### 3.3. Uygulama Anasayfası

Bu uygulamanın ana sayfası, kullanıcıların konum bilgisini kullanarak kişiselleştirilmiş gezi önerileri sunmak üzere tasarlanmıştır. Uygulama başlatıldığında, kullanıcıdan konum izni istenir ve konuma bağlı olarak anasayfada gösterilecek veriler belirlenir. Anasayfa üzerinde kullanıcılar, 3 farklı şehrin ve her bir şehir için belirlenen rotaların gezilecek yerleri hakkında bilgi alabilirler.



Şekil 15 - Uygulama anasayfası

Uygulama anasayfasında Rotalar kısmı konuma göre belirlenir. Konum Sivas olduğu için Sivas'taki gezilecek yerler gösterilmiştir.

```
fun getVeri(sehir : String, listeyiDondur: (ArrayList<Place>) -> Unit){
    db.collection("Şehirler")
        .document(sehir)
        .collection("GezilecekYerler")
        .get()
        .addOnSuccessListener {
            val postList = ArrayList<Place>()
            for (document in it){
                val enlem = document.getString("enlem")?.toDouble()
                val boylam = document.getString("boylam")?.toDouble()
                val urlGorsel = document.getString("url")
                val name = document.getString("name")

                if (enlem != null && boylam != null && urlGorsel != null &&
name != null) {
                    val place = Place(enlem, boylam, urlGorsel, name)
                    postList.add(place)
                }else{
                    println("null")
                }
            }
            postList.sortBy { it.name }
            listeyiDondur(postList)
        }
}
```

Bu fonksiyon, belirli bir şehir adıyla tanımlanan *sehir* parametresi aracılığıyla Firestore veritabanından veri çekmek için kullanılır. Veritabanında *Sehirler* koleksiyonu altında bulunan belirtilen şehir belgesinin içindeki *GezilecekYerler* koleksiyonundan veriler getirilir. Başarılı bir şekilde veri çekildiğinde, her belge için *enlem*, *boylam*, *url* ve *name* alanları kontrol edilerek uygun olanlar *Place* sınıfından nesneler oluşturulur ve bir listeye eklenir. Son olarak, bu liste isme göre sıralanır ve *listeyiDondur* adlı geri çağırım fonksiyonu aracılığıyla döndürülür. Bu işlev, uygulamanın anasayfasında kullanıcıya belirli bir şehirdeki gezilecek yerlerin listesini sunmak için tasarlanmıştır.

```
fusedLocationClient =
LocationServices.getFusedLocationProviderClient(requireContext())

/// Konum izni kontrolü ve konum bilgisini alma
if (ContextCompat.checkSelfPermission(requireContext(),
Manifest.permission.ACCESS_FINE_LOCATION)
== PackageManager.PERMISSION_GRANTED) {
    fusedLocationClient.lastLocation
        .addOnSuccessListener { location ->
            if (location != null) {
                val latitude = location.latitude
                val longitude = location.longitude
                getCityName(latitude, longitude) { cityName ->
                    cityName?.let { city ->
                        fetchRotaData(city)
                    } ?: run {
                        // Konum bulunamazsa varsayılan verileri yükle
                        loadDefaultData()
                        konumUyari()
                    }
                }
            }
        }
    }else{
        loadDefaultData()
        konumUyari()
    }
} else {
    // İzin yoksa izin iste
    ActivityCompat.requestPermissions(requireActivity(),
        arrayOf(Manifest.permission.ACCESS_FINE_LOCATION), 1)
}

private fun fetchRotaData(cityName: String) {
    db.getVeri(cityName.toLowerCase()) { dataList ->
        if (dataList.isEmpty()){
            loadDefaultData()
            konumUyari()
        }else{
            rotaList.clear()
            for (data in dataList) {
                rotaList.add(RotaModel(name = data.name, url =
data.urlGorsel))
            }
        }
    }
}
```

```

        }

        val layoutManager = LinearLayoutManager(requireContext(),
        LinearLayoutManager.HORIZONTAL, false)
        binding.recyclerRotalar.layoutManager = layoutManager
        recyclerViewAdapter = RecyclerRotaAdapter(rotalList)
        binding.recyclerRotalar.adapter = recyclerViewAdapter

        progressBar.visibility = View.GONE
    }
}

}

private fun loadDefaultData(){
    db.getVeri("istanbul") { dataList ->
        rotalList.clear()
        for (data in dataList) {
            rotalList.add(RotaModel(name = data.name, url = data.urlGorsel))
        }

        val layoutManager = LinearLayoutManager(requireContext(),
        LinearLayoutManager.HORIZONTAL, false)
        binding.recyclerRotalar.layoutManager = layoutManager
        recyclerViewAdapter = RecyclerRotaAdapter(rotalList)
        binding.recyclerRotalar.adapter = recyclerViewAdapter

        progressBar.visibility = View.GONE
    }
}

private fun getCityName(latitude: Double, longitude: Double, callback:
(String?) -> Unit) {
    val geocoder = Geocoder(requireContext(), Locale.getDefault())
    val addresses = geocoder.getFromLocation(latitude, longitude, 1)
    if (addresses!!.isEmpty()) {
        val cityName = addresses!![0].adminArea
        binding.textView2.text = cityName
        callback(cityName)
    } else {
        callback(null)
    }
}
}

```

Yukarıdaki kod bloğu, uygulamanın konum hizmetlerini kullanarak kullanıcı konumunu belirlemek ve konuma göre verileri getirmek için tasarlanmıştır. Başlangıçta, *LocationServices.GetFusedLocationProviderClient(requireContext())* ile konum sağlayıcı hizmeti başlatılır. Daha sonra, kullanıcının *ACCESS\_FINE\_LOCATION* iznini verip vermediği kontrol edilir. İzin verilmişse, *fusedLocationClient.lastLocation* ile en son bilinen konum alınır. Konum başarılı bir şekilde alındığında, *getCityName* fonksiyonu aracılığıyla konumun şehir adı çıkarılır ve bu şehre göre rotaların verileri



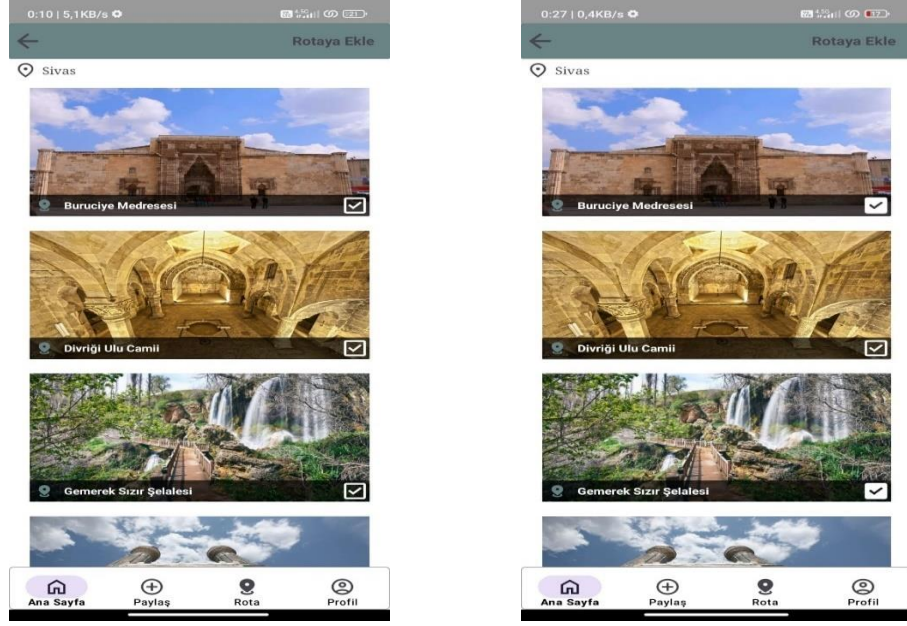
*fetchRotaData* ile getirilir. Eğer konum bilgisi alınamaz veya şehir adı null ise, varsayılan veriler *loadDefaulData* ile yüklenir ve kullanıcıya bir konum uyarısı *konumUyari* gösterilir. Kullanıcı izin vermemişse, *ACCESS\_FINE\_LOCATION* izni *ActivityCompat.requestPermissions* ile talep edilir, bu da kullanıcının konum temelli özellikleri kullanabilmesini sağlar. Bu işlevsel yapı, uygulamanın konuma dayalı içerikleri kullanıcıya sunmasını ve kullanıcı deneyimini iyileştirmeyi amaçlamaktadır

Recycler, Android uygulamalarında listeleri görüntülemek için kullanılan güçlü bir bileşendir. Genellikle veri kümesini düzenli ve tekrarlanabilir bir şekilde göstermek için kullanılır. RecyclerView, performansı artıran ve kullanıcı arayüzünde akıcı kaydırmalar sağlayan ViewHolder mimarisiyle çalışır. RecyclerView, büyük veri setlerini yönetmek için optimize edilmiştir ve veri değişikliklerini etkin bir şekilde yönetir.

