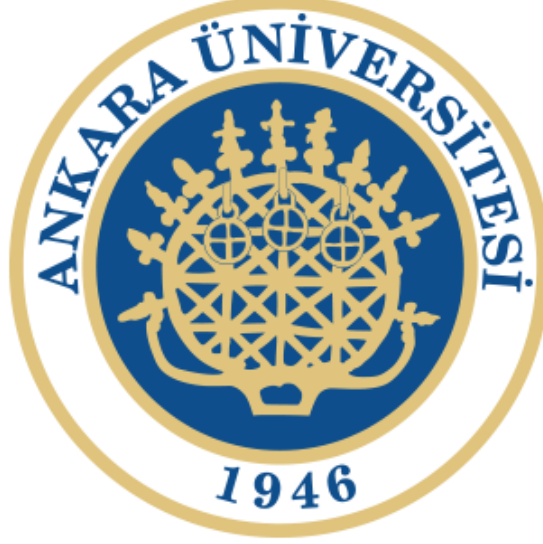


BLM 4538 PROJE FİNAL RAPORU



GERÇEK ZAMANLI PLAKA TANIMA VE YÖNETİM SİSTEMİ UYGULAMASI

GitHub Linki:

https://github.com/MetehanTRN/IOS_PROJECT

Video Linki:

<https://drive.google.com/drive/folders/1NQsvOWwi9v26vI6hYCwJ84S1GxFL0fmQ?usp=sharing>

Metehan TURAN

20291263

ÖZET

Bu rapor, araç plaka tanıma sistemine yönelik bir yazılım projesinin tüm geliştirme sürecini kapsamaktadır. Projenin temel amacı, farklı kaynaklardan alınan plaka görsellerini işleyerek, tanımlı plakalarla karşılaştırma yapan ve bu doğrultuda çeşitli işlemler gerçekleştiren bütünlük bir sistem oluşturmaktır. Sistem; plaka tanıma motoru, veritabanı, kullanıcı arayüzü ve otomasyon modüllerini içerecek şekilde tasarlanmıştır.

Geliştirme sürecinde farklı plaka tanıma teknolojileri test edilmiştir. İlk aşamalarda açık kaynaklı çözümler olan Tesseract OCR ve YOLOv4 kullanılarak plaka tespiti ve karakter tanıma işlemleri gerçekleştirilmiştir. Ancak bu yöntemlerde karşılaşılan doğruluk oranı düşüklüğü, özellikle ışık-gölge, açı ve çözünürlük gibi etkenlere karşı sistemin hassasiyeti, bu çözümlerin proje hedeflerine uygun olmadığını göstermiştir. Bu nedenle daha güvenilir sonuçlar sunan PlateRecognizer API tercih edilmiştir.

Plaka görsellerinin sisteme aktarılması için çeşitli yöntemler denenmiştir. Gerçek zamanlı canlı kamera akışı, yerel ağ üzerinden video yayını ve USB bağlantısı gibi yöntemler test edilmiştir. Bu yöntemlerde bağlantı sorunları, çözünürlük düşüklüğü veya sürekli izleme senaryolarında kararsızlık gözlemlenmiştir. En verimli ve tutarlı sonuç, Google Drive ile senkronize edilen klasör tabanlı sistem sayesinde elde edilmiştir. Bu yöntemle telefon kamerasıyla çekilen görseller otomatik olarak sisteme aktarılabilmiş ve arka planda işlenmiştir.

Proje kapsamında Python ile plaka işleme modülü, Firebase veritabanı altyapısı ve React Native ile geliştirilen mobil kullanıcı arayüzü entegre edilmiştir. Kullanıcılar sisteme yeni plaka ekleyebilmekte, kara liste tanımlayabilmekte ve geçmiş loglara erişebilmektedir. Tanımlı olmayan plakalar için kullanıcı onayı alınmakta, sistem karar mekanizmasını insan girdisiyle desteklemektedir.

Sonuç olarak, proje yalnızca işlevsel değil; aynı zamanda kullanıcı etkileşimine açık, senaryolarla test edilmiş ve genişletilebilir bir yapıya sahiptir. Süreç boyunca yapılan deneysel denemeler, karşılaşılan zorluklar ve tercih edilen çözümler raporun ilgili bölümlerinde ayrıntılı olarak sunulmuştur.

1. GİRİŞ

1.1. Problem Tanımı, Amaç ve Kapsam

Yoğun şehir yaşamı ve artan araç sayısı, otopark yönetimini daha karmaşık ve zaman alıcı hâle getirmiştir. Araçların giriş-çıkışlarının manuel yöntemlerle kontrol edilmesi hem güvenlik açıklarına yol açmakta hem de süreçlerin verimini düşürmektedir. Bu bağlamda, plaka tanıma sistemleri; güvenlik, otomasyon ve izlenebilirlik açısından modern otopark yönetiminin vazgeçilmez bir parçası hâline gelmiştir.

Bu projenin temel amacı, otoparklarda araç giriş-çıkışlarının plaka tanıma yoluyla otomatikleştirildiği, kullanıcı tarafından kolayca yönetilebilen ve mobil destekli bir sistem geliştirmektir. Sistem, plakaları görsel kaynaklardan otomatik olarak tanıyacak, tanımlı plakalar için giriş izni verecek, tanımsız plakalar için kullanıcıdan onay alarak işlem yapılmasını sağlayacaktır. Giriş-çıkış kayıtları sistemde saklanacak ve kullanıcı tarafından erişilebilecektir.

Proje kapsamında hem Python tabanlı arka plan işlemleri hem de React Native ile geliştirilen mobil uygulama arayüzü bir bütün olarak çalışacak şekilde tasarlanmıştır. Firebase altyapısı üzerinden veriler senkronize edilecek; plaka listesi, kara liste ve log kayıtları merkezi olarak yönetilecektir.

Bu sistem yalnızca test görselleriyle değil, gerçek senaryoya yakın yöntemlerle de çalışacak şekilde yapılandırılmıştır. Ancak sistem; doğrudan IP kamera bağlantısı, canlı video akışında gerçek zamanlı kare seçme gibi gelişmiş görüntü yönetimi özelliklerini kapsamaz. Görsel işleme, belirli klasörlere aktarılan resimler üzerinden yürütülmektedir. Bu sınırlama, sistemin kararlılığı ve test edilebilirliği açısından tercih edilmiştir.

1.2. Kullanılan Teknolojiler ve Geliştirme Ortamı

Proje, farklı işlevleri yerine getiren birden fazla bileşenden oluştuğu için, geliştirme sürecinde birden çok teknoloji ve platformdan yararlanılmıştır. Her bileşen kendi görevine uygun teknoloji ile tasarlanmış; entegrasyon aşamasında uyumluluk gözetilmiştir.

1.2.1. Kullanılan Diller ve Frameworkler

Python: Plaka tanıma, dosya izleme, API ile etkileşim ve otomasyon işlemleri için kullanılmıştır.

React Native (JavaScript): Mobil kullanıcı arayüzü geliştirme sürecinde tercih edilmiştir. Hem iOS hem Android uyumlu yapıdadır.

TypeScript: React Native bileşenlerinde daha güvenli ve okunabilir bir yapı sağlamak amacıyla kullanılmıştır.

1.2.2. Platformlar ve Servisler

Firebase (Firestore & Storage): Plaka verileri, kara liste kayıtları ve loglar için merkezi bulut veritabanı çözümüdür. Gerçek zamanlı senkronizasyon sağlamaktadır.

PlateRecognizer API: Plaka tanıma işlemi bu servis üzerinden yürütülmektedir. Görsel yükleme yoluyla yüksek doğrulukta çıktı alınmaktadır.

Google Drive: Görsellerin mobil cihazdan sisteme senkronize edilmesi amacıyla tercih edilmiştir. Otomatik klasör eşitleme sağlanmıştır.

1.2.3. Geliştirme Ortamı ve Araçlar

VS Code: Kod geliştirme sürecinde tercih edilen ana editördür.

Fiziksel Cihaz: Mobil uygulamanın test ve önizlemesi için kullanılmıştır.

Expo Go: React Native uygulamasının hızlı test edilmesini sağlamıştır.

