Yazılım Laboratuvarı Bulut Tabanlı Nesne Tespiti

Özge POYRAZ
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
180202025@kocaeli.edu.tr

Burak Can TEMİZEL
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
180202024@kocaeli.edu.tr

***Özet*—Bu çalışmada, çapraz platform destekli, kullanıcıdan aldığı resimleri bulut tabanlı bir api üzerinde nesne tespiti işleminden geçirerek daha sonrasında resim üzerinde gerekli işaretlemeleri yapan bir mobil uygulama geliştirdik. Uygulamanın istemci tarafını React Native ile geliştiririrken bulut platformu olarak Google Cloud kullandık. Bulut üzerinde kendi Node.js api sunucumuzu oluşturduk ve istemci mobil uygulama ile haberleşmesini sağladık.**

Anahtar kelimeler—Mobil Uygulama, Çapraz Platform, React Native, Bulut Teknolojileri, Google Cloud, API, Server Node.js, Görüntü İşleme, Nesne Tespiti.

I. GIRIŞ VE PROBLEM TANIMI

Gerçekleştirdiğimiz projenin tanımını yaparsak, kullanıcı mobil uygulama üzerinden isteğe bağlı olarak galeri veya kamera aracılığıyla bir resim seçiyor ardından bu resim bulut teknolojileri kullanılarak, kendi geliştirdiğimiz api üzerinde nesne tespiti işlemine sokuluyor ve daha sonrasında elde edilen sonuçlar ile bu resim üzerinde gerekli olan objelerin işaretlenmeleri gerçekleştirip yine uygulama üzerinde kullanıcıya sunuluyor. Proje en temelinde mobil uygulama ve api kısımlarından oluşuyor.

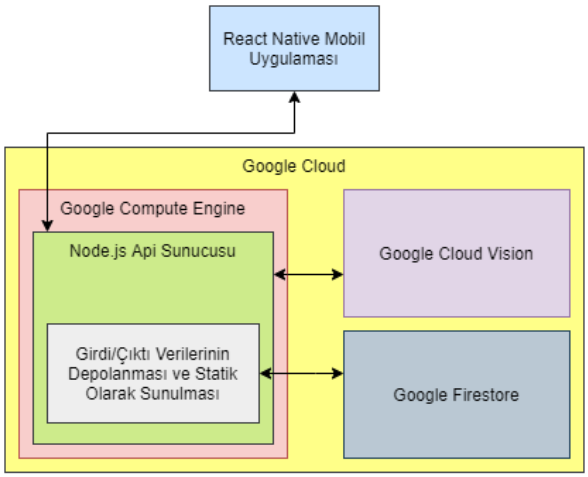
Mobil uygulama tarafı React Native frameworku kullanılarak çapraz platform olarak gerçeklenirken, Bulut teknolojisi olarak Google Cloud'ı tercih ettik. Google Cloud üzerinde bir Node.js web serverı kullandık. Bu web server hem api olarak hizmet ediyor hem de veritabanımız için depolama görevi görüyordu. Api, mobil uygulama ile haberleşerek ilgili resimleri alıyor, görüntü işleme, veritabanı ve resim işaretleme gibi işlemleri gerçekleştirerek tekrardan uygulamaya dönüyor. Temel olarak bu şekilde çalışan programımızın mimarisi ilerleyen kısımlarda daha detaylı açıklanacaktır.

II. YÖNTEMLER, YAKLAŞIMLAR VE PROGRAM MIMARISI

Bu kısımda programın farklı özelliklerini oluşturmak için kullandığımız araçlar ve yöntemler üzerinde durularak ayrıntılı olarak bilgi verilecektir. Program mimarisi daha detaylı bir şekilde açıklanacaktır.

A. Genel Program Mimarisi

Projeyi temel olarak iki kısma ayırsak da aslında daha alt kademelerde farklı hizmetler ve teknolojilerden de faydalandık. İstemci tarafı tamamen React Native ile geliştirilirken, api tarafına baktığımızda teknolojilerin çeşitlendiğini görebiliriz. Temel olarak apimiz bir Node.js web serverı ve Google Cloud Compute Engine üzerinde bir Linux sanal makinesinde çalışmakta. Api görüntü işleme için yine bir Google Cloud hizmeti olan Vision Api'yi kullanmakta. Web serverımız, veritabanımız için gelen ve çıktı resimlerin depolamasını statik sunum tekniğiyle yapmakta fakat veritabanının ilişkisel olarak sunulması yine bir Google Cloud hizmeti olan Firestore ile sağlanmaktadır. Aşağıda program mimarisinin daha detaylı bir şeması bulunmaktadır.