Geocoder ise Android'in konum hizmetlerinden biridir ve coğrafi konum verilerini almak ve bu verileri adreslere veya adresleri coğrafi konumlara dönüştürmek için kullanılır. Bir Geocoder örneği oluşturulduğunda, belirli bir coğrafi konum için adres bilgilerini alabilir veya bir adresi coğrafi konuma dönüştürebiliriz.

### **3.3.1. Şehirdeki Rotalar**

Uygulamanın anasayfasında kullanıcıların şehirleri görüntülediği bir arayüz bulunmakta ve her bir şehre tıkladıklarında Firebase Firestore üzerinden o şehre ait gezilecek yerlerin verileri çekilmektedir. Bu veriler, kullanıcıların seçtikleri şehre özgü olarak dinamik olarak gösterilmektedir. Kullanıcılar böylece uygulama içinde gezilecek yerleri keşfederken, her şehrin kendine özgü turistik noktalarını kolayca görebilir ve detaylarına ulaşabilirler. Bu yapı sayesinde kullanıcı deneyimi kişiselleştirilmiş ve şehirlere göre özelleştirilmiş içeriklerle zenginleştirilmiş olur.



Şekil 16 - Şehirdeki gezilecek yerler

Kullanıcılar uygulamanın rota oluşturma özelliği sayesinde gezmek istedikleri yerleri seçip "Rotaya ekle" butonuna basarak kolayca bir rota oluşturabiliyorlar. Seçilen her bir yer, rota menüsüne eklenerek kullanıcının planını kişiselleştirmesine olanak tanır.

```
arguments?.let {
    sehirName = it.getString("name")
}

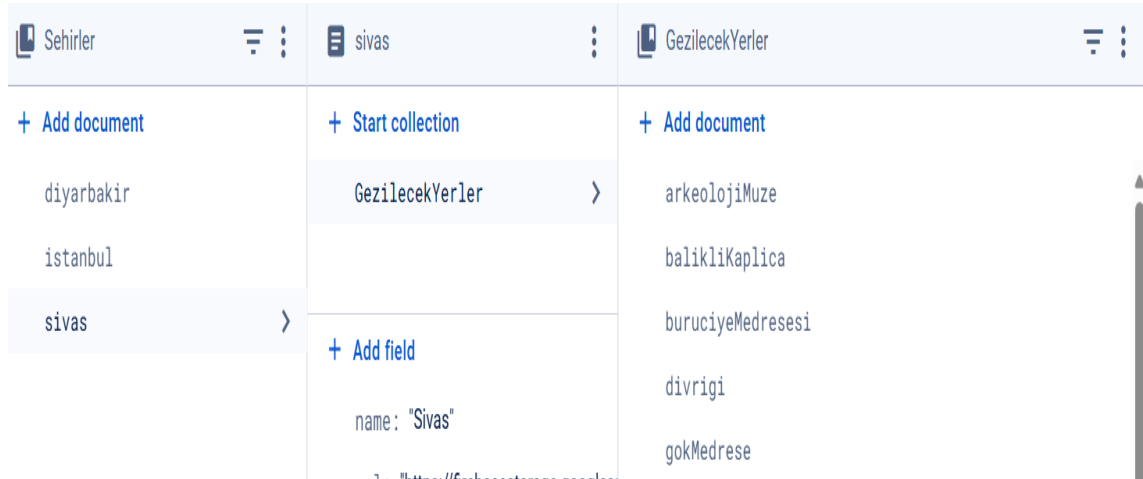
data.getVeri(sehirName!!) {

    val layoutManager = LinearLayoutManager(requireContext(),
        LinearLayoutManager.VERTICAL, false)
    binding.recyclerView.layoutManager = layoutManager
    recyclerAdapter = RecyclerViewAdapter(it)
    binding.recyclerView.adapter = recyclerAdapter

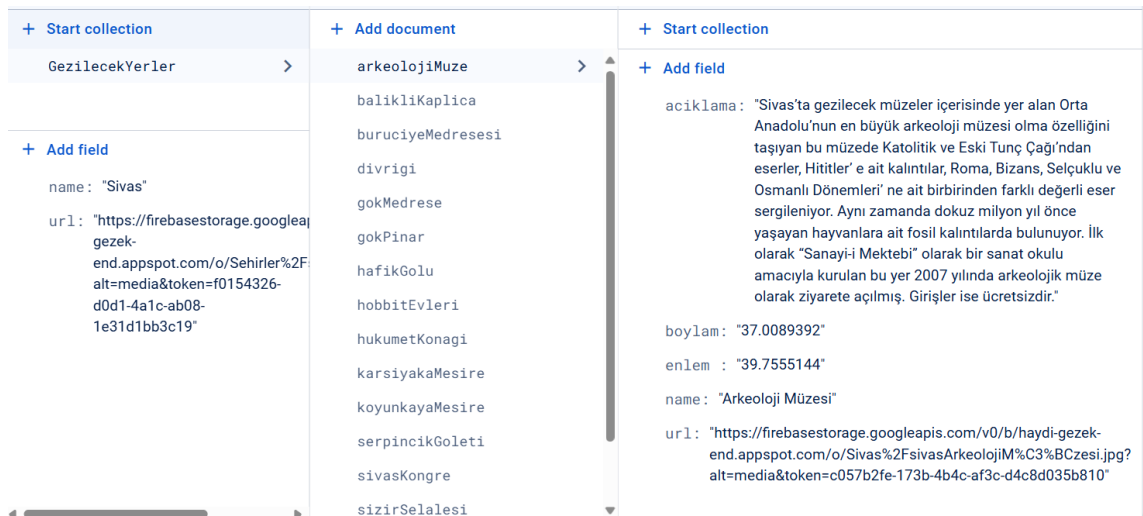
    binding.progressBar2.visibility = View.GONE
}
```

HomeFragment'te kullanıcı tarafından tıklanan şehre göre veri gönderme işlemi Navigation Component kullanılarak gerçekleştirilir. Bu işlem, fragment içindeki *arguments* nesnesi üzerinden "name" anahtarına sahip veriyi (*sehirName* olarak atanmış değişkene) alma işlemini içerir. Daha sonra *getVeri* fonksiyonu, aldığı şehir adı

parametresiyle Firestore Database'den ilgili şehre ait verileri çeker. Çekilen veriler, bir liste olarak döner ve bu liste RecyclerView kullanılarak ekranda görüntülenir.



Şekil 17 - Şehirlerin verilerinin veritabanında gösterilmesi



Şekil 18 - Gezilecek yerlere ait bilgilerin veritabanında gösterimi

```
override fun onBindViewHolder(holder: SehirHolder, position: Int) {

    holder.binding.sehirName.text = sehirList[position].name
    Picasso.get().load(sehirList[position].url).into(holder.binding.sehirUrl)

    holder.binding.cardBas.setOnClickListener {
```

```

        val navController = holder.itemView.findNavController()
        val action =
            HomeFragmentDirections.actionHomeFragmentToSehirFragment(sehirList[position].
            name)
        navController.navigate(action)
    }
}

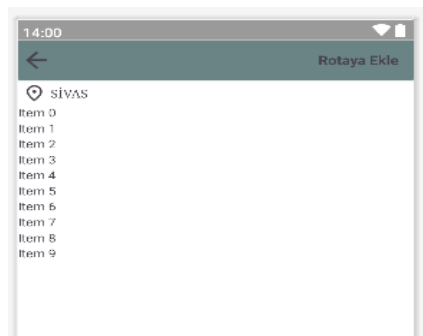
```