Postman: API testleri ve veri yapısının kontrolü amacıyla kullanılmıştır.

Node.js & npm: React Native geliştirme ortamı kurulumu ve bağımlılık yönetimi için kullanılmıştır.

Bu teknolojiler sayesinde sistem hem esnek hem de geliştirici dostu bir yapıda oluşturulmuştur. Parçalı modül yapısı sayesinde her bileşen gerektiğinde bağımsız olarak test edilebilmekte ve geliştirilebilmektedir.

1.3. Sistem Mimarisi ve Bileşenleri

Geliştirilen plaka tanıma sistemi, modüler yapıda tasarlanmış dört temel bileşenden oluşmaktadır: **mobil kullanıcı arayüzü**, **arka plan işlem birimi**, **veritabanı altyapısı** ve **plaka tanıma servisi**. Bu yapı sayesinde sistem, parçalı olarak yönetilebilir, kolay güncellenebilir ve gerektiğinde genişletilebilir hâle getirilmiştir.

1.3.1. Mobil Uygulama Arayüzü

React Native kullanılarak geliştirilen mobil uygulama, kullanıcıya sistemin kontrol panelini sunmaktadır. Uygulama üzerinden kullanıcı;

- Yeni plaka ekleyebilir,
- Kara listeyi düzenleyebilir,
- Giriş-çıkış loglarını görüntüleyebilir,
- Tema değişikliği yapabilir,
- Uyarı mesajlarına yanıt vererek sistemle etkileşime geçebilir.

1.3.2. Arka Plan İşlem Birimi (Python)

Bu bölüm, görsellerin izlenmesi ve tanınması süreçlerinden sorumludur. Python ile geliştirilen script, sistem klasörlerini izleyerek yeni eklenen görselleri PlateRecognizer API'ye gönderir. API çıktısına göre plakanın sisteme tanımlı olup olmadığı kontrol edilir ve Firebase veritabanı ile işlem yapılır. Tanımsız plaka varsa, mobil uygulamaya bildirim gönderilir.

1.3.3. PlateRecognizer API

Plaka tanıma işlemleri bu servis aracılığıyla gerçekleştirilir. API, kendisine gönderilen görsellerdeki plaka kutularını belirler ve plaka metnini JSON formatında döner. Bu sayede sistemin içinde OCR veya nesne tespiti modeline gerek kalmadan, kararlı ve güvenilir bir sonuç alınır.

1.3.4. Firebase Veritabanı

Veritabanı tarafında **Firestore** kullanılmıştır. Plaka listesi, kara liste ve giriş-çıkış logları burada tutulur. Ayrıca kullanıcılar tarafından yapılan işlemler (plaka ekleme, kara listeden çıkarma gibi) gerçek zamanlı olarak burada güncellenmektedir. Firebase ile sistemin tüm verileri merkezi ve senkronize bir yapıda yönetilmektedir.

1.3.5. Veri Akışı ve Genel Etkileşim

Sistemin temel akışı aşağıdaki gibidir:

1. Kullanıcı bir aracı park alanına yönlendirir.
2. Görsel, belirlenen klasöre düşer (manuel ya da senkronize yöntemle).
3. Python script klasörü izler ve yeni görseli PlateRecognizer API'ye gönderir.
4. API'den gelen sonuç Firebase veritabanı ile karşılaştırılır.
5. Giriş yetkisi varsa log'a kayıt yapılır; yoksa kullanıcıya uyarı gönderilir.
6. Mobil uygulama üzerinden işlem onaylanır veya kara listeye alınır.



Şekil 1.3. Projenin Genel Bileşen Mimarisi ve Veri Akışı

Çizelge 1.3. Kullanılan Ana Bileşenler ve Görevleri

Bileşen	Açıklama
Mobil Uygulama	Kullanıcı arayüzü, işlem onayı
Python Script	Görsel işleme ve API bağlantısı
API	Plaka tanıma hizmeti
Firebase	Veritabanı ve kayıt yönetimi

1.4. Önceki Denemeler ve Nihai Yöntem

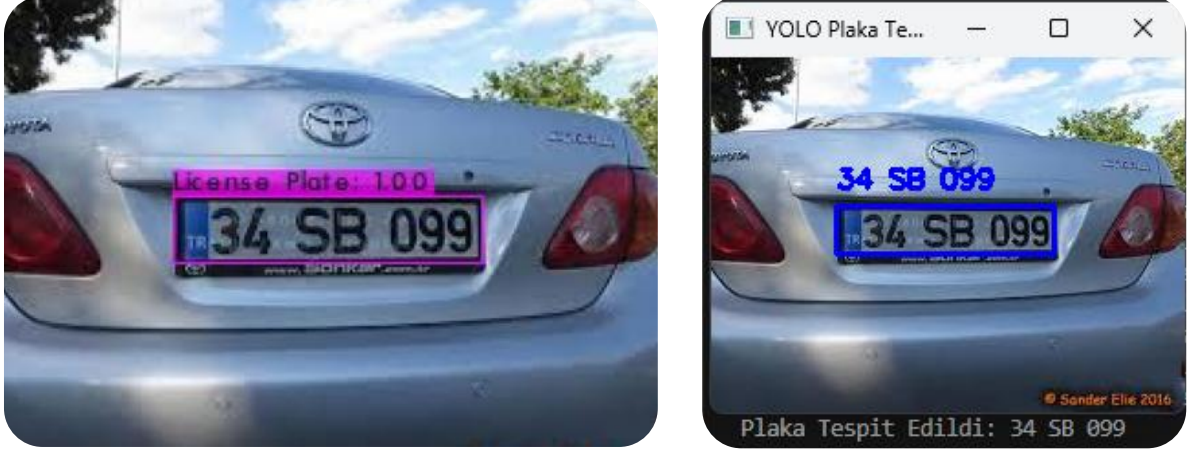
Projenin başlangıç aşamasında, plaka tanıma işlemi için açık kaynaklı çözümler üzerinde çeşitli denemeler yapılmıştır. Amaç, dış servislere bağlı kalmadan, düşük kaynak tüketimiyle çalışan bir tanıma sistemi geliştirmektir. Bu bağlamda **Tesseract OCR**, **YOLOv4** ve farklı kamera bağlantı sistemleri test edilmiştir. Ancak bu yöntemler, pratikte karşılaşılan bazı sorunlar nedeniyle tercih edilmemiştir.

1.4.1. Tesseract OCR

Tesseract, metin tanıma konusunda güçlü bir açık kaynak motorudur. İlk denemelerde plakaların manuel olarak kırılıp Tesseract'a gönderilmesiyle karakter tanıma yapılmıştır. Ancak:

- Işık-gölge farkları,
- Açık bozulmaları,

- Plaka fontunun standart dışı olması gibi nedenlerle okuma başarı oranı çok düşmüştür. Tanıma ancak yüksek çözünürlüklü ve net görsellerde mümkün olmuştur.



Şekil 1.4.1 YOLOv4 ve Tesseract OCR ile plaka bölgesi tespiti ve karakter okuma işlemi

1.4.2. YOLOv4

YOLOv4 nesne tespiti için kullanılan güçlü bir modeldir. Plaka konumunun otomatik olarak tespit edilmesi amacıyla kullanılmıştır. Ancak:

- Eğitilmemiş haliyle Türkiye plakalarını tespit edememiştir,
- Model eğitimi için geniş ve dengeli bir veri seti gereklidir,
- Tespit edilen plaka kutularının içinde çevresel gürültüler olması OCR kalitesini düşürmüştür.

Dolayısıyla YOLO + OCR entegrasyonu yüksek doğrulukta çalışamamıştır.



Şekil 1.4.2 Bozuk görüntülerde eski sistemin performansı

1.4.3. PlateRecognizer API ile Nihai Çözüm

Yukarıdaki yöntemlerin her biri belirli alanlarda fayda sağlasa da sistemin **kararlı, hızlı ve yüksek doğrulukta** çalışması için yeterli olmamıştır. Bu noktada karar verilerek, plaka tanıma görevini özel olarak üstlenen, güvenilir bir servis olan **PlateRecognizer API** sistemin merkezine alınmıştır.

