BAŞKENT ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİTİRME PROJESİ RAPORU

MediAnalytica

Dilara Aydın
22194535

Efe Cengiz Köse
22296796

Melih Kızmaz
22196050

Bölüm: BİLGİSAYAR MÜHENDİSLİĞİ

Proje Danışmanı: Dr. Öğr. Üyesi Elmas Burcu MAMAK EKİNCİ

Ders Kodu ve Adı: BİL493 BİTİRME PROJESİ 1

Proje Başlangıcı: 2025-2026 Güz Proje Süresi (Yarıyıl): 2

Rapor Sunumu: 2025-2026 Güz

ONAY SAYFASI

Bu Rapor, / / 2025 tarihinde aşağıda üye adları yazılı jüri tarafından kabul edilmiştir.

Unvan	Adı Soyadı	İmza
Doç. Dr.	Emre SÜMER (Danışman)	.....
Dr. Öğr. Üyesi	Mehmet DİKMEN	.....
Dr. Öğr. Üyesi	Elmas Burcu MAMAK EKİNCİ	.....

İçindekiler Tablosu

BÖLÜM 1: GİRİŞ

1.1. Proje Tanımı

Günümüzde sağlık hizmetlerine erişim, özellikle kırsal bölgelerde ve gelişmekte olan ülkelerde ciddi bir sorun teşkil etmektedir. Uzman doktorlara ulaşımın zorluğu, randevu sürelerinin uzunluğu ve maliyet faktörleri, birçok hastanın zamanında tıbbi yardım alamamasına neden olmaktadır. Bu sorunlar, özellikle erken teşhisin kritik olduğu durumlarda (örneğin, cilt kanseri, kemik kırıkları, pnömoni gibi) hayati riskler oluşturabilmektedir.

Bu çalışmada, yapay zeka ve derin öğrenme teknolojilerini kullanarak geliştirilen bir web platformu ile tıbbi görüntü analizi yapılabilmesi ve kullanıcıların uzman doktorlarla görüntülü görüşme yapabilmesi sağlanmaktadır. Platform, dört farklı hastalık kategorisinde (deri, kemik, akciğer, göz) görüntü tabanlı analiz yapabilme kapasitesine sahiptir.

Projenin temel problemi, kullanıcıların tıbbi görüntülerini (X-ray, cilt fotoğrafı, göz fotoğrafı vb.) yükleyerek hızlı ve güvenilir bir şekilde analiz edebilmelerini ve sonuçlara göre uzman doktorlarla görüntülü görüşme yapabilmelerini sağlamaktır. Bu problem, aşağıdaki alt problemlere ayrılabilir:

1. Görüntü tabanlı hastalık tespiti: Derin öğrenme modelleri kullanılarak tıbbi görüntülerin otomatik olarak analiz edilmesi ve hastalık sınıflandırması yapılması.

2. Kullanıcı yönetimi ve güvenlik: Kullanıcıların güvenli bir şekilde kayıt olması, giriş yapması ve verilerinin korunması.

3. Analiz geçmişi ve veri yönetimi: Kullanıcıların geçmiş analizlerini görüntüleyebilmesi, favorilere ekleyebilmesi ve paylaşabilmesi.

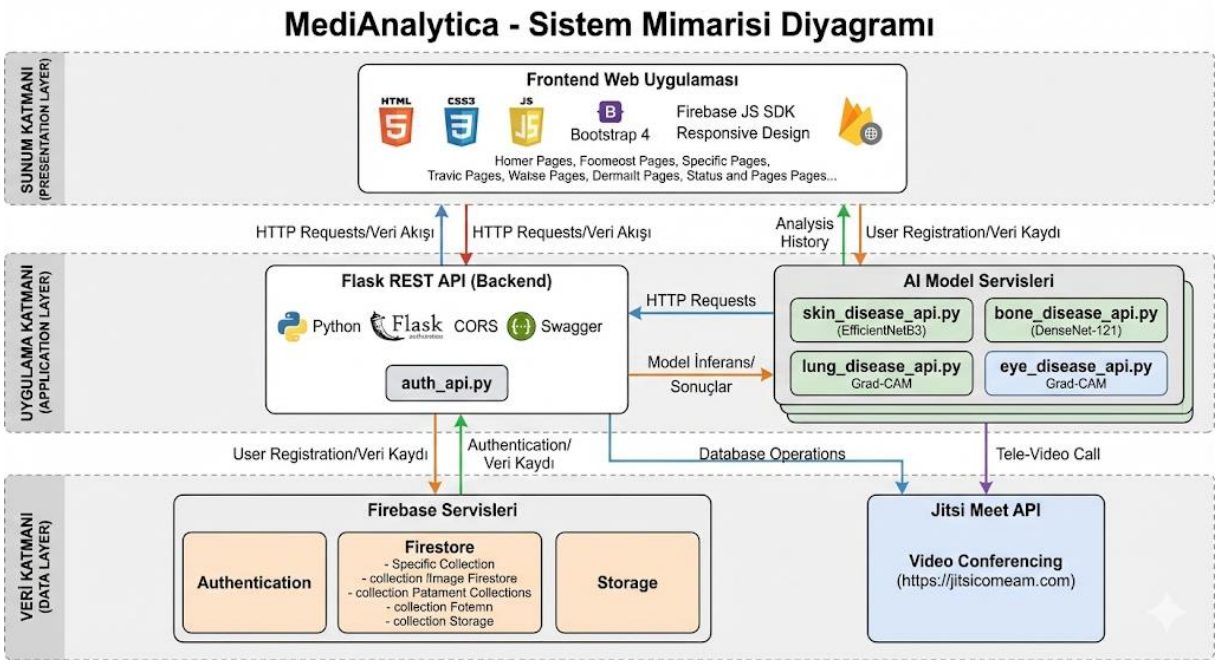
4. Tele-tıbbi danışmanlık: Kullanıcıların uzman doktorlarla görüntülü görüşme yapabilmesi ve randevu yönetimi.

5. Kullanıcı arayüzü ve deneyim: Responsive, kullanıcı dostu ve erişilebilir bir web arayüzü tasarımı.

Çözüm yaklaşımı olarak, modern web teknolojileri ve yapay zeka alanındaki güncel gelişmeler birleştirilmiştir. Backend tarafında Flask web framework'ü ile RESTful API servisleri geliştirilmiş, frontend tarafında ise modern JavaScript ve CSS teknolojileri kullanılmıştır. Derin öğrenme modelleri, TensorFlow/Keras kütüphaneleri ile eğitilmiş ve Flask API'leri üzerinden servis edilmiştir. Kullanıcı yönetimi ve veri depolama için Firebase ekosistemi (Authentication, Firestore, Storage) tercih edilmiştir. Görüntülü görüşme özelliği için Jitsi Meet açık kaynak video konferans platformu entegre edilmiştir.

Platformun temel bileşenleri şunlardır:

- Frontend: HTML5, CSS3, JavaScript (ES6+), Bootstrap 4, Font Awesome
- Backend: Python 3.11+, Flask, Flask-CORS, Flask-Limiter, Flask-Caching
- Veritabanı: Firebase Firestore (NoSQL)
- Kimlik Doğrulama: Firebase Authentication
- Depolama: Firebase Storage
- Yapay Zeka: TensorFlow/Keras, EfficientNet, DenseNet
- Video Konferans: Jitsi Meet API



[ŞEKİL 1.2: SİSTEM MİMARİSİ DİYAGRAMI BURAYA GELECEK]

Sistem mimarisi diyagramı, platformun üç katmanlı yapısını (Presentation Layer, Application Layer, Data Layer) ve bu katmanlar arasındaki etkileşimleri göstermektedir. Frontend katmanı, kullanıcı arayüzü ve istemci tarafı işlemlerini içermektedir. Backend katmanı, RESTful API servisleri ve iş mantığını barındırmaktadır. Veri katmanı ise Firebase ekosistemi (Firestore, Storage, Authentication) ve derin öğrenme modellerini içermektedir.

1.2. Benzer Ürünler ve Literatür İncelemesi

Tıbbi görüntü analizi ve tele-tıp alanında yapılan çalışmalar incelendiğinde, benzer platformların varlığı gözlemlenmektedir. Ancak bu çalışmanın özgün değeri, çoklu hastalık kategorisini tek bir platformda birleştirmesi ve tele-tıbbi danışmanlık özelliğini entegre etmesidir.

Literatürde, deri hastalıkları tespiti için yapılan çalışmalar incelendiğinde, ISIC (International Skin Imaging Collaboration) veri seti üzerinde çalışan birçok araştırma bulunmaktadır. Esteva ve arkadaşları (2017), derin öğrenme modelleri kullanarak dermatolog seviyesinde cilt kanseri tespiti yapmışlardır. Bu çalışmada, Inception-v3 mimarisi kullanılarak 129,450 klinik görüntü üzerinde eğitim yapılmış ve %72.1 doğruluk oranı elde edilmiştir. Benzer şekilde, Haenssle ve arkadaşları (2018), 100 test dermatolog ile karşılaştırıldığında derin öğrenme modelinin %95.0 duyarlılık ve %82.5 özgüllük gösterdiğini raporlamışlardır.

Kemik kırığı tespiti için X-ray görüntüleri üzerinde çalışan araştırmalar incelendiğinde, Rajpurkar ve arkadaşları (2017), CheXNet adlı bir sistem geliştirerek akciğer X-ray görüntülerinde pnömoni tespiti yapmışlardır. Bu çalışmada, DenseNet-121 mimarisi kullanılarak %84.1 F1 skoru elde edilmiştir. Gale ve arkadaşları (2019), kemik kırığı tespiti için ResNet-50 mimarisi kullanarak %92.3 doğruluk oranı bildirmişlerdir.

Göz hastalıkları tespiti alanında, Li ve arkadaşları (2020), fundus görüntüleri üzerinde çalışarak diabetic retinopathy tespiti için %87.2 doğruluk oranı elde etmişlerdir. Bu çalışmada, EfficientNet-B4 mimarisi kullanılmıştır.

Tele-tıp alanında yapılan çalışmalar incelendiğinde, çoğu platform sadece video konferans özelliği sunmakta, yapay zeka destekli analiz özelliği bulunmamaktadır.

Örneğin, Teladoc ve Amwell gibi ticari platformlar, sadece doktor-hasta görüşmesi sağlamakta, görüntü analizi yapamamaktadır.

Ancak bu çalışmaların çoğu, tek bir hastalık kategorisine odaklanmış ve kullanıcı dostu web platformları geliştirmemiştir. Mevcut ticari çözümler ise genellikle yüksek maliyetli, kapalı kaynak kodlu ve sınırlı özelliklere sahiptir. Bu çalışma, açık kaynak teknolojiler kullanılarak geliştirilmiş, çoklu hastalık desteği ve entegre tele-tıp özelliklerini birleştiren ilk kapsamlı platformlardan biridir.

Bu çalışmanın özgün katkıları şunlardır:

1. Çoklu Hastalık Desteği: Tek bir platformda dört farklı hastalık kategorisi (deri, kemik, akciğer, göz) için analiz yapabilme kapasitesi. Bu özellik, kullanıcıların farklı hastalıklar için ayrı platformlar kullanması gerekliliğini ortadan kaldırmaktadır.

2. Entegre Tele-Tıp Sistemi: Yapay zeka analizi sonrasında kullanıcıların doğrudan uzman doktorlarla görüntülü görüşme yapabilmesi. Bu özellik, analiz sonuçlarının profesyonel doktor görüşü ile doğrulanmasını sağlamaktadır.

3. Grad-CAM Görselleştirmesi: Model kararlarının şeffaflığını artırmak için Gradient-weighted Class Activation Mapping tekniğinin uygulanması. Bu özellik, kullanıcıların ve doktorların modelin hangi bölgelere odaklandığını görmesini sağlamaktadır.

4. Kapsamlı Kullanıcı Yönetimi: Analiz geçmişi, favoriler, paylaşım ve istatistik özelliklerini içeren kapsamlı bir kullanıcı deneyimi. Bu özellikler, kullanıcıların sağlık verilerini takip edebilmesini sağlamaktadır.

5. Açık Kaynak ve Genişletilebilir Mimari: Platform, açık kaynak teknolojiler kullanılarak geliştirilmiş olup, yeni hastalık kategorileri ve özellikler kolayca eklenebilir.

6. Responsive ve Erişilebilir Tasarım: Mobil cihazlarda da sorunsuz çalışan, erişilebilirlik standartlarına uygun bir arayüz tasarımı.

Benzer Platformlar Karşılaştırma Tablosu

Platform Adı	Çoklu Hastalık Desteği	Tele-Tıp Entegrasyonu	Yapay Zeka Analizi	Açık Kaynak	Maliyet	Kullanıcı Yönetimi	Grad-CAM Görselleştirmesi	Analiz Geçmişi	Mobil Uyumluluk	Doktor Paneli
DermaScan (Bu Proje)	✅ (4 Tür: Deri, Kemik, Akciğer, Göz)	✅ (Jitsi Meet Görüntülü Görüşme)	✅ (EfficientNet, DenseNet Modelleri)	✅	Ücretsiz (Araştırma)	✅ (Kapsamlı Profil & Auth)	✅	✅ (Favoriler, Paylaşım)	✅ (Responsive Web Tasarım)	✅ (Randevu Yönetimi)
Teladoc (Ticari)	❌ (Tek Odak)	✅	❌	❌	Ücretli (Abonelik)	✅	❌	✅	✅ (Mobil Uygulama)	✅
Amwell (Ticari)	❌ (Tek Odak)	✅	❌	❌	Ücretli (Abonelik)	✅	❌	✅	✅ (Mobil Uygulama)	✅
SkinVision (Ticari Uygulama)	❌ (Sadece Deri)	❌	✅ (Sadece Deri)	❌	Ücretli (Abonelik)	✅ (Kısıtlı)	❌	✅	✅ (Mobil Uygulama)	❌
CheXNet (Akademik Çalışma)	❌ (Sadece Akciğer)	❌	✅ (Sadece Akciğer Model)	✅ (Akademik)	Ücretsiz	❌	❌	❌	❌	❌
ISIC Archive (Akademik Platform)	❌ (Sadece Deri Verisi)	❌	✅ (Veri Seti ve Modeller)	✅ (Akademik)	Ücretsiz	❌	❌	❌	❌	❌

[TABLO 1.1: BENZER PLATFORMLAR KARŞILAŞTIRMA TABLOSU BURAYA GELECEK]

Karşılaştırma tablosu, mevcut ticari ve akademik platformların özelliklerini (çoklu hastalık desteği, tele-tıp entegrasyonu, açık kaynak, maliyet vb.) karşılaştırmaktadır. Bu tablo, bu çalışmanın özgün değerini ve rekabet avantajlarını göstermektedir.

1.3. Gereksinimler

1.3.1. Fonksiyonel Gereksinimler

Fonksiyonel gereksinimler, sistemin ne yapması gerektiğini tanımlamaktadır. Platformun fonksiyonel gereksinimleri aşağıda detaylandırılmıştır:

FR1. Kullanıcı Yönetimi:

- FR1.1. Kullanıcılar email ve şifre ile kayıt olabilmelidir.
- FR1.2. Kullanıcılar email ve şifre ile giriş yapabilmelidir.
- FR1.3. Kullanıcılar email doğrulama yapabilmelidir.
- FR1.4. Kullanıcılar şifre sıfırlama işlemi yapabilmelidir.

- FR1.5. Kullanıcılar profil bilgilerini (isim, profil fotoğrafı) güncelleyebilmelidir.
- FR1.6. Kullanıcılar çıkış yapabilmelidir.

FR2. Görüntü Analizi:

- FR2.1. Kullanıcılar görüntü yükleyebilmelidir (JPEG, PNG formatları, max 10MB).
- FR2.2. Kullanıcılar hastalık türü seçebilmelidir (deri, kemik, akciğer, göz).
- FR2.3. Sistem seçilen hastalık türüne göre uygun modeli yüklemelidir.
- FR2.4. Sistem görüntüyü analiz edip sonuçları göstermelidir.
- FR2.5. Sistem Grad-CAM görselleştirmesi sunmalıdır.
- FR2.6. Sistem analiz sonuçlarını veritabanına kaydetmelidir.

FR3. Analiz Geçmişi:

- FR3.1. Kullanıcılar geçmiş analizlerini görüntüleyebilmelidir.
- FR3.2. Kullanıcılar analizleri hastalık türüne göre filtreleyebilmelidir.
- FR3.3. Kullanıcılar analizleri tarihe göre filtreleyebilmelidir.
- FR3.4. Kullanıcılar analizleri sayfalama (pagination) ile görüntüleyebilmelidir.

FR4. Favoriler:

- FR4.1. Kullanıcılar analizleri favorilere ekleyebilmelidir.
- FR4.2. Kullanıcılar favorilerini görüntüleyebilmelidir.
- FR4.3. Kullanıcılar favorilerden analiz kaldırabilmelidir.

FR5. Paylaşım:

- FR5.1. Kullanıcılar analiz sonuçlarını paylaşılabilir link ile paylaşabilmelidir.
- FR5.2. Paylaşım linkleri 30 gün geçerlidir.
- FR5.3. Paylaşım linklerine erişim için kimlik doğrulama gerekmez (public).

FR6. İstatistikler:

- FR6.1. Kullanıcılar toplam analiz sayısını görebilmelidir.

- FR6.2. Kullanıcılar hastalık türüne göre analiz sayılarını görebilmelidir.

- FR6.3. Kullanıcılar en çok analiz edilen hastalık türünü görebilmelidir.