Önceki sayfada yer alan kod parçası, uygulamanın ana ekranında bulunan RecyclerView içindeki her bir şehir ögesi için görsel ve metinsel içerikleri bağlamak ve kullanıcı etkileşimini yönetmek amacıyla kullanılır. Her bir öge için, *onBindViewHolder* metodunda şehrin adı ilgili TextView'e (*sehirName*) yerleştirilir ve şehre ait görsel URL, Picasso kütüphanesi aracılığıyla ilgili ImageView'e (*sehirUrl*) yüklenir. Kullanıcı bu şehir kartlarına tıkladığında, *cardBas* adlı view üzerindeki tıklama dinleyicisi aktif hale gelir. Bu dinleyici içinde, Navigation Component kullanılarak ilgili şehrin detaylarına geçiş yapılır (*SehirFragment*'e yönlendirilir), bu geçişte şehrin adı argüman olarak iletilir. Bu şekilde, kullanıcılar uygulamada gezilecek yerler hakkında bilgi alabilir ve görsellerini inceleyebilirler.

```

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/sehirName" />

```



Şekil 19 - RecyclerView gösterimi

### 3.3.2. Bilgi Ekranı

Uygulamada rota bilgilerini görüntülemek için, kullanıcılar resmin olduğu alana tıklayarak bilgi sayfasına yönlendirilirler. Bu bilgi sayfasında, gezilecek yerin adı ve detaylı bilgileri sunulur. Bu işlem, kullanıcıların rotalar hakkında daha fazla bilgi edinmelerini ve gezmek istedikleri yerler hakkında detaylı bilgilere ulaşmalarını sağlar.



Şekil 20 - Gezilecek yere ait bilgi ekranı

Bilgi ekranında, kullanıcının görmek istediği yerin detayları Firestore Database'den çekilerek ekranda gösterilir. Bu detaylar genellikle gezilecek yerin adı, açıklaması, fotoğrafı gibi bilgileri içerir. Kullanıcı bu bilgi sayfasında yer alan 'Rotaya Ekle' butonu ile o gezilecek yeri kendi oluşturduğu rotaya ekleyebilir.

```
val user = auth.currentUser?.email
var isRotaEkleCliked = false

// Kullanıcının seçtiği yerler koleksiyonunu al
val selectedPlacesRef = database.collection("usersRota").document(user!!)
    .collection("selectedPlaces")

// Veritabanında bu yeri daha önce ekleyip eklenmediğini kontrol et
selectedPlacesRef.whereEqualTo("name", placeName).get().addOnSuccessListener {
    querySnapshot ->
        if (!querySnapshot.isEmpty) {
            // Zaten bu yeri eklemişse işlem yapma
            binding.rotaEkle.text = "Rotaya eklendi "
            binding.rotaEkle.setCompoundDrawablesWithIntrinsicBounds(0, 0,
                R.drawable.check_24px, 0)
        }
    }
}
```

```

        return@addOnSuccessListener
    }else{
        binding.rotaEkle.text = "Rotaya ekle "
        binding.rotaEkle.setCompoundDrawablesWithIntrinsicBounds(0, 0,
            R.drawable.paylas, 0)
    }
}

```

Bu kod parçası, kullanıcının seçtiği bir yerin daha önce rotasına eklenip eklenmediğini kontrol eder. Öncelikle, mevcut kullanıcının Firebase Authentication üzerinden e-posta adresine erişilir (*auth.currentUser.email*). *isRotaEkleClicked* değişkeni, kullanıcının 'Rotaya Ekle' butonuna tıkladığını belirtmek için kullanılır.

Sonrasında, *selectedPlacesRef* değişkeniyle Firebase Firestore'da kullanıcının rotasını tuttuğu koleksiyonuna erişilir. *whereEqualTo("name", placeName)* sorgusuyla veritabanında o adı taşıyan bir yerin olup olmadığı kontrol edilir.

Eğer böyle bir yer zaten varsa, *addOnSuccessListener* içinde o yeri daha önce eklediğini belirten bir mesaj ve işaret gösterilir (*binding.rotaEkle.text* ve *binding.rotaEkle.setCompoundDrawablesWithIntrinsicBounds*).

Eğer bu yer daha önce eklenmemişse, *else* bloğunda 'Rotaya ekle' butonu gösterilir ve ilgili simge eklenir. Bu durumda kullanıcı, bu yeri rotasına eklemek isteyebilir ve butona tıklayarak bu işlemi gerçekleştirebilir.

```

binding.rotaEkle.setOnClickListener {
    binding.rotaEkle.text = "Rotaya eklendi"
    binding.rotaEkle.setCompoundDrawablesWithIntrinsicBounds(0, 0,
        R.drawable.check_24px, 0)

    if (!isRotaEkleClicked) {
        isRotaEkleClicked = true

        // Eğer yoksa, yeri rotaya ekle
        database.collection("Sehirler").get().addOnSuccessListener {
            cityDocuments ->
                for (document in cityDocuments) {

                    document.reference.collection("GezilecekYerler").whereEqualTo("name",
                        placeName).get()

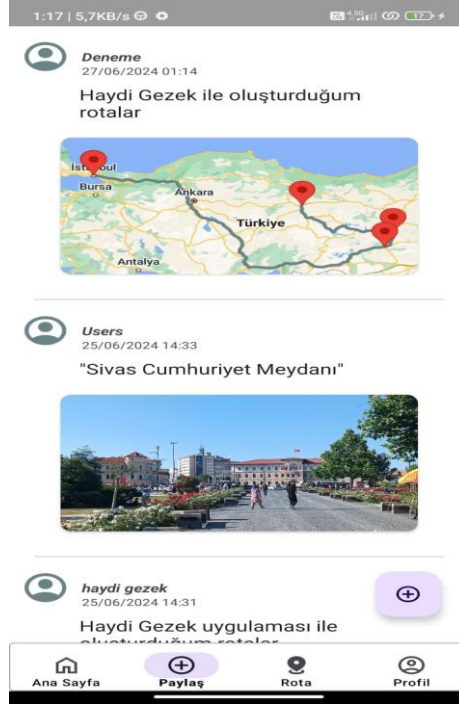
                        .addOnSuccessListener { placeDocuments ->
                            for (i in placeDocuments) {
                                val hashMap = hashMapOf(

```



### 3.4. Uygulama Paylaş Ekranı

Uygulamanın "Paylaş" ekranı, kullanıcıların gezdikleri yerler hakkında duygu, düşünce ve çektikleri resimleri paylaşabilecekleri bir platform sunmaktadır. Bu ekran, kullanıcıların deneyimlerini diğer kullanıcılarla paylaşmalarını ve gezdikleri yerler hakkında yorum yapmalarını sağlar.



Şekil 21- Uygulama Paylaş Ekranı

Burda kullanıcılar ekranda gözüken '+' butonuna basarak paylaşım ekranına geçer.

```
database.getUsers {  
    var sayac = 0  
    listUser = it  
    for (i in listUser){  
        if (listUser[sayac].email == postList[position].kullaniciEmail){  
            holder.binding.currentUser.text = listUser[sayac].name  
        }  
        sayac++  
    }  
}
```

```
holder.binding.yorumSatir.setText(postList[position].kullaniciYorum)
```

```
// Timestamp'i String olarak formatlayarak kullanma  
val sdf = SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.getDefault())
```



```
val date = postList[position].tarih.toDate()
val formattedDate = sdf.format(date)
holder.binding.tarih.text = formattedDate
```

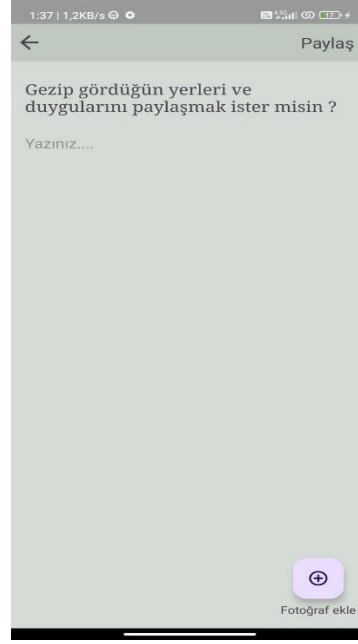
```
Picasso.get().load(postList[position].gorselUrl).into(holder.binding.resim)
```

Bu kod parçası, Firebase Firestore'dan kullanıcı verilerini çekerek, kullanıcı adının ve yorumunun gösterilmesini sağlar. İlk olarak, *getUsers* fonksiyonuyla Firebase Firestore'dan kullanıcı verileri çekilir ve *listUser* adlı bir liste değişkenine atanır. Daha sonra, *postList* içindeki her bir öge için bir döngü oluşturulur. Her döngü adımı, *listUser* listesindeki kullanıcıların e-posta adresi, *postList* içindeki *position* indeksindeki ögenin *kullaniciEmail* değeriyle karşılaştırılır. Eğer eşleşme bulunursa, bu kullanıcının adı (*listUser[sayac].name*) *currentUser* alanına atanır ve ekranda gösterilir.