PlateRecognizer, gönderilen görselden plaka kutusunu ve plaka metnini tespit ederek JSON çıktısı sunmaktadır. Türkiye plakaları dahil olmak üzere çok sayıda formatı desteklemekte; ışık, açı gibi değişkenlerden büyük oranda bağımsız çalışmaktadır.

- %90+ doğrulukla çalışan yapı
- JSON formatında okunabilir çıktı
- API üzerinden kolay entegrasyon
- Geliştirici dökümantasyonu ve ücretsiz kullanım limiti gibi nedenlerle bu çözüm tercih edilmiştir.



Şekil 1.4.3 PlateRecognizer ile başarıyla tanınan plaka çıktısı

1.5. Görüntü Elde Etme Süreci ve Otomasyon

Plaka tanıma sistemlerinin işleyişi için en temel ihtiyaç, işlenecek görsellerin sisteme düzenli ve doğru biçimde aktarılmasıdır. Bu projede, kamera görüntülerinin sisteme aktarılması için farklı yöntemler denenmiş ve çeşitli senaryolar test edilmiştir. Her yöntemin getirdiği avantajlar ve karşılaşılan zorluklar değerlendirilmiş; en uygun yaklaşım sistemin yapısına entegre edilmiştir.

1.5.1. Canlı Kamera Bağlantısı Denemeleri

Projenin başında, gerçek zamanlı kamera görüntüsünden doğrudan kare alarak plaka tanıma yapılması hedeflenmiştir. Bunun için şu yöntemler test edilmiştir:

- **IP Webcam** uygulaması kullanılarak telefon kamerası kablosuz bağlantı üzerinden bilgisayara aktarılmıştır. Ancak Android 10 ve üzeri sürümlerde erişim izinleri nedeniyle kamera görüntüsü siyah kalmış, Python tarafında cv2.VideoCapture() ile kare alınamamıştır.
- **DroidCam** uygulamasıyla telefon bilgisayara USB üzerinden bağlanmış ve webcam gibi kullanılmıştır. Bu yöntem daha kararlı olsa da sürekli kablolu bağlantı ihtiyacı, cihaz ısınması ve düşük çözünürlük gibi pratik zorluklar nedeniyle tercih edilmemiştir.

Bu denemeler sonucunda, gerçek zamanlı stream üzerinden işlem yapmanın sistem kararlılığını olumsuz etkilediği görülmüştür.



Şekil 1.5.1. IP Webcam uygulamasında alınan siyah ekran örneği

1.5.2. Klasör Bazlı Görsel Aktarımı

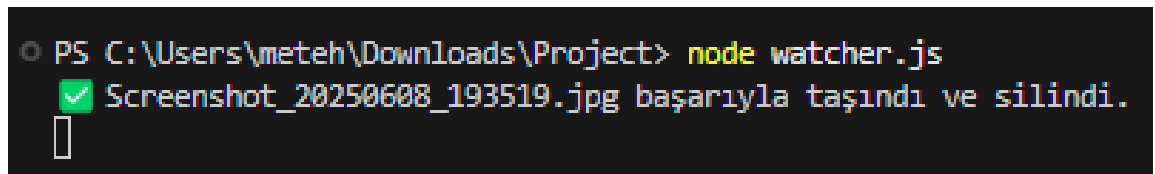
Gerçek zamanlı stream yerine daha kontrol edilebilir ve sade bir yöntem olarak, görsellerin belirli bir klasöre aktarılması ve sistemin bu klasörü izlemesi temelli bir yapı geliştirilmiştir. Bu yapı, hem mobil cihazlardan çekilen görsellerin sisteme taşınmasını hem de sistemin bu görselleri belirli aralıklarla işleyebilmesini sağlamaktadır.

1.5.3. Google Drive ile Otomatik Senkronizasyon

En verimli sonuç, **Google Drive tabanlı otomatik senkronizasyon yöntemiyle** elde edilmiştir. Bu yapıda:

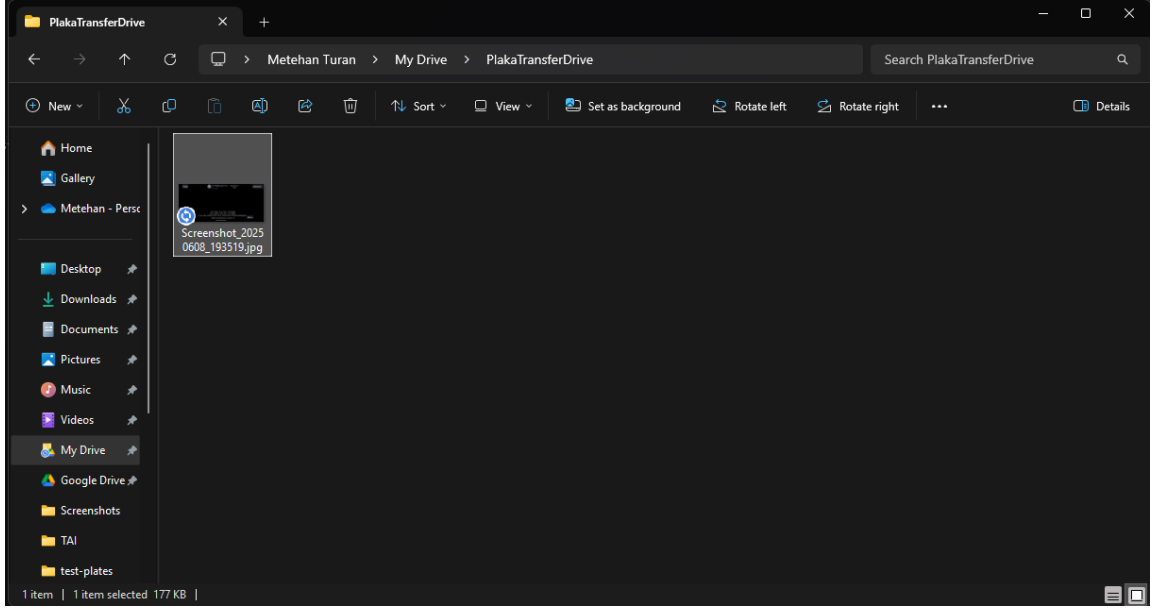
- Telefona kurulan bir senkronizasyon uygulaması (örneğin **AutoSync**) aracılığıyla, kamerayla çekilen her görsel otomatik olarak belirli bir klasöre yüklenmektedir.
- Bilgisayarda bu Drive klasörü eşlenerek, sistemin izlediği test-plates/ klasörüyle senkronize edilmiştir.
- Python tarafındaki **watcher script**, bu klasörü izleyerek yeni gelen her görseli otomatik olarak PlateRecognizer API'ye gönderip sonuçları işlemiştir.

Bu yöntem, kararlılığı, taşınabilirliği ve doğruluğu ile projenin otomasyon ihtiyacına en uygun yapı olmuştur.

A screenshot of a terminal window with a dark background. The prompt is 'PS C:\Users\meteh\Downloads\Project>'. The command 'node watcher.js' has been executed. The output is 'Screenshot_20250608_193519.jpg başarıyla taşındı ve silindi.' (Screenshot_20250608_193519.jpg successfully moved and deleted). A green checkmark icon is visible to the left of the output text. A cursor is visible on the line below the output.

```
PS C:\Users\meteh\Downloads\Project> node watcher.js
✓ Screenshot_20250608_193519.jpg başarıyla taşındı ve silindi.
```

Şekil 1.5.2. Watcher script'in otomatik olarak işlediği bir örnek görsel



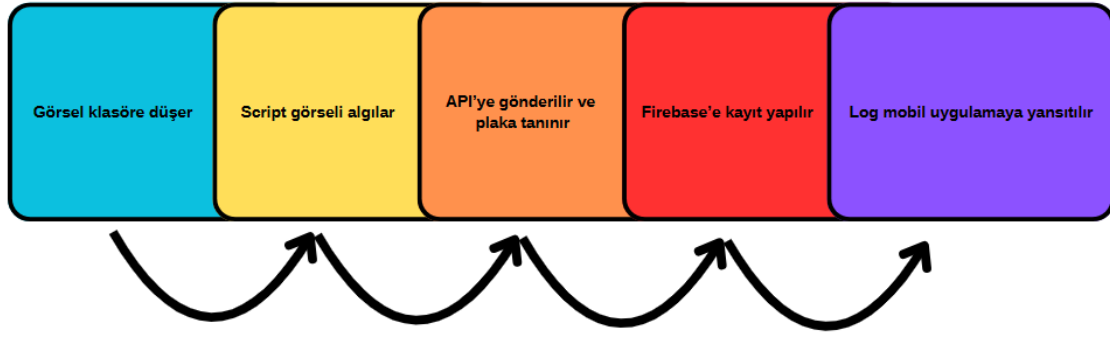
Şekil 1.5.3. Drive ile senkronize edilen klasördeki yeni görsel