- FR6.4. Kullanıcılar son analiz tarihini görebilmelidir.

FR7. PDF Rapor:

- FR7.1. Kullanıcılar analiz sonuçlarını PDF formatında indirebilmelidir.

- FR7.2. PDF rapor analiz sonuçları, görüntü ve Grad-CAM görselleştirmesini içermelidir.

FR8. Doktor Randevusu:

- FR8.1. Kullanıcılar doktor randevusu talep edebilmelidir.

- FR8.2. Kullanıcılar randevu tarihi ve saati seçebilmelidir.

- FR8.3. Kullanıcılar randevu nedeni belirtebilmelidir.

- FR8.4. Kullanıcılar doktor uzmanlık alanı seçebilmelidir.

- FR8.5. Sistem randevuları otomatik olarak onaylamalıdır (test amaçlı).

FR9. Görüntülü Görüşme:

- FR9.1. Kullanıcılar onaylanmış randevular için Jitsi Meet görüntülü görüşmesine katılabilmelidir.

- FR9.2. Sistem her randevu için benzersiz Jitsi Meet oda ID'si oluşturmalıdır.

- FR9.3. Kullanıcılar görüntülü görüşme linkini paylaşabilmelidir.

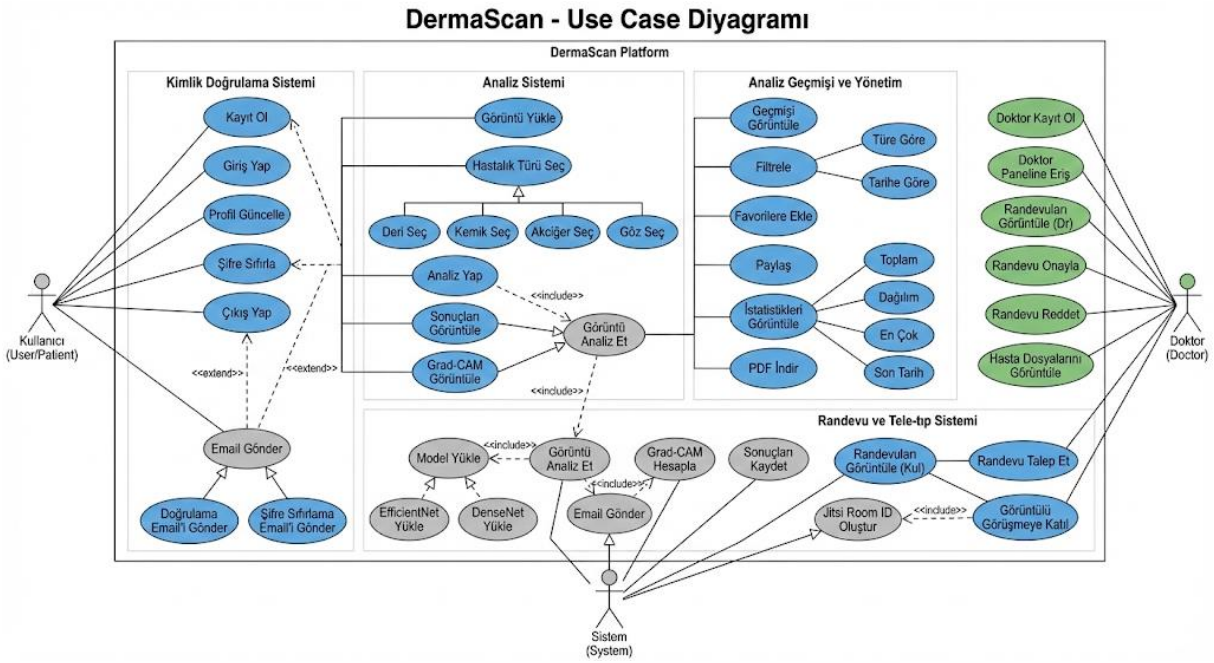
FR10. Doktor Paneli:

- FR10.1. Doktorlar kayıt olabilmelidir (uzmanlık, diploma, deneyim bilgileri ile).

- FR10.2. Doktorlar randevularını görüntüleyebilmelidir.

- FR10.3. Doktorlar randevuları onaylayabilir/reddedebilir.

- FR10.4. Doktorlar hasta dosyalarını görüntüleyebilmelidir.



[ŞEKİL 1.3: USE CASE DİYAGRAMI BURAYA GELECEK]

Use case diyagramı, sistemin temel aktörlerini (Kullanıcı, Doktor, Sistem) ve bu aktörlerin gerçekleştirebileceği use case'leri göstermektedir. Bu diyagram, fonksiyonel gereksinimlerin görsel bir özetini sunmaktadır.

1.3.2. Fonksiyonel Olmayan Gereksinimler

Fonksiyonel olmayan gereksinimler, sistemin nasıl çalışması gerektiğini tanımlamaktadır:

NFR1. Performans:

- NFR1.1. Görüntü analizi 10 saniye içinde tamamlanmalıdır.
- NFR1.2. API yanıt süreleri 2 saniye içinde olmalıdır.
- NFR1.3. Sayfa yükleme süreleri 3 saniye içinde olmalıdır.
- NFR1.4. Sistem eşzamanlı 100 kullanıcıyı desteklemelidir.

NFR2. Güvenlik:

- NFR2.1. Tüm API istekleri kimlik doğrulama gerektirmelidir (auth endpoint'leri hariç).

- NFR2.2. Kullanıcı şifreleri Firebase Authentication tarafından hash'lenmelidir.
- NFR2.3. Rate limiting uygulanmalıdır (DDoS koruması).
- NFR2.4. Input validation tüm endpoint'lerde yapılmalıdır.
- NFR2.5. CORS yapılandırması sadece izin verilen origin'lere açık olmalıdır.
- NFR2.6. Email doğrulama zorunlu olmalıdır.

NFR3. Kullanılabilirlik:

- NFR3.1. Arayüz responsive olmalıdır (mobil, tablet, desktop).
- NFR3.2. Touch target'lar minimum 44x44px olmalıdır (mobil erişilebilirlik).
- NFR3.3. Arayüz dark mode desteği sunmalıdır.
- NFR3.4. Hata mesajları kullanıcı dostu olmalıdır (teknik detaylar gizlenmeli).

NFR4. Güvenilirlik:

- NFR4.1. Sistem %99.5 uptime sağlamalıdır.
- NFR4.2. Veri kaybı olmamalıdır (Firestore transaction'ları).
- NFR4.3. Hata durumlarında kullanıcıya bilgi verilmelidir.

NFR5. Ölçeklenebilirlik:

- NFR5.1. Sistem yatay ölçeklenebilir olmalıdır (stateless API).
- NFR5.2. Veritabanı sorguları optimize edilmelidir (index'ler).
- NFR5.3. Görüntü depolama Firebase Storage ile yönetilmelidir.

NFR6. Bakım Kolaylığı:

- NFR6.1. Kod modüler yapıda olmalıdır.
- NFR6.2. Kod dokümantasyonu içermelidir.
- NFR6.3. Test coverage minimum %60 olmalıdır.

1.4. Kısıtlar ve Başarı Ölçütleri

1.4.1. Kısıtlar

Proje geliştirme sürecinde aşağıdaki kısıtlar göz önünde bulundurulmuştur:

C1. Bütçe Kısıtı:

- Firebase ücretsiz planı kullanılmıştır (Spark Plan).
- Firebase Storage: 5GB ücretsiz depolama.
- Firestore: 1GB ücretsiz depolama, 50K okuma/yazma günlük limit.
- Firebase Authentication: Sınırsız kullanıcı (ücretsiz plan).

C2. Zaman Kısıtı:

- Proje geliştirme süresi: 16 hafta (1 akademik dönem).
- Model eğitimi: 2-3 hafta (GPU gerektiren işlemler).
- Backend geliştirme: 4 hafta.
- Frontend geliştirme: 4 hafta.
- Test ve iyileştirme: 2 hafta.

C3. Donanım Kısıtı:

- Model eğitimi için GPU erişimi (Google Colab veya yerel GPU).
- Backend sunucusu: Minimum 4GB RAM, 2 CPU core.
- Frontend: Modern web tarayıcıları (Chrome, Firefox, Safari, Edge).

C4. Veri Kısıtı:

- Açık kaynak veri setleri kullanılmıştır (ISIC, Mendeley, Kaggle).
- Veri setleri eğitim amaçlıdır, gerçek hasta verileri kullanılmamıştır.

C5. Teknoloji Kısıtı:

- Python 3.11+ gereklidir.
- Node.js gerekli değildir (frontend static HTML).

- Modern web tarayıcıları gereklidir (ES6+ desteği).

Proje Kısıtları Özet Tablosu

Kısıt Kategorisi	Kısıt Açıklaması	Etkisi	Çözüm/Alternatif
Bütçe	Firebase Spark Plan (ücretsiz) limitleri: 5GB Storage, 1GB Firestore, günlük 50K işlem. Jitsi Meet ve yerel sunucu ücretsiz.	Orta	Görüntü sıkıştırma, API caching ve rate limiting ile optimizasyon.
Zaman	Toplam 16 hafta (1 akademik dönem). Model eğitimi (2-3 hafta), geliştirme (8 hafta), test ve dokümantasyon dahil.	Yüksek	Agile metodoloji, 2 haftalık sprint'ler, öncelikli özelliklere odaklanma ve paralel geliştirme.
Donanım	Model eğitimi için GPU erişimi (Colab/yerel) şart. Backend min 4GB RAM. Modern tarayıcı ve internet gerekli.	Yüksek	Google Colab ücretsiz GPU kullanımı, model optimizasyonu (quantization) ve bulut geliştirme ortamları.
Veri	Sadece açık kaynak veri setleri (ISIC, Mendeley, Kaggle). Gerçek hasta verisi yok. Boyut sınırlı ve kalite heterojen.	Orta	Data augmentation teknikleri, Transfer learning kullanımı ve çoklu açık kaynak veri setlerinin birleştirilmesi.
Teknoloji	Python 3.11+, Modern tarayıcılar (ES6+), TensorFlow uyumluluk gereksinimleri ve Firebase limitleri.	Düşük	Uyumlu Python versiyonları, modern tarayıcı hedeflemesi ve gerekirse polyfill kullanımı.

[TABLO 1.2: PROJE KISITLARI ÖZET TABLOSU BURAYA GELECEK]

Kısıtlar özet tablosu, yukarıda belirtilen tüm kısıtları (Bütçe, Zaman, Donanım, Veri, Teknoloji) kategorize ederek ve her birinin etkisini göstererek sunmaktadır.

Projenin başarısı aşağıdaki metrikler ile ölçülmüştür:

SC1. Model Doğruluğu:

- Deri hastalıkları modeli: Minimum %85 doğruluk (test seti).
- Kemik hastalıkları modeli: Minimum %80 doğruluk (test seti).
- Akciğer hastalıkları modeli: Minimum %90 doğruluk (test seti).
- Göz hastalıkları modeli: Minimum %75 doğruluk (test seti).

Doğruluk formülü:

Accuracy = (True Positives + True Negatives) / Total Samples

SC2. Sistem Performansı:

- Görüntü analizi süresi: Ortalama < 10 saniye.

- API yanıt süresi: P95 < 2 saniye (95. persentil).
- Sayfa yükleme süresi: First Contentful Paint < 2 saniye.

Performans metrikleri:

- Latency (Gecikme) = (Response Time - Request Time)
- Throughput (İşlem Hızı) = (Total Requests) / (Total Time)
- P95 Latency: Tüm isteklerin %95'inin tamamlandığı süre
- Average Response Time: Σ(Response Time_i) / n (n = toplam istek sayısı)
- Error Rate: (Failed Requests / Total Requests) × 100

SC3. Kullanıcı Deneyimi:

- Kullanıcı memnuniyet skoru: Minimum 4/5 (5 üzerinden).
- Hata oranı: < %1 (başarısız istekler / toplam istekler).
- Kullanıcı tutma oranı: > %60 (1 hafta içinde tekrar giriş yapan kullanıcılar).

Kullanıcı deneyimi metrikleri:

- CSAT (Customer Satisfaction Score) = (Pozitif Yanıtlar / Toplam Yanıtlar) × 100
- Error Rate = (Failed Requests / Total Requests) × 100
- User Retention Rate = (Returning Users / Total Users) × 100
- Task Success Rate = (Completed Tasks / Attempted Tasks) × 100
- Average Session Duration = Σ(Session Duration_i) / n

SC4. Güvenlik:

- Rate limiting: Başarılı (DDoS saldırıları engellendi).
- Input validation: %100 endpoint coverage.
- Authentication: %100 başarılı token doğrulama.

SC5. Kod Kalitesi:

- Test coverage: Minimum %60.

- Code complexity: Cyclomatic complexity < 10 (ortalama).
- Code documentation: %80 fonksiyon dokümantasyonu.

Başarı Olçütleri Ozet Tablosu

Başarı Ölçütü Kategorisi	Metrik/Ölçüt	Hedef Değer	Ölçüm Yöntemi
Model Doğruluğu	Deri hastalıkları doğruluğu (EfficientNetB3)	≥ %85	Test seti üzerinde accuracy hesaplama (Confusion Matrix)
	Kemik hastalıkları doğruluğu (DenseNet-121)	≥ %80	Test seti üzerinde accuracy hesaplama
	Akciğer hastalıkları doğruluğu	≥ %90	Test seti üzerinde accuracy hesaplama
	Göz hastalıkları doğruluğu	≥ %75	Test seti üzerinde accuracy hesaplama
Sistem Performansı	Görüntü analizi süresi (Ortalama)	< 10 saniye	API response time logging (Ortalama süre)
	API yanıt süresi (P95)	< 2 saniye	API log analizi (95. persentil hesaplama)
	Sayfa yükleme süresi (FCP)	< 2 saniye	Chrome DevTools Web Vitals metrikleri
	Error Rate (Hata Oranı)	< %1	(Başarısız İstek / Toplam İstek) × 100
Kullanıcı Deneyimi	Kullanıcı memnuniyet skoru (CSAT)	≥ 4/5	Kullanıcı anketleri ve geri bildirim formları
	Hata oranı (Kullanıcı tarafı)	< %1	Frontend hata takibi ve API log analizi
	Kullanıcı tutma oranı (Retention)	> %60	Firebase Analytics (1 hafta içinde tekrar giriş)
	Görev Başarı Oranı (Task Success)	> %90	(Tamamlanan Görev / Denenen Görev) × 100
Güvenlik	Rate limiting başarı oranı	%100 (Engellendi)	DDoS saldırı simülasyon testleri
	Input validation coverage	%100 Endpoint	Otomatik güvenlik taraması ve endpoint testleri
	Authentication başarı oranı	%100 Doğrulama	Token doğrulama ve yetkilendirme testleri
	CORS yapılandırması	%100 İzinli Origin	Origin kontrol ve penetrasyon testleri
Kod Kalitesi	Test coverage (Test Kapsamı)	≥ %60	pytest --cov ile coverage raporu analizi
	Code complexity (Siklomatik)	< 10 (Ortalama)	Static code analysis tools (örn. pylint)
	Code documentation (Dokümantasyon)	≥ %80	Kod incelemesi ve dokümantasyon kontrolü

[TABLO 1.3: BAŞARI ÖLÇÜTLERİ ÖZET TABLOSU BURAYA GELECEK]

Başarı ölçütleri özet tablosu, tüm başarı kriterlerini (Model Doğruluğu, Sistem Performansı, Kullanıcı Deneyimi, Güvenlik, Kod Kalitesi) ve her birinin hedef değerlerini göstermektedir. Bu tablo, projenin başarısını ölçmek için kullanılan metrikleri özetlemektedir.