Ayrıca, *yorumSatir*, *tarih* ve *resim* alanları da *postList* içindeki verilere göre doldurulur. *yorumSatir* alanına *kullaniciYorum* verisi atanır. *tarih* alanı için ise Firestore'dan gelen *tarih* verisi *toDate()* metoduyla bir *Date* nesnesine dönüştürülür, ardından *SimpleDateFormat* kullanılarak bu tarih formatlanır ve *formattedDate* olarak *tarih* alanına atanır. Son olarak, *resim* alanı için *Picasso* kütüphanesi kullanılarak *gorselUrl* adresindeki görsel *resim* alanında gösterilir.

### 3.4.1. Gönderi Ekleme Ekranı

Gönderi ekleme ekranı, kullanıcıların uygulama üzerindeki deneyimlerini resim ve metin içeriğiyle paylaşabilecekleri bir arayüz sunar. Kullanıcıların resim seçme ve metin girişi yapma imkanı vardır.

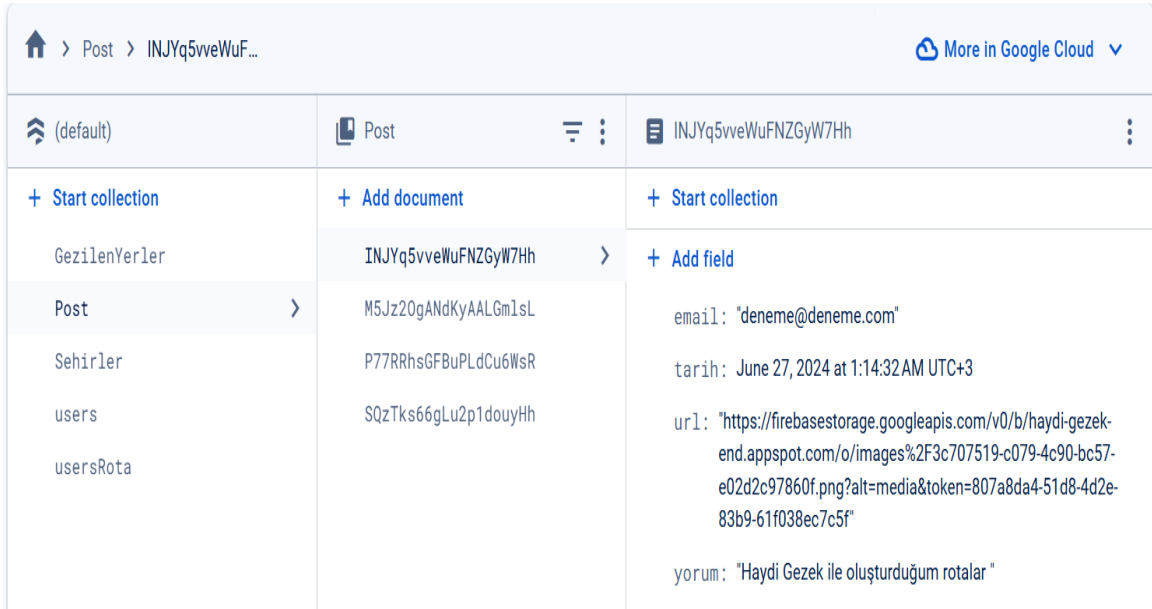


Şekil 22 - Yeni gönderi ekleme ekranı

```
fun fotografEkle(view: View){
    if (Build.VERSION.SDK_INT >= 32){
        val intent = Intent()
        intent.action = Intent.ACTION_GET_CONTENT
        intent.type = "image/*"
        startActivityForResult(intent,2)
    }else{
        if (ContextCompat.checkSelfPermission(requireContext(),
            Manifest.permission.READ_EXTERNAL_STORAGE) !=
            PackageManager.PERMISSION_GRANTED){
            //izin verilmemiş izin almalıyız
            ActivityCompat.requestPermissions(requireContext() as
            Activity,arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE) ,1)
        }else{
            //galeriye gidecez izin verildiyse
            val galeriIntent = Intent(Intent.ACTION_PICK,
            MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
            startActivityForResult(galeriIntent,2)
        }
    }
}
```

*fotografEkle* fonksiyonu, kullanıcının fotoğraf eklemek için uygulama içinde galeri veya fotoğraf seçme ekranını başlatmasını sağlar. Android 12 ve üzeri sürümlerde, kullanıcıya cihazındaki herhangi bir fotoğrafı seçme imkanı sunar (*ACTION\_GET\_CONTENT* kullanılarak). Eğer kullanıcı cihazında bu sürümden daha

eski bir Android sürümü kullanıyorsa, izin verilmişse galeri uygulaması açılarak fotoğraf seçme işlemi gerçekleştirilir; izin verilmemişse kullanıcıdan önce izin istenir (*READ\_EXTERNAL\_STORAGE* izni için).

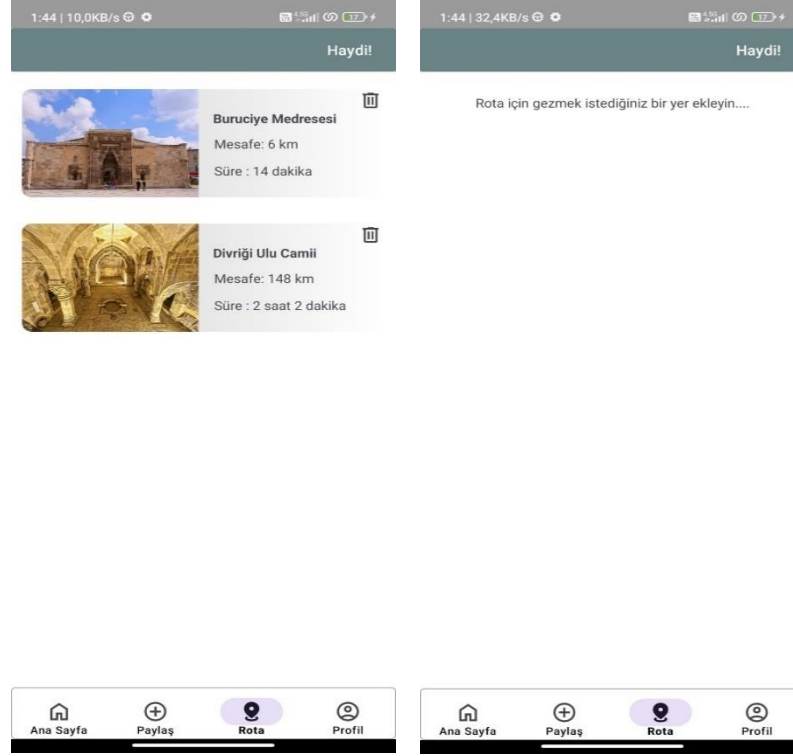


🏠 > Post > INJYq5vveWuF...			More in Google Cloud
(default)	Post	INJYq5vveWuFNZGyW7Hh	
+ Start collection	+ Add document	+ Start collection	
GezilenYerler	INJYq5vveWuFNZGyW7Hh	+ Add field	
Post	M5Jz20gANdKyAALGmIsL	email: "deneme@deneme.com"	
Sehirler	P77RRhsGFBuPLdCu6WsR	tarih: June 27, 2024 at 1:14:32 AM UTC+3	
users	SQzTks66gLu2p1douyHh	url: "https://firebasestorage.googleapis.com/v0/b/haydi-gezek-end.appspot.com/o/images%2F3c707519-c079-4c90-bc57-e02d2c97860f.png?alt=media&token=807a8da4-51d8-4d2e-83b9-61f038ec7c5f"	
usersRota		yorum: "Haydi Gezek ile oluşturduğum rotalar"	

Şekil 23 - Kullanıcı gönderilerinin veritabanında gösterilmesi

### 3.5. Uygulama Rota Ekranı

Kullanıcılar, oluşturdukları rotaları yönetebildikleri Rota Menüsü üzerinden rotaya ekledikleri yerleri görüntüleyebilirler. Her rota ögesi, rota adıyla birlikte fotoğrafını, kullanıcının konumu ile rotaya eklenen yer arasındaki mesafe ve ulaşım süresi gibi detayları içerir.