1.5.4. Otomasyon İş Akışı

Otomasyon yapısı genel olarak şu adımlarla çalışmaktadır:

1. Yeni görsel klasöre düşer (Drive ya da manuel yükleme ile),
2. Watcher script belirli aralıklarla klasörü tarar,
3. Daha önce işlenmemiş görsel bulunursa PlateRecognizer API'ye gönderilir,
4. Tanıma sonucuna göre giriş izni verilir veya kullanıcı onayı beklenir,
5. İşlenen görsel işaretlenir, tekrar işlenmez.

Bu yapı sayesinde sistem, minimum kullanıcı müdahalesiyle sürekli çalışabilir hâle getirilmiş ve gerçek bir otopark otomasyon sistemi davranışı elde edilmiştir.



Şekil 1.5.4. Görselin klasöre düşmesi → işlenmesi → loglanması akışı

1.6. Genel Kullanıcı Etkileşimi

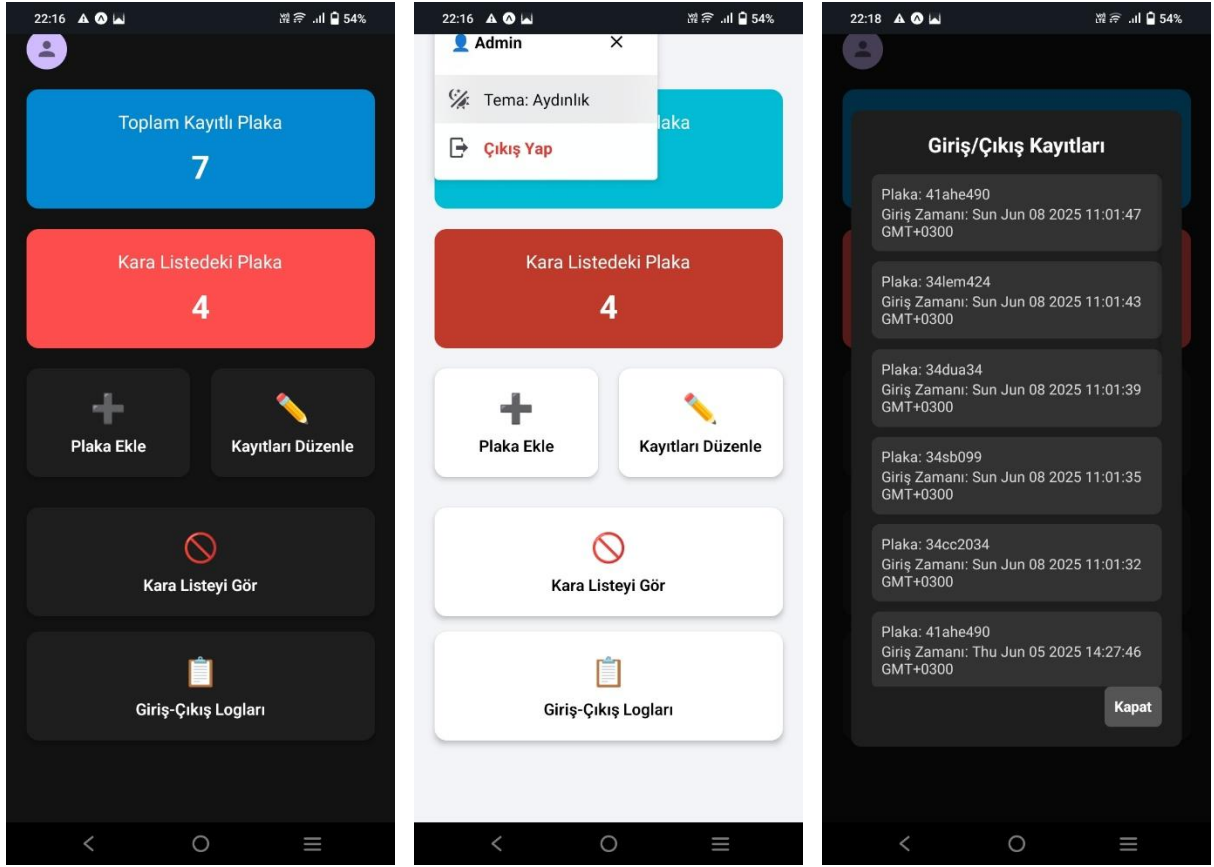
Sistemin son kullanıcıya dönük yüzü, React Native ile geliştirilen mobil uygulama arayüzüdür. Bu arayüz, kullanıcıya sistem üzerinde doğrudan kontrol sunmakta; plaka yönetimi, kayıt inceleme ve sistem etkileşimi gibi işlemleri hızlı ve pratik şekilde gerçekleştirmesine olanak tanımaktadır.

1.6.1. Ana İşlemler

Mobil uygulama üzerinden kullanıcı aşağıdaki temel işlemleri gerçekleştirebilmektedir:

- **Yeni Plaka Ekleme:** Sistemde bulunmayan bir plaka tanındığında, kullanıcıya "kaydetmek ister misiniz?" uyarısı çıkar. Kullanıcı onay verirse plaka veritabanına eklenir.
- **Kara Liste Yönetimi:** Tanımlı olmayan plaka, reddedilmek yerine kara listeye alınabilir. Kara listedeki plakaların uygulama içinden görüntülenmesi, düzenlenmesi ve kaldırılması mümkündür.
- **Log Kayıtlarını Görüntüleme:** Son 15 giriş-çıkış olayının log kayıtları uygulama arayüzünde gösterilir. Her logda plaka bilgisi, saat ve işlem sonucu yer alır.
- **Tema Değiştirme:** Uygulama, kullanıcı deneyimini iyileştirmek amacıyla karanlık ve aydınlık tema arasında geçiş yapılmasına izin verir.

- **Profil ve Oturum Yönetimi:** Kullanıcı, profil simgesi üzerinden sistemden çıkış yapabilir veya hesap bilgilerine ulaşabilir.



Şekil 1.6. Ana sayfa, Tema ve Log kayıt ekranları

1.6.2. Uyarı ve Bildirim Mekanizması

Sistem, tanımlı olmayan bir plaka algıladığında mobil uygulamaya bildirim gönderir ve kullanıcıya seçenek sunar:

- **Evet** → Plaka sistem veritabanına eklenir (ekrana otomatik olarak yazılı gelir).
- **Hayır** → İkinci bir soru ile kara listeye almak isteyip istemediği sorulur.
- Kullanıcı bu adımlarda etkileşimde bulunmazsa sistem beklemeye alınır.

Bu yapı sayesinde kullanıcı, sistem üzerinde **tam kontrol sağlayabilmekte ancak manuel iş yüküne maruz kalmamaktadır**. Sistem yalnızca gerektiğinde kullanıcıyı sürece dâhil eder.

1.6.3. Arayüz ve Kullanılabilirlik

Arayüz tasarımı sade ve işlevsellik odaklıdır. Her bir işlem için kart tasarımı kullanılmış; sezgisel ikonlar ve açıklayıcı başlıklar ile kullanıcıya net yönlendirme sunulmuştur. Renk geçişleri, tema uyumu ve responsive yapı mobil deneyimi destekleyecek şekilde optimize edilmiştir.

2. UYGULAMA GELİŞTİRME SÜRECİ

2.1. Uygulama Mimarisi ve Teknik Detaylar

Proje iki temel yapı üzerine kurulmuştur: mobil arayüz (frontend) ve arka plan işlem modülü (backend). Uygulamanın genel mimarisi, veri işleme ve kullanıcı etkileşimini birbirinden ayıracak şekilde modüler olarak tasarlanmıştır. Bu yapı sayesinde hem geliştirme hem de test süreçleri yönetilebilir hâle gelmiştir.