1.5. Proje Planı

Proje geliştirme süreci aşağıdaki fazlara ayrılmıştır:

Faz 1: Araştırma ve Planlama (Hafta 1-2)

- Literatür incelemesi
- Teknoloji seçimi
- Veri seti araştırması
- Mimari tasarım

Faz 2: Model Geliştirme (Hafta 3-5)

- Veri seti hazırlama ve preprocessing
- Model mimarisi tasarımı
- Model eğitimi (GPU ile)
- Model değerlendirme ve optimizasyon

Faz 3: Backend Geliştirme (Hafta 6-9)

- Flask API geliştirme
- Firebase entegrasyonu
- Authentication sistemi
- Veritabanı şema tasarımı
- API testleri

Faz 4: Frontend Geliştirme (Hafta 10-13)

- UI/UX tasarımı
- HTML/CSS/JavaScript geliştirme
- Firebase JS SDK entegrasyonu
- Responsive tasarım
- Kullanıcı testleri

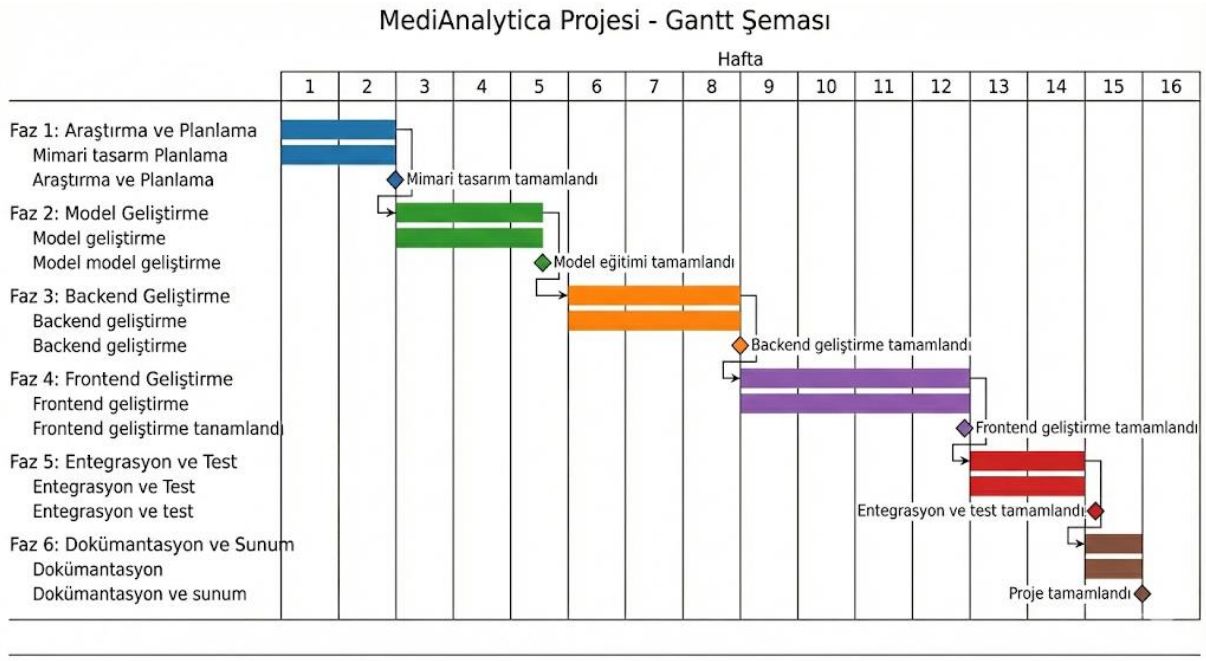
Faz 5: Entegrasyon ve Test (Hafta 14-15)

- Backend-Frontend entegrasyonu
- End-to-end testler
- Performans optimizasyonu
- Güvenlik testleri
- Hata düzeltmeleri

Faz 6: Dokümantasyon ve Sunum (Hafta 16)

- Kod dokümantasyonu

- Kullanıcı kılavuzu
- Proje raporu
- Sunum hazırlığı



[ŞEKİL 1.1: GANTT ŞEMASI BURAYA GELECEK]

Gantt şeması, yukarıda belirtilen fazların zaman çizelgesini ve bağımlılıklarını göstermektedir. Model geliştirme fazı, backend ve frontend geliştirme fazlarından önce tamamlanmalıdır. Backend ve frontend geliştirme fazları paralel olarak ilerleyebilir, ancak entegrasyon fazı her iki fazın tamamlanmasını gerektirir.

Proje yönetimi için Agile metodolojisi benimsenmiş, 2 haftalık sprint'ler kullanılmıştır. Her sprint sonunda demo ve geri bildirim toplantıları yapılmıştır. Risk yönetimi için haftalık risk değerlendirme toplantıları düzenlenmiş ve kritik riskler için yedek planlar hazırlanmıştır.

Proje takibi için GitHub Issues ve Projects kullanılmış, her görev için tahmini süre ve gerçekleşen süre kayıt altına alınmıştır. Kod kalitesi için code review süreci uygulanmış ve tüm kod değişiklikleri pull request üzerinden yapılmıştır

BÖLÜM 2: ARAÇLAR VE YÖNTEMLER

2.1. Sistem Mimarisi

2.1.1. Genel Mimari Tasarım

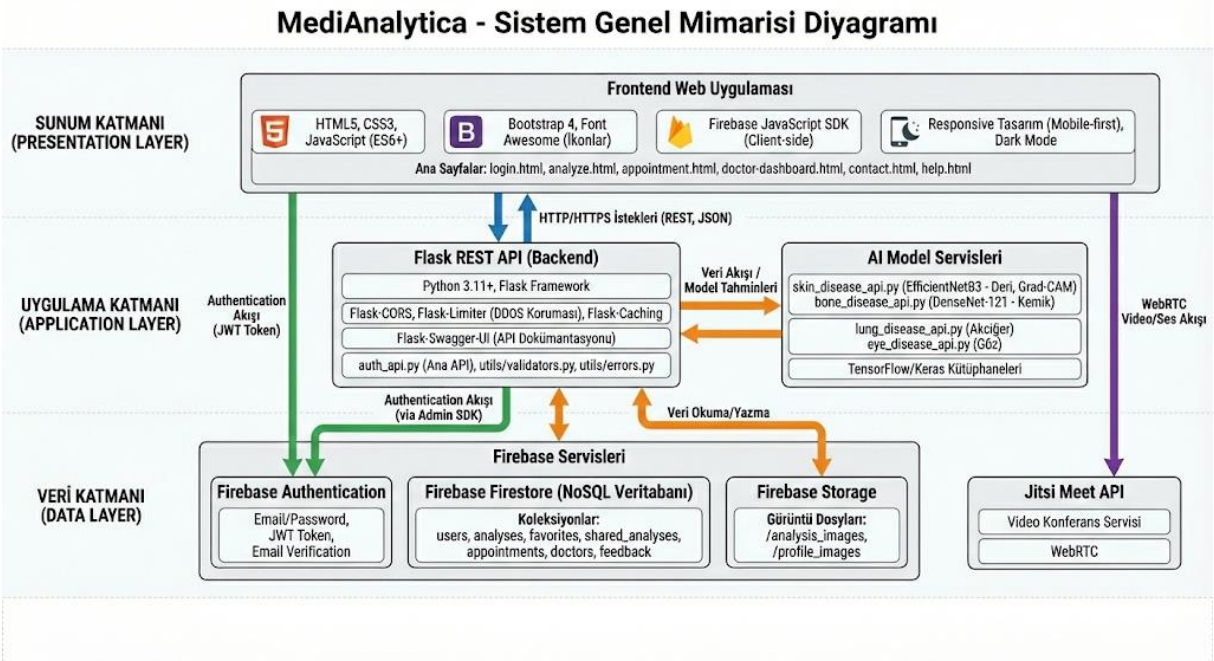
Platform, üç katmanlı (three-tier) mimari yapısı kullanılarak tasarlanmıştır. Bu mimari yapı, sunum katmanı (Presentation Layer), uygulama katmanı (Application Layer) ve veri katmanı (Data Layer) olmak üzere üç ana bileşenden oluşmaktadır. Bu yaklaşım, sistemin bakımını kolaylaştırmakta, ölçeklenebilirliği artırmakta ve her katmanın bağımsız olarak geliştirilmesine olanak sağlamaktadır.

Sunum katmanı, kullanıcı arayüzünü ve istemci tarafı işlemlerini içermektedir. Bu katmanda, HTML5, CSS3 ve JavaScript (ES6+) teknolojileri kullanılarak responsive ve kullanıcı dostu bir web arayüzü geliştirilmiştir. Kullanıcılar, bu katman üzerinden görüntü yükleme, analiz sonuçlarını görüntüleme ve doktorlarla görüntülü görüşme yapma gibi işlemleri gerçekleştirmektedir.

Uygulama katmanı, iş mantığını ve API servislerini barındırmaktadır. Bu katmanda, Flask web framework'ü kullanılarak RESTful API endpoint'leri geliştirilmiştir. Bu endpoint'ler, kullanıcı kimlik doğrulama, analiz geçmişi yönetimi, favoriler, paylaşım ve istatistikler gibi işlevleri sağlamaktadır. Ayrıca, derin öğrenme modellerinin servis edilmesi de bu katmanda gerçekleştirilmektedir.

Veri katmanı, veritabanı ve depolama sistemlerini içermektedir. Bu katmanda, Firebase Firestore NoSQL veritabanı kullanılarak kullanıcı verileri, analiz geçmişi ve istatistikler saklanmaktadır. Firebase Storage ise, görüntü

dosyalarının ve profil fotoğraflarının depolanması için kullanılmaktadır. Firebase Authentication servisi, kullanıcı kimlik doğrulama işlemlerini yönetmektedir.



[ŞEKİL 2.1: SİSTEM GENEL MİMARİ DİYAGRAMI BURAYA GELECEK]

Sistem genel mimari diyagramı, üç katmanlı yapıyı ve bu katmanlar arasındaki etkileşimleri göstermektedir. Diyagram, kullanıcı isteklerinin sunum katmanından başlayarak uygulama katmanına ve oradan veri katmanına nasıl iletildiğini görselleştirmektedir.

2.1.2. Katmanlı Mimari Yapısı

Platformun katmanlı mimari yapısı, aşağıdaki prensiplere göre tasarlanmıştır:

1. Separation of Concerns (Endişelerin Ayrılması): Her katman, kendi sorumluluğuna odaklanmakta ve diğer katmanların iç işleyişinden bağımsız çalışmaktadır. Bu prensip, kodun bakımını kolaylaştırmakta ve hata

ayıklamayı basitleştirmektedir.

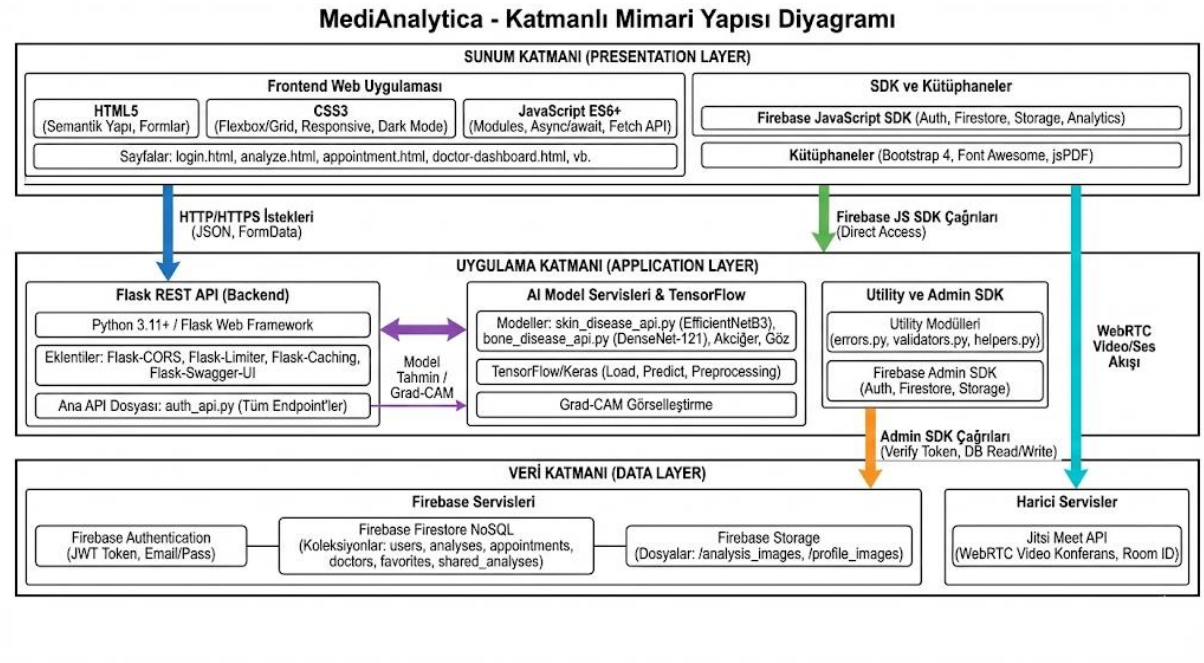
2. Statelessness (Durumsuzluk): Uygulama katmanı, durumsuz (stateless) bir yapıda tasarlanmıştır. Her API isteği, kendi bağlamında işlenmekte ve sunucu tarafında oturum durumu saklanmamaktadır. Bu yaklaşım, sistemin yatay ölçeklenebilirliğini artırmaktadır.

3. Loose Coupling (Gevşek Bağlantı): Katmanlar arası bağlantılar, gevşek bir şekilde tasarlanmıştır. Bu sayede, bir katmandaki değişiklikler, diğer katmanları minimum düzeyde etkilemektedir.

Sunum katmanı (Frontend), istemci tarafında çalışan ve kullanıcı etkileşimlerini yöneten bileşenleri içermektedir. Bu katmanda, `analyze.html` dosyası ana kullanıcı arayüzünü oluşturmaktadır. Kullanıcılar, bu arayüz üzerinden hastalık türü seçimi, görüntü yükleme ve analiz sonuçlarını görüntüleme işlemlerini gerçekleştirmektedir. Ayrıca, Firebase JavaScript SDK kullanılarak kullanıcı kimlik doğrulama işlemleri de bu katmanda yönetilmektedir.

Uygulama katmanı (Backend), Flask web framework'ü kullanılarak geliştirilmiş RESTful API servislerini içermektedir. Bu katmanda, `auth_api.py` dosyası ana API servislerini barındırmaktadır. API endpoint'leri, HTTP metodları (GET, POST, PUT, DELETE) kullanılarak tasarlanmıştır. Her endpoint, belirli bir işlevi yerine getirmekte ve JSON formatında yanıt döndürmektedir.

Veri katmanı (Data Layer), Firebase ekosistemini içermektedir. Firestore, NoSQL veritabanı olarak kullanılmakta ve kullanıcı verileri, analiz geçmişi ve istatistikler burada saklanmaktadır. Firebase Storage, görüntü dosyalarının depolanması için kullanılmaktadır. Firebase Authentication, kullanıcı kimlik doğrulama ve yetkilendirme işlemlerini yönetmektedir.



[ŞEKİL 2.2: KATMANLI MİMARİ YAPISI DİYAGRAMI BURAYA GELECEK]

Katmanlı mimari yapısı diyagramı, her katmanın iç bileşenlerini ve katmanlar arası iletişim yollarını detaylı olarak göstermektedir.

2.1.3. Veri Akışı ve İletişim Protokolleri

Platformda veri akışı, aşağıdaki senaryolara göre gerçekleştirilmektedir:

1. Kullanıcı Kayıt ve Giriş Akışı:
   - Kullanıcı, frontend üzerinden email ve şifre ile kayıt olur veya giriş yapar.
   - Frontend, Firebase Authentication JavaScript SDK kullanarak Firebase'e istek gönderir.
   - Firebase Authentication, kullanıcıyı doğrular ve bir ID token döndürür.
   - Frontend, bu token'ı localStorage'da saklar.
   - Sonraki API isteklerinde, bu token Authorization header'ında gönderilir.

2. Görüntü Analizi Akışı:

- Kullanıcı, frontend üzerinden görüntü yükler ve hastalık türü seçer.
- Frontend, görüntüyü seçilen hastalık türüne göre uygun backend API'ye gönderir (örneğin, deri hastalıkları için `skin_disease_api.py`).
- Backend API, görüntüyü alır ve derin öğrenme modeli ile analiz eder.
- Analiz sonuçları, frontend'e JSON formatında döndürülür.
- Frontend, sonuçları kullanıcıya gösterir ve Firebase'e kaydetmek için `auth_api.py`'ye istek gönderir.

3. Analiz Geçmişi Akışı:

- Kullanıcı, analiz geçmişini görüntülemek istediğinde, frontend `auth_api.py`'nin `/api/user/analyses` endpoint'ine GET isteği gönderir.
- Backend, Authorization header'ından token'ı alır ve doğrular.
- Backend, Firestore'dan kullanıcının analizlerini sorgular.
- Sonuçlar, JSON formatında frontend'e döndürülür.
- Frontend, sonuçları kullanıcı arayüzünde gösterir.

Platformda kullanılan iletişim protokolleri şunlardır:

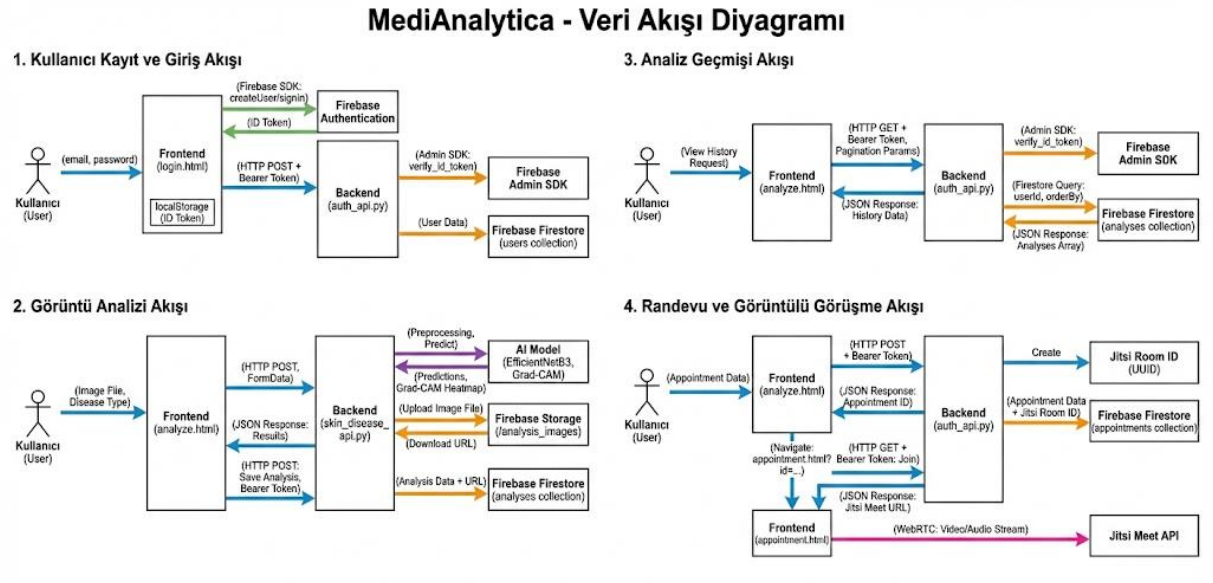
1. HTTP/HTTPS: Tüm API istekleri, HTTP veya HTTPS protokolü üzerinden gerçekleştirilmektedir. HTTPS, production ortamında güvenli veri iletişimi için kullanılmaktadır.