Genel Program Mimarisinin Basitleştirilmiş Şeması

B. React Native Mobil Uygulaması

Projenin istemci kısmı olan mobil uygulama için ilk olarak bir React Native projesi oluşturduk ve genel hatlarıyla uygulamayı tasarladık. İsterlerden ötürü kullanıcı hem galeriden hem de kameradan resim seçimi yapabilmeliydi. Bu kısımda hem ios hem Android için kamera ve galeriye direkt olarak ulaşabilmek adına React Native Image Picker kütüphanesinden faydalandık.[1] Projemize ilgili kütüphaneyi kurduk ve linkledik. Daha sonra programımızın ana ekranı için bir tasarım oluşturduk. Bazı görseller, yazılar ve butonlar ile bir ana ekran oluşturup Image Pickerın launchCamera ve launchImageLibrary metotlarını bu butonlara atadık. Bu metotlar bizden bir ayar objesi alıp daha sonrasında response olarak sistemden seçilen resme ait çeşitli bilgileri bir obje halinde dönüyordu. Bu response objesinin içerisinde dosyanın ismi, adresi, boyutları, base64 hali gibi çok çeşitli veriler bulunuyordu. ImagePicker son versiyonunda standart ayarlarda android tarafında herhangi bir permission ayarına ihtiyaç duymuyordu bu yüzden herhangi bir ek ayar yapmamıza gerek kalmadı. Eğer resimleri kaydetmek istersek buna ihtiyaç doğuyordu fakat zaten biz cachede olan resmi direkt olarak apimize yollayacaktık. Ayrıca bu metotlar içerisinde varsayılan çeşitli hata yakalama işlemlerini yaparak oluşabilecek hatalar sonucunda gerektiğinde metodun sonlanmasını sağlıyorduk.

Bu noktadan sonra kullanıcı kameradan ya da galeriden resim seçme butonuna bastığında sistem üzerinde native bileşenlerle, varsayılan sistem uygulamaları çalışacak ve kullanıcı resmini seçecekti. Ardından bize dönen resimle ilgili olan javascript objesi apiye yollanmak üzere ilgili fonksiyona gönderilecekti. Bu response objesini direkt olarak

apimize yollayacaktık çünkü bizim veritabanımızdaki ögelerde api içerisinde depolanıyordu. Ardından ilgili fonksiyon içerisinde bir post isteği oluşturuluyordu. Burada resmi direkt olarak apiye yollayacağımız için biçim olarak form-data kullanıyorduk. Form-data oluşturan bir metot hazırladık ve içerisinde image pickerdan dönen response objesinden gerekli 3 parametre olan url, isim ve tip bilgilerini alarak form-data objesini oluşturduk. Artık geriye sadece bunu apiye postlamak kalmıştı. Fetch fonksiyonu ile api üzerinde "/upload" endpointimize bir post isteği yolladık. Bu noktadan sonra api gerekli işlemleri yapacak ve bize işlenmiş resmin ve bilgilerin son halini bir obje olarak dönecekti. Mobil uygulamanın tek yapması gereken apiden dönen response'u ve bilgileri son bir ekranda kullanıcıya göstermekti. Daha sonrasında bu ekran geçişleri için state objesi altında birtakım değişkenlerle sahne yapısını oluşturduk.

Program menü, işlem ve çıktı olarak 3 sahneden oluşuyordu. Apiye istek gönderildiğinde sahne menüden işleme çevriliyor, apiden response geldiğinde işlemden, çıktıya çevriliyor ve tekrar işlem yapılmak istendiğinde çıktıdan menüye çevriliyordu. Sahne sistemi oluştuktan sonra geri kalan tüm işlem render metodu altında gerekli komponentleri ilgili sahne değişkenlerinin değerlerine göre çizdirmekti. Tüm komponentleri ayarlayarak programı bitirdik. Daha sonrasında stil düzenlemeleri yaparak programı görselleştirdik. İstemci olan mobil uygulamayı bu şekilde tamamladık.

