

ANKARA ÜNİVERSİTESİ  
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM3522-BULUT BİLİŞİM VE UYGULAMALARI  
DERSİ PROJE ÖDEVİ



EZGİ SANKIR- 21290431  
ZİYA EREN ALTAŞ -21290571

Github Linki: <https://github.com/Ezgiis02>

Videolar:

Video Akışı ve İşleme Uygulaması: <https://youtu.be/5lTEJcss0RE>

Diyabet Tahmin Uygulaması: <https://youtu.be/oDQqbMmw08I>

E-Ticaret Web Sitesi: <https://youtu.be/JJ6fFrX5CJI>

Çift Katmanlı Web Uygulaması : <https://youtu.be/HIUXbOTlhPk>

Hocam Diyabet Tahmin Uygulamasını Azure aboneliğinde yaşadığım sıkıntıdan dolayı baştan yapmak durumunda kaldım. Öğrenci aboneliğine geçtiğim için sanal makine ben çalıştırdıktan sonra sadece 1 saat çalışıyor.

Videoları mümkün olduğunca kısa tutmaya çalıştım. 2 dakika ile 3 dakika arasında oldular.

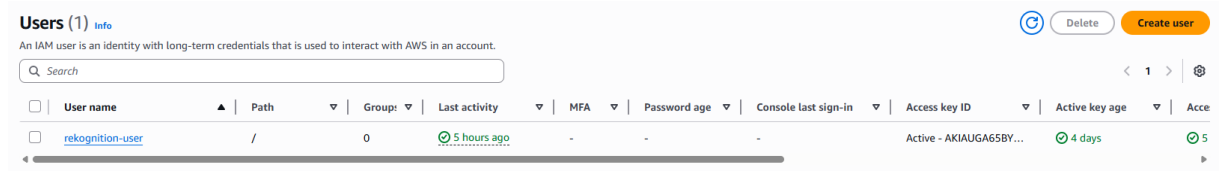
## Proje: Video Akışı ve İşleme Uygulaması

**Projenin Amacı:** Bu projede, gerçek zamanlı video akışlarının yönetilmesi ve bu akışlar üzerinden nesne tanıma, etiketleme gibi analizlerin yapılması amaçlanmıştır. Bulut tabanlı video işleme çözümleri kullanılarak, esnek ve ölçeklenebilir bir sistem geliştirilmiştir.

## GELİŞTİRME SÜRECİ

### Kurulumlar

AWS hesabı açıldı ve IAM (Identity and Access Management) bölümünden Users kısmından kullanıcı oluşturuldu .(rekognition-user)



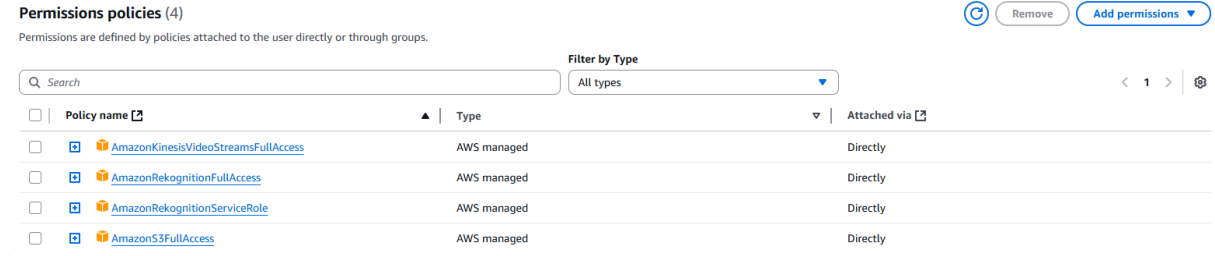
**Users (1)** Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

Search

<input type="checkbox"/>	User name	Path	Group	Last activity	MFA	Password age	Console last sign-in	Access key ID	Active key age	Access key status
<input type="checkbox"/>	<a href="#">rekognition-user</a>	/	0	5 hours ago	-	-	-	Active - AKIAUGA65BY...	4 days	5

AmazonKinesisVideoStreamsFullAccess, AmazonRekognitionFullAccess, AmzmonRekognitionServiceRole, AmazonS3FullAccess izinleri verildi.