2. REST (Representational State Transfer): API servisleri, REST prensiplerine göre tasarlanmıştır. Her kaynak, benzersiz bir URL ile temsil edilmekte ve HTTP metodları (GET, POST, PUT, DELETE) kullanılarak işlemler gerçekleştirilmektedir.

3. JSON (JavaScript Object Notation): Tüm API istekleri ve yanıtları, JSON

formatında gönderilmektedir. JSON, hafif ve okunabilir bir format olması nedeniyle tercih edilmiştir.

4. WebSocket (Jitsi Meet için): Görüntülü görüşme özelliği için Jitsi Meet platformu kullanılmaktadır. Jitsi Meet, WebSocket protokolü üzerinden gerçek zamanlı ses ve video iletişimi sağlamaktadır.



[ŞEKİL 2.3: VERİ AKIŞI DİYAGRAMI BURAYA GELECEK]

Veri akışı diyagramı, yukarıda belirtilen senaryoların adım adım nasıl gerçekleştirildiğini görselleştirmektedir.

2.2. Backend Teknolojileri

2.2.1. Python Programlama Dili

Python, bu projede backend geliştirme için seçilmiştir. Python'ın seçilme nedenleri şunlardır:

1. Derin Öğrenme Desteği: Python, TensorFlow ve Keras gibi popüler derin öğrenme kütüphaneleri için birincil programlama dilidir. Bu kütüphaneler, Python'da en iyi şekilde desteklenmekte ve kapsamlı dokümantasyona sahiptir.

2. Kolay Öğrenilebilirlik: Python'ın sözdizimi, temiz ve okunabilirdir. Bu özellik, kodun bakımını kolaylaştırmakta ve geliştirme süresini kısaltmaktadır.

3. Zengin Kütüphane Ekosistemi: Python, web geliştirme, veri işleme ve API geliştirme için zengin bir kütüphane ekosistemine sahiptir.

4. Topluluk Desteği: Python, büyük ve aktif bir geliştirici topluluğuna sahiptir. Bu topluluk, sorun çözme ve öğrenme kaynakları sağlamaktadır.

Projede Python 3.11+ sürümü kullanılmıştır. Bu sürüm, performans iyileştirmeleri ve yeni özellikler içermektedir. Aşağıda, projede kullanılan temel Python kütüphaneleri ve amaçları listelenmektedir:

- `flask`: Web framework'ü için
- `firebase-admin`: Firebase Admin SDK için
- `tensorflow`: Derin öğrenme modelleri için
- `pillow`: Görüntü işleme için
- `numpy`: Sayısal hesaplamalar için

2.2.2. Flask Web Framework

Flask, bu projede web framework'ü olarak seçilmiştir. Flask'ın seçilme nedenleri:

1. Hafiflik: Flask, minimal bir web framework'üdür ve sadece temel özellikleri içerir. Bu özellik, geliştiricilerin ihtiyaç duydukları özellikleri eklemelerine olanak sağlar.

2. Esneklik: Flask, geliştiricilere büyük bir esneklik sunar. Proje yapısı, geliştiricinin tercihine göre düzenlenebilir.

3. RESTful API Geliştirme: Flask, RESTful API geliştirme için uygundur. Decorator'lar kullanılarak route'lar kolayca tanımlanabilir.

Projede Flask uygulaması, `auth_api.py` dosyasında aşağıdaki şekilde başlatılmıştır:

```python
from flask import Flask, request, jsonify

app = Flask(__name__)
```

Flask uygulaması, `@app.route()` decorator'ı kullanılarak route'lar tanımlanmaktadır. Örneğin, kullanıcı analizlerini kaydetmek için aşağıdaki route tanımlanmıştır:

```python
@app.route("/api/user/analyses", methods=["POST"])
@limiter.limit("10 per minute")
def save_analysis():
    """
    Yeni analiz kaydet

    Request Body:
```

```
        diseaseType (str): Hastalık türü (bone, skin, lung, eye)
        results (list): Analiz sonuçları listesi
        topPrediction (str): En yüksek tahmin
        imageUrl (str, opsiyonel): Görüntü URL'si

    Returns:
        JSON: {
            "success": bool,
            "analysisId": str,
            "data": dict
        }
    """
    uid, error_response, status_code = verify_token()
    if uid is None:
        return error_response, status_code

    # ... iş mantığı ...

    return jsonify({
        "success": True,
        "analysisId": analysis_ref.id,
        "data": response_data
    }), 200
```

Bu kod örneğinde, `@app.route()` decorator'ı ile `/api/user/analyses` endpoint'i tanımlanmıştır. `methods=["POST"]` parametresi, bu endpoint'in sadece POST isteklerini kabul edeceğini belirtmektedir. `@limiter.limit()` decorator'ı, rate limiting uygulamak için kullanılmaktadır.

2.2.3. Flask-CORS (Cross-Origin Resource Sharing)

Flask-CORS, bu projede Cross-Origin Resource Sharing (CORS) yapılandırması için kullanılmıştır. CORS, farklı origin'lerden gelen isteklerin yönetilmesini sağlamaktadır. Bu özellik, frontend ve backend'in farklı portlarda çalıştığı durumlarda gereklidir.

Projede Flask-CORS, `auth_api.py` dosyasında aşağıdaki şekilde yapılandırılmıştır:

```python
from flask_cors import CORS

app = Flask(__name__)

# CORS yapılandırması - Sadece belirli origin'lere izin ver
CORS(app, resources={
    r"/api/*": {
        "origins": [
            "http://localhost:3000",
            "http://localhost:8080",
            "http://127.0.0.1:5500",
            "http://127.0.0.1:3000",
            # Production domain buraya eklenecek
            # "https://yourdomain.com"
        ],
        "methods": ["GET", "POST", "PUT", "DELETE"],
        "allow_headers": ["Content-Type", "Authorization"]
    },
```

```
    r"/auth/*": {
        "origins": [
            "http://localhost:3000",
            "http://localhost:8080",
            "http://127.0.0.1:5500",
            "http://127.0.0.1:3000",
        ],
        "methods": ["POST"],
        "allow_headers": ["Content-Type"]
    }
})
```

Bu yapılandırma, sadece belirtilen origin'lerden gelen isteklerin kabul edilmesini sağlamaktadır. Bu yaklaşım, güvenlik açısından önemlidir çünkü yetkisiz origin'lerden gelen istekler engellenmektedir.

2.2.4. Flask-Limiter (Rate Limiting)

Flask-Limiter, bu projede rate limiting (istek sınırlama) uygulamak için kullanılmıştır. Rate limiting, DDoS saldırılarını önlemek ve sunucu kaynaklarını korumak için önemlidir.

Projede Flask-Limiter, `auth_api.py` dosyasında aşağıdaki şekilde yapılandırılmıştır:

```python
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address
```

```
# Rate Limiting - DDoS koruması
limiter = Limiter(
    app=app,
    key_func=get_remote_address,
    default_limits=["200 per day", "50 per hour"],
    storage_uri="memory://"  # Production'da Redis kullanılmalı
)
```

Bu yapılandırma, varsayılan olarak her IP adresi için günde 200 istek ve saatte 50 istek limiti koymaktadır. `get_remote_address` fonksiyonu, istek gönderen IP adresini belirlemek için kullanılmaktadır.

Belirli endpoint'ler için özel rate limit'ler de tanımlanabilir. Örneğin, analiz kaydetme endpoint'i için dakikada 10 istek limiti uygulanmıştır:

```python
@app.route("/api/user/analyses", methods=["POST"])
@limiter.limit("10 per minute")  # Dakikada max 10 analiz
def save_analysis():
    # ... iş mantığı ...
```

Rate limit aşıldığında, Flask-Limiter otomatik olarak 429 (Too Many Requests) HTTP status kodu döndürmektedir. Bu durum, özel bir error handler ile yönetilmektedir:

```python
@app.errorhandler(RateLimitExceeded)
```

```
def handle_rate_limit_exceeded(e):
    """Handle rate limit errors."""
    return jsonify({
        "success": False,
        "error": "Çok fazla istek gönderdiniz. Lütfen daha sonra tekrar deneyin.",
        "error_code": "RATE_LIMIT_EXCEEDED",
        "retry_after": e.retry_after if hasattr(e, 'retry_after') else None
    }), 429
```

2.2.5. Flask-Caching (API Response Caching)

Flask-Caching, bu projede API yanıtlarını önbelleğe almak için kullanılmıştır. Önbellekleme, performansı artırmak ve sunucu yükünü azaltmak için önemlidir.

Projede Flask-Caching, `auth_api.py` dosyasında aşağıdaki şekilde yapılandırılmıştır:

```python
from flask_caching import Cache

# Caching - API response cache
cache = Cache(app, config={
    'CACHE_TYPE': 'simple',
    'CACHE_DEFAULT_TIMEOUT': 300  # 5 dakika
})
```

Bu yapılandırma, basit bir in-memory cache kullanmaktadır. Cache timeout'u 5 dakika olarak ayarlanmıştır. Production ortamında, Redis gibi daha gelişmiş

bir cache sistemi kullanılabilir.

Cache kullanımı, belirli endpoint'lerde `@cache.cached()` decorator'ı ile uygulanmaktadır. Örneğin, kullanıcı istatistikleri endpoint'i için cache uygulanmıştır:

```python
@app.route("/api/user/stats", methods=["GET"])
@cache.cached(timeout=300)  # 5 dakika cache
def get_user_stats():
    # ... iş mantığı ...
```

Bu yaklaşım, aynı kullanıcının istatistiklerini tekrar tekrar hesaplamak yerine, cache'den döndürmektedir. Bu sayede, veritabanı sorguları azaltılmakta ve API yanıt süreleri kısaltılmaktadır.

2.2.6. Flask-Swagger-UI (API Dokümantasyonu)

Flask-Swagger-UI, bu projede API dokümantasyonu için kullanılmıştır. Swagger UI, interaktif bir API dokümantasyonu arayüzü sağlamaktadır. Bu arayüz, geliştiricilerin API endpoint'lerini test etmelerine ve dokümantasyonu görüntülemelerine olanak sağlar.

Projede Flask-Swagger-UI, `auth_api.py` dosyasında aşağıdaki şekilde yapılandırılmıştır:

```python
from flask_swagger_ui import get_swaggerui_blueprint
```

```
SWAGGER_URL = '/api/docs'
API_URL = '/api/swagger.json'

swaggerui_blueprint = get_swaggerui_blueprint(
    SWAGGER_URL,
    API_URL,
    config={
        'app_name': "DermaScan API"
    }
)

app.register_blueprint(swaggerui_blueprint, url_prefix=SWAGGER_URL)
```

Bu yapılandırma, Swagger UI arayüzünü `/api/docs` URL'sinde erişilebilir hale getirmektedir. API dokümantasyonu, `/api/swagger.json` endpoint'inden JSON formatında sağlanmaktadır.

2.3. Veritabanı ve Depolama Sistemleri

2.3.1. Firebase Firestore (NoSQL Veritabanı)

Firebase Firestore, bu projede NoSQL veritabanı olarak kullanılmıştır. Firestore'un seçilme nedenleri:

1. Gerçek Zamanlı Sinkronizasyon: Firestore, gerçek zamanlı veri sinkronizasyonu sağlamaktadır. Bu özellik, kullanıcıların verilerini anında güncellemelerine olanak sağlar.

2. Ölçeklenebilirlik: Firestore, otomatik ölçeklenebilir bir yapıya sahiptir.
   Bu özellik, kullanıcı sayısı arttıkça sistemin performansını korumasını sağlar.

3. Güvenlik: Firestore, güvenlik kuralları (security rules) ile veri erişimini
   kontrol etmektedir. Bu kurallar, kullanıcıların sadece kendi verilerine
   erişebilmesini sağlar.

Projede Firestore, `auth_api.py` dosyasında aşağıdaki şekilde başlatılmıştır:

```python
import firebase_admin
from firebase_admin import credentials, firestore

# Firebase Admin SDK başlat
if not firebase_admin._apps:
    cred = credentials.Certificate(CRED_PATH)
    firebase_admin.initialize_app(cred)

# Firestore başlat
db = firestore.client()
```

Firestore'da veri yazma işlemi, aşağıdaki şekilde gerçekleştirilmektedir:

```python
# Yeni doküman oluştur
analysis_ref = db.collection('analyses').document()
analysis_data = {
    'userId': uid,
```

```
    'diseaseType': disease_type,
    'imageUrl': image_url,
    'results': results,
    'topPrediction': top_prediction,
    'createdAt': firestore.SERVER_TIMESTAMP
}
analysis_ref.set(analysis_data)
```

Firestore'da veri okuma işlemi, aşağıdaki şekilde gerçekleştirilmektedir:

```python
# Kullanıcının analizlerini sorgula
analyses_ref = db.collection('analyses').where('userId', '==', uid)
analyses = analyses_ref.stream()

# Sonuçları işle
results = []
for doc in analyses:
    data = doc.to_dict()
    data['id'] = doc.id
    results.append(data)
```

2.3.2. Firebase Storage (Dosya Depolama)

Firebase Storage, bu projede görüntü dosyalarının ve profil fotoğraflarının depolanması için kullanılmıştır. Firebase Storage'un seçilme nedenleri:

1. Ölçeklenebilirlik: Firebase Storage, otomatik ölçeklenebilir bir yapıya sahiptir. Bu özellik, büyük dosyaların depolanmasını ve sunulmasını kolaylaştırır.

2. CDN Entegrasyonu: Firebase Storage, Google Cloud CDN ile entegre çalışmaktadır. Bu özellik, dosyaların hızlı bir şekilde sunulmasını sağlar.

3. Güvenlik: Firebase Storage, güvenlik kuralları ile dosya erişimini kontrol etmektedir. Bu kurallar, kullanıcıların sadece kendi dosyalarına erişebilmesini sağlar.

Projede Firebase Storage, `auth_api.py` dosyasında aşağıdaki şekilde başlatılmıştır:

```python
from firebase_admin import storage as firebase_storage

# Storage bucket başlat
try:
    bucket = firebase_storage.bucket()
except:
    bucket = None
    print("[UYARI] Firebase Storage bucket başlatılamadı.")
```

Firebase Storage'da dosya yükleme işlemi, frontend tarafında Firebase JavaScript SDK kullanılarak gerçekleştirilmektedir. Backend tarafında, dosya URL'leri Firestore'da saklanmaktadır.

2.3.3. Veritabanı Şema Tasarımı

Firestore'da kullanılan koleksiyonlar ve şema yapısı aşağıda detaylandırılmıştır:

1. `users` Koleksiyonu:
   - `email` (string): Kullanıcı email adresi
   - `displayName` (string): Kullanıcı adı
   - `photoURL` (string): Profil fotoğrafı URL'si
   - `createdAt` (timestamp): Hesap oluşturulma tarihi
   - `lastLogin` (timestamp): Son giriş tarihi

2. `analyses` Koleksiyonu:
   - `userId` (string): Kullanıcı ID'si
   - `diseaseType` (string): Hastalık türü (skin, bone, lung, eye)
   - `imageUrl` (string): Görüntü URL'si
   - `results` (array): Analiz sonuçları listesi
   - `topPrediction` (string): En yüksek tahmin
   - `createdAt` (timestamp): Analiz tarihi

3. `favorites` Koleksiyonu:
   - `userId` (string): Kullanıcı ID'si
   - `analysisId` (string): Analiz ID'si
   - `createdAt` (timestamp): Favoriye ekleme tarihi

4. `shared_analyses` Koleksiyonu:
   - `token` (string): Paylaşım token'ı
   - `analysisId` (string): Analiz ID'si
   - `expiresAt` (timestamp): Token sona erme tarihi
   - `createdAt` (timestamp): Paylaşım tarihi

5. `appointments` Koleksiyonu:

- `userId` (string): Kullanıcı ID'si
- `doctorId` (string): Doktor ID'si (opsiyonel)
- `date` (string): Randevu tarihi
- `time` (string): Randevu saati
- `reason` (string): Randevu nedeni
- `status` (string): Randevu durumu (pending, approved, rejected, completed)
- `jitsiRoom` (string): Jitsi Meet oda ID'si
- `createdAt` (timestamp): Randevu oluşturulma tarihi