Şekil 24 - Rota menüsü

```
val directionsResult = DirectionsApi.newRequest(getGeoApiContext(apiKey))

if (directionsResult.routes.isNotEmpty() &&
directionsResult.routes[0].legs.isNotEmpty()) {
    val distanceInMeters =
directionsResult.routes[0].legs[0].distance.inMeters
    val distanceInKm = distanceInMeters / 1000.0
    roundedDistance = distanceInKm.toInt().toString()
    withContext(Dispatchers.Main) {
        holder.binding.textView12.text = "Mesafe: $roundedDistance km"
    }
}

val durationInMinutes = directionsResult.routes[0].legs[0].duration.inSeconds
/ 60
val formattedDuration = formatDurationToTurkish(durationInMinutes)
withContext(Dispatchers.Main) {
    holder.binding.textView21.text = "Süre : $formattedDuration"
}
```

Yukarıdaki kodda, Google Directions API üzerinden alınan yönlendirme sonuçlarıyla bir rota için mesafe ve süre bilgilerini görselleştirir. *directionsResult* nesnesi, belirli bir rota için yönlendirme bilgilerini içerir. Kod, bu bilgileri kullanarak ilk olarak

rota üzerindeki mesafeyi hesaplar. Mesafe, rota bacağının metre cinsinden uzunluğudur ve kilometreye dönüştürülerek tam sayı olarak *roundedDistance* değişkenine aktarılır.

Aynı şekilde, kod süre bilgisini de işler. Yönlendirme sonuçlarındaki rota bacağı süresi saniye cinsinden verilir ve *durationInMinutes* değişkeni aracılığıyla dakika cinsine dönüştürülür. Bu süre bilgisi *formattedDuration* olarak formatlanarak *holder.binding.textView21.text* üzerinde "Süre: [formattedDuration]" şeklinde kullanıcıya sunulur. Kullanıcılar bu bilgileri görerek, belirli bir rotanın tahmini mesafesini ve süresini uygulama üzerinde gözlemleyebilirler.

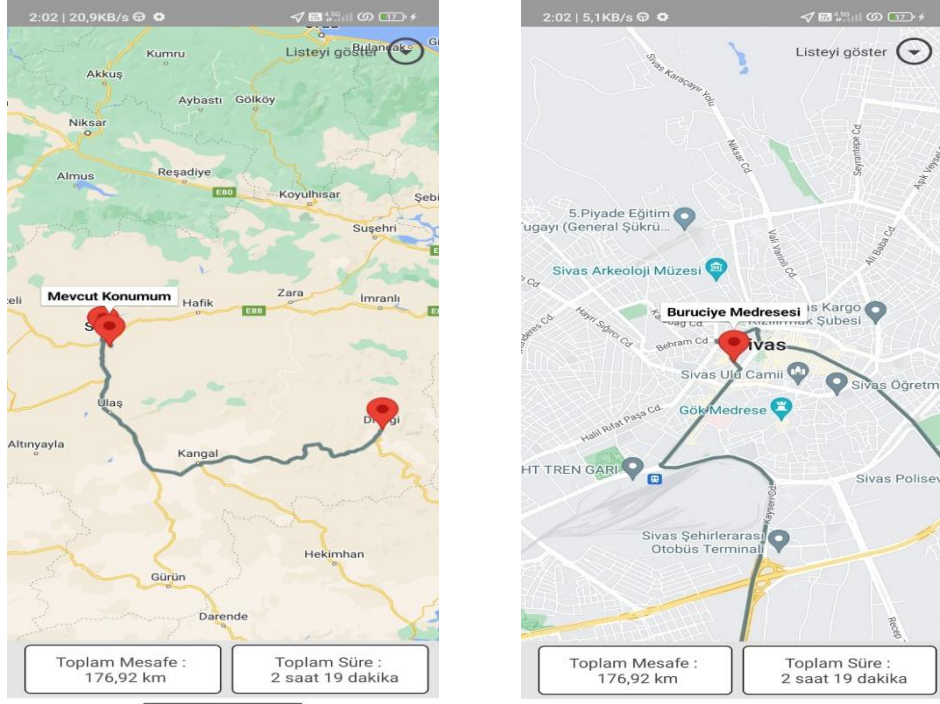
deneme@deneme.com	selectedPlaces	buruciye+medresesi
+ Start collection	+ Add document	+ Start collection
selectedPlaces	buruciye+medresesi	+ Add field
+ Add field	divri%c4%9fi+ulu+camii	boylam: "37.0129122"
		enlem: "39.7490268"
		name: "Buruciye Medresesi"
		url: "https://firebasestorage.googleapis.com/v0/b/haydi-gezek-end.appspot.com/o/Sivas%2Fburuciye.jpeg?alt=media&token=28fa36d4-68b4-4590-be58-64a3e6ea8212"
This document does not exist, it will not appear in queries or snapshots. <a href="#">Learn more</a>		

Şekil 25 - Kullanıcın seçtiği yerlerin veritabanında gösterimi

### 3.5.1. Harita Ekranı

Kullanıcı 'Rota' ekranında seçtiği yerleri gördükten sonra 'Haydi!' butonuna bastıktan sonra harita sayfasına yönlendirilir.

Harita sayfası, kullanıcıya seçtiği rotanın detaylarını ve harita üzerinde gösterilen rotanın grafiksel temsilini sunarak yolculuk öncesi bilgi sağlar. Bu sayede kullanıcılar, seyahat rotalarını daha iyi planlayabilir ve yolculukları boyunca navigasyon desteğinden faydalanabilirler.



Şekil 26- Harita sayfası

Kullanıcı, harita üzerinde kırmızı ile gösterilen işarete bastığında, o konumdaki yerin ismi görünür hale gelir ve haritada seçilen konuma yaklaşılr. Bu işlev genellikle kullanıcıların belirli bir noktayı detaylı olarak incelemesini ve bulundukları konuma göre bu noktaya olan uzaklığı veya ilişkili bilgileri öğrenmelerini sağlar.

Haritada mevcut konumdan seçilen konumlar arasında sırayla en yakından en uzağa doğru yol çizilir. Bu yola göre mesafe ve süre belirlenir.

```
database.collection("usersRota")
  .document(auth.currentUser?.email!!)
  .collection("selectedPlaces")
  .get()
  .addOnSuccessListener { documents ->
    for (document in documents) {
      val enlem = document.getString("enlem")?.toDouble()
      val boylam = document.getString("boylam")?.toDouble()
      if (enlem != null && boylam != null) {
        selectedLocations.add(LatLng(enlem, boylam))
      }
    }
  }
```

Kullanıcının seçtiği rotalar Firestore Database'den çekilir. Bu verilerden enlem ve boylamı alınır ve harita üzerinde bununla ilgili işlemler yapılır.

```
private lateinit var mMap: GoogleMap
locationCallback = object : LocationCallback() {
    override fun onLocationResult(locationResult: LocationResult) {
        locationResult.lastLocation?.let { location ->
            val myKonum = LatLng(location.latitude, location.longitude)
        }
    }
}
mMap.addMarker(MarkerOptions().position(myLocation).title("Mevcut Konum"))
```

Yukarıdaki kodda `mMap` değişkeni, Google Haritalar API'sinde kullanılan ve harita nesnesini temsil eden bir değişken olarak tanımlanmıştır. `addMarker` metodu, `MarkerOptions` nesnesi aracılığıyla belirtilen konum ve başlık bilgileriyle haritaya bir işaretçi ekler. Bu işaretçi, kullanıcının mevcut konumunu veya belirli bir noktayı gösterir. `moveCamera` metodu ise `CameraUpdateFactory` kullanarak haritanın belirli bir konuma ve belirli bir yakınlaştırma düzeyine odaklanmasını sağlar.