**Permissions policies (4)**

Permissions are defined by policies attached to the user directly or through groups.

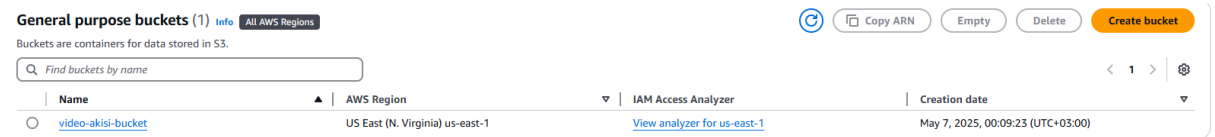
Search

Filter by Type: All types

<input type="checkbox"/>	Policy name	Type	Attached via
<input type="checkbox"/>	<a href="#">AmazonKinesisVideoStreamsFullAccess</a>	AWS managed	Directly
<input type="checkbox"/>	<a href="#">AmazonRekognitionFullAccess</a>	AWS managed	Directly
<input type="checkbox"/>	<a href="#">AmazonRekognitionServiceRole</a>	AWS managed	Directly
<input type="checkbox"/>	<a href="#">AmazonS3FullAccess</a>	AWS managed	Directly

Access key, Secret Access key ve Region bilgileri alındı.

Video dosyalarının geçici olarak saklanması ve analiz edilebilmesi amacıyla bir S3 Bucket oluşturuldu. (video-akisi-bucket)



**General purpose buckets (1)** Info All AWS Regions

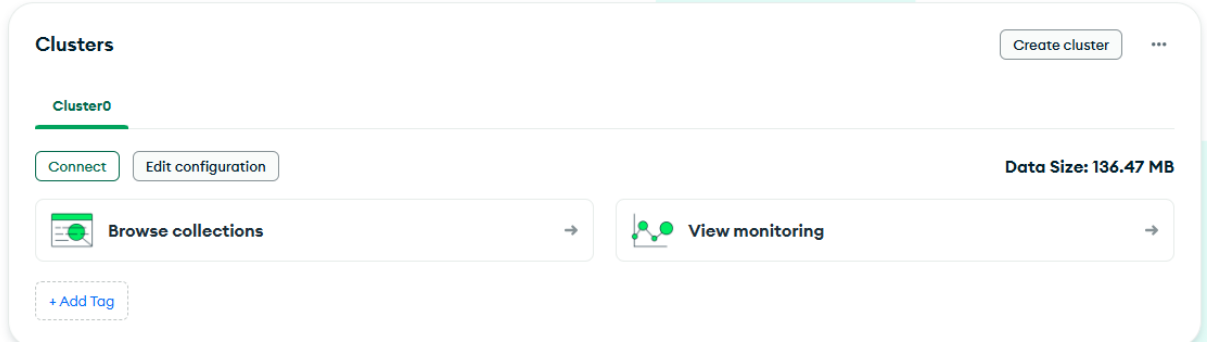
Buckets are containers for data stored in S3.

Find buckets by name

<input type="radio"/>	Name	AWS Region	IAM Access Analyzer	Creation date
<input type="radio"/>	<a href="#">video-akisi-bucket</a>	US East (N. Virginia) us-east-1	<a href="#">View analyzer for us-east-1</a>	May 7, 2025, 00:09:23 (UTC+03:00)

MongoDB Atlas hesabı açıldı. Proje oluşturuldu. Cluster oluşturuldu. (Cluster 0)