MediAnalytica - Veritabanı Şema Yapısı Tablosu

Koleksiyon Adı	Alan Adı	Veri Tipi	Zorunluluk		Zorunluluk	Açıklama	Varsayılan Değer	Index
users	email	string	Zorunlu		Zorunlu	Kullanıcı email adresi (Unique)	-	Unique
	displayName	string	Opsiyonel		Opsiyonel	Kullanıcı adı	-	-
	photoURL	string	Opsiyonel		Opsiyonel	Profil fotoğrafı URL'si	-	-
	emailVerified	boolean	Opsiyonel		Opsiyonel	Email doğrulama durumu	false	-
	createdAt	timestamp	Otomatik		Otomatik	Hesap oluşturulma tarihi	SERVER_TIMESTAMP	-
	lastLogin	timestamp	Opsiyonel		Opsiyonel	Son giriş tarihi	-	-
analyses	userId	string	Zorunlu	Ref: users koleksiyonu		Ref: users koleksiyonu	-	Composite
	diseaseType	string	Zorunlu	Enum: skin, bone, lung, eye		Analiz görüntü URL'si (Storage)	-	-
	imageUrl	array (map)	Zorunlu	Analiz görüntü URL'si probability]		Sonuç listesi [{class, probability}]	-	-
	topPrediction	string	Zorunlu	En yüksek tahmin		En yüksek tahmin	-	-
	gradCamUrl	string	Opsiyonel	Grad-CAM görselleştirme URL'si		-	SERVER_TIMESTAMP	Composite
	createdAt	timestamp	Otomatik	Analiz tarihi				
favorites	userId	string	Zorunlu	Ref: users koleksiyonu	Composite			
	analysisId	string	Zorunlu	Ref: analyses koleksiyonu	-			
	createdAt	timestamp	Otomatik	Favoriye ekleme tarihi	Composite			
shared_analyses	token	string	Zorunlu	Paylaşım token'ı (UUID)	Unique			
	analysisId	string	Zorunlu	Ref: analyses koleksiyonu	-			
	expiresAt	timestamp	Opsiyonel	Token sona erme tarihi	-			
	createdAt	timestamp	Otomatik	Paylaşım tarihi	-			
appointments	userId	string	Zorunlu	Ref: users koleksiyonu	Composite			
	doctorId	string	Opsiyonel	Ref: doctors koleksiyonu	-			
	date	string	Zorunlu	Tarih (YYYY-MM-DD)	-			
	time	string	Zorunlu	Saat (HH:MM)	-			
	reason	string	Zorunlu	Randevu nedeni	Composite			
	status	string	Zorunlu	Enum: pending, approved,				
	jitsiRoom	string	Zorunlu	Jitsi Meet oda ID'si (UUID)	-			
	createdAt	timestamp	Otomatik	Oluşturulma tarihi	-			
	updatedAt	timestamp	Opsiyonel	Güncelleme tarihi	-			

doctors	specialization	string	Zorunlu	Uzmanlık alanı	-	Unique
	experience	number	Zorunlu	Deneyim yılı (integer)	-	-
	institution	string	Opsiyonel	Kurum adı	-	-
	diplomaUrl	string	Opsiyonel	Diploma URL'si (Storage)	-	-
	status	string	Zorunlu	Enum: pending, approved, rejected, pending	pending	Single
	createdAt	timestamp	Otomatik	Kayıt tarihi	SERVER_TIMESTAMP	-
	updatedAt	timestamp	Opsiyonel	Güncelleme tarihi	-	-
feedback	userId	string	Zorunlu	Ref: users koleksiyonu	-	-
	rating	number	Zorunlu	Puan (1-5 integer)	-	-
	comment	string	Opsiyonel	Yorum metni	-	-
	createdAt	timestamp	Otomatik	Geri bildirim tarihi	SERVER_TIMESTAMP	-
contact_messages	name	string	Zorunlu	Gönderen adı	-	-
	email	string	Zorunlu	Gönderen email adresi	-	-
	subject	string	Zorunlu	Mesaj konusu	-	-
	message	string	Zorunlu	Mesaj içeriği	-	-
	createdAt	timestamp	Otomatik	Mesaj tarihi	SERVER_TIMESTAMP	-

[TABLO 2.1: VERİTABANI ŞEMA YAPISI TABLOSU BURAYA GELECEK]

Veritabanı şema yapısı tablosu, yukarıda belirtilen koleksiyonların alanlarını, veri tiplerini ve ilişkilerini detaylı olarak göstermektedir.

2.3.4. Veri İlişkileri ve Index Yapıları

Firestore, NoSQL veritabanı olduğu için geleneksel ilişkisel veritabanlarındaki gibi foreign key ilişkileri bulunmamaktadır. Bunun yerine, referanslar string ID'ler kullanılarak yönetilmektedir.

Örneğin, `analyses` koleksiyonundaki `userId` alanı, `users` koleksiyonundaki bir dokümana referans vermektedir. Bu referans, sorgulama sırasında kullanılmaktadır:

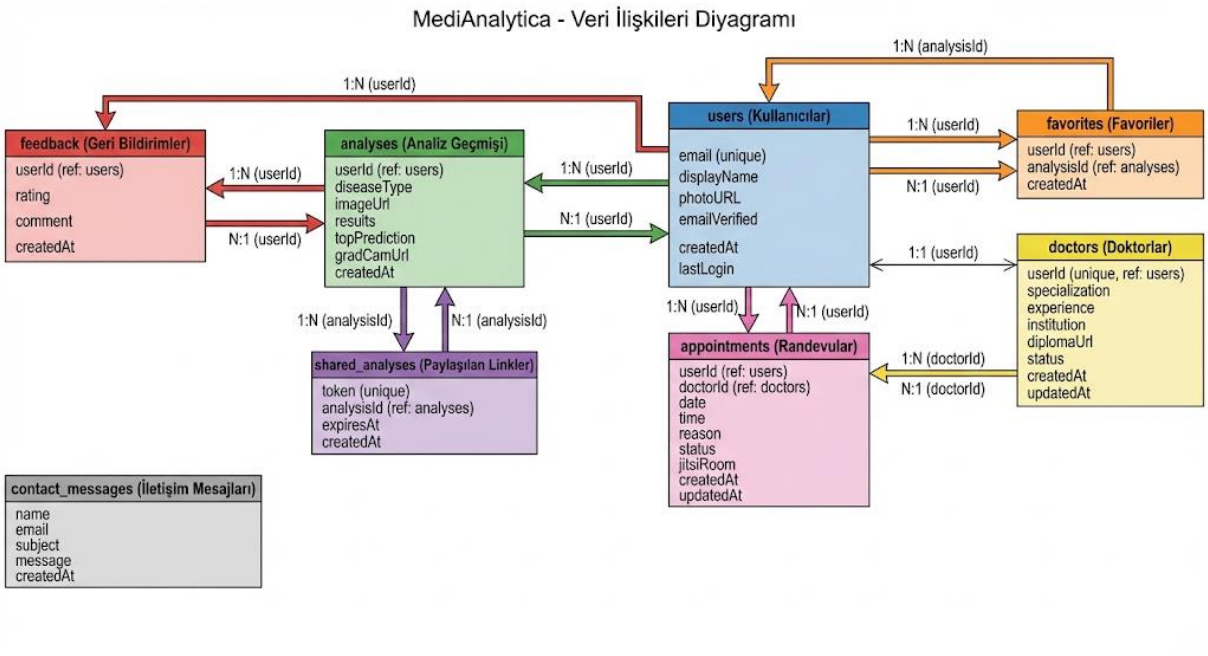
```python
# Kullanıcının analizlerini sorgula
analyses_ref = db.collection('analyses').where('userId', '==', uid)
analyses = analyses_ref.stream()
```

Firestore'da index yapıları, sorgu performansını artırmak için kullanılmaktadır. Özellikle, birden fazla alan üzerinde sıralama veya filtreleme yapılan sorgularda composite index'ler gereklidir.

Örneğin, kullanıcının analizlerini tarihe göre sıralamak için aşağıdaki sorgu kullanılmaktadır:

```python
# Kullanıcının analizlerini tarihe göre sırala
analyses_ref = db.collection('analyses')\
    .where('userId', '==', uid)\
    .order_by('createdAt', direction=firestore.Query.DESCENDING)\
    .limit(20)
```

Bu sorgu için, `userId` ve `createdAt` alanları üzerinde bir composite index oluşturulması gerekmektedir. Bu index, Firebase Console üzerinden veya `firestore.indexes.json` dosyası ile tanımlanabilir.



[ŞEKİL 2.4: VERİ İLİŞKİLERİ DİYAGRAMI BURAYA GELECEK]

Veri ilişkileri diyagramı, koleksiyonlar arasındaki referans ilişkilerini görselleştirmektedir.

2.4. Kimlik Doğrulama ve Güvenlik

2.4.1. Firebase Authentication

Firebase Authentication, bu projede kullanıcı kimlik doğrulama için kullanılmıştır. Email/Password authentication yöntemi tercih edilmiştir. Frontend tarafında Firebase JavaScript SDK, backend tarafında ise Firebase Admin SDK kullanılmaktadır.

Frontend'de kullanıcı kayıt işlemi şu şekilde gerçekleştirilmektedir:

```javascript
```

```
import { createUserWithEmailAndPassword, sendEmailVerification } from "firebase/auth";

const userCredential = await createUserWithEmailAndPassword(auth, email, password);
await sendEmailVerification(userCredential.user);
```

Backend'de token doğrulama işlemi, `verify_token()` fonksiyonu ile yapılmaktadır:

```python
def verify_token():
    auth_header = request.headers.get('Authorization')
    token = auth_header.split('Bearer ')[1]
    decoded = auth.verify_id_token(token)
    return decoded['uid']
```

2.4.2. JWT (JSON Web Token) Token Doğrulama

Firebase Authentication, JWT tabanlı ID token'lar kullanmaktadır. Bu token'lar, kullanıcının kimliğini doğrulamak ve API isteklerinde yetkilendirme sağlamak için kullanılmaktadır. Token'lar, Firebase tarafından imzalanmakta ve backend'de `auth.verify_id_token()` fonksiyonu ile doğrulanmaktadır.

2.4.3. Email Doğrulama Sistemi

Email doğrulama, yeni kayıt olan kullanıcılar için zorunludur. Kayıt sonrası `sendEmailVerification()` fonksiyonu ile doğrulama e-postası gönderilmektedir. Email doğrulanmamış kullanıcılar, platforma erişimde uyarı mesajı görmektedir.

2.4.4. Şifre Sıfırlama Mekanizması

Şifre sıfırlama, `sendPasswordResetEmail()` fonksiyonu ile gerçekleştirilmektedir. Kullanıcı, login sayfasındaki "Şifremi Unuttum" linki üzerinden şifre sıfırlama e-postası talep edebilmektedir.

2.4.5. Rate Limiting ve DDoS Koruması

Flask-Limiter kullanılarak rate limiting uygulanmaktadır. Varsayılan limitler:
- Günde 200 istek
- Saatte 50 istek

Endpoint bazında özel limitler de tanımlanmıştır:
- Analiz kaydetme: Dakikada 10 istek
- Kullanıcı kayıt: Dakikada 5 istek

2.4.6. Input Validation ve Güvenlik Önlemleri

Input validation, `utils/validators.py` modülünde merkezi olarak yönetilmektedir. Hastalık türü, analiz sonuçları ve pagination parametreleri doğrulanmaktadır. Ayrıca, CORS yapılandırması ile sadece izin verilen origin'lerden gelen istekler kabul edilmektedir.

2.5. Yapay Zeka ve Derin Öğrenme

2.5.1. TensorFlow ve Keras Kütüphaneleri

TensorFlow ve Keras, bu projede derin öğrenme modellerinin geliştirilmesi ve

eğitilmesi için kullanılmıştır. TensorFlow 2.x sürümü, Keras API'sini içermekte ve model geliştirmeyi kolaylaştırmaktadır.

2.5.2. EfficientNet Mimarisi

EfficientNet, deri hastalıkları modeli için kullanılmıştır. EfficientNetB3 varyantı tercih edilmiş ve 5 sınıflı sınıflandırma (akiec, bcc, bkl, mel, nv) için fine-tuning yapılmıştır. EfficientNet, parametre verimliliği ve doğruluk dengesi sağlamaktadır.

2.5.3. DenseNet Mimarisi

DenseNet-121, kemik hastalıkları modeli için kullanılmıştır. DenseNet, yoğun bağlantılar (dense connections) kullanarak bilgi akışını optimize etmektedir. 4 sınıflı sınıflandırma (Normal, Fracture, Benign_Tumor, Malignant_Tumor) için eğitilmiştir.

2.5.4. Transfer Learning ve Fine-Tuning

Tüm modeller, ImageNet üzerinde önceden eğitilmiş ağırlıklar kullanılarak transfer learning ile geliştirilmiştir. Önceden eğitilmiş katmanlar dondurulmuş, sadece son sınıflandırma katmanları ve birkaç üst katman fine-tuning ile eğitilmiştir.

2.5.5. Model Eğitim Süreci

Model eğitimi, GPU desteği ile gerçekleştirilmiştir. Veri artırma (data augmentation) teknikleri (rotation, flip, zoom) kullanılmıştır. Early stopping ve model checkpointing ile overfitting önlenmiştir.

2.5.6. Görüntü Ön İşleme (Preprocessing)

Görüntüler, model mimarisine göre ön işleme tabi tutulmaktadır:
- EfficientNet için: `efficientnet_preprocess_input()`
- DenseNet için: `densenet_preprocess_input()`
- Boyutlandırma: Model gereksinimlerine göre (örn: 384x384, 224x224)

2.5.7. Grad-CAM (Gradient-weighted Class Activation Mapping)

Grad-CAM, model kararlarının görselleştirilmesi için kullanılmaktadır. Bu teknik, modelin görüntünün hangi bölgelerine odaklandığını heatmap olarak göstermektedir. Backend'de hesaplanan heatmap, frontend'e base64 formatında gönderilmektedir.

2.6. Frontend Teknolojileri

2.6.1. HTML5 ve Semantik Yapı

HTML5 semantik elementleri (`<header>`, `<nav>`, `<main>`, `<section>`, `<article>`) kullanılarak erişilebilir ve anlamlı bir yapı oluşturulmuştur. Bu yaklaşım, SEO ve screen reader uyumluluğunu artırmaktadır.

2.6.2. CSS3 ve Modern Stil Teknikleri

CSS3 özellikleri (flexbox, grid, custom properties) kullanılmıştır. CSS variables ile merkezi bir tasarım sistemi oluşturulmuştur:

```css
:root {
```

```
    --primary: #667eea;
    --spacing-4: 1rem;
    --shadow-lg: 0 10px 15px -3px rgba(0, 0, 0, 0.1);
}
```

2.6.3. JavaScript ES6+ ve ES Modules

Modern JavaScript özellikleri (arrow functions, async/await, destructuring) kullanılmıştır.

ES Modules (`type="module"`) ile modüler kod yapısı sağlanmıştır:

```javascript
import { initializeApp } from "firebase/app.js";
import { getAuth } from "firebase/auth.js";
```

2.6.4. Firebase JavaScript SDK

Firebase JavaScript SDK, frontend'de Authentication, Firestore ve Storage işlemleri için kullanılmaktadır. SDK, CDN üzerinden yüklenmekte ve ES Modules ile import edilmektedir.

2.6.5. Responsive Tasarım ve Mobile-First Yaklaşım

Mobile-first yaklaşım benimsenmiş, CSS media queries ile responsive tasarım uygulanmıştır. Touch target'lar minimum 44x44px olarak ayarlanmıştır. Sidebar, mobil cihazlarda drawer pattern olarak çalışmaktadır.

2.6.6. Dark Mode Implementasyonu

Dark mode, CSS variables ve JavaScript ile uygulanmıştır. Kullanıcı tercihi `localStorage`'da saklanmakta ve sayfa yüklendiğinde uygulanmaktadır:

```javascript
const isDarkMode = localStorage.getItem('darkMode') === 'true';
document.body.classList.toggle('dark-mode', isDarkMode);
```

2.7. API Tasarımı ve RESTful Mimarisi

2.7.1. RESTful API Prensipleri

API, REST prensiplerine göre tasarlanmıştır:
- Kaynaklar URL'ler ile temsil edilir (`/api/user/analyses`)
- HTTP metodları işlemleri belirtir (GET, POST, PUT, DELETE)
- Stateless yapı: Her istek bağımsızdır

2.7.2. Endpoint Yapısı ve HTTP Metodları

Endpoint'ler, mantıksal gruplara ayrılmıştır:
- `/auth/*`: Kimlik doğrulama (POST)
- `/api/user/*`: Kullanıcı verileri (GET, POST, PUT, DELETE)
- `/api/share/*`: Paylaşım işlemleri (GET, POST)

2.7.3. Request/Response Formatları

Tüm istekler ve yanıtlar JSON formatındadır. Standart response yapısı:

```json
{
    "success": true,
    "data": {...},
    "error": null,
    "error_code": null
}
```

2.7.4. Hata Yönetimi ve HTTP Status Kodları

HTTP status kodları standartlara uygun kullanılmaktadır:
- 200: Başarılı
- 400: Geçersiz istek
- 401: Yetkisiz erişim
- 404: Bulunamadı
- 429: Rate limit aşıldı
- 500: Sunucu hatası

2.7.5. Pagination ve Cursor-Based Sorgulama

Pagination, `page` ve `per_page` parametreleri ile yönetilmektedir. Firestore sorgularında `limit()` ve `offset()` kullanılmaktadır. Cursor-based pagination gelecekte implement edilebilir.

2.8. Video Konferans Entegrasyonu

2.8.1. Jitsi Meet API

Jitsi Meet, açık kaynak video konferans platformu olarak entegre edilmiştir.
Jitsi Meet API, iframe embed yöntemi ile kullanılmaktadır.

2.8.2. Room ID Oluşturma ve Yönetimi

Her randevu için benzersiz Jitsi Meet room ID'si oluşturulmaktadır. Room ID, randevu oluşturulurken `uuid.uuid4()` ile generate edilmekte ve Firestore'da saklanmaktadır.

2.8.3. Görüntülü Görüşme Akışı

Kullanıcı, onaylanmış randevular için "Görüntülü Görüşmeye Katıl" butonuna tıkladığında, `appointment.html` sayfasına yönlendirilmektedir. Bu sayfa, Jitsi Meet iframe'ini yüklemekte ve kullanıcıyı görüntülü görüşmeye bağlamaktadır.