C. Bulut Platformu, API ve Veritabanı

Daha önce bahsettiğimiz gibi projemizde Google Cloud teknolojilerini kullandık. Apimiz için Node.js ile bir web server yazmaya karar verdik. Bunu deploylamak için Google Cloud Compute Engine'i seçtik. App Engineden farklı olarak depolama işlemlerini api üzerinde gerçekleştirip veritabanına referans vermek bizim tasarladığımız mimari için daha uygundu. Compute Engine üzerinde bir linux sanal makinesi oluşturduk daha sonrasında sunucu programımızı buraya git ile aktaracaktık fakat öncesinde sunucu üzerinde bazı yapılandırmalar yaptık. Öncelikle Cloud ayarlarından sunucumuzun firewall yapılandırmasını yaptık ve apimiz için ihtiyaç duyduğumuz 4000 portuna gerekli izinleri verdik. Daha sonra apimiz için sabit bir ip adresi ayırdık. Bu ip adresine istemciden post isteği atılacaktı ve sunucuda bu adreste çalışacaktı. Api üzerinde işlemlerin gerçekleştiği endpointi "/upload" olarak belirledik. Api sunucusu üzerinde uploads ve outputs adı altında iki adet dizin oluşturduk. Node.js uygulamamızı express ile bir web server haline getirerek bu iki dizini statik olarak sunduk. Verilerimiz bu dizinlerde depolanıp veritabanına referans verilecekti.

Api üzerindeki işlemler "/upload" endpointine gelen post isteği ile başlıyordu. Öncelikle form-data biçimi içerisinden resim dosyası yakalanıp bir değişkene atanıyor. Daha sonrasında uploads dizini altında her gelen istek için eşi benzeri olmayan bir isim uuid kütüphanesi ile oluşturuluyor ve dizin altına kaydediliyor. Artık resim sunucu üzerinde ve hemen ardından görüntü işleme için Google Cloud Vision Api'ye bir request isteği oluşturuluyor ve resim yollanıyor.[2] Burada gerekli nesne tespiti işlemleri gerçekleştirilip ilgili response objesi sunucuya döndürülüyor. Resim dosyası üzerinde işaretlemelerin gerçekleştirilmesi için node.js canvas sınıfından faydalanıyor. Sizeof kütüphanesi ile resmin boyutları alınarak bu boyutlarda yeni bir canvas oluşturuluyor. Canvas içerisine sunucudaki resim çizdiriliyor ve Vision Api'den dönen response objesi üzerinde tespit edilen objelerin köşe konumlarından faydalanılarak, primitive çizim fonksiyonları ile resim üzerinde işaretlemeler gerçekleştiriliyor. Cloud Vision'ın sağladığı obje isminden faydalanılarak resim üzerine objelerin isimleri de çizdiriliyor. Toplam obje adeti sol alt köşeye çizdiriliyor. Çizim işlemleri bitip resim son halini aldıktan sonra canvas bufferlanarak bir resim dosyası olarak aynı isim ile outputs klasörüne kaydediliyor.

Bu noktadan sonra sunucu üzerinde hem resmin işlenmemiş hem de işlenmiş hali oluşmuş oluyor. Daha sonrasında istemciye bir response dönülüyor bu response içerisinde resmin işlenmiş halinin adres referansı, isteğin başarılı mı başarısız mı olduğunun bilgisi, tespit edilen obje adeti ve objelerin listesi de bulunuyor. Resim üzerinde işaretlenmiş olsa da bilgiler istemci programa arayüz bileşenleriyle de çizdirilebilmeleri için gönderiliyor. Api sunucumuzun statik dizinlerinde işlenmiş ve işlenmemiş resimleri tuttuğunu söylemiştik bu resim dosyalarının adres referansları api üzerinden firestore veritabanına kaydediliyor ve obje listesi ve adeti de veritabanına ekleniyor. Bu noktadan sonra istemciye işlenmiş resim ve nesne tespitine ait tüm bilgiler başarılı bir şekilde dönülmüş oluyor. Her post isteği için bu adımlar tekrar tekrar gerçekleşiyor