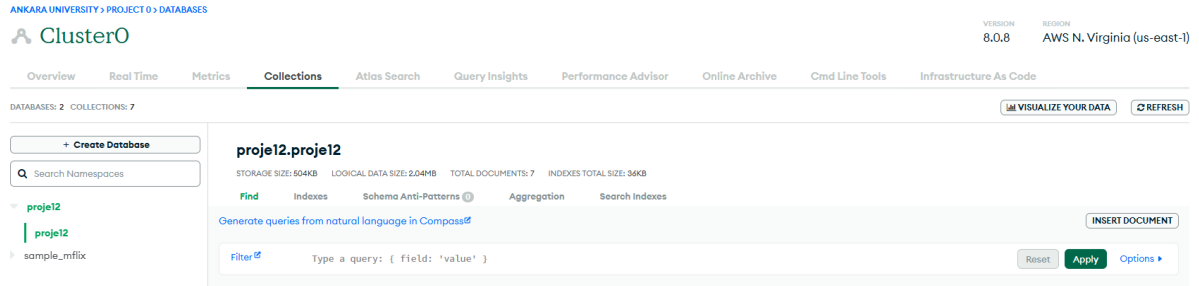
## Overview



Connect kısmına girip, ardından Drivers kısmına girip Cluster'a bağlanabilmek için URI alındı.

```
mongodb+srv://<db_username>:<db_password>@cluster0.18atnrk.mongodb.net/?
retryWrites=true&w=majority&appName=Cluster0
```

Collection ve Database oluşturuldu. Verilerimiz buraya kaydedilecek.



## Kodlar

### AWS S3'e Video Yükleme (S3\_uploader.py)

Projede video dosyalarının bulutta saklanması amacıyla AWS S3 (Simple Storage Service) kullanılmıştır. Python programlama diliyle birlikte boto3 kütüphanesi aracılığıyla, yerel dosyaların doğrudan S3 Bucket içerisine yüklenmesi sağlanmıştır.

boto3.client('s3', ...) komutu ile S3 servisine erişim sağlanmıştır.

AWS hesabı üzerinden alınan Access Key, Secret Access Key ve Region bilgileri kullanılarak kimlik doğrulama yapılmıştır.

upload\_file fonksiyonu, belirtilen file\_path'teki yerel dosyayı bucket\_name adlı S3 klasörüne, s3\_key ile tanımlanan konuma yüklemektedir.

İşlem başarıyla tamamlandığında kullanıcıya bilgi veren bir çıktı mesajı yazdırılmıştır. Bu sayede, video analizinden önce veya sonra içerik güvenli bir şekilde bulut ortamında saklanabilmektedir.

```

s3_uploader.py > ...
1  import boto3
2
3  def upload_video_to_s3(file_path, bucket_name, s3_key):
4      import boto3
5
6      # Kimlik bilgilerini manuel olarak ayarlamak
7      s3 = boto3.client('s3',
8                          aws_access_key_id='AKIAUGA65BY340A7AVX2',
9                          aws_secret_access_key='oHEKrgwzKSumdVMVCA3XSy4AY/4npu9jIMZt5BcY',
10                         region_name='us-east-1')
11
12     s3.upload_file(file_path, bucket_name, s3_key)
13     print(f"{file_path} dosyası {bucket_name}/{s3_key} olarak yüklendi.")
14

```

### MongoDB ile Video Analiz Sonuçlarının Saklanması (mongo\_handler.py)

Projede video analizlerine ait çıktıların saklanması ve gerektiğinde tekrar erişilebilmesi amacıyla MongoDB Atlas kullanılmıştır. Python ile veritabanı bağlantısı kurmak için pymongo kütüphanesi tercih edilmiştir.

MongoDBHandler sınıfı, video analiz sonuçlarının MongoDB'ye kaydedilmesi, sorgulanması ve bağlantının güvenli bir şekilde kapatılması işlemlerini gerçekleştirmektedir.

\_\_init\_\_ fonksiyonu ile MongoDB Atlas üzerinde oluşturulan proje12 adlı veritabanına bağlantı sağlanmaktadır.

Bağlantı için MongoClient kullanılmış ve bağlantı URI'si doğrudan sınıf içinde tanımlanmıştır.

```

mongo_handler.py > ...
1  from pymongo import MongoClient
2  from datetime import datetime
3  import json
4
5  class MongoDBHandler:
6      def __init__(self):
7          # Doğrudan bağlantı bilgilerini burada tanımlayın
8          self.MONGODB_URI = "mongodb+srv://user11:test11@cluster0.18atnrk.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
9          self.DB_NAME = "proje12"
10         self.COLLECTION_NAME = "proje12"
11
12         self.client = MongoClient(self.MONGODB_URI)
13         self.db = self.client[self.DB_NAME]
14