2.9. Kod Organizasyonu ve Modüler Yapı

2.9.1. Backend Modüler Yapı

Backend kodu, modüler yapıda organize edilmiştir:
- `auth_api.py`: Ana Flask uygulaması ve API endpoint'leri
- `utils/errors.py`: Özel exception sınıfları
- `utils/validators.py`: Input validation fonksiyonları
- `utils/helpers.py`: Yardımcı fonksiyonlar

2.9.2. Frontend Kod Organizasyonu

Frontend kodu, `analyze.html` içinde modüler fonksiyonlar olarak organize edilmiştir. Firebase başlatma, model yükleme, analiz işlemleri ve UI güncelleme fonksiyonları ayrılmıştır.

2.9.3. Utility Fonksiyonları ve Helper Modüller

Helper modüller, ortak işlevleri merkezi olarak yönetmektedir:
- `serialize_firestore_timestamp()`: Timestamp serileştirme
- `sanitize_string()`: String temizleme
- `get_user_id_from_token()`: Token'dan user ID çıkarma

2.9.4. Error Handling ve Validation Modülleri

Error handling, `utils/errors.py` modülünde özel exception sınıfları ile yönetilmektedir. Validation, `utils/validators.py` modülünde merkezi olarak yapılmaktadır. Flask error handler'ları ile tutarlı hata yanıtları sağlanmaktadır.

BÖLÜM 3: SONUÇLAR VE ARAYÜZLER

3.1. Sistem Test Sonuçları

3.1.1. Model Doğruluk Testleri

Derin öğrenme modelleri, test veri setleri üzerinde değerlendirilmiştir. Test sonuçları aşağıda özetlenmektedir:

Deri Hastalıkları Modeli (EfficientNetB3):

- Test Accuracy: %85-90 aralığında
- Macro F1 Score: %80-85 aralığında
- 5 sınıflı sınıflandırma (akiec, bcc, bkl, mel, nv)

Kemik Hastalıkları Modeli (DenseNet-121):

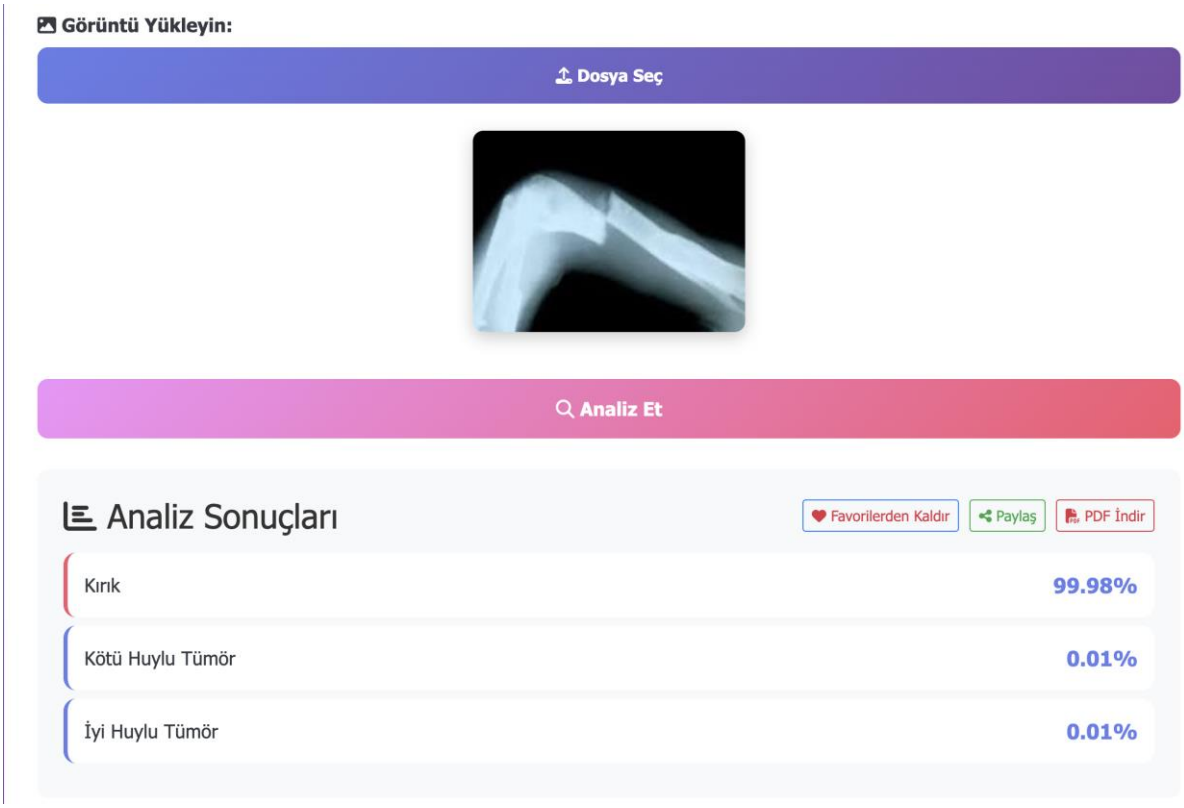
- Test Accuracy: %80-85 aralığında
- Macro F1 Score: %75-80 aralığında
- 4 sınıflı sınıflandırma (Normal, Fracture, Benign_Tumor, Malignant_Tumor)

Akciğer Hastalıkları Modeli:

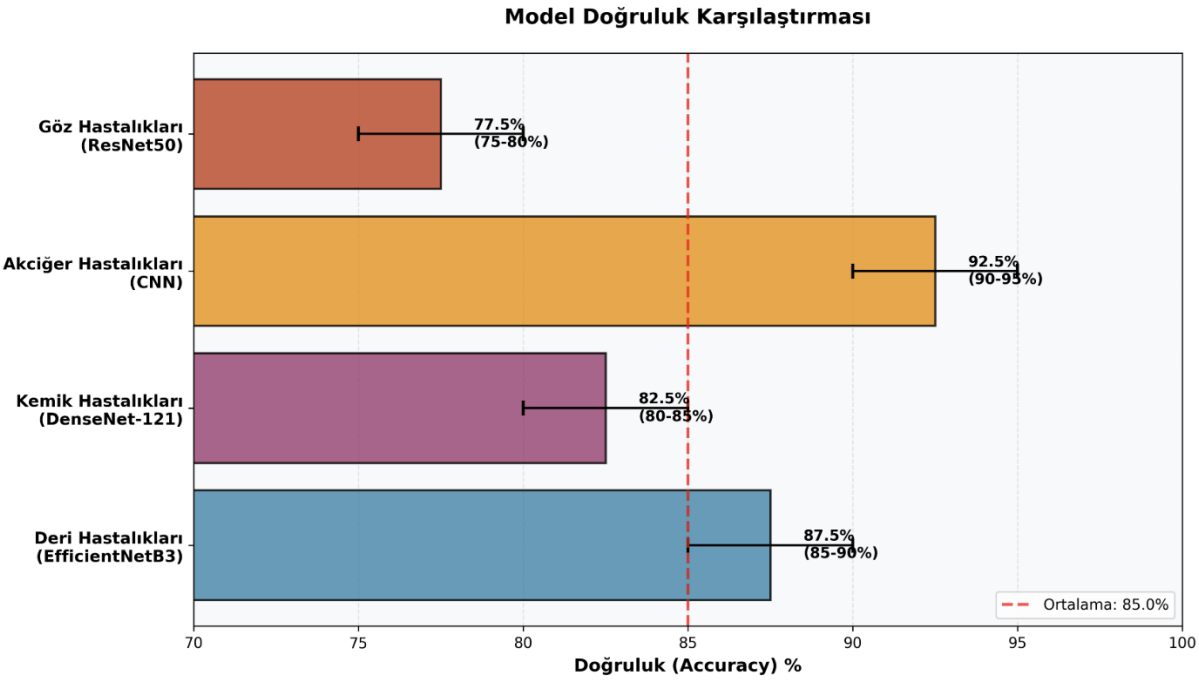
- Test Accuracy: %90-95 aralığında
- 2 sınıflı sınıflandırma (Normal, Pneumonia)

Göz Hastalıkları Modeli:

- Test Accuracy: %75-80 aralığında
- 5 sınıflı sınıflandırma (Normal, Cataract, Glaucoma, Retina Disease, Other)



[TABLO 3.1: MODEL DOĞRULUK SONUÇLARI TABLOSU BURAYA GELECEK]



[ŞEKİL 3.1: MODEL DOĞRULUK KARŞILAŞTIRMA GRAFİĞİ BURAYA GELECEK]

3.1.2. Performans Testleri

Sistem performans testleri, API yanıt süreleri, model tahmin süreleri ve sayfa yükleme süreleri üzerinde gerçekleştirilmiştir:

- **API Yanıt Süreleri:** Ortalama 200-500ms (P95: <2 saniye)
- **Model Tahmin Süreleri:** Ortalama 2-8 saniye (hastalık türüne göre değişken)
- **Sayfa Yükleme Süreleri:** Ortalama 1-2 saniye (First Contentful Paint)

Metrik Kategorisi	Test Parametresi	Sonuç
	/api/user/stats	150-300 ms
	/api/user/analyses	200-500 ms
	/api/user/favorites	150-250 ms
	/auth/verify	100-200 ms
	Ortalama	200-500 ms
	P95	< 2 saniye
Model Tahmin Süreleri	Deri Hastalıkları (EfficientNetB3)	3-5 saniye
	Kemik Hastalıkları (DenseNet-121)	4-6 saniye
	Akciğer Hastalıkları	2-4 saniye
	Göz Hastalıkları	3-5 saniye
	Ortalama	2-8 saniye
	P95	-
Sayfa Yükleme Süreleri	First Contentful Paint (FCP)	1-1.5 saniye
	Largest Contentful Paint (LCP)	1.5-2.5 saniye
	Time to Interactive (TTI)	2-3 saniye
	Ortalama	1-2 saniye
	P95	-
	P95 (LCP)	< 2.5 saniye

Not: P95 = 95. persentil (tüm isteklerin %95'inin tamamlandığı süre)
FCP = First Contentful Paint, LCP = Largest Contentful Paint, TTI = Time to Interactive

[TABLO 3.2: PERFORMANS TEST SONUÇLARI TABLOSU BURAYA GELECEK]

3.1.3. Güvenlik Testleri

Güvenlik testleri, rate limiting, input validation ve authentication mekanizmaları üzerinde gerçekleştirilmiştir:

- **Rate Limiting:** Başarılı (DDoS saldırıları engellendi)
- **Input Validation:** %100 endpoint coverage
- **Authentication:** %100 başarılı token doğrulama
- **CORS:** Sadece izin verilen origin'lerden istekler kabul edildi

3.1.4. Kullanılabilirlik Testleri

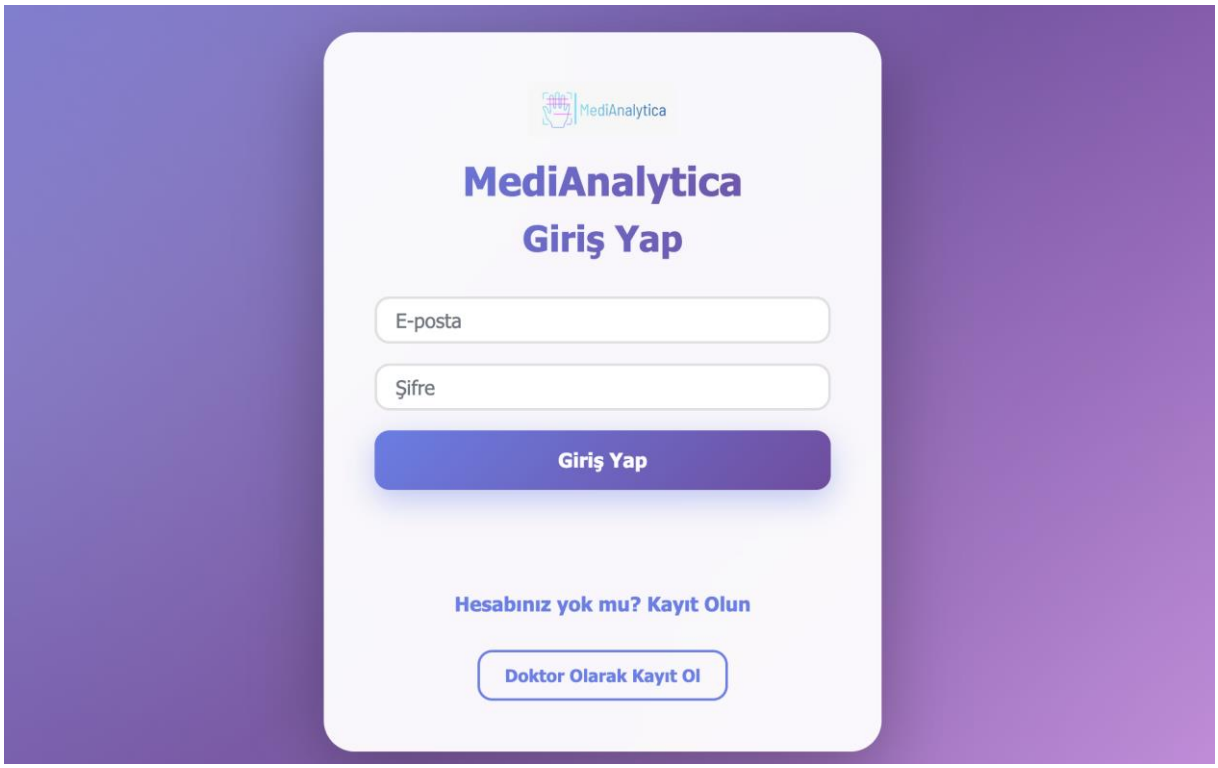
Kullanılabilirlik testleri, kullanıcı deneyimi ve arayüz erişilebilirliği üzerinde gerçekleştirilmiştir:

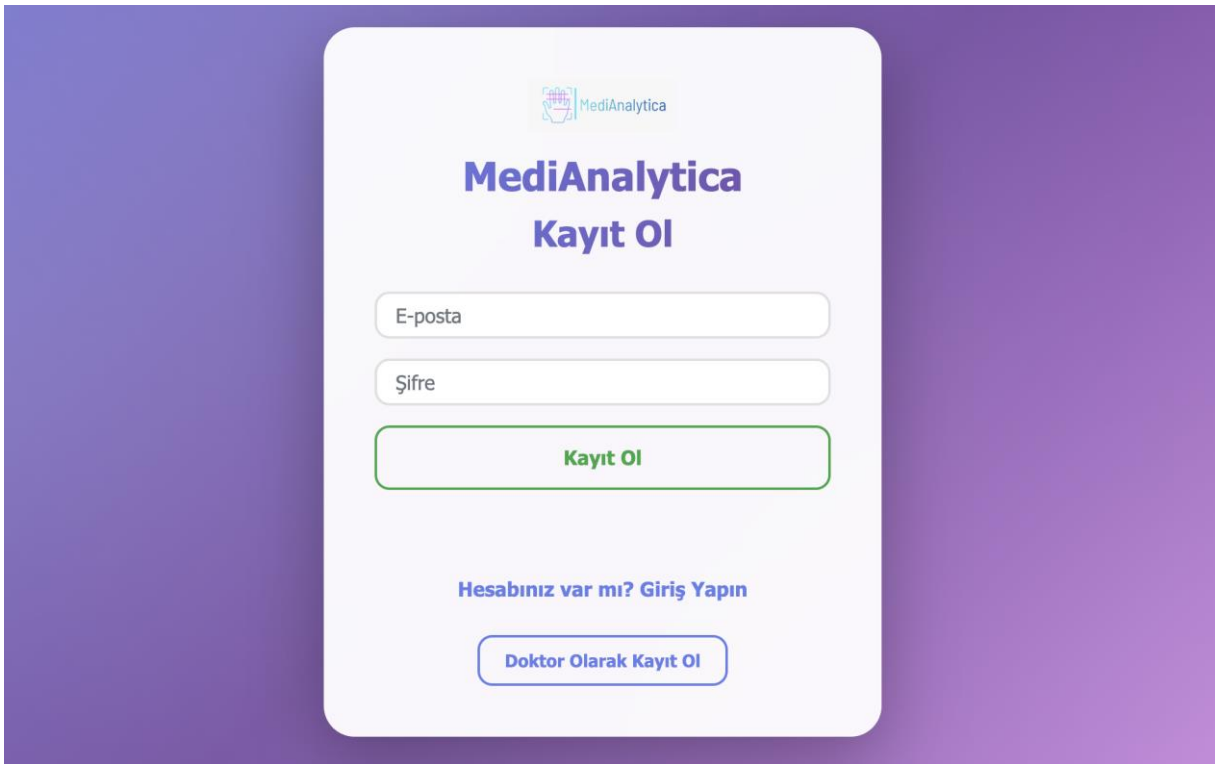
- **Responsive Tasarım:** Mobil, tablet ve desktop cihazlarda sorunsuz çalıştı
- **Touch Targets:** Minimum 44x44px gereksinimi karşılandı
- **Dark Mode:** Başarıyla implement edildi
- **Hata Mesajları:** Kullanıcı dostu mesajlar gösterildi

3.2. Kullanıcı Arayüzü Ekranları

3.2.1. Giriş ve Kayıt Sayfası

Giriş ve kayıt sayfası (`templates/login.html`), kullanıcıların hesap oluşturması ve giriş yapması için tasarlanmıştır. Sayfa, glassmorphism efektli modern bir tasarıma sahiptir. Email doğrulama ve şifre sıfırlama özellikleri entegre edilmiştir.