D. Görüntü İşleme

Daha önce de bahsettiğimiz gibi ana api sunucumuz görüntü üzerinde obje tespitini yapmak için yine bir Google Cloud hizmeti olan Cloud Vision Api'yi kullanıyor. Aslında projeyi tasarlarken sunucu üzerinde tenserflow.js ve önceden eğitilmiş bir modeli kullanarak obje tespitini yapacaktık fakat hazır apileri kullanabilme fırsatını tarafımıza verildiği için ve zaten Google Cloud ekosisteminde bulunduğu için Vision Api'yi tercih ettik. Ana api sunucumuzun Vision api'ye attığı istek ve cevap mekanizmasından zaten bahsetmiştik. Bu kısımda obje tespit işleminin nasıl gerçekleştiğini araştırmalarımızdan yola çıkarak anlatacağız.[3][4]

Vision Api üzerinde görüntüdeki nesnelerin tespit süreci bazı aşamalardan geçer. İlk olarak nesne tespiti yapılacak görüntü alınır ve matris formatına dönüştürülür. Matristeki elemanlar görüntünün piksellerindeki özellikleri tutar. Daha sonra veri ön işleme ile görüntüyü doğru halde işlememizi engelleyebilecek eksik veriler, tutarsız veriler gibi sorunlar düzeltilir. Daha önceden belirlenmiş olan öznitelikleri(renk, şekil gibi) sayesinde görüntüdeki cisimler saptanır. Bu öznitelik çıkarımıdır. Nesnenin ne olduğu belirlenirken sınflandırma, kümeleme kullanılır. Sınıflandırma ile nesne daha önceden belirlenmiş sınıflara dahil edilerek tespit edilir. Kümeleme de görüntülerden elde edilen nesneler özelliklerine göre kümeler oluştururlar. Benzer özelliklere sahip nesneler aynı küme içerisinde yer alır. Böylece nesne tespit edilebilir. Google Cloud Vision API, önceden eğitilmiş modelleri ve önceden tanımlanmış öznitelikleri kullanarak görüntüleri sınıflandırır ve bize json olarak elde ettiği sonuçları döner. Bu json dosyası içerisinde tespit edilen her obje için isim bilgisi, tespit edilen objenin orijinal resim üzerindeki köşe konumlarının normalize edilmiş değerler halindeki listesi, olasılık bilgisi gibi değerleri içerir. Daha sonrasında resim üzerinde işaretlemeleri gerçekleştirirken normalize edilmiş değerler üzerinde birtakım matematiksel işlemler gerçekleştirmemiz gerekiyor.

Elde ettiğimiz bu json dosyasını Node.js api sunucumuzda kullanarak ilgili obje verileri üzerinden hem

istemciye döneceğimiz son json yapımızı hazırladık hem de işlenmiş resim üzerindeki işaretlemeleri gerçekleştirdik.

III. KAZANIMLAR

Bu projeyi gerçekleştirerek, ilk defa çapraz platform bir mobil uygulama geliştirdik. Değişik bulut teknolojileri hakkında bilgiler edindik ve farklı hizmetleri kullandık. Görüntü işleme apilerinin genel işleyişi hakkında araştırmalar yapıp temel çalışma mantığını anladık. Node ile kendi api sunucumuzu yazdık. İstemci uygulamamız ile apimizin haberleşmesini sağladık. Resimler üzerinde çeşitli işaretlemeler gerçekleştirdik. Bulut veritabanlarıyla çalıştık..