`LocationCallback()` kullanarak kullanıcı konumu alındığında, konum bilgisi `LatLng` tipinde enlem ve boylam olarak elde edilir. Bu bilgiler, kullanıcının mevcut konumunu belirlemek ve harita üzerinde göstermek için kullanılır.

```
if (!selectedLocations.isNullOrEmpty()) {
    val sortedLocations = listOf(myKonum) + selectedLocations.sortedBy {
        calculateDistance(myKonum, it)
    }
}

for (i in 0 until sortedLocations.size - 1) {
    drawRoute(sortedLocations[i], sortedLocations[i + 1])
}
addMarkers(selectedLocations + listOf(myKonum))

private fun calculateDistance(location1: LatLng, location2: LatLng): Float {
    val result = floatArrayOf(0f)
    Location.distanceBetween(location1.latitude, location1.longitude,
        location2.latitude, location2.longitude, result)
    return result[0] / 1000 // Metreyi kilometreye çevir
}
```

Bu kod, bir harita üzerinde birden fazla konuma gitmenin en kısa rotasını bulmak ve rotayı çizmek için kullanılan bir algoritmayı gösterir.

*myKonum* (kullanıcının mevcut konumu) ile *selectedLocations* listesi, *calculateDistance* fonksiyonu kullanılarak sıralanır. Bu fonksiyon, iki konum arasındaki mesafeyi hesaplar. Daha sonra, sıralanmış konumlar üzerinde bir döngü ile her bir konumdan diğerine rotalar çizilir (*drawRoute* fonksiyonu kullanılarak). Son olarak, *addMarkers* fonksiyonuyla seçilen konumlar ve mevcut konum haritaya işaretlenir.

```
private fun drawRoute(origin: LatLng, destination: LatLng) {
    val apiKey = "api_key"

    val directionsResult =
        DirectionsApi.newRequest(getGeoApiContext(apiKey))
            .mode(TravelMode.DRIVING)
            .origin("${origin.latitude},${origin.longitude}")
            .destination("${destination.latitude},${destination.longitude}")
            .await()

    val polylineOptions = PolylineOptions()
        .color(0xFF6A8385.toInt()) // Çizgi rengi
        .width(11f) // Çizgi kalınlığı

    for (step in directionsResult.routes[0].legs[0].steps) {
        val points = step.polyline.decodePath()
        for (point in points) {
            polylineOptions.add(LatLng(point.lat, point.lng))
        }
    }
}
```

"drawRoute" fonksiyonu, harita üzerinde iki konum (origin ve destination) arasındaki en kısa rotayı çizmek için "Polyline" adı verilen bir çizgi kullanır. Bu çizgiyi oluşturmak için öncelikle bir harita servisi ile iletişim kurulur. API anahtarı kullanılarak ("getGeoApiContext") bir rota isteği ("DirectionsApi.newRequest") oluşturulur. Bu istek, başlangıç ve hedef konum bilgilerini ("origin" ve "destination") ve seyahat modunu ("TravelMode.DRIVING") içerir. Harita servisi, en kısa rotayı tanımlayan noktaları döndürür. Bu noktalar daha sonra çizgi özelliklerine ("PolylineOptions") eklenerek, belirli bir renk ve kalınlıkta ("color" ve "width") bir çizgi oluşturulur. Böylece, harita üzerinde kullanıcı için görsel bir rota oluşturulmuş olur.

```
private fun startLocationUpdates() {
    val locationRequest = LocationRequest.create().apply {
        interval = 60000 // 1 dakika
        //Konum güncellemelerinin alınma süresi. Burada, 60.000 milisaniye (1
        dakika) olarak ayarlanmıştır.
        fastestInterval = 5000 // 5 saniye
        //Konum güncellemelerinin alınma süresi için minimum süre. Bu değer,
```



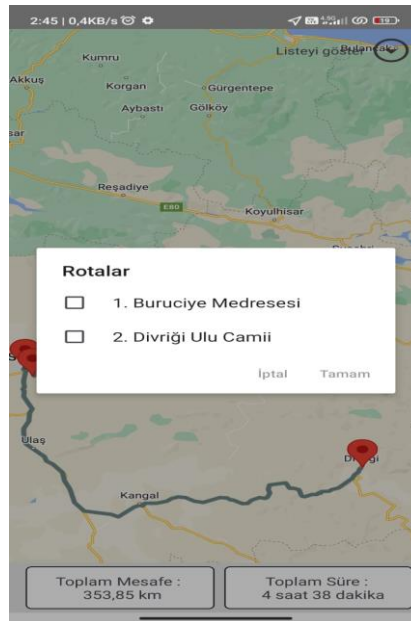
```

5.000 milisaniye (5 saniye) olarak belirlenmiştir.
    priority = LocationRequest.PRIORITY_HIGH_ACCURACY
}

if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED) {
    fusedLocationClient.requestLocationUpdates(locationRequest,
locationCallback, null)
}
}

```

"startLocationUpdates" fonksiyonu, kullanıcının konumunu belirli aralıklarla güncellemek için kullanılır. Bu fonksiyon, konum güncellemelerinin alınma süresini ve minimum süresini belirtir ve konum doğruluğunu yüksek olarak ayarlar. Uygulama, konum izinlerine sahipse, konum güncellemeleri başlatılır ve alınan konum bilgileri "locationCallback" fonksiyonuna aktarılır. Bu fonksiyon, alınan konum bilgilerini işlemeyi sağlar.

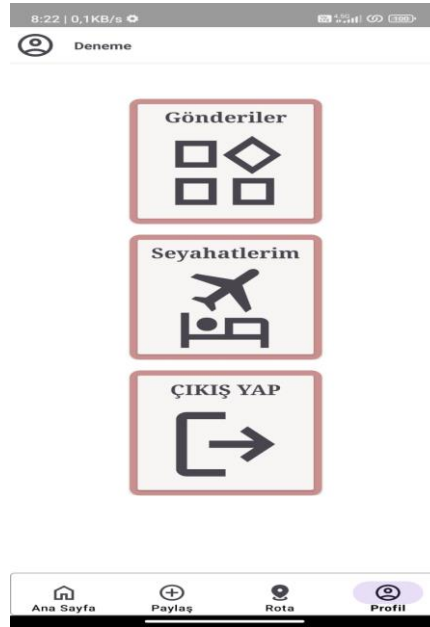


Şekil 27 - Haritada gezilen yerler listesi

"Listeyi Göster" butonuna basıldığında, oluşturulan rota içerisindeki yerlerin bir listesi görüntülenir. Bu liste, her bir yerin kullanıcıya olan mesafesine göre en yakından en uzağa doğru sıralanmıştır. Kullanıcı, bu listeden bir veya daha fazla yeri seçerek rotadan çıkarabilir. Seçilen yerler, gezilmiş olarak kabul edilir ve "Seyahatlerim" sayfasına eklenir.

### 3.6. Uygulama Profil Ekranı

Uygulamanın profil sayfası, kullanıcıların gezilerini ve paylaşımlarını yönetmelerine olanak tanıyan sade ve kullanıcı dostu bir arayüz sunmaktadır. Profil sekmesi altında, kullanıcılar "Gönderiler", "Seyahatlerim" ve "Çıkış Yap" seçeneklerine kolayca erişebilirler. "Gönderiler" bölümü, kullanıcının uygulamada paylaştığı içeriklerin tümünü görüntülemesini sağlar. "Seyahatlerim" seçeneği, kullanıcının gerçekleştirdiği seyahatlerin bir özetini sunar, böylece kullanıcılar gezilerini kolayca yönetebilirler. "Çıkış Yap" butonu ise, kullanıcıların uygulamadan güvenli bir şekilde çıkış yapmalarını sağlar. Bu yapı, kullanıcı deneyimini artırmak ve gezinmeyi kolaylaştırmak amacıyla tasarlanmıştır.