2.1.1. Uygulama Yapısı

Mobil Arayüz (React Native):

Uygulamanın kullanıcıya sunulan arayüzü React Native ile geliştirilmiştir. Uygulama içindeki ekranlar component bazlı olarak ayrılmış, örneğin:

- HomeScreen.tsx → ana kontrol ve bilgi ekranı,
- AddPlateModal.tsx → plaka ekleme penceresi,
- LogModal.tsx → logları gösteren bileşen,
- ThemeContext.tsx → tema kontrol mekanizması.

Bu yapı, okunabilirliği ve test edilebilirliği artırmıştır.

Arka Plan Modülü (Python):

Plaka tanıma, klasör izleme ve otomasyon işlemleri Python tarafında yürütülmüştür. Burada temel bileşenler:

- watcher.py → klasör izleme ve işlem tetikleyici,
- plateRecognizer.ts → API bağlantısını yöneten fonksiyon,
- normalizePlate() → tanınan plaka değerlerini standartlaştırmak için kullanılan yardımcı fonksiyon.

2.1.2. Uygulamanın Çalışma Senaryosu

1. Kullanıcı bir plaka görseli çeker (veya bu işlem simüle edilir).
2. Görsel, test-plates/ klasörüne düşer (Drive senkron veya manuel).
3. watcher.py klasörü izler ve yeni görseli fark eder.
4. Görsel PlateRecognizer API'ye gönderilir ve tanımlama yapılır.
5. Gelen sonuç Firebase ile karşılaştırılır.
6. Sistem giriş izni verirse log'a kaydeder, değilse kullanıcı onayı bekler.
7. Mobil uygulama kullanıcıya uyarı gösterir ve işlem yapılır.

Bu akış sayesinde sistem sadece otomatik çalışan bir yapı değil, aynı zamanda kullanıcı kontrolüne açık, uyarlanabilir bir platform sunmaktadır.

2.1.3. Teknik Detaylar

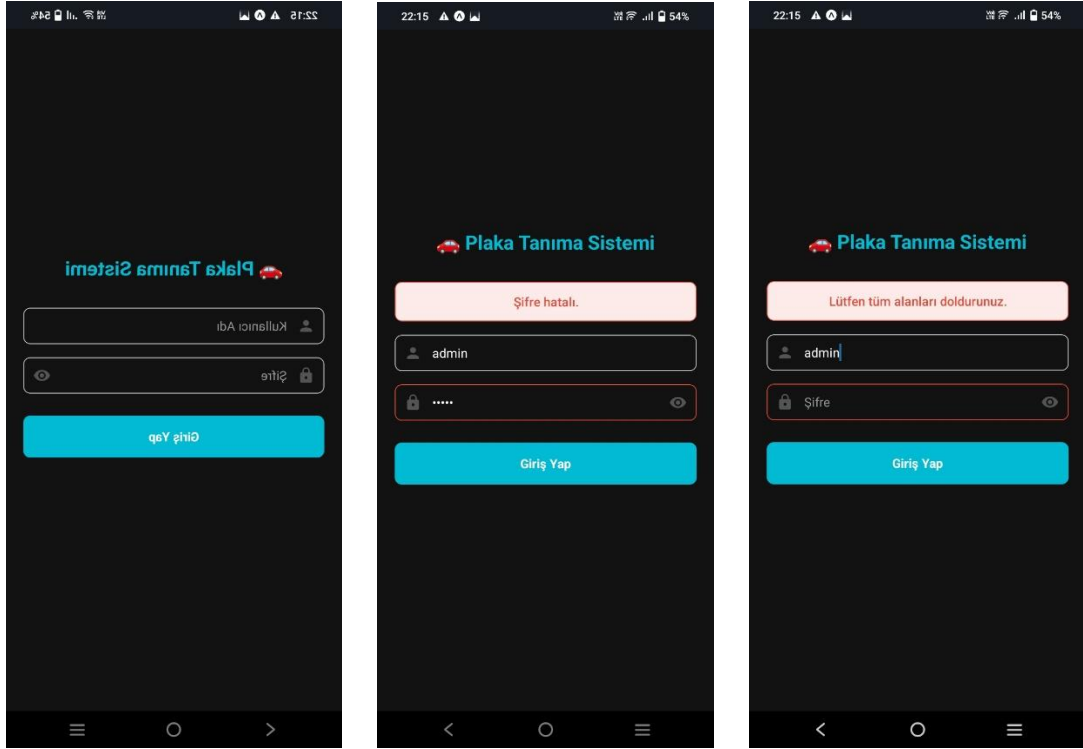
- API anahtarı ve bağlantılar .env veya utils/plateRecognizer.ts dosyasında saklanır.
- Her işlem için hata yönetimi tanımlıdır. Bağlantı kopması, boş cevap gibi durumlarda kullanıcı bilgilendirilir.
- Firebase bağlantıları async yapıda çalışır ve mobil uygulamayla anlık senkronizasyon sağlar.
- Uygulama başlatıldığında loglar yüklenir, kara liste ve plaka sayısı dinamik olarak güncellenir.

2.2. Arayüz Görselleri ve Uygulama Özellikleri

Mobil uygulama, kullanıcı dostu ve işlevsel bir arayüz sunacak şekilde tasarlanmıştır. Arayüzde görsel sadelik ve işlem erişilebilirliği ön planda tutulmuş; giriş-çıkış takibi, plaka yönetimi ve sistem bildirimleri kullanıcıyı yormayacak şekilde düzenlenmiştir.

2.2.1. Giriş (Login) Ekranı

Kullanıcının sisteme erişimini sağlayan ilk ekran giriş ekranıdır. Kullanıcı adı ve şifre alanları dışında, şifre göster/gizle butonu, hata uyarıları ve yanlış giriş durumunda kırmızı çerçeveli input alanları bulunur. Kullanıcı girişi sırasında sistemin doğrulama yaptığı sürece özel bir **yükleniyor animasyonu** gösterilir.

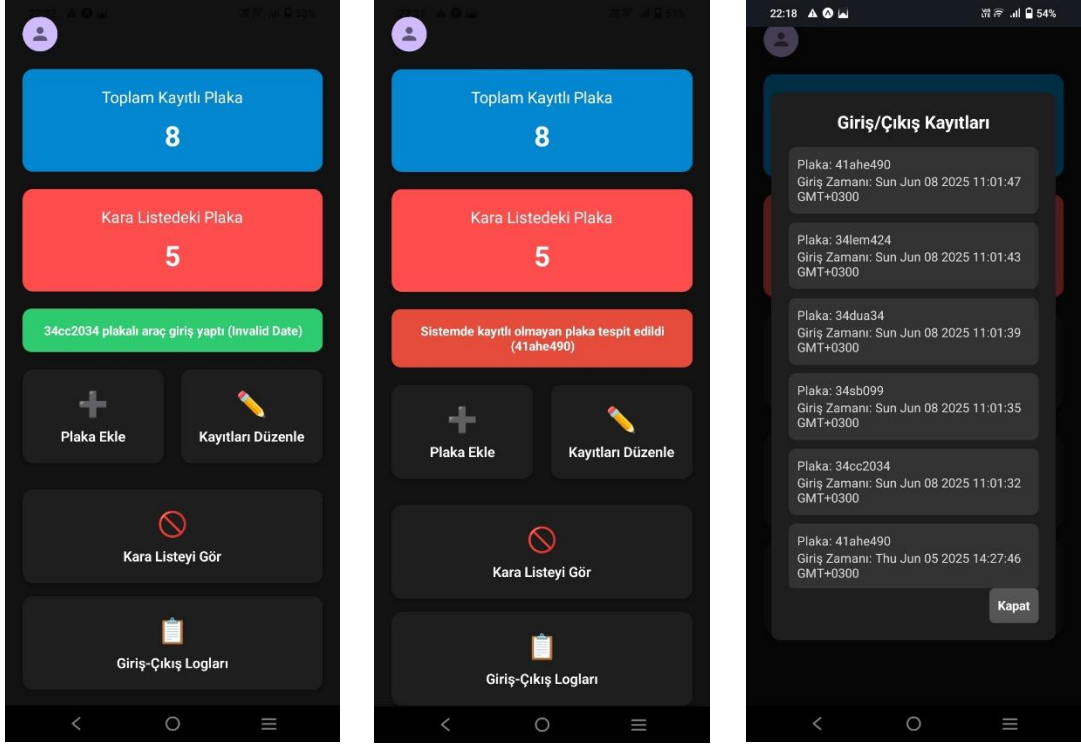


Şekil 2.2.1. Giriş ekranı – başarılı ve hatalı giriş durumu

2.2.2. Ana Sayfa ve Log Alanı

Kullanıcı giriş yaptıktan sonra yönlendirildiği ekran, sistemin kontrol panelidir. Burada:

- Son 15 plaka girişi log şeklinde listelenir,
- Giriş izni varsa yeşil, değilse kırmızı arka planla bildirim verilir,
- Girişler arasında plaka numarası, saat ve işlem durumu yer alır,
- Uygulama her açıldığında en güncel logları yükler.