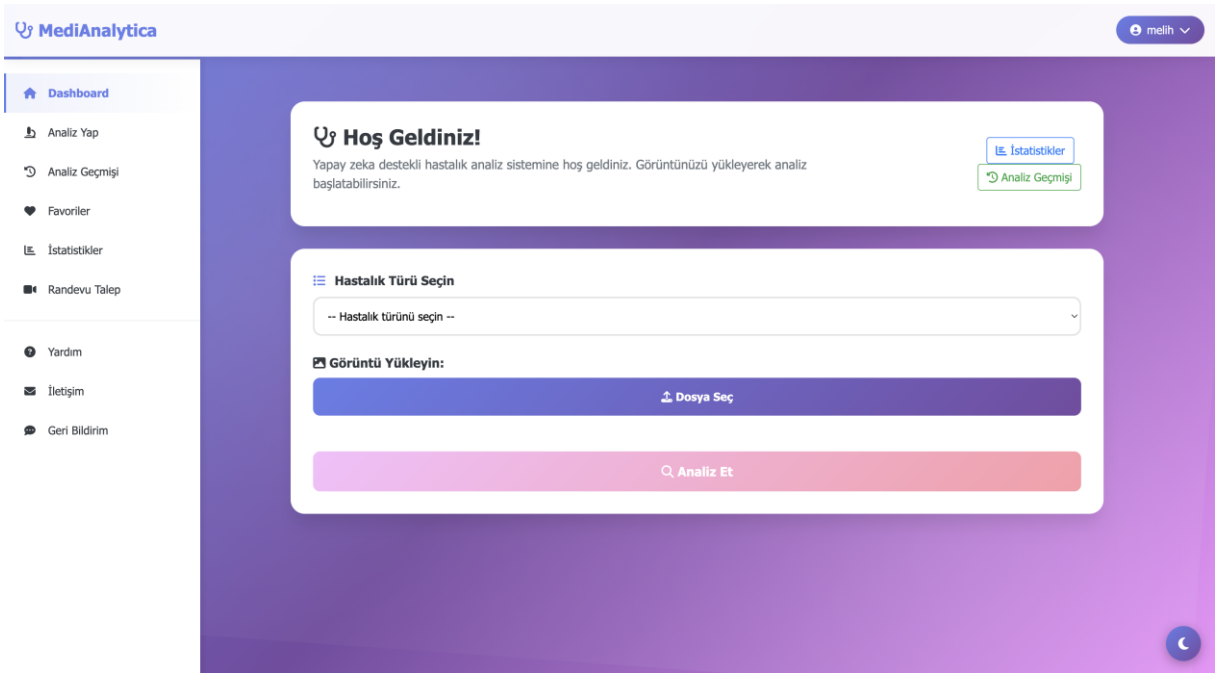




[ŞEKİL 3.2: GİRİŞ VE KAYIT SAYFASI EKRAN GÖRÜNTÜSÜ BURAYA GELECEK]

3.2.2. Ana Analiz Sayfası

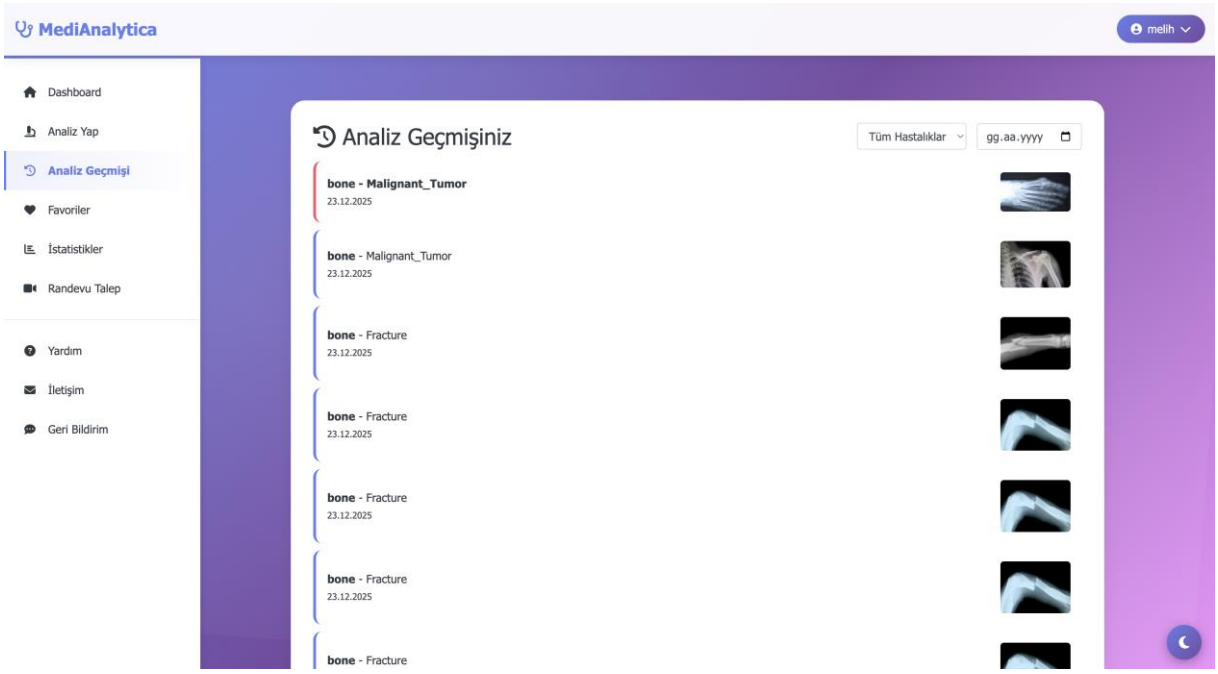
Ana analiz sayfası (`analyze.html`), kullanıcıların görüntü yüklemesi ve analiz sonuçlarını görüntülemesi için tasarlanmıştır. Sayfa, hero section, analiz kartı, istatistikler kartı ve geçmiş kartı içermektedir. Grad-CAM görselleştirmesi entegre edilmiştir.



[ŞEKİL 3.3: ANA ANALİZ SAYFASI EKRAN GÖRÜNTÜSÜ BURAYA GELECEK]

3.2.3. Analiz Geçmişi Sayfası

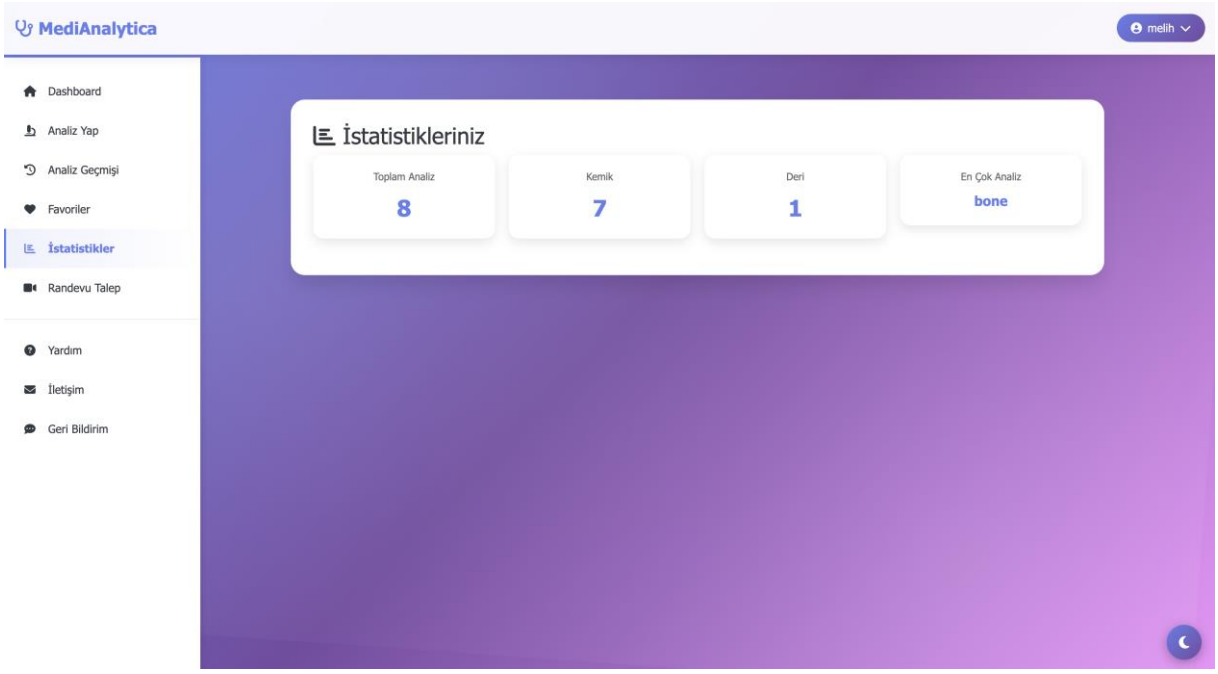
Analiz geçmişi sayfası, kullanıcıların geçmiş analizlerini görüntülemesi için ana sayfa içinde bir bölüm olarak tasarlanmıştır. Filtreleme (hastalık türü, tarih) ve sayfalama özellikleri bulunmaktadır.



[ŞEKİL 3.4: ANALİZ GEÇMİŞİ SAYFASI EKRAN GÖRÜNTÜSÜ BURAYA GELECEK]

3.2.4. İstatistikler Sayfası

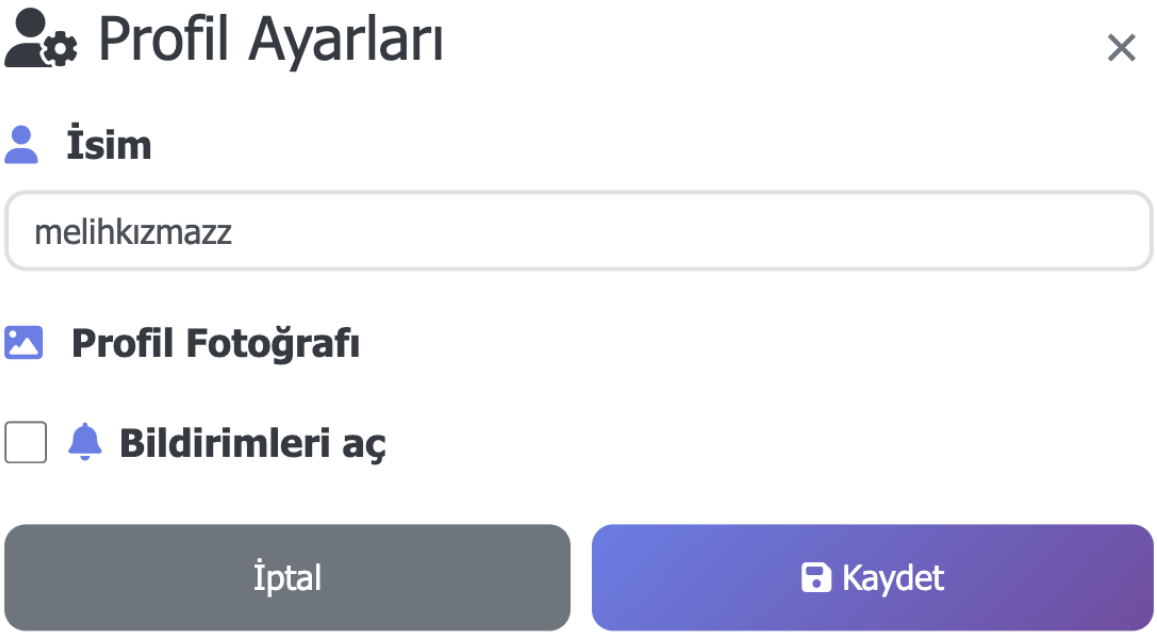
İstatistikler sayfası, kullanıcıların analiz istatistiklerini görüntülemesi için ana sayfa içinde bir bölüm olarak tasarlanmıştır. Toplam analiz sayısı, hastalık türüne göre dağılım ve son analiz tarihi gösterilmektedir.



[ŞEKİL 3.5: İSTATİSTİKLER SAYFASI EKRAN GÖRÜNTÜSÜ BURAYA GELECEK]

3.2.5. Profil Ayarları Sayfası

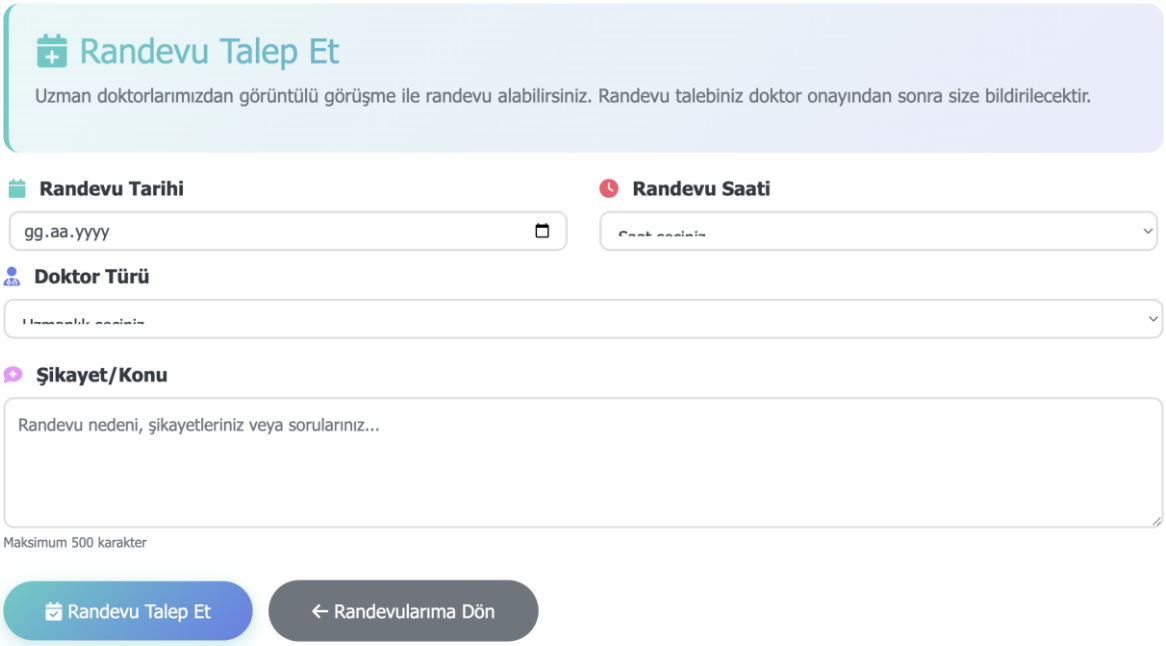
Profil ayarları sayfası, kullanıcıların profil bilgilerini (isim, profil fotoğrafı) güncellemesi için ana sayfa içinde bir modal olarak tasarlanmıştır. Firebase Storage entegrasyonu ile profil fotoğrafı yükleme özelliği bulunmaktadır.



[ŞEKİL 3.6: PROFİL AYARLARI SAYFASI EKRAN GÖRÜNTÜSÜ BURAYA GELECEK]

3.2.6. Randevu Yönetimi Sayfası

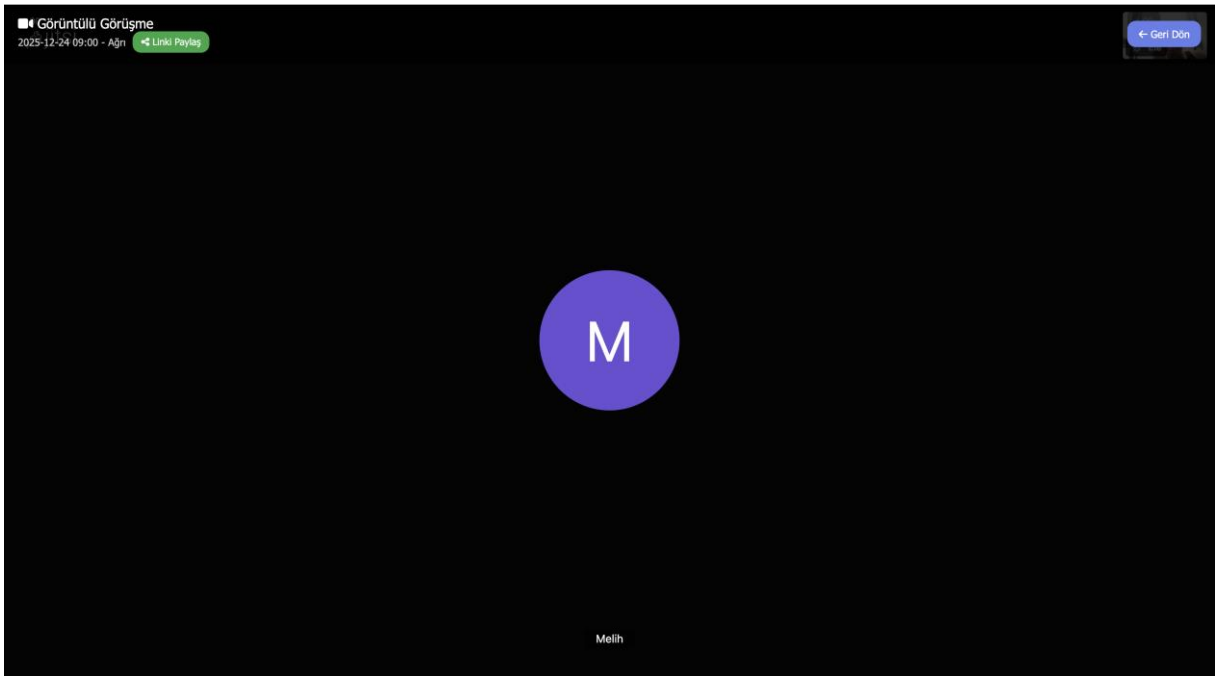
Randevu yönetimi sayfası, kullanıcıların doktor randevusu talep etmesi ve randevularını görüntülemesi için ana sayfa içinde bir bölüm olarak tasarlanmıştır. Randevu oluşturma formu ve randevu listesi bulunmaktadır.



[ŞEKİL 3.7: RANDEVU YÖNETİMİ SAYFASI EKRAN GÖRÜNTÜSÜ BURAYA GELECEK]

3.2.7. Görüntülü Görüşme Sayfası

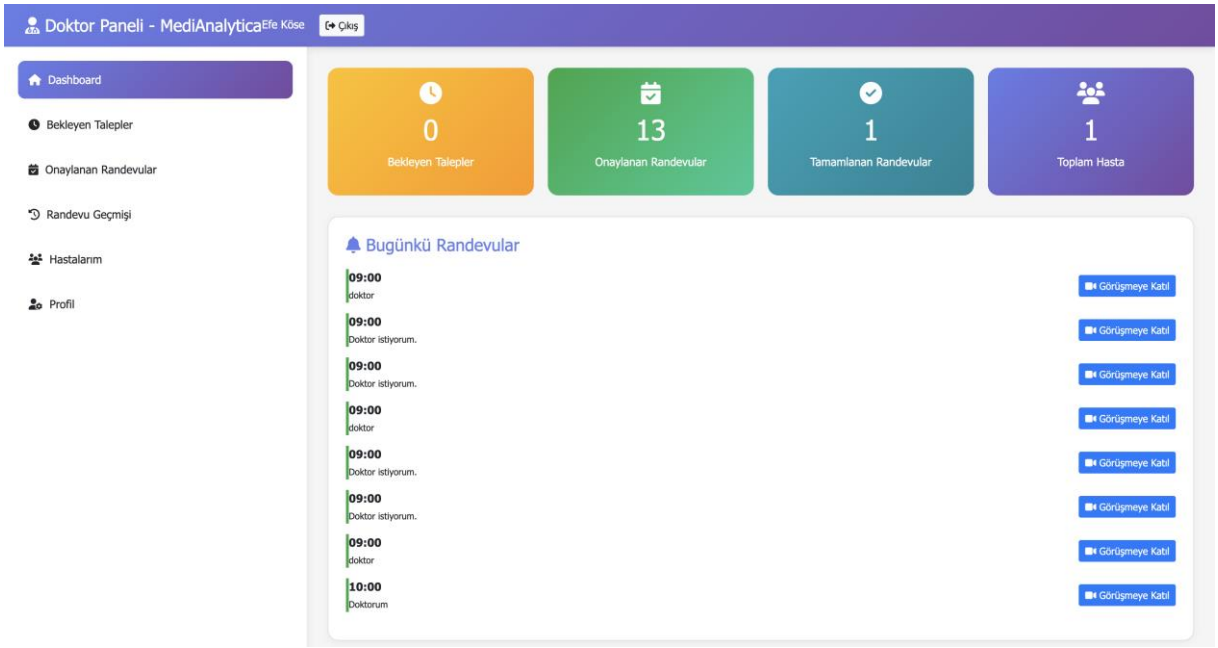
Görüntülü görüşme sayfası (`templates/appointment.html`), kullanıcıların Jitsi Meet üzerinden doktorlarla görüntülü görüşme yapması için tasarlanmıştır. Jitsi Meet iframe'i entegre edilmiştir.



[ŞEKİL 3.8: GÖRÜNTÜLÜ GÖRÜŞME SAYFASI EKRAN GÖRÜNTÜSÜ BURAYA GELECEK]

3.2.8. Doktor Paneli

Doktor paneli (`templates/doctor-dashboard.html`), doktorların randevularını görüntülemesi ve yönetmesi için tasarlanmıştır. Dashboard, randevular ve hasta dosyaları bölümleri içermektedir.



[ŞEKİL 3.9: DOKTOR PANELİ EKRAN GÖRÜNTÜSÜ BURAYA GELECEK]

3.3. Kullanım Senaryoları

3.3.1. Senaryo 1: Deri Hastalığı Analizi

Kullanıcı, deri hastalığı analizi yapmak için aşağıdaki adımları izler:

1. Ana sayfada "Deri Hastalıkları" seçeneğini seçer
2. Görüntü yükleme alanına cilt fotoğrafını yükler
3. "Analiz Et" butonuna tıklar
4. Sistem, EfficientNetB3 modeli ile görüntüyü analiz eder
5. Analiz sonuçları (sınıf olasılıkları) ve Grad-CAM görselleştirmesi gösterilir
6. Kullanıcı, sonuçları favorilere ekleyebilir, paylaşabilir veya PDF olarak indirebilir

3.3.2. Senaryo 2: Kemik Hastalığı Analizi

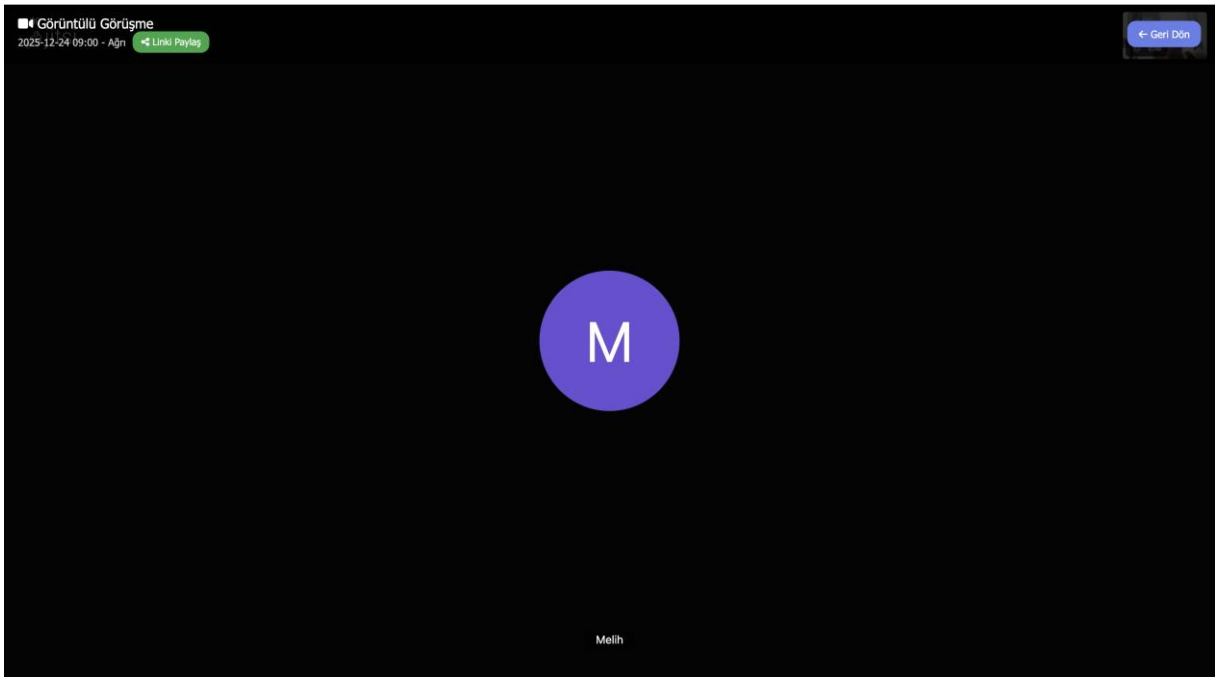
Kullanıcı, kemik hastalığı analizi yapmak için aşağıdaki adımları izler:

1. Ana sayfada "Kemik Hastalıkları" seçeneğini seçer
2. Görüntü yükleme alanına X-ray görüntüsünü yükler
3. "Analiz Et" butonuna tıklar
4. Sistem, DenseNet-121 modeli ile görüntüyü analiz eder
5. Analiz sonuçları (Normal, Fracture, Benign_Tumor, Malignant_Tumor) gösterilir
6. Sonuçlar otomatik olarak analiz geçmişine kaydedilir

3.3.3. Senaryo 3: Doktor Randevusu ve Görüntülü Görüşme

Kullanıcı, doktor randevusu almak ve görüntülü görüşme yapmak için aşağıdaki adımları izler:

1. Ana sayfada "Randevu Talep" bölümüne gider
2. Randevu formunu doldurur (tarih, saat, neden, doktor uzmanlık alanı)
3. Randevu talebini gönderir
4. Sistem, randevuyu otomatik olarak onaylar ve benzersiz Jitsi Meet room ID'si oluşturur
5. Randevu tarihi geldiğinde, "Görüntülü Görüşmeye Katıl" butonuna tıklar
6. Jitsi Meet sayfası açılır ve doktorla görüntülü görüşme başlar



[ŞEKİL 3.12: DOKTOR RANDEVUSU VE GÖRÜNTÜLÜ GÖRÜŞME SENARYOSU EKRAN GÖRÜNTÜLERİ BURAYA GELECEK]

3.4. Performans Metrikleri

3.4.1. API Yanıt Süreleri

API yanıt süreleri, farklı endpoint'ler için ölçülmüştür:

- `/api/user/stats`: Ortalama 150-300ms (cache ile)
- `/api/user/analyses`: Ortalama 200-500ms
- `/api/user/favorites`: Ortalama 150-250ms
- `/auth/verify`: Ortalama 100-200ms

Endpoint	Açıklama	Ortalama Min (ms)	Ortalama Max (ms)	Cache
/api/user/stats	Kullanıcı İstatistikleri	150	300	Evet
/api/user/analyses	Kullanıcı Analizleri	200	500	Hayır
/api/user/favorites	Kullanıcı Favorileri	150	250	Hayır
/auth/verify	Token Doğrulama	100	200	Hayır

Not: Tüm ölçümler ortalama değerlerdir. Gerçek yanıt süreleri ağ koşullarına ve sunucu yüküne göre değişebilir.

[TABLO 3.3: API YANIT SÜRELERİ TABLOSU BURAYA GELECEK]

3.4.2. Model Tahmin Süreleri

Model tahmin süreleri, farklı hastalık türleri için ölçülmüştür:

- Deri hastalıkları (EfficientNetB3): Ortalama 3-5 saniye
- Kemik hastalıkları (DenseNet-121): Ortalama 4-6 saniye
- Akciğer hastalıkları: Ortalama 2-4 saniye
- Göz hastalıkları: Ortalama 3-5 saniye

Hastalık Türü	Model	Ortalama Min (saniye)	Ortalama Max (saniye)
Deri Hastalıkları	EfficientNetB3	3	5
Kemik Hastalıkları	DenseNet-121	4	6
Akciğer Hastalıkları	-	2	4
Göz Hastalıkları	-	3	5

Not: Tüm ölçümler ortalama değerlerdir. Gerçek tahmin süreleri görüntü boyutuna, model yüküne ve sunucu performansına göre değişebilir.

[TABLO 3.4: MODEL TAHMİN SÜRELERİ TABLOSU BURAYA GELECEK]

3.4.3. Sayfa Yükleme Süreleri

Sayfa yükleme süreleri, Web Vitals metrikleri kullanılarak ölçülmüştür:

- First Contentful Paint (FCP): Ortalama 1-1.5 saniye
- Largest Contentful Paint (LCP): Ortalama 1.5-2.5 saniye
- Time to Interactive (TTI): Ortalama 2-3 saniye

Metrik	Açıklama	Ortalama Min (saniye)	Ortalama Max (saniye)
First Contentful Paint *(FCP)*	İlk İçerik Boyaması	1.0	1.5
Largest Contentful Paint *(LCP)*	En Büyük İçerik Boyaması	1.5	2.5
Time to Interactive *(TTI)*	Etkileşimli Olma Süresi	2.0	3.0

Not: Tüm ölçümler Web Vitals metrikleri kullanılarak yapılmıştır. Gerçek yükleme süreleri ağ hızına, cihaz performansına ve tarayıcıya göre değişebilir.

[TABLO 3.5: SAYFA YÜKLEME SÜRELERİ TABLOSU BURAYA GELECEK]

3.5. Sistem Karşılaştırması

3.5.1. Benzer Sistemlerle Karşılaştırma

Bu çalışma, literatürdeki benzer sistemlerle karşılaştırıldığında aşağıdaki farklılıklar gözlemlenmektedir:

1. **Çoklu Hastalık Desteği:** Çoğu çalışma tek bir hastalık kategorisine odaklanırken, bu çalışma dört farklı hastalık kategorisini desteklemektedir.

2. Entegre Tele-Tıp: Yapay zeka analizi sonrasında doğrudan doktor görüşmesi yapabilme özelliği, benzer çalışmalarda bulunmamaktadır.

3. Kapsamlı Kullanıcı Yönetimi: Analiz geçmişi, favoriler, paylaşım ve istatistikler gibi özellikler, kullanıcı deneyimini artırmaktadır.

Özellik/Kriter	Bu Çalışma	Literatürdeki Benzer Sistemler
Çoklu Hastalık Desteği	4 Kategori (Deri, Kemik, Akciğer, Göz)	Tek Kategori (Genellikle)
Entegre Tele-Tıp	Var (Jitsi Meet Entegrasyonu)	Yok
Kapsamlı Kullanıcı Yönetimi	Var (Geçmiş, Favoriler, Paylaşım, İstatistikler)	Sınırlı veya Yok

Not: Bu karşılaştırma, literatürdeki benzer yapay zeka destekli hastalık tespit sistemleri ile yapılmıştır. Bu çalışma, çoklu hastalık desteği ve entegre tele-tıp özellikleri ile farklılaşmaktadır.

[TABLO 3.6: BENZER SİSTEMLERLE KARŞILAŞTIRMA TABLOSU BURAYA GELECEK]

3.5.2. Özgün Katkılar

Bu çalışmanın özgün katkıları şunlardır:

1. Çoklu Hastalık Platformu: Tek bir platformda dört farklı hastalık kategorisi için analiz yapabilme kapasitesi.

2. Entegre Tele-Tıp Sistemi: Yapay zeka analizi ve doktor görüşmesi entegrasyonu.

3. Grad-CAM Görselleştirmesi: Model kararlarının şeffaflığını artıran görselleştirme tekniği.

4. Açık Kaynak Mimari: Genişletilebilir ve özelleştirilebilir açık kaynak kod yapısı.

5. Responsive ve Erişilebilir Tasarım: Mobil cihazlarda da sorunsuz çalışan, erişilebilirlik standartlarına uygun arayüz.

KAYNAKLAR

[1] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep

neural networks. Nature, 542(7639), 115-118.

[2] Haenssle, H. A., Fink, C., Schneiderbauer, R., Toberer, F., Buhl, T., Blum, A., ... & Reader Study Level I and Level II Groups. (2018). Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. Annals of oncology, 29(8), 1836-1841.

[3] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Ng, A. Y. (2017). CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning. arXiv preprint arXiv:1711.05225.

[4] Gale, W., Oakden-Rayner, L., Carneiro, G., Bradley, A. P., & Palmer, L. J. (2019). Detecting hip fractures with radiologist-level performance using deep neural networks. arXiv preprint arXiv:1711.06504.

[5] Li, X., Hu, X., Yu, L., Zhu, L., Fu, C. W., & Heng, P. A. (2020). CANet: Cross-disease attention network for joint diabetic retinopathy and diabetic macular edema grading. IEEE transactions on medical imaging, 39(5), 1483-1493.

[6] Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning (pp. 6105-6114).

[7] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and

pattern recognition (pp. 4700-4708).

[8] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017).

Grad-cam: Visual explanations from deep networks via gradient-based localization. In

Proceedings of the IEEE international conference on computer vision (pp. 618-626).

[9] Flask Documentation. (2024). Flask Web Framework. https://flask.palletsprojects.com/

[10] Firebase Documentation. (2024). Firebase Authentication, Firestore, Storage.

https://firebase.google.com/docs

[11] TensorFlow Documentation. (2024). TensorFlow and Keras API.

https://www.tensorflow.org/api_docs

[12] Jitsi Meet Documentation. (2024). Jitsi Meet API.

https://jitsi.github.io/handbook/docs/dev-guide/dev-guide-iframe

[13] Bootstrap Documentation. (2024). Bootstrap 4 Framework.

https://getbootstrap.com/docs/4.6/

[14] Font Awesome Documentation. (2024). Font Awesome Icons.

https://fontawesome.com/

[15] Python Software Foundation. (2024). Python Programming Language.

https://www.python.org/

[16] World Health Organization. (2020). Telemedicine: Opportunities and developments in

Member States. WHO Global Observatory for eHealth.

[17] ISIC Archive. (2024). International Skin Imaging Collaboration.

https://www.isic-archive.com/

[18] Mendeley Data. (2024). Medical Image Datasets.

https://data.mendeley.com/

[19] Kaggle. (2024). Medical Image Analysis Competitions.

https://www.kaggle.com/

[20] Web Content Accessibility Guidelines (WCAG) 2.1. (2018). W3C Recommendation.

https://www.w3.org/TR/WCAG21/