ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Bulut Bilişim Ve Uygulamaları

Proje 6: Video Akışı ve İşleme Uygulaması

Emrah Oğuz Ordu
21290298

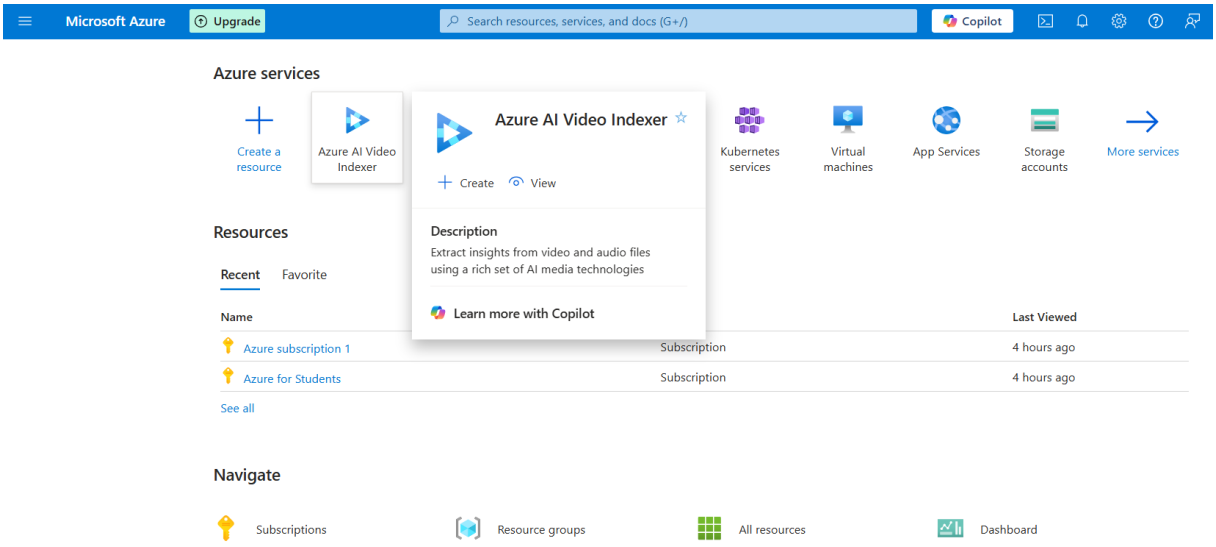
Video Linki: https://www.youtube.com/watch?v=oBoHbbkwjj0
GitHub Uygulama Linki: https://github.com/oguzordu/Video-Akisi-ve-Isleme-Uygulamasi