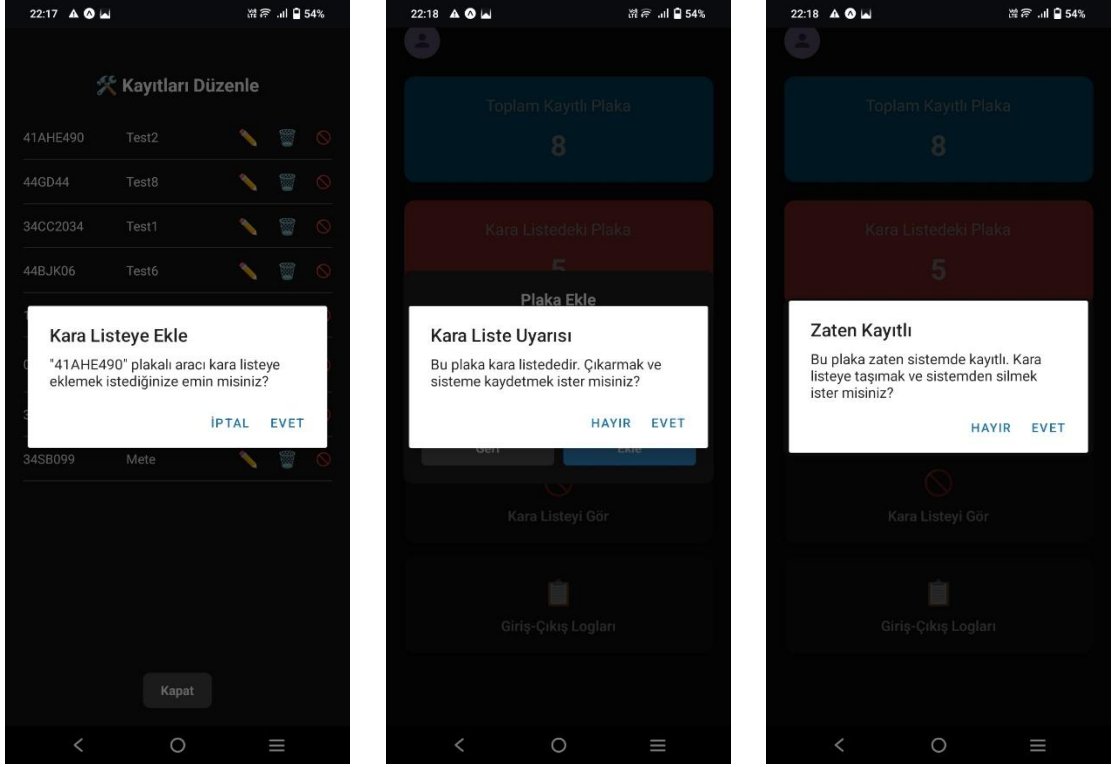
Şekil 2.2.2. Ana ekran – son girişler, başarı/başarısız renkli uyarılar

2.2.3. Modal Yapılar ve Uyarılar

Sisteme tanımlı olmayan bir plaka algılandığında kullanıcıya bir modal ekran sunulur:

- “Bu plakayı kaydetmek ister misiniz?”
- “Hayır” denirse ikinci modal çıkar: “Kara listeye eklemek ister misiniz?”

Bu yapılar sade butonlarla desteklenmiş, işlem sonrası ekran otomatik kapanarak süreci tamamlamaktadır.



Şekil 2.2.3. Plaka ekleme ve kara liste uyarı ekranları

2.2.4. Tema Seçimi ve Menü Sistemi

Uygulama, kullanıcıya koyu ve açık tema arasında geçiş imkânı sunar. Profil ikonu üzerinden açılan menüde;

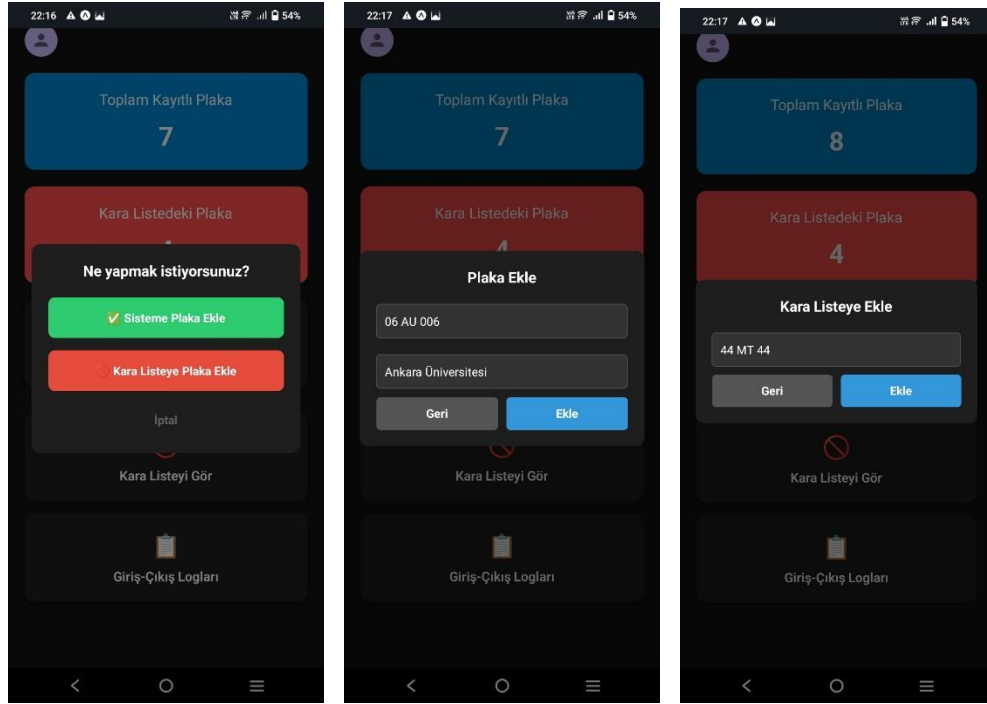
- Tema geçişi yapılabilir,
- “Çıkış Yap” butonuyla oturum sonlandırılır.

Bu sistem, kullanıcı deneyimini kişiselleştirilebilir hâle getirerek modern bir mobil uygulama davranışı kazandırır.