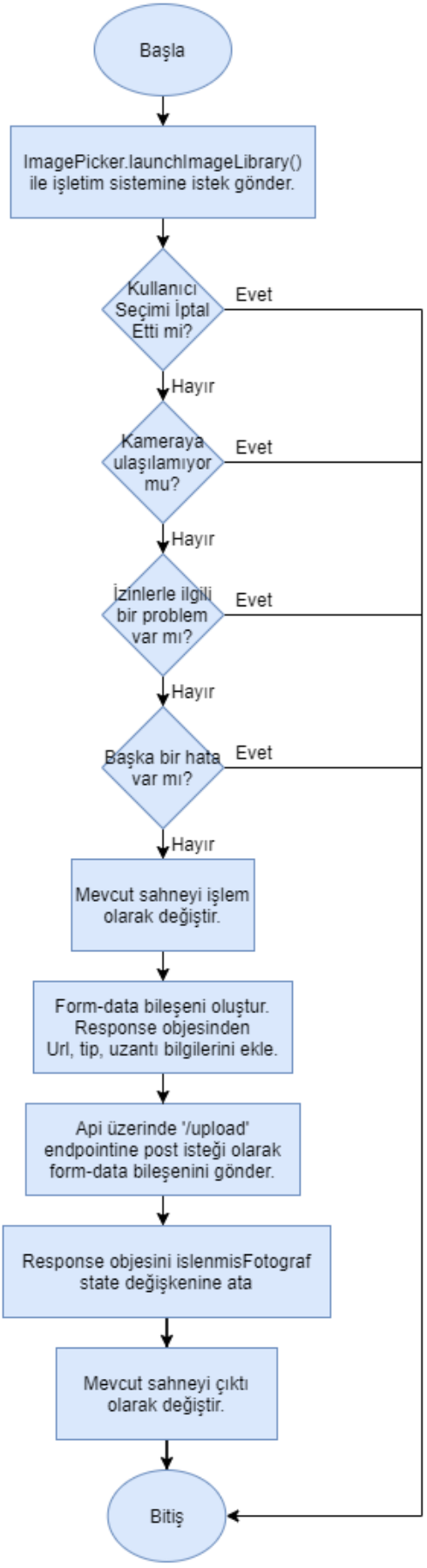
IV. EKSIKLIKLER

Projede istenen tüm isterleri yerine getirdik.İsterler açısından bir eksik olmasa da proje bir okul projesi olduğu için api üzerinde herhangi bir güvenlik önlemi almadık. Herhangi bir Kullanıcı doğrulaması gerçekleştirilmiyor bu da büyük bir güvenlik zaafı oluşturuyor. Bunun dışında arayüzde işlevselliği gösterecek kadar minimum komponenti kullandık. Arayüz tasarımı daha da geliştirilebilir.

V. AKIŞ DIYAGRAMLARI VE YALANCI KOD

```
boyutlar = boyutHesapla(resim)
cizimAlani = cizimAlaniOlustur(boyutlar.genislik,
boyutlar.yükseklik)
cizimAlani.resimCiz(resim);

referansGenislik = 1080
referansCizgiGenisligi = 30
cizgiOrani = referansCizgiGenisligi / referansGenislik

Tüm Elemanlar İçin Döngü : tespitEdilenNesneler => nesne:

        renk = renk(rastgele(255, 255, 255))

        noktalar = nesne.normalizeEdilmisNoktalar

        x1 = noktalar[0].x * boyutlar.genislik
        y1 = noktalar[0].y * boyutlar.yükseklik
        x2 = (noktalar[1].x - noktalar[0].x) *
        boyutlar.genislik
        y2 = (noktalar[2].y - noktalar[1].y) *
        boyutlar.yukseklik

        cizgiGenisligi = x2 * cizgiOrani

        Eğer cizgiGenisligi <= 5:
                cizgiGenisligi = 5
        Değilse Eğer cizgiGenisligi >= 20:
                cizgiGenisligi = 20

        cizimAlani.cizgiGenisligi = cizgiGenisligi
        cizimAlani.kenarlikRengi = renk
        cizimAlani.kenarlikliKareCiz(x1, y1, x2, y2)
```

Cloud Visiondan Dönen Normalize Edilmiş Noktalara Göre Resim Üzerinde Karelerin Çizdirilmesinin Kaba Kodu