1. **Proje Özeti**

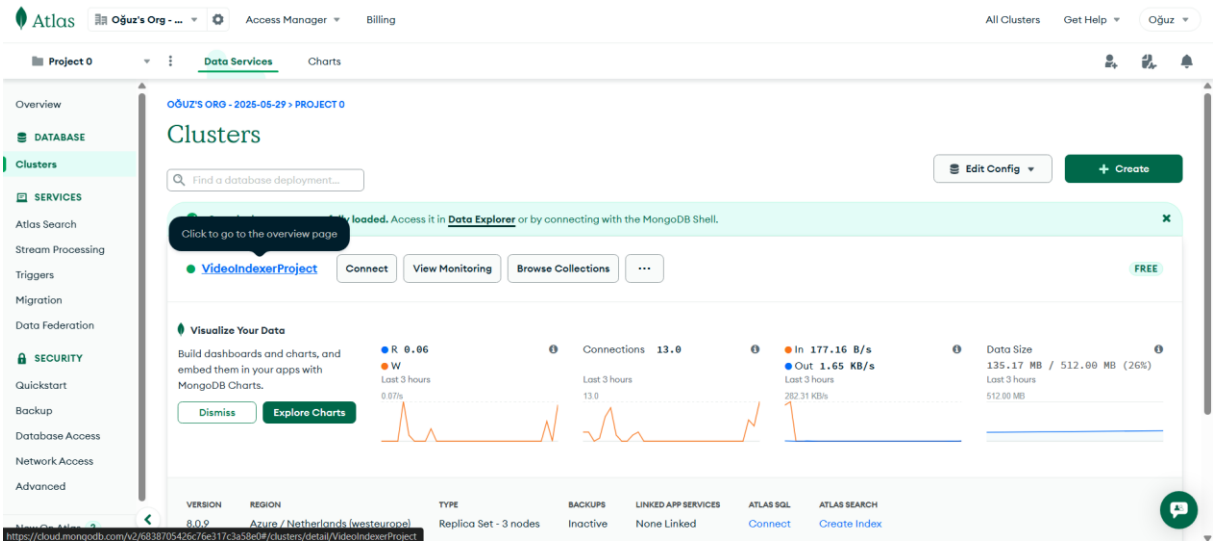
   Bu projede, kullanıcıların video dosyalarını (örn: .mp4) bir web arayüzü üzerinden yükleyebildiği, bu videoların Microsoft Azure Video Indexer servisi kullanılarak bulutta işlenip analiz edildiği ve analiz sonuçlarının kullanıcıya sunulduğu bir Flask tabanlı web uygulaması geliştirilmiştir. Yüklenen videolarla ilgili meta veriler (dosya adı, Azure video ID, yükleme tarihi, işlem durumu vb.) bulut tabanlı bir NoSQL veritabanı olan MongoDB Atlas üzerinde saklanmıştır. Proje kapsamında, ek olarak WebRTC teknolojisi kullanılarak tarayıcı üzerinden canlı kamera görüntüsü akışını gösteren bir demo özelliği de entegre edilmiştir. Temel amaç, Azure bulut platformunun video işleme yeteneklerini ve MongoDB Atlas gibi yönetilen veritabanı servislerini kullanarak basit ama işlevsel bir video yönetim ve analiz sistemi prototipi oluşturmak olmuştur.

2. **Kullanılan Teknolojiler, Servisler ve Yapılandırmalar**

- **Programlama Dili:** Python
- **Framework:** Flask (Web uygulaması çatısı için kullanıldı)
- **Bulut Platformu:** Microsoft Azure
- **Azure Video Indexer:** Videoların yüklenmesi, transkripsiyonu, çevirisi, yüz tanıma, nesne tanıma gibi çeşitli analizlerin yapıldığı ana işleme servisi olarak kullanıldı.
- **Yapılandırma Detayları:**
- Azure Video Indexer AI portalı üzerinden ücretsiz (trial) bir hesap oluşturuldu.
- Hesap Ayarları (Account Settings) bölümünden API Anahtarları (Primary Key) ve Hesap ID'si (Account ID) temin edildi. Bu bilgiler, geliştirilen uygulamanın Video Indexer API'sine güvenli erişimi için .env dosyasında yapılandırıldı.



- **Veritabanı:** MongoDB Atlas (Bulut tabanlı NoSQL veritabanı servisi)
- **Yapılandırma Detayları:**
- MongoDB Atlas platformunda ücretsiz (M0 Sandbox) bir cluster oluşturuldu.



- Database Access bölümünden uygulama için özel bir veritabanı kullanıcısı (dbUser) tanımlandı ve güvenli bir parola atandı.
- Network Access bölümünde, uygulamanın geliştirildiği ve çalıştırılacağı IP adreslerine (geliştirme aşamasında "Allow Access From Anywhere" seçeneği ile tüm IP'lere) erişim izni tanımlandı.
- Oluşturulan Cluster'a bağlanmak için gerekli olan bağlantı dizesi (Connection String) elde edildi ve uygulamanın .env dosyasında MONGODB_CONNECTION_STRING olarak saklandı.
- **Frontend Teknolojileri:**
- HTML (Sayfa yapısı oluşturuldu)
- JavaScript (Dinamik işlemler, API çağrıları ve WebRTC entegrasyonu için kullanıldı)
- **Temel Python Kütüphaneleri:**
- Flask: Web sunucusu ve uygulama yönlendirmeleri için kullanıldı.
- python-dotenv: Ortam değişkenlerini (.env dosyası) yönetmek amacıyla projeye dahil edildi.
- requests: Azure Video Indexer API'sine HTTP istekleri göndermek için kullanıldı.

- pymongo: MongoDB Atlas veritabanı ile etkileşim kurmak için kullanıldı.