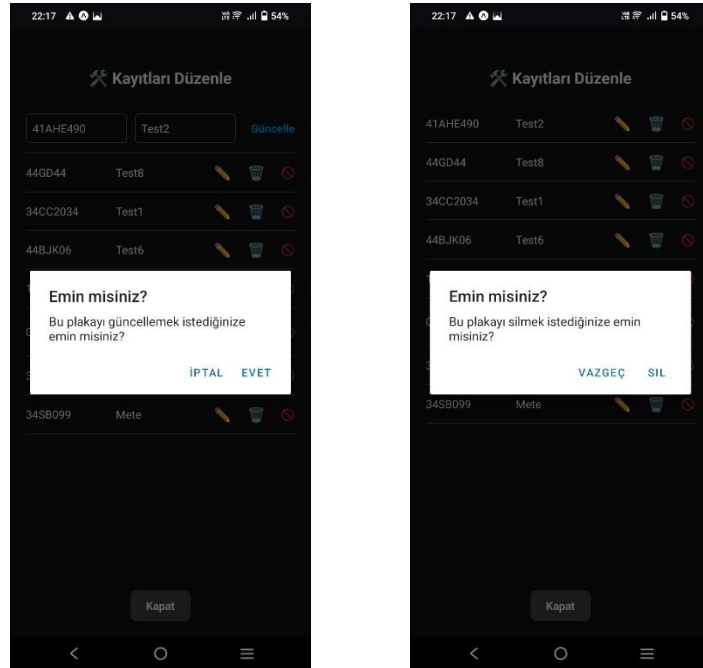
2.2.5. Ekstra Özellikler

- Uygulama responsive yapıdadır ve ekran boyutuna göre bileşenler orantılı yerleşir.
- Butonlar dokunmaya duyarlı ve görsel geribildirim (ripple effect) ile tasarlanmıştır.

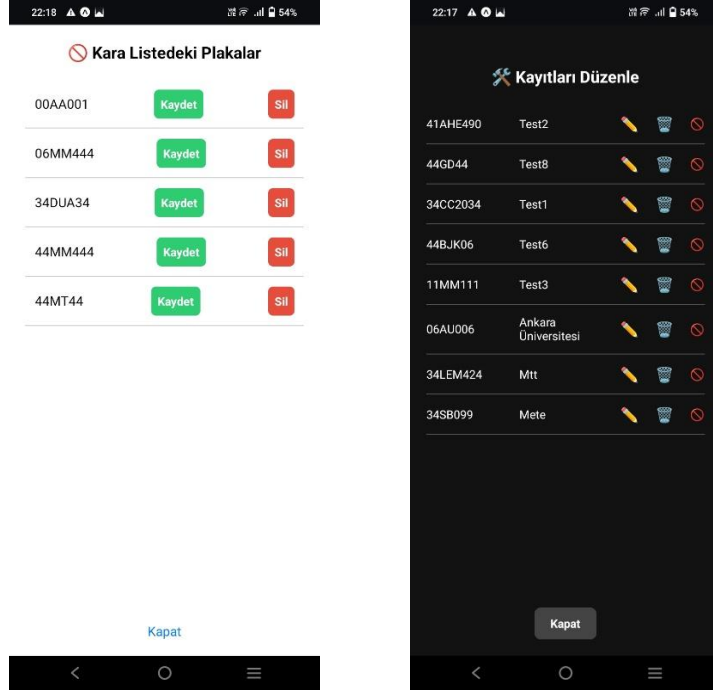
- Uygulama, varsayılan olarak karanlık tema ile başlar ve tema tercihi sistem genelinde korunur.



Şekil 2.2.4. Plaka ekleme ve kara liste modal ekranları



Şekil 2.2.5. Plaka güncelleme ve silme uyarı ekranları



Şekil 2.2.6. Kara Liste ve Plaka Düzenleme ekranları

2.3. Firestore ile Veri Etkileşimi

Bu projede, veri saklama ve senkronizasyon ihtiyaçları için **Google Firebase** altyapısı tercih edilmiştir. Firebase'in sunduğu bulut tabanlı veritabanı hizmetleri, kullanıcı ile sistem arasındaki tüm veri etkileşimini gerçek zamanlı olarak yönetebilmeyi sağlamaktadır. Uygulama, Firebase üzerinde **Firestore** kullanarak üç temel koleksiyon ile çalışmaktadır:

2.3.1. Plates (Kayıtlı Plakalar)

Sisteme manuel olarak veya kullanıcı onayıyla eklenen plakalar bu koleksiyonda tutulur. Her plaka kaydı, plakayı büyük harfle saklar ve tanımlandığı zaman bilgisini içerir.

- Kullanım Örneği: Tanınan bir plakanın sistemde kayıtlı olup olmadığı buradan kontrol edilir.
- Yapı Örneği:

plates > o09lyqgdrJUOf...		
(default)	plates	o09lyqgdrJUOfdVB537
+ Start collection	+ Add document	+ Start collection
blacklist	K7RB6oyTQ0fszDQ0U2FI	+ Add field
entries	Y4ZqjrScqs8FwPXBzNfo	createdAt: June 8, 2025 at 10:17:07 PM UTC+3
plates >	a75om6ÜsRx11T1gl6bje	owner: "Ankara Üniversitesi"
processed_images	iTU5W2mmUxYUvJA1nCXs	plate: "06AU006"
	19u1VuVvk9bqfwzy6oA8n	
	o09IyqgdrJUOfdVB537 >	
	thq4PE1PW8oohzYK0bg	
	yvQpFbxjT6fBL2CgK7n1	

Şekil 2.3.1 Firebase plaka kayıtları

2.3.2. Blacklist (Kara Liste)

Kullanıcının sisteme girmesini istemediği plakalar burada tutulur. Kullanıcı dilerse bu plakaları daha sonra sistemden kaldırabilir veya normal plakaya dönüştürebilir.

- Kullanım Örneği: Tanınan bir plaka kara listedeyse giriş reddedilir ve log buna göre oluşturulur.
- Yapı Örneği:

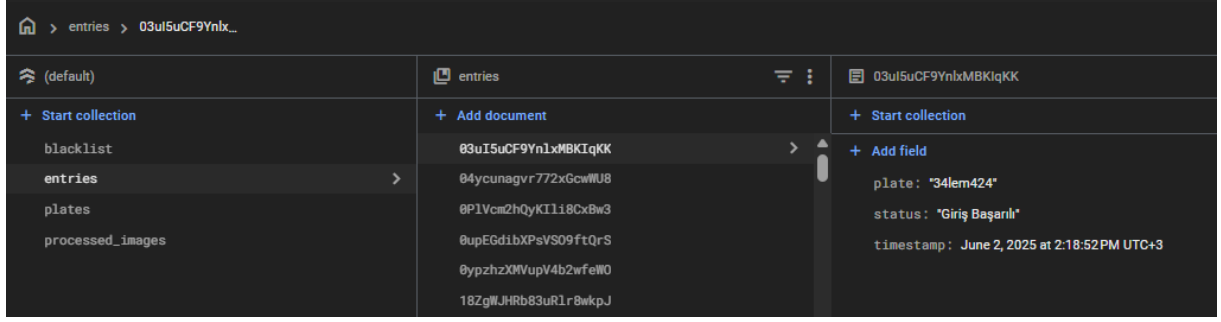
blacklist > 00AA001		
(default)	blacklist	00AA001
+ Start collection	+ Add document	+ Start collection
blacklist >	00AA001 >	+ Add field
entries	06MM444	plate: "00AA001"
plates	34DUA34	timestamp: June 8, 2025 at 11:10:28 AM UTC+3
processed_images	44MM444	
	44MT44	

Şekil 2.3.2 Firebase karaliste kayıtları

2.3.3. Entries (Giriş-Çıkış Kayıtları)

Her plaka tanıma işleminden sonra sonuç ne olursa olsun, sistem bir log oluşturur. Bu log, mobil uygulama üzerinden kullanıcıya listelenir. Her log, plakayı, giriş zamanını ve işlem durumunu içerir.

- Kullanım Örneği: Kullanıcı giriş-çıkış geçmişini görebilir.
- Yapı Örneği:



Şekil 2.3.3 Firebase giriş durum (log) kayıtları

2.3.4. Gerçek Zamanlı Etkileşim

Mobil uygulama, Firebase verilerini **onSnapshot** fonksiyonları ile dinlemektedir. Bu sayede:

- Kara liste ve plaka sayısı anlık olarak güncellenir,
- Yeni loglar otomatik olarak kullanıcı ekranında görüntülenir,
- Kullanıcının yaptığı işlemler hemen veritabanına yansır.

Bu yapı sayesinde sistem hem dinamik hem de kullanıcı odaklı bir deneyim sunmaktadır. Aynı zamanda Firebase'in kolay yönetilebilir yapısı, test ve geliştirme sürecinde büyük kolaylık sağlamıştır.