Galeriden Fotoğraf Seçme Ve Apiye Gönderme İşleminin Akış Şeması

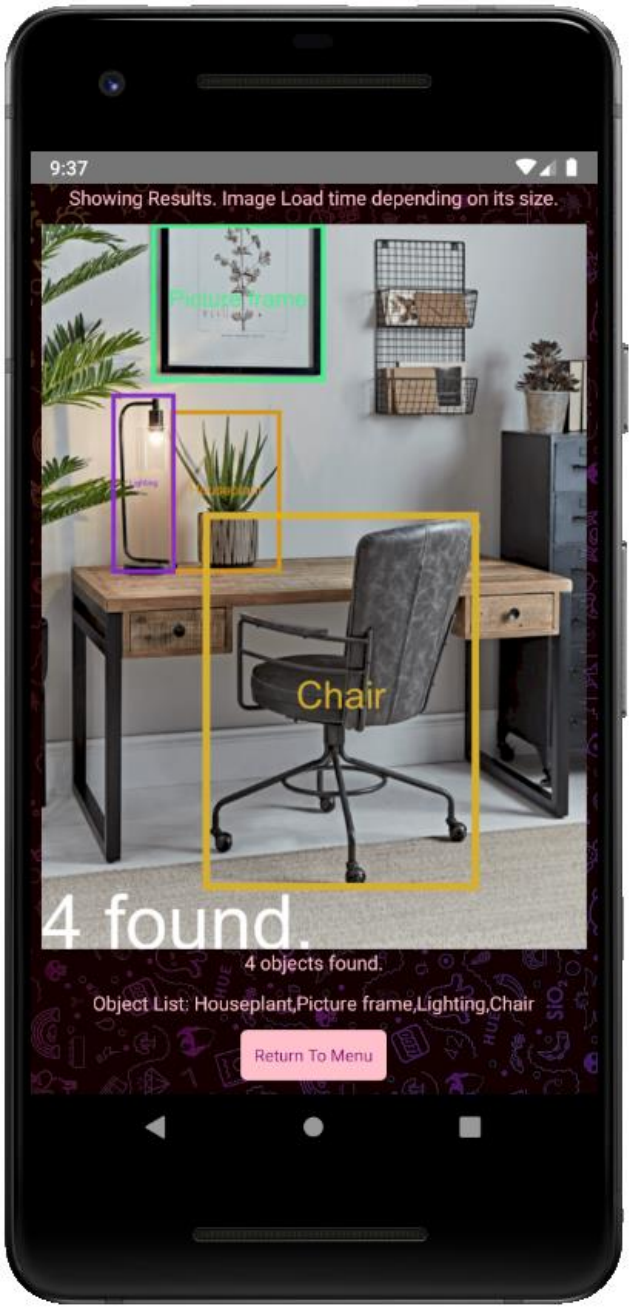
VI. GELIŞTIRME ORTAMI VE KULLANILAN DIL

Projeyi geliştirirken mobil uygulama için React-Native Frameworkünü, Bulut üzerinde çalışan Api sunucumuz için ise Node.js kullandık. Projeyi Windows İşletim Sistemi üzerinde gerçekleştirirken, geliştirme ortamı olarak Visual Studio Code Editorunu kullandık.

VII. PROGRAMIN GENEL YAPISI VE TASARIMI

Program temel olarak menu sahnesi, islem sahnesi ve cikti sahnesi olmak üzere üç adet sahne içeriyor. Her sahne farklı bir tasarıma sahip ve basit bir sahne yöneticisi aracılığıyla sahneler arası geçiş gerçekleştiriliyor.

.



Program Çıktı Ekranının Genel Tasarımı

Program Menüsünün Genel Tasarımı

VIII. DENEYSEL SONUÇLAR

Mobil uygulama geliştirirken yaşanan test aşamalarının, masaüstü uygulaması geliştirmekten farklı yanları bulunuyordu. Uygulamayı test ederken çeşitli emulatörlerden ve gerçek cihazlardan faydalandık. Bazı testlerimizde mobil işletim sisteminin api versiyonundan kaynaklanan uyumsuzluk problemlerini fark ettik ve api sürümünü arttırarak bu uyumsuzlukların önüne geçtik. Henüz mobil uygulamamız tamamen bitmemiş haldeyken paralel olarak api sunucumuzu geliştiriyorduk ve request işlemlerini test etmemiz gerekiyordu. Bu noktada postman gibi yardımcı programlar kullanarak manuel olarak api ile haberleştirerek mobil uygulamamızı taklit ettirdik. Daha sonrasında apiyi çalışır hale getirdikten sonra mobil uygulama ile aradaki haberleşmeyi sağladık

IX. SONUÇ

Bu projeyi gerçekleştirerek bulut üzerinde nesne tespiti yapabilen bir mobil uygulama ortaya çıkardık. Çarpraz platform frameworkler, bulut teknolojileri, görüntü işleme , api geliştirme, sunucu-istemci mimarisi, veritabanları gibi çeşitli konularda çok sayıda yeni bilgi edindik.

X. KAYNAKLAR

(1) https://github.com/react-native-image-picker/react-native-image-picker (Erişim Tarihi: 25.12.2020)
(2) https://cloud.google.com/vision/docs/libraries (Erişim Tarihi: 26.12.2020)
(3) https://mesutpiskin.com/blog/nesne-tespiti-ve-nesne-tanima.html (Erişim Tarihi: 3.1.2021)
(4) https://dergipark.org.tr/en/download/article-file/796408 (Erişim Tarihi: 3.1.2021)