- **Ortam Değişkenleri (.env dosyası):**

- Proje konfigürasyonlarını ve hassas bilgileri (API anahtarları, bağlantı dizeleri) saklamak için bir .env dosyası oluşturuldu ve aşağıdaki değişkenler tanımlandı:

- API_KEY_1: Azure Video Indexer API anahtarı.

- ACCOUNT_ID: Azure Video Indexer Hesap ID'si.

- AZURE_LOCATION: Azure Video Indexer hesabının bulunduğu bölge (proje için "trial" olarak ayarlandı).

- MONGODB_CONNECTION_STRING: MongoDB Atlas bağlantı dizesi.

- MONGODB_DB_NAME: MongoDB Atlas'ta oluşturulan veritabanının adı.

3. Sistem Mimarisi ve Veri Akışı

1. **Kullanıcı Arayüzü (Web Tarayıcısı):** Kullanıcı, Flask uygulaması tarafından sunulan HTML arayüzü üzerinden sisteme erişim sağladı.

2. **Video Yükleme (MP4):**

- Kullanıcı, yerel sisteminden bir MP4 video dosyası seçerek "Upload" butonu aracılığıyla yükleme işlemini başlattı.

- Tarayıcı, seçilen videoyu Flask backend'indeki /upload_video endpoint'ine HTTP POST isteği ile gönderdi.

3. **Flask Backend (app.py):**

- /get_access_token endpoint'i, Azure Video Indexer'dan periyodik olarak bir erişim token'ı aldı. Bu token, sonraki API çağrılarında yetkilendirme için kullanıldı.

- /upload_video endpoint'i, kullanıcıdan gelen video dosyasını ve geçerli erişim token'ını kullanarak Azure Video Indexer'a yükledi. Yükleme işlemi sırasında video adı ve gizlilik ayarı (Private) gibi parametreler de Azure API'sine iletildi.

- Video Azure'a başarıyla yüklendikten sonra, Azure Video Indexer'dan dönen benzersiz videoId ve videoyla ilgili diğer bilgiler (orijinal dosya adı, yükleme zaman damgası, başlangıç durumu "Uploaded" olarak) MongoDB Atlas'taki videos adlı koleksiyona insert_one operasyonu ile kaydedildi.

4. **Azure Video Indexer:**

- Platform, yüklenen videoyu alarak asenkron bir şekilde video işleme (transkripsiyon, nesne tanıma vb.) sürecini başlattı.

- İşlem tamamlandığında, videonun durumu platform üzerinde "Processed" olarak güncellendi. Bu durum değişikliği, uygulama tarafından periyodik sorgularla takip edilmedi; bunun yerine, kullanıcı analiz istediğinde güncel durum ve sonuçlar çekildi.

5. **Video Listeleme ve Analiz Sonuçlarının Gösterimi:**

- Uygulamanın ana sayfasında (/),
  MongoDB'deki videos koleksiyonundan find operasyonu ile çekilen, daha önce yüklenmiş videoların bir listesi kullanıcıya sunuldu.

- Kullanıcı, listedeki herhangi bir video için "Get Analysis" butonuna tıkladığında, ilgili videonun videoId'si ile Flask
  backend'indeki /get_video_analysis/<video_id> endpoint'ine bir istek gönderildi.

- Bu endpoint, alınan videoId ve geçerli erişim token'ını kullanarak Azure Video Indexer'dan videonun detaylı analiz sonuçlarını (JSON formatında) çekti ve bu veriyi frontend'e ileterek kullanıcının görmesi sağlandı.

6. **WebRTC Canlı Yayın Demosu:**

- Kullanıcı "Start Live Stream" butonuna tıkladığında,
  tarayıcı getUserMedia API'si aracılığıyla kullanıcının kamera ve mikrofonuna erişim izni istedi.

- İzin verilmesi durumunda, canlı kamera görüntüsü doğrudan tarayıcıdaki <video> HTML elementine yansıtıldı. "Stop Live Stream" butonu ile yayın sonlandırıldı. Bu özellik, sunucu tarafında herhangi bir kayıt veya işleme yapmaksızın, sadece istemci tarafında canlı bir önizleme demosu olarak geliştirildi.

4. Yazılım Geliştirme (app.py ve Frontend)

- **app.py (Flask Backend Mimarisi):**

- **Flask Uygulaması ve Ortam Kurulumu:** Projenin temeli olarak bir Flask uygulaması (app = Flask(__name__)) başlatıldı. python-dotenv kütüphanesi ile .env dosyasındaki ortam değişkenleri uygulamaya yüklendi.

- **MongoDB Bağlantısının Kurulması:** pymongo.MongoClient kullanılarak .env dosyasından okunan bağlantı dizesi aracılığıyla MongoDB Atlas cluster'ına bağlantı sağlandı ve hedef veritabanı ile videos koleksiyonu referans alındı.

- **Ana Rotaların (Endpoints) Tanımlanması:**

- @app.route('/'): Uygulamanın ana sayfasını (index.html) render etmek için tanımlandı. Bu rota, MongoDB'den mevcut videoları çekip şablona göndererek kullanıcıya listeledi.

- @app.route('/get_access_token', methods=['GET']): Azure Video Indexer API'sine erişim için gerekli olan access token'ı almak üzere bir GET isteği işleyen rota oluşturuldu.

- @app.route('/upload_video', methods=['POST']): Kullanıcının gönderdiği video dosyasını almak, Azure Video Indexer'a yüklemek ve yükleme bilgilerini MongoDB'ye kaydetmek üzere bir POST isteği işleyen rota geliştirildi.

- @app.route('/get_video_analysis/<video_id>', methods=['GET']): Belirli bir video_id için Azure Video Indexer'dan analiz sonuçlarını çekmek ve JSON olarak döndürmek üzere dinamik bir GET rotası tanımlandı.

- **Yardımcı Fonksiyonlar:** Azure API'lerine istek gönderme (get_azure_access_token, upload_to_azure, get_video_analysis_from_azure) ve MongoDB işlemleri (insert_video_metadata, get_all_videos) gibi tekrarlayan görevler için modüler Python fonksiyonları yazıldı. Bu fonksiyonlar, kodun daha okunabilir ve yönetilebilir olmasını sağladı.

- **Hata Yönetimi ve İyileştirmeler:** requests kütüphanesi ile yapılan API çağrılarında try-except blokları ve response.raise_for_status() kullanılarak olası HTTP hataları (örn: 4xx, 5xx) yakalandı. Eksik konfigürasyon (örn: .env dosyasında anahtar olmaması) durumları için başlangıç kontrolleri eklendi.

- **Frontend (templates/index.html ve Statik JavaScript Dosyası):**

- **HTML Yapısının Oluşturulması:** Video yükleme formu (<input type="file">), yüklenen videoların listeleneceği bir alan (<ul id="videoList">), analiz sonuçlarının gösterileceği bir bölüm ( ) ve WebRTC demo butonları için temel HTML5 elementleri kullanılarak index.html şablonu oluşturuldu.

- **JavaScript ile Dinamik Etkileşimlerin Sağlanması:**

- **Video Yükleme Mantığı:** Video yükleme formunun gönderilmesi fetch API'si ile asenkron olarak gerçekleştirildi. FormData nesnesi kullanılarak dosya verisi backend'e iletildi. Yükleme başarılı olduğunda dönen videoId kullanıcıya gösterildi ve video listesi güncellendi.

- **Analiz Sonuçlarını Alma ve Gösterme:** "Get Analysis" butonlarına olay dinleyicileri (event listeners) eklendi. Tıklandığında, ilgili videoId ile backend'e fetch isteği yapıldı ve dönen JSON formatındaki analiz verisi parse edilerek okunabilir bir şekilde sayfada görüntülendi.

- **Video Listesinin Dinamik Oluşturulması:** Sayfa ilk yüklendiğinde ve yeni bir video başarıyla yüklendiğinde, backend'den alınan video listesi kullanılarak HTML listesi dinamik olarak JavaScript ile oluşturuldu ve güncellendi.

- **WebRTC Entegrasyonu:** navigator.mediaDevices.getUserMedia() API'si çağrılarak kullanıcının kamera ve mikrofonuna erişim istendi. İzin alındıktan sonra, medya akışı (MediaStream)
  bir <video> elementinin srcObject özelliğine atanarak canlı görüntü akışı sağlandı. "Start" ve "Stop" butonları ile akışın başlatılması ve durdurulması yönetildi.

5. Projenin Hayata Geçirilmesi ve Test Süreci

1. **Geliştirme Ortamının Hazırlanması:**

- Proje geliştirme için Python 3.9 ve pip paket yöneticisi kullanıldı.

2. **Proje Kodlarının Edinilmesi:**

- Proje, Git versiyon kontrol sistemi kullanılarak geliştirildi ve GitHub üzerinde [GitHub Depo Linkiniz] adresinde bir depo oluşturularak kodlar buraya yüklendi. Geliştirme ve test için bu depo yerel makineye klonlandı (git clone).

3. **Gerekli Python Kütüphanelerinin Kurulumu:**

- Projenin çalışması için gerekli olan Python kütüphaneleri (Flask, python-dotenv, requests, pymongo) bir requirements.txt dosyasında listelenerek (veya doğrudan) pip install -r requirements.txt (veya pip install Flask python-dotenv requests pymongo) komutu ile geliştirme ortamına kuruldu.

4. **Ortam Değişkenlerinin Yapılandırılması:**

- Proje ana dizininde .env adında bir dosya oluşturuldu. Bu dosyaya, Bölüm 2'de detaylandırılan Azure Video Indexer API Anahtarı, Hesap ID'si, Azure Bölgesi, MongoDB Atlas Bağlantı Dizesi ve Veritabanı Adı gibi konfigürasyon bilgileri girildi.

5. **Uygulamanın Yerel Geliştirme Sunucusunda Çalıştırılması:**

- Flask'in gömülü geliştirme sunucusu, proje ana dizininde python app.py komutu çalıştırılarak başlatıldı.

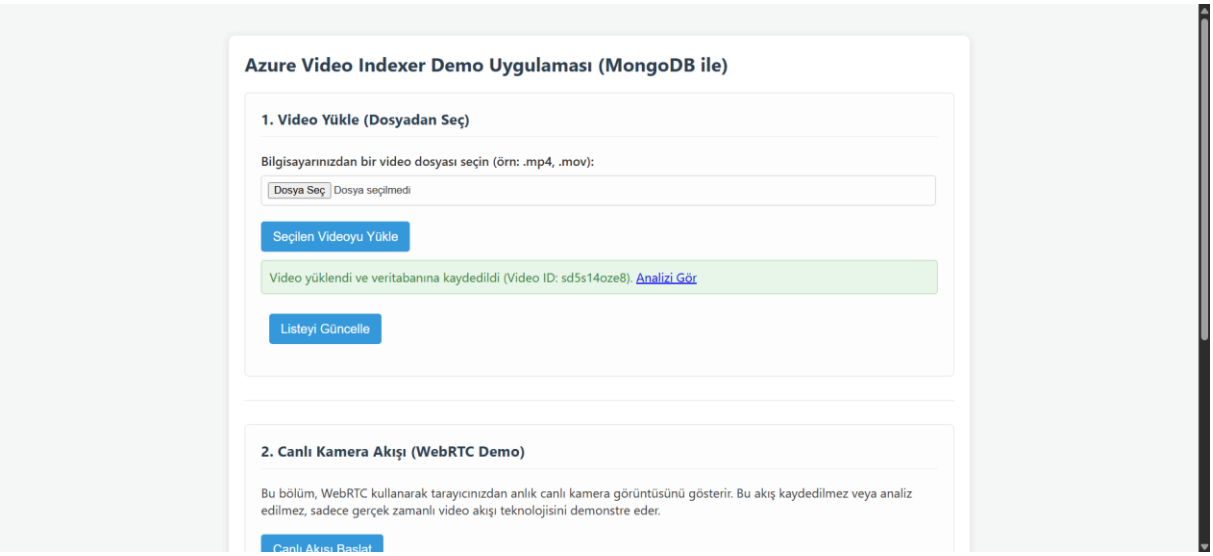
- Uygulamaya, bir web tarayıcısı üzerinden varsayılan olarak http://127.0.0.1:5000/ adresi ziyaret edilerek erişildi.

6. **Uygulama Fonksiyonlarının Test Edilmesi ve Sonuçlar:**

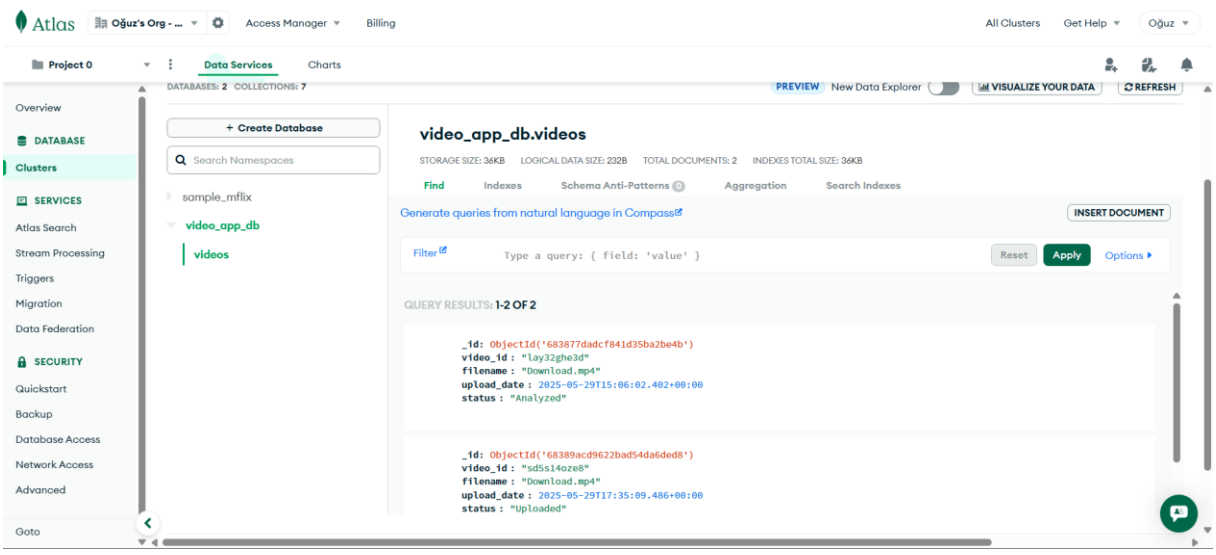
- **Video Yükleme Fonksiyonunun Testi:**

- Web arayüzündeki form aracılığıyla geçerli bir .mp4 formatında video dosyası seçildi ve "Upload Video" butonu ile yükleme işlemi başlatıldı.

- Yükleme işleminin başarıyla tamamlandığı, arayüzde "Video uploaded successfully! Video ID: [dönen_id]" şeklinde bir mesajın göründüğü ve yüklenen videonun sayfanın alt kısmındaki listede yer aldığı gözlemlendi.



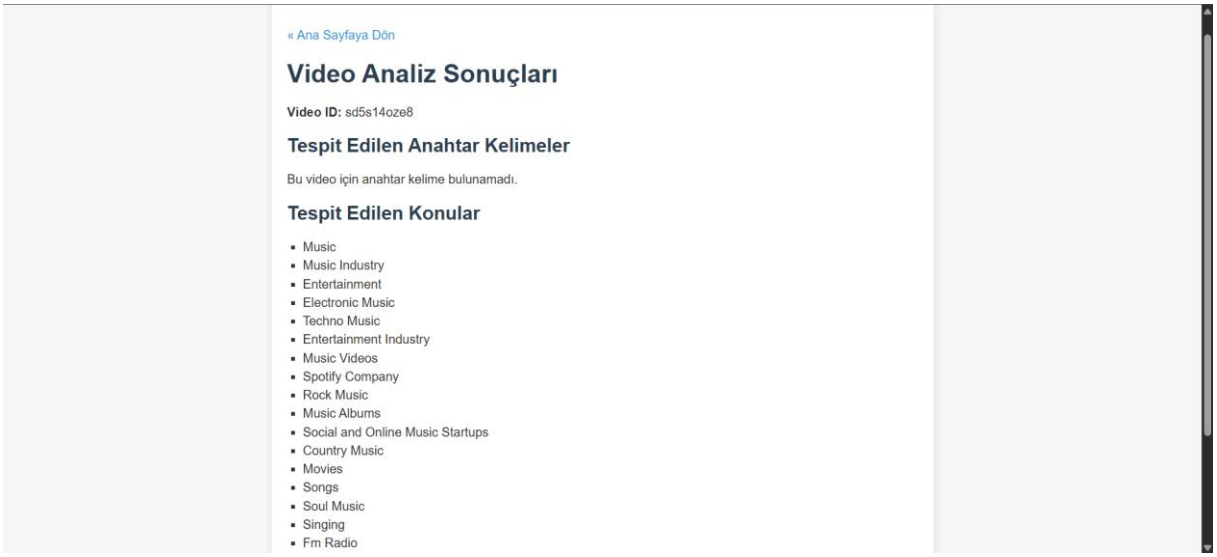
- MongoDB Atlas veritabanındaki videos koleksiyonu kontrol edildiğinde, yüklenen videoya ait yeni bir dökümanın (kaydın) oluştuğu; bu dökümanda filename, azure_video_id, upload_date ve status: 'Uploaded' gibi alanların doğru bilgilerle populate edildiği teyit edildi.



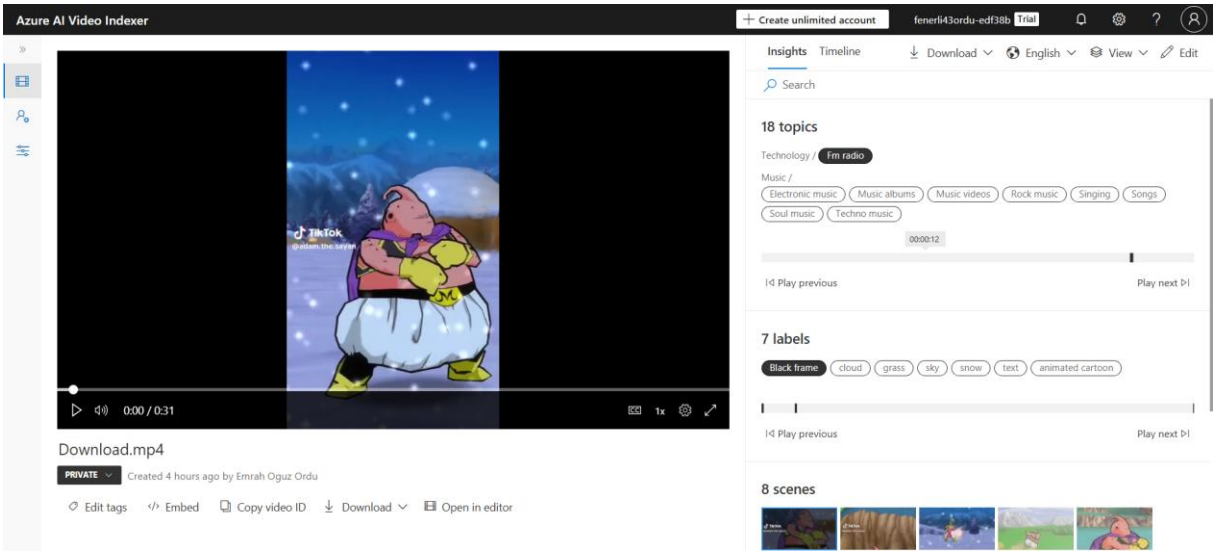
- **Video Analiz Sonuçlarını Görüntüleme Testi:**

- Daha önce yüklenmiş ve Azure Video Indexer tarafından işleme süreci tamamlanmış (bu işlem videonun süresine göre birkaç dakika alabilir) bir video için listedeki "Get Analysis" butonuna tıklandı.

- Azure Video Indexer'dan gelen analiz sonuçlarının (örneğin, videonun transkripti, tespit edilen anahtar kelimeler, duygusal analiz vb. JSON verisinin bir kısmı) web arayüzündeki ilgili bölümde başarıyla gösterildiği doğrulandı.



- Bu test sırasında, Azure Video Indexer portalı üzerinden de ilgili videonun analiz durumunun "Processed" (İşlendi) olduğu ve detaylı "Insights" (içgörüler) sekmesinin erişilebilir olduğu kontrol edildi.

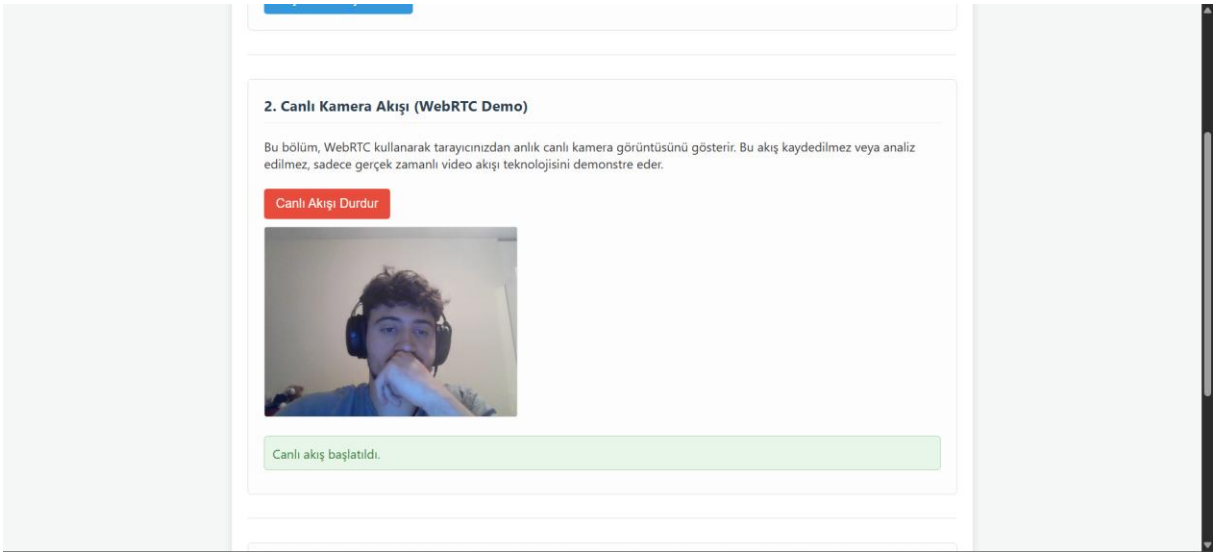


- **WebRTC Canlı Kamera Akışı Demosunun Testi:**

- Web arayüzündeki "Start Live Stream" butonuna tıklandı.

- Tarayıcının kamera ve mikrofon erişimi için gösterdiği izin istemi onaylandı.

- Kullanıcının bilgisayarına bağlı kameradan alınan görüntünün web sayfasındaki <video> elementinde canlı olarak aktığı gözlemlendi.

- 

  "Stop Live Stream" butonuna tıklanarak canlı yayının başarıyla sonlandırıldığı teyit edildi.

6. Sonuç ve Değerlendirme Bu proje ile Microsoft Azure Video Indexer ve MongoDB Atlas gibi güçlü bulut servisleri kullanılarak, video yükleme, bulutta otomatik işleme, analiz sonuçlarını web arayüzünde sunma ve video meta verilerini yönetme yeteneklerine sahip fonksiyonel bir web uygulaması başarıyla geliştirilmiştir. Python tabanlı Flask framework'ü ile oluşturulan backend, Azure ve MongoDB servisleriyle etkili bir şekilde entegre edilerek kullanıcıya interaktif ve zengin bir deneyim sunulmuştur. Ek olarak geliştirilen WebRTC tabanlı canlı yayın demosu, modern tarayıcıların gerçek zamanlı video iletişim yeteneklerini sergilemiştir. Proje, bulut bilişim teknolojilerinin video içerik yönetimi, analizi ve dağıtımı alanlarındaki temel prensiplerini ve pratik uygulamalarını ortaya koymaktadır. Geliştirilen prototip, basit bir yapıda olmasına rağmen, daha kapsamlı ve özellikli video işleme platformları için ölçeklenebilir bir başlangıç noktası ve değerli bir öğrenme deneyimi sunmuştur.