2.4. Test ve Performans Analizi

Geliştirilen plaka tanıma sistemi hem senaryo tabanlı hem de rastgele örneklerle farklı test ortamlarında denenmiştir. Testler sırasında sistemin genel doğruluk oranı, kullanıcı etkileşimi, hız ve kararlılık gibi kriterler dikkate alınmıştır.

2.4.1. Test Senaryoları

Aşağıdaki koşullar altında sistemin işleyişi gözlemlenmiştir:

- Net ve merkezi konumlandırılmış plaka görselleri
- Düşük ışık, gölgeli ortamlar

- Açık bozulması içeren fotoğraflar
- Arka arkaya aynı plakayla test (işlenmemiş tekrarlar)
- Kara listedeki plakalarla geçiş denemesi

Her testte sistemin PlateRecognizer API'den aldığı sonuçlar Firebase veritabanı ile eşleştirilmiş ve mobil uygulamaya yansıyan davranışlar incelenmiştir.

2.4.2. Hata Örnekleri

Testler sırasında şu hatalarla karşılaşılmıştır:

- **Görselin bulanık olması** → tanınamayan plaka, sistem “plaka algılanamadı” uyarısı verir.
- **Boş ya da desteklenmeyen dosya formatı** → sistem kullanıcıyı uyarır, hata loglanır.
- **Klasöre aynı isimli dosya tekrar geldiğinde** → önceki işlem kontrol edilir, tekrar işlenmez.

Sistem bu hataları engellemek yerine **yönetmeye odaklanmış** olup, hata anında kullanıcı bilgilendirilmekte ve sistem çalışmaya devam etmektedir.

2.4.3. Performans Gözlemleri

- Ortalama API yanıt süresi: **2–3 saniye**
- watcher.py klasörü her **5 saniyede bir** taramaktadır
- Firebase logları **anlık** olarak uygulamaya yansıtılmaktadır
- Uygulama açılışında log ve sayaçlar **2 saniye içinde yüklenmektedir**

Bu değerler, sistemin küçük ölçekli bir otopark için yeterli tepki süresine ve doğruluğa sahip olduğunu göstermektedir.

2.4.4. Doğruluk Oranı

Çizelge 2.4 Test koşullarına göre başarı oranı tablosu

Durum	Başarı Oranı
Net ve uygun açılı görseller	%95+
Orta seviye çözünürlük, az bozulma	%85
Kötü ışık ve açı koşulları	%60-70

2.4.5. Kullanıcı Geri Bildirimleri (Geliştirici Testleri)

- Arayüz sade ve yönlendirici bulunmuştur.
- Plaka ekleme/kara liste süreci sezgiseldir.
- Uygulamanın karanlık tema ile başlaması, kullanıcı deneyimini olumlu etkilemiştir.
- Giriş sonrası gelen son log bildirimi, anlık bilgi akışını netleştirmiştir.

3. SONUÇ

Bu çalışmada, otoparklarda araç giriş-çıkışlarını otomatikleştirmek amacıyla plaka tanıma tabanlı bir sistem geliştirilmiştir. Geliştirilen sistem; plaka görsellerinin tanınması, yetki kontrolü yapılması, kullanıcı onayı alınması ve bu işlemlerin mobil arayüz üzerinden yönetilmesini sağlayan bütünlük bir yapıya sahiptir.

Proje kapsamında hem teknik hem de kullanıcı odaklı birçok bileşen başarıyla entegre edilmiştir. Python ile geliştirilen arka plan işlem modülü, plaka görsellerini işleyerek PlateRecognizer API üzerinden plaka tanıma gerçekleştirmektedir. React Native ile hazırlanan mobil uygulama ise kullanıcıya plaka ekleme, kara liste yönetimi ve log görüntüleme gibi işlevler sunmaktadır. Firebase altyapısı sayesinde tüm veriler gerçek zamanlı olarak senkronize edilmekte ve güvenli bir şekilde saklanmaktadır.

Yapılan testler, sistemin iyi ışık koşullarında ve net görsellerle %95'in üzerinde doğruluk oranıyla çalıştığını göstermiştir. Karar sürecine kullanıcıyı dahil eden modal yapılar sayesinde sistem, otomatik işleyiş ile manuel kontrol arasında esnek bir yapı sunmuştur. Aynı zamanda Google Drive senkronizasyonu ile plaka görsellerinin sisteme aktarımı başarıyla otomatikleştirilmiştir.

Güçlü Yönler:

- Yüksek doğruluk oranı ve kullanıcıya açık işlem süreci
- Platform bağımsız mobil arayüz
- Gerçek zamanlı veri yönetimi
- Modüler ve genişletilebilir sistem yapısı

Sınırlamalar:

- Canlı video akışından doğrudan kare alma işlemi kapsam dışıdır
- Kamera görüntüleri çözünürlüğe ve açığa bağlı olarak değişken başarı gösterebilir
- Sistem, dış bağlantıya açık bir API değil; yalnızca tanımlı kullanıcılar içindir

Gelecekte Yapılabilecek Geliştirmeler:

- Doğrudan canlı video stream üzerinden kare yakalama desteği
- Web arayüzü veya yönetici paneli entegrasyonu
- Yerel sunucu üzerinden çalışan bir OCR modeli ile offline kullanım
- Plaka dışı veri (araç tipi, renk, hız vb.) tespiti ile sistemin genişletilmesi

Sonuç olarak, bu proje hem akademik hem uygulamalı düzeyde başarılı bir otopark otomasyon sisteminin temellerini ortaya koymuş, gerçek hayatta uygulanabilir, sürdürülebilir ve geliştirilebilir bir yapı sunmuştur.

KAYNAKLAR

- Goodfellow, I., Bengio, Y., & Courville, A. 2016. Deep Learning. MIT Press, 800 s., Cambridge, Massachusetts, ABD.
- Day, R.A. 1996. Bilimsel bir makale nasıl yazılır ve yayımlanır (çeviri: G.A. Altay). 2. Baskı, TÜBİTAK Yayını, 223 s., Ankara.
- Redmon, J., & Farhadi, A. 2018. YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767, 1-6.
- Smith, R. 2007. An Overview of the Tesseract OCR Engine. Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR), 629-633.
- OpenCV. 2023. Open-Source Computer Vision Library. OpenCV.org. <https://opencv.org/>
Erişim Tarihi: 15.10.2024
- Tesseract OCR. 2023. Tesseract Optical Character Recognition. GitHub Repository. <https://github.com/tesseract-ocr/tesseract>
Erişim Tarihi: 18.10.2024
- Python Software Foundation. 2023. Python Programming Language. <https://www.python.org/>
Erişim Tarihi: 21.10.2024
- Darknet YOLO. 2023. YOLO: Real-Time Object Detection. <https://pjreddie.com/darknet/yolo/>
Erişim Tarihi: 25.10.2024
- EasyOCR. 2023. EasyOCR Documentation. <https://github.com/JaidedAI/EasyOCR>
Erişim Tarihi: 27.10.2024
- Open Images Dataset. 2023. Dataset for Object Detection and Classification. <https://storage.googleapis.com/openimages/web/index.html>
Erişim Tarihi: 01.11.2024
- Kaggle License Plate Dataset. 2023. <https://www.kaggle.com/>
Erişim Tarihi: 03.11.2024
- Redmon, J. 2023. YOLOv4 Documentation. <https://github.com/AlexeyAB/darknet>
Erişim Tarihi: 23.11.2024
- Smith, R. 2023. Tesseract OCR User Guide. <https://github.com/tesseract-ocr/tesseract/wiki>
Erişim Tarihi: 01.12.2024
- Aytekin, A. (2020). *Görüntü İşleme ile Nesne Tanıma Sistemleri*. İstanbul: Teknoloji Yayınları
- Bilgin, M. (2019). *Python ile Görüntü İşleme ve Uygulamaları*. Ankara: Kodlab.
- Firebase. (2024). *Cloud Firestore Documentation*. Google Developers. <https://firebase.google.com/docs/firestore>
- Google Developers. (2024). *Google Drive API Overview*. <https://developers.google.com/drive>
- PlateRecognizer. (2024). *PlateRecognizer API Documentation*. <https://app.platerecognizer.com>