```

save\_analysis\_results fonksiyonu, analiz edilen videoya ait JSON dosyasını okuyarak içeriğini MongoDB'ye kaydeder. Kayıt sırasında videonun adı, analiz tarihi ve etiket verileri (labels) birlikte tutulur.

get\_results fonksiyonu ile daha önce veritabanına kaydedilmiş analiz sonuçları istenirse video adına göre, istenirse tümüyle sorgulanabilir.

close\_connection fonksiyonu ile veritabanı bağlantısının sistem kaynaklarını gereksiz yere tüketmemesi için güvenli bir şekilde kapatılması sağlanır.

```

def save_analysis_results(self, video_name, json_path):
    """Video analiz sonuçlarını MongoDB'ye kaydeder"""
    collection = self.db[self.COLLECTION_NAME]

    with open(json_path) as f:
        labels = json.load(f)

    document = {
        "video_name": video_name,
        "analysis_date": datetime.now(),
        "labels": labels,
        "status": "completed"
    }

    return collection.insert_one(document).inserted_id

def get_results(self, video_name=None):
    """Sonuçları sorgular"""
    collection = self.db[self.COLLECTION_NAME]
    if video_name:
        return collection.find_one({"video_name": video_name})
    return list(collection.find({}))

def close_connection(self):
    """Bağlantıyı güvenli şekilde kapat"""
    if hasattr(self, 'client') and self.client:
        self.client.close()
        self.client = None
        self.db = None

```

## AWS Rekognition ile Video Etiketleme (Label Detection) (rekognition\_handler.py)

AWS Rekognition servisi kullanılarak S3'e yüklenen videolar üzerinde nesne tanıma (label detection) işlemleri gerçekleştirilmiştir. Python ve boto3 kütüphanesi aracılığıyla, videonun işlenmesi, etiketlerin alınması ve çıktının dosyaya kaydedilmesi sağlanmıştır.

AWS erişim bilgileri (Access Key, Secret Key, Region) kullanılarak Rekognition servisine bağlantı kurulmuştur.

```

rekognition_handler.py > ...
1  import boto3
2  import time
3  import json
4
5  # 📌 Buraya kendi bilgilerini yaz:
6  AWS_ACCESS_KEY = 'AKIAUGA65BY340A7AVX2'
7  AWS_SECRET_KEY = 'oHEKrgwzKSumdVMVCA3XSy4AY/4npu9jIMZt5BcY'
8  AWS_REGION = 'us-east-1' # veya kendi bölgen ne ise
9
10 # Rekognition client'ını oluştur
11 rekognition = boto3.client(
12     'rekognition',
13     aws_access_key_id=AWS_ACCESS_KEY,
14     aws_secret_access_key=AWS_SECRET_KEY,
15     region_name=AWS_REGION
16 )
17

```

start\_label\_detection fonksiyonu, belirtilen S3 bucket içerisindeki bir video dosyasında etiket algılama işlemini başlatır. AWS tarafından bu işlem için bir JobId döndürülür. Bu ID, ilerleyen aşamada sonucu sorgulamak için gereklidir.

get\_label\_detection\_result fonksiyonu, JobId yardımıyla etiket algılama işleminin tamamlanıp tamamlanmadığını periyodik olarak kontrol eder. İşlem başarıyla tamamlandığında sonuçları

içeren veri döndürülür. Eğer işlem başarısız olursa bir hata gösterilir, işlem devam ediyorsa 5 saniyede bir tekrar kontrol edilir.

save\_result\_to\_file fonksiyonu ile elde edilen sonuçlar bir JSON dosyasına kaydedilir.

```
def start_label_detection(bucket_name, video_name):
    response = rekognition.start_label_detection(
        Video={'S3Object': {'Bucket': bucket_name, 'Name': video_name}}
    )
    return response['JobId']

def get_label_detection_result(job_id):
    print("AWS Rekognition sonucu bekleniyor...")
    while True:
        result = rekognition.get_label_detection(JobId=job_id)
        status = result['JobStatus']
        if status == 'SUCCEEDED':
            print("Etiketleme tamamlandı ✅")
            return result
        elif status in ['FAILED', 'ERROR']:
            raise Exception(f"İşlem başarısız: {status}")
        else:
            print("Bekleniyor... İşlem devam ediyor...")
            time.sleep(5)

def save_result_to_file(result, filename):
    with open(filename, 'w') as f:
        json.dump(result, f, indent=4)
    print(f"Etiketler {filename} dosyasına kaydedildi.")
```

### Etiketli Video Görselleştirme Modülü (video\_drawer.py)

AWS Rekognition tarafından analiz edilmiş ve etiketlenmiş bir videonun üzerine nesne sınırlayıcı kutularını (bounding box) ve etiket adlarını çizerek videoyu kullanıcıya görsel olarak sunar. Python'da OpenCV ve JSON verisi kullanılarak gerçekleştirilmiştir.

Video ve JSON Etiket Dosyasını Okuma: Verilen video\_path adresindeki video okunur. AWS Rekognition tarafından üretilmiş JSON dosyası açılır ve içindeki etiket verileri yüklenir.

Etiketleri Timestamp'e Göre Gruplama: AWS etiketi her Timestamp (zaman damgası) için ayrı şekilde gruplandırılmıştır. Her zaman noktasındaki etiketler, bir sözlükte saklanır. Böylece her frame ile doğru etiket eşleşmesi yapılabilir.

```

video_drawer.py > ...
1  import cv2
2  import json
3
4  def draw_labels_on_video(video_path, json_path):
5      # Video yakalayıcıyı başlat
6      video_capture = cv2.VideoCapture(video_path)
7      if not video_capture.isOpened():
8          raise ValueError("Video açılmadı")
9
10     with open(json_path, 'r') as f:
11         data = json.load(f)
12
13     # Video özelliklerini al
14     fps = video_capture.get(cv2.CAP_PROP_FPS)
15     width = int(video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
16     height = int(video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
17
18     # Etiketleri timestamp'e göre grupla
19     labels_by_timestamp = {}
20     for item in data['Labels']:
21         timestamp = item.get('Timestamp', 0)
22         label_info = item.get('Label', {})
23         name = label_info.get('Name', 'Unknown')
24         instances = label_info.get('Instances', [])
25         if timestamp not in labels_by_timestamp:
26             labels_by_timestamp[timestamp] = []
27             labels_by_timestamp[timestamp].append({
28                 'name': name,
29                 'instances': instances
30             })
31
32     current_labels = [] # En sonki aktif etiketler
33

```

**Videoyu Kare Kare Okuma ve Etiketleri Çizme:** Video, kare kare okunur. Her karenin zaman damgası alınır ve varsa o zamana ait etiketler kareye çizilir. Her etiketin bounding box bilgisi varsa kutu çizilir, yoksa sadece metin olarak etiketi gösterilir.

**Görüntüyü JPEG Formatında Yayınlama:** Her frame JPEG formatına dönüştürülerek bir web arayüzünde ya da canlı yayında kullanılabilir hâle getirilmiştir. yield komutu sayesinde video canlı olarak gösterilebilir.



```

while True:
    ret, frame = video_capture.read()
    if not ret:
        break

    current_time_ms = video_capture.get(cv2.CAP_PROP_POS_MSEC)
    current_timestamp = int(current_time_ms)

    # Eğer yeni bir timestamp varsa güncelle
    if current_timestamp in labels_by_timestamp:
        current_labels = labels_by_timestamp[current_timestamp]

    # Mevcut aktif etiketleri çiz
    y_offset = 30
    for label in current_labels:
        name = label['name']
        instances = label['instances']
        if instances:
            for inst in instances:
                box = inst.get('BoundingBox', None)
                if box:
                    x1 = int(box['Left'] * width)
                    y1 = int(box['Top'] * height)
                    x2 = x1 + int(box['Width'] * width)
                    y2 = y1 + int(box['Height'] * height)

                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                    cv2.putText(frame, name, (x1, y1 - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
                else:
                    cv2.putText(frame, name, (10, y_offset),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)
                    y_offset += 30

    # Frame'i JPEG'e dönüştür
    ret, jpeg = cv2.imencode('.jpg', frame)
    if not ret:
        break
    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n\r\n\r\n')

video_capture.release()

```

## Proje Arayüzü (index.html)

Kullanıcıdan bir video dosyası seçmesi istenir, ardından video sunucuya yüklenir ve işlendikten sonra görsel çıktısı arayüzde gösterilir. Sayfa Türkçe dilinde hazırlanmıştır.

HTML Yapısı:

- `<h2>AWS Rekognition Video Etiketleyici</h2>`: Sayfa başlığıdır. AWS Rekognition hizmetiyle video işleme yapılacağı belirtilmiştir.
- `<input type="file">`: Kullanıcının video dosyası seçmesi için bir alan sunar.
- `<button onclick="upload()">Gönder ve İşle</button>`: Seçilen videonun sunucuya gönderilmesini ve işlenmesini başlatır.
- `<p id="status"></p>`: Kullanıcıya yükleme durumu, hata ya da başarı mesajı göstermek için kullanılır.
- `<div id="videoContainer">`: İşlenen videonun görsel çıktısını göstermek için oluşturulmuştur.

CSS Stili:

- Renk değişkenleri `:root` içinde tanımlanmıştır. Bu, sayfa genelinde tutarlı renk kullanımı sağlar.
- `.container`, `.upload-section`, ve `#videoStream` gibi sınıflarla kullanıcı arayüzü görsel olarak şık ve responsive (mobil uyumlu) hale getirilmiştir.
- Butonlar ve dosya seçici için hover ve aktif efektler ile kullanıcı deneyimi geliştirilmiştir.
- `@media (max-width: 768px)` kısmıyla mobil cihazlara özel düzenlemeler yapılmıştır.

JavaScript Fonksiyonu (upload):

Kodun sonunda yer alan JavaScript kısmı, video dosyasının yüklenmesi ve işlenmesi sürecini yönetir:

- upload() fonksiyonu çağrıldığında önce dosya seçilip seçilmediği kontrol edilir.
- Dosya varsa FormData nesnesine eklenir ve /upload adresine POST isteği gönderilir.
- Sunucudan başarı yanıtı alınırsa videoStream alanına /video\_feed adresinden gelen görsel veri gösterilir.
- İşlem sırasında ve sonrasında kullanıcıya bilgi mesajları gösterilir.
- Hata durumunda kullanıcı uyarılır.

Sunucu Tarafı:

- /upload: Backend (Flask) bu POST isteğini karşılayıp videoyu işler.
- /video\_feed: İşlenen videonun çıktısını sürekli olarak (örneğin bir etiketleme sonucu) sağlayan bir akış URL'sidir.

## Main kod (app.py)

Python tabanlı web uygulaması, Flask framework'ü kullanılarak geliştirilmiş ve bir video dosyasının Amazon S3'e yüklenmesi, AWS Rekognition servisi ile etiketlenmesi, bu etiketlerin MongoDB'ye kaydedilmesi ve sonuçların görsel olarak kullanıcıya sunulması işlemlerini gerçekleştirmektedir.

Gerekli modüller ve yardımcı dosyalar içe aktarılır. mongo\_handler, s3\_uploader, rekognition\_handler, video\_drawer gibi dosyalar özel görevler için modülerleştirilmiş durumdadır.

```
app.py > ...
1 from flask import Flask, request, jsonify, render_template, Response
2 from flask_cors import CORS
3 import os
4 from mongo_handler import MongoDBHandler
5 from s3_uploader import upload_video_to_s3
6 from rekognition_handler import start_label_detection, get_label_detection_result, save_result_to_file
7 from video_drawer import draw_labels_on_video
8 import atexit
9 import signal
10 import sys
```

Flask uygulaması oluşturulur. CORS aktif edilerek farklı portlardan gelen istekler kabul edilir.

Video dosyalarının yükleneceği uploads klasörü tanımlanır ve yoksa oluşturulur.

MongoDB ile bağlantı kurmak için MongoDBHandler sınıfından bir nesne oluşturulur.

Uygulamada geçici olarak kullanılacak video ve etiket dosya yolları current\_video\_path ve current\_json\_path de tutulur.

```

app = Flask(__name__)
CORS(app)

# Yapılandırma
UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# MongoDB bağlantısı
mongo_handler = MongoDBHandler()

# Global değişkenler
current_video_path = None
current_json_path = None

```

Kaynak temizlenir.

```

def cleanup(signum=None, frame=None):
    """Kaynakları temizle ve bağlantıyı kapat"""
    try:
        if mongo_handler:
            mongo_handler.close_connection()
            print("\nKaynaklar temizlendi, uygulama kapatılıyor...")
    except Exception as e:
        print(f"Temizleme sırasında hata: {e}")

    if signum is not None:
        sys.exit(0)

```

Uygulama kapatılırken MongoDB bağlantısı düzgün bir şekilde sonlandırılır.

Ctrl+C (SIGINT) veya sistemden gelen sonlandırma sinyalleri (SIGTERM) yakalanarak cleanup fonksiyonu çalıştırılır.

```

# Çıkış sinyallerini yakala
atexit.register(cleanup)
signal.signal(signal.SIGINT, cleanup)
signal.signal(signal.SIGTERM, cleanup)

```

Ana sayfa için index.html şablonu döndürülür.

```

@app.route('/')
def index():
    return render_template('index.html')

```

Kullanıcı video yüklediğinde bu endpoint çalışır.

Global değişkenler güncelleneceği için global olarak tanımlanır.

```
@app.route('/upload', methods=['POST'])
def upload():
    global current_video_path, current_json_path
```

Formda 'video' alanı yoksa hata döndürülür.

Dosya seçilmediyse hata döndürülür.

```
if 'video' not in request.files:
    return jsonify({"error": "Video dosyası bulunamadı"}), 400

file = request.files['video']
if file.filename == '':
    return jsonify({"error": "Dosya seçilmedi"}), 400
```

Dosya uploads klasörüne kaydedilir.

AWS işlemleri için gerekli dosya adları belirlenir.

JSON dosyası MongoDB'ye kaydedilir.

Global değişkenler güncellenir.

Kullanıcıya başarılı yanıt ve veritabanı ID'si döndürülür.

```
try:
    video_filename = file.filename
    video_path = os.path.join(UPLOAD_FOLDER, video_filename)
    file.save(video_path)

    # AWS işlemleri
    bucket_name = 'video-akisi-bucket'
    s3_key = video_filename
    json_filename = os.path.splitext(video_filename)[0] + '_labels.json'
    json_path = os.path.join(UPLOAD_FOLDER, json_filename)

    upload_video_to_s3(video_path, bucket_name, s3_key)
    job_id = start_label_detection(bucket_name, s3_key)
    result = get_label_detection_result(job_id)
    save_result_to_file(result, json_path)

    # MongoDB'ye kaydet
    mongo_id = mongo_handler.save_analysis_results(video_filename, json_path)

    # Global değişkenleri güncelle
    current_video_path = video_path
    current_json_path = json_path

    return jsonify({
        "success": True,
        "message": "İşlem başarıyla tamamlandı",
        "mongo_id": str(mongo_id)
    })
```

```

except Exception as e:
    return jsonify({
        "error": str(e),
        "message": "İşlem sırasında hata oluştu"
    }), 500

```

Önce video ve JSON dosyasının yüklü olup olmadığı kontrol edilir.

OpenCV kullanarak video üzerine etiketler çizilir ve tarayıcıya canlı akış yapılır.

```

@app.route('/video_feed')
def video_feed():
    global current_video_path, current_json_path

    if not current_video_path or not current_json_path:
        return jsonify({"error": "Video veya etiket dosyası bulunamadı"}), 404

    try:
        return Response(
            draw_labels_on_video(current_video_path, current_json_path),
            mimetype='multipart/x-mixed-replace; boundary=frame'
        )
    except Exception as e:
        return jsonify({"error": str(e)}), 500

```

MongoDB'den tüm analiz sonuçları getirilir.

```

@app.route('/results')
def get_all_results():
    try:
        results = mongo_handler.get_results()
        return jsonify({
            "success": True,
            "results": results
        })
    except Exception as e:
        return jsonify({"error": str(e)}), 500

```

Verilen video\_name parametresi ile MongoDB'den tek bir analiz sonucu döndürülür.

```

@app.route('/results/<video_name>')
def get_single_result(video_name):
    try:
        result = mongo_handler.get_results(video_name=video_name)
        if result:
            return jsonify({
                "success": True,
                "result": result
            })
        return jsonify({
            "success": False,
            "message": "Sonuç bulunamadı"
        }), 404
    except Exception as e:
        return jsonify({"error": str(e)}), 500

```

Flask sunucusu başlatılır.

Hata oluşursa loglanır.

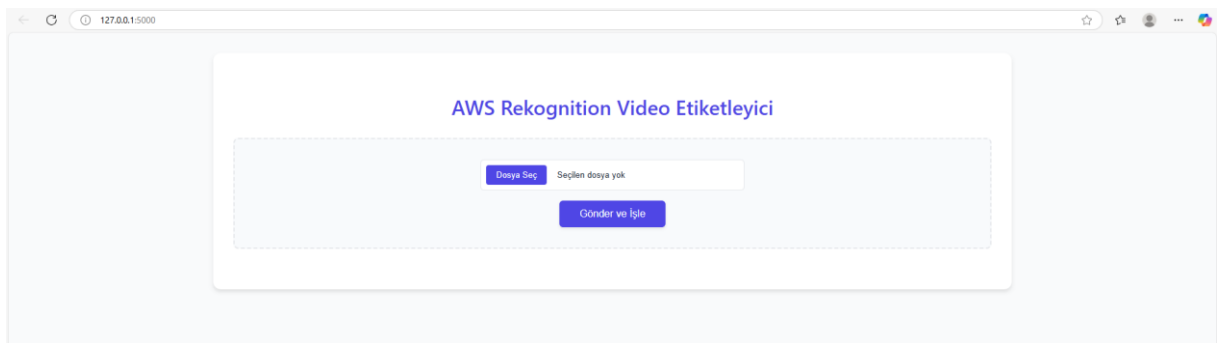
Uygulama sonlandığında kaynaklar temizlenir.

```
if __name__ == '__main__':  
    try:  
        app.run(  
            host='0.0.0.0',  
            port=5000,  
            debug=True,  
            use_reloader=False  
        )  
    except Exception as e:  
        print(f"Uygulama hatası: {e}")  
    finally:  
        cleanup()
```

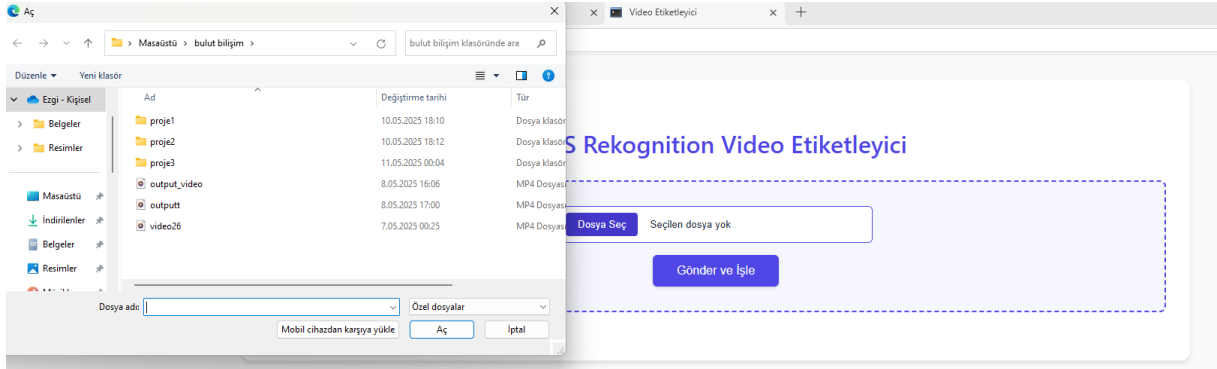
### Uygulamanın Çalışması

app.py dosyası çalıştırılır. Terminaldeki <http://127.0.0.1:5000> adresine Ctrl +click ile erişilir.