Şekil 28 - Profil ekranı

```
<LinearLayout
    android:id="@+id/linearLayout3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    android:gravity="center"
    android:orientation="horizontal"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <androidx.cardview.widget.CardView
        android:id="@+id/gonderiBas"
        android:layout_width="175dp"
```

```

        android:layout_height="175dp"
        android:layout_gravity="center"
        app:cardBackgroundColor="#CA8F8F"
        app:cardCornerRadius="10dp">

        <androidx.cardview.widget.CardView
            android:layout_width="160dp"
            android:layout_height="160dp"
            android:layout_gravity="center"
            app:cardBackgroundColor="#F8F5F5">

            <androidx.constraintlayout.widget.ConstraintLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent">

                <TextView
                    android:id="@+id/textView23"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Gönderiler"
                    android:textSize="20sp"
                    android:textStyle="bold"
                    android:typeface="serif"
                    app:layout_constraintBottom_toTopOf="@+id/imageView16"
                    app:layout_constraintEnd_toEndOf="parent"
                    app:layout_constraintStart_toStartOf="parent"
                    app:layout_constraintTop_toTopOf="parent"
                    app:layout_constraintVertical_bias="0.484" />

                <ImageView
                    android:id="@+id/imageView16"
                    android:layout_width="120dp"
                    android:layout_height="120dp"
                    android:layout_marginBottom="4dp"
                    android:scaleType="centerCrop"
                    android:src="@drawable/widgets_48px"
                    app:layout_constraintBottom_toBottomOf="parent"
                    app:layout_constraintEnd_toEndOf="parent"
                    app:layout_constraintHorizontal_bias="0.485"
                    app:layout_constraintStart_toStartOf="parent" />
            </androidx.constraintlayout.widget.ConstraintLayout>
        </androidx.cardview.widget.CardView>

    </androidx.cardview.widget.CardView>

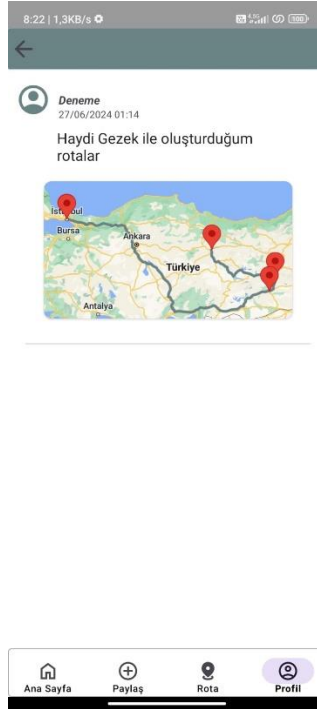
</LinearLayout>

```

Yukarıdaki kodda “Gönderiler” kısmının nasıl oluştuğu gösterilmektedir. Bu kodda ‘*LinearLayout*’ içinde yer alan iki katmanlı ‘*CardView*’ yapısı kullanılmıştır. Dıştaki ‘*CardView*’, yuvarlak köşeleri ve belirgin arka plan rengi ile kullanıcıya güzel görsel bir deneyim sunar. İç kısımda yer alan ‘*CardView*’ ise beyaz arka planıyla içerik vurgusunu artırır. Üst kısımda ‘*TextView*’ ile “Gönderiler” başlığı belirtilmiştir ve alt kısmında ‘*ImageView*’ ile görsel öğe eklenmiştir.

### 3.6.1. Gönderiler

Gönderiler sayfasında kullanıcın kendine ait gönderileri yer alır. Böylelikle daha kolay gönderilerine erişebilir.



Şekil 29 - Kullanıcı Gönderileri

Kullanıcı gönderileri Firestore Database'den gerekli veriler çekilir. Ve gönderiler sayfasına RecyclerView kullanılarak gösterilir.

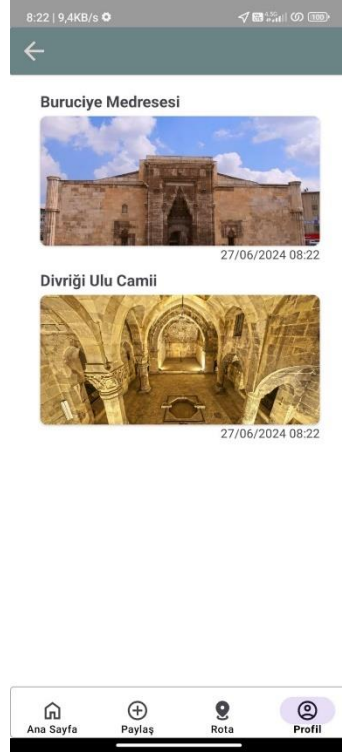
```
val currentUserEmail = auth.currentUser?.email
if (currentUserEmail != null) {
    database.collection("Post")
        .whereEqualTo("email", currentUserEmail)
        .get()
        .addOnSuccessListener { documents ->
            for (document in documents) {
                val email = document.get("email") as String
                val yorum = document.get("yorum") as String
                val url = document.get("url") as String
                val tarih = document.getTimestamp("tarih") ?:
Timestamp(Date())

                val postListesi = Post(email,yorum,url,tarih)

                postList.add(postListesi)
            }
            adapter.notifyDataSetChanged()
        }
}
```

### 3.6.2. Seyahatlerim

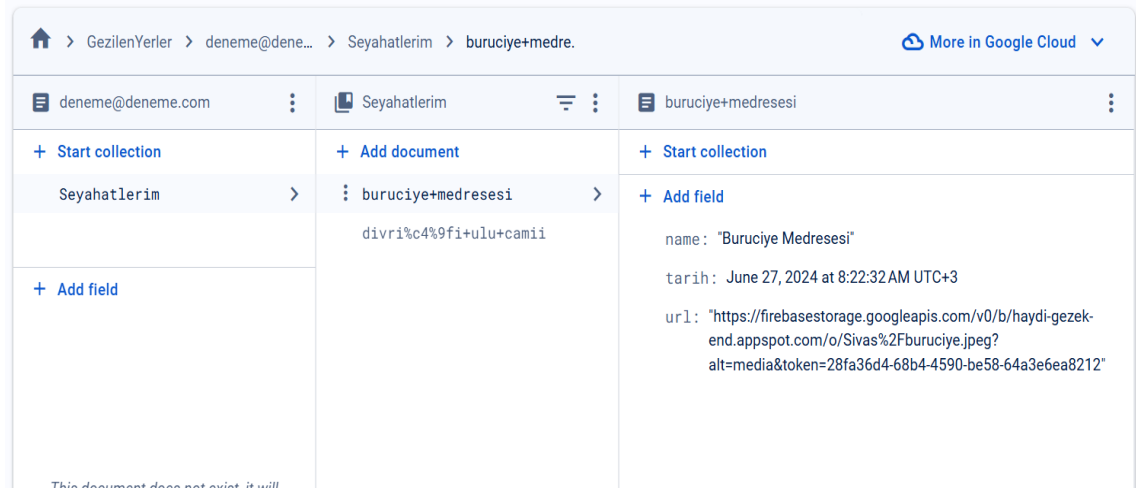
"Seyahatlerim" bölümü, kullanıcıların uygulama içerisinde oluşturdukları ve gezdikleri rotaları görüntülemelerine olanak tanır. Bu bölüm, kullanıcıların geçmişte ziyaret ettikleri yerleri kolayca takip edebilmeleri için tasarlanmıştır. Kullanıcılar, daha önce oluşturdukları rotalarda gezdikleri yerleri bu alanda görebilirler.



Şekil 30 - Kullanıcının gezdiği yerler

Şekil 30'da, kullanıcıların gezdiği yerlerin tarih bilgisiyle birlikte nasıl görüntülendiğini göstermektedir. Bu bölümde, kullanıcılar gezdikleri yerlerin fotoğraflarını ve ziyaret tarihlerini görebilirler.

Şekil 31'de ise Firebase Database'de verilerin nasıl görüntülediği gösterilmiştir.



Şekil 31 - Seyahatlerim veritabanı gösterimi

### 3.6.3. Çıkış

Kullanıcı, oturumunu kapatmak istediğinde "Çıkış Yap" butonuna basarak uygulamadan güvenli bir şekilde çıkış yapabilir. Bu özellik, kullanıcının hesabını ve kişisel verilerini koruma amacı taşır. Çıkış işlemi, kullanıcının profil sayfasında yer alan basit ve erişilebilir bir buton ile gerçekleştirilir. Kullanıcı bu butona bastığında, oturumu sonlandırılır ve uygulamanın ana giriş ekranına yönlendirilir.

```
fun cikisYap(view: View){
    //çıkış yapınca authenticationdan da çıkış yapıldı
    val build = AlertDialog.Builder(requireContext())
    build.setTitle("Çıkış")
    build.setMessage("Çıkmak istediğine emin misin?")
    build.setPositiveButton("Evet", DialogInterface.OnClickListener { dialog,
which ->
        auth.signOut()
        val intent = Intent(requireContext(), LoginActivity::class.java)
        startActivity(intent)
    })
    build.setNegativeButton("Hayır", DialogInterface.OnClickListener {
dialog, which ->
        dialog.dismiss()
    })
    build.setCancelable(false)
    build.create().show()
}
```

Uygulamada, kullanıcının oturumunu güvenli bir şekilde kapatabilmesi için "Çıkış Yap" butonu bulunmaktadır. Kullanıcı bu butona bastığında, *cikisYap()*

fonksiyonu çalışır ve bir *AlertDialog* ile onay ekranı görüntülenir (Şekil 32). Onay ekranında, kullanıcının çıkış yapma isteği doğrulanır. Kullanıcı "Evet" butonuna bastığında, *auth.signOut()* fonksiyonu çağrılarak kullanıcının oturumu kapatılır ve kullanıcı giriş ekranına yönlendirilir. Bu yönlendirme, *LoginActivity'e* bir *Intent* ile gerçekleştirilir. Kullanıcı "Hayır" butonuna bastığında ise dialog kapatılır ve oturum kapatma işlemi iptal edilir. *AlertDialog'un setCancelable(false)* özelliği ile kullanıcının dialog dışında bir yere dokunarak dialogu kapatması engellenir, böylece kullanıcıya net bir seçim yapma fırsatı sunulur.



Şekil 32 - Çıkış ekranı

#### 4. SONUÇLAR VE ÖNERİLER

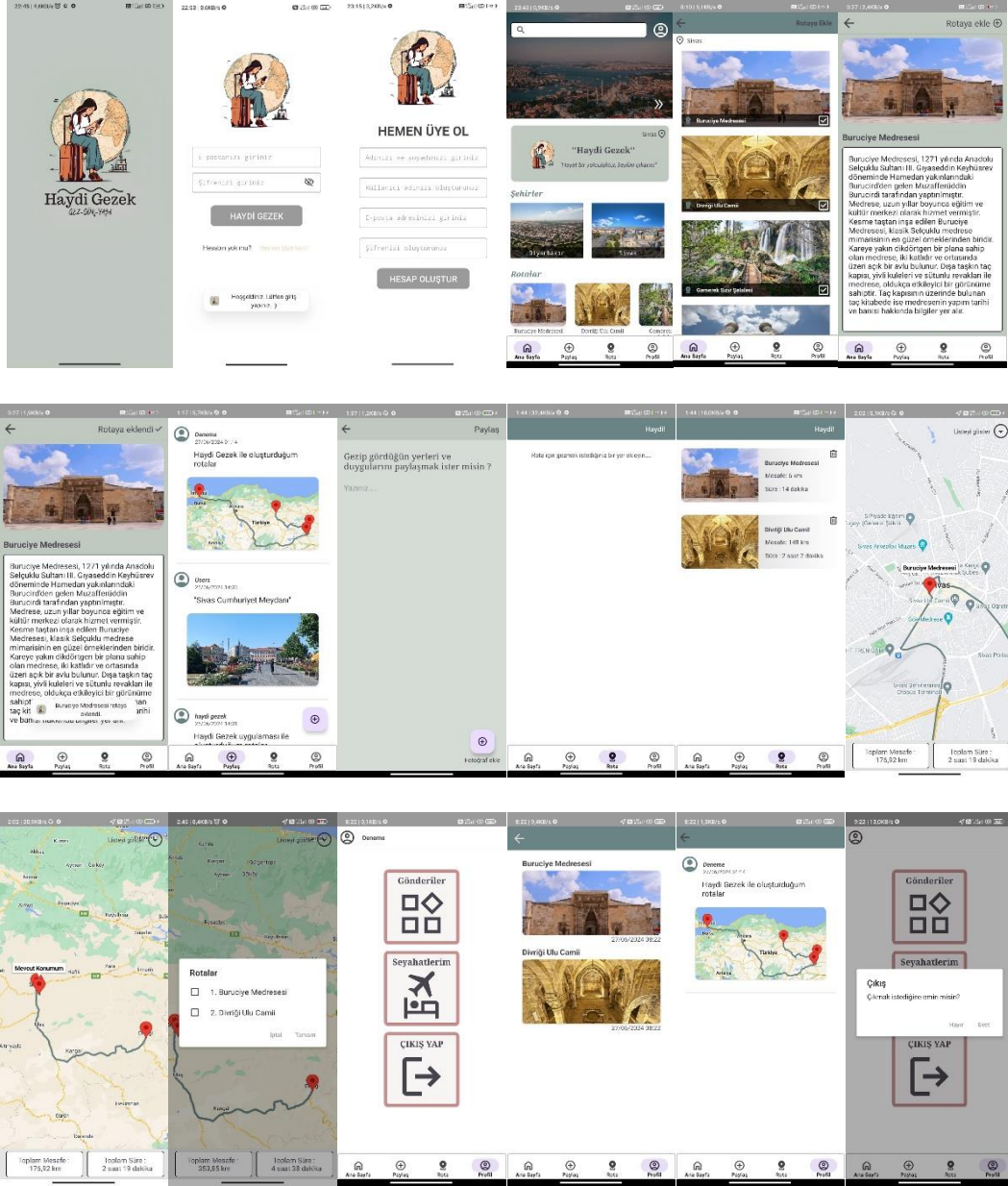
Şehir içi gezi planlama uygulaması olan Haydi Gezek, kullanıcılara basit ve anlaşılabilir bir arayüz sunarak gezmek istedikleri yerlere rahatlıkla ulaşabilmelerini ve bu yerler hakkında detaylı bilgilere erişebilmelerini sağlamaktadır. Uygulama, kullanıcılara ilgi çekici yerleri keşfetme imkanı sunarak kişiselleştirilmiş rotalar oluşturabilme olanağı tanır. Kullanıcılar, seçtikleri yerleri "Rotaya Ekle" butonuna basarak rotalarına ekleyebilir ve bu rotaları kolayca takip edebilirler. Bu özellik, kullanıcıların seyahatlerini planlamalarını ve düzenlemelerini daha etkili hale getirir.

Uygulama geliştirilirken Firebase Firestore veri tabanı kullanılmıştır. Kullanıcılar eş zamanlı olarak gezilecek yerler hakkında güncel bilgilere ulaşabilir. Uygulama internet tabanlı olduğu için internet erişimi olmadığında bazı sorunlarla karşılaşılabilir, bu da uygulamanın kullanılabilirliğini etkileyebilir. Proje geliştirilirken internet erişimi sorunlarıyla sık sık karşılaşılmıştır. Kotlin dilinde Android Studio kullanılarak geliştirilen uygulama, Android platformu üzerinde sorunsuz çalışması hedeflenmiştir.

Kayıt oluşturma aşamasında kullanıcılardan sadece ad, soyad ve e-posta gibi gerekli ve yeterli bilgiler istenmektedir. Kullanıcılar hakkında daha fazla bilgi istemek, veri gizliliği ve güvenliği açısından risk oluşturabilir. Bu nedenle, geliştiriciler ve kullanıcılar bu riski almamakta ve böylece veri güvenliği sağlanmaktadır. Android bileşenlerinin kullanımı ve yapısının basit olması, geliştirici açısından büyük kolaylık sağlamaktadır. Ancak birden fazla bileşen kullanıldığında kodlar karmaşık bir hal alabilir, bu nedenle işlemleri farklı sınıflara bölerek karmaşıklık azaltılabilir.

Uygulamanın kullanıcı geri bildirimleri olumlu yöndedir ve kullanıcılar tarafından kolaylıkla kullanılabilmektedir. Ancak, performans iyileştirmeleri ve kullanıcı deneyimini artırmak için önerilerde bulunulabilir. Örneğin, harita üzerindeki işaretçilerin daha etkili bir şekilde yönetilmesi ve kullanıcıların gezdikleri yerler hakkında daha kapsamlı bilgiler paylaşmalarına olanak tanınması gibi öneriler geliştirilebilir. Bu şekilde, uygulamanın kullanıcı memnuniyetini ve kullanıcı katılımını artırmak için ileriye dönük adımlar atılabilir.





## KAYNAKÇA

- 1- <https://developer.android.com/studio>
- 2- <https://firebase.google.com/>
- 3- <https://www.udemy.com/course/android-mobil-uygulama-gelistirme-egitimi-kotlin/?couponCode=ST18MT62524>
- 4- <https://www.udemy.com/course/android-o-mobil-uygulama-dersi-kotlin-java/?couponCode=ST18MT62524>
- 5- <https://www.btkakademi.gov.tr/portal/course/kotlin-ile-android-mobil-uygulama-gelistirme-ileri-seviye-10359>
- 6- <https://www.btkakademi.gov.tr/portal/course/kotlin-ile-android-mobil-uygulama-gelistirme-temel-seviye-10274>
- 7- <https://www.canva.com>
- 8- <https://stackoverflow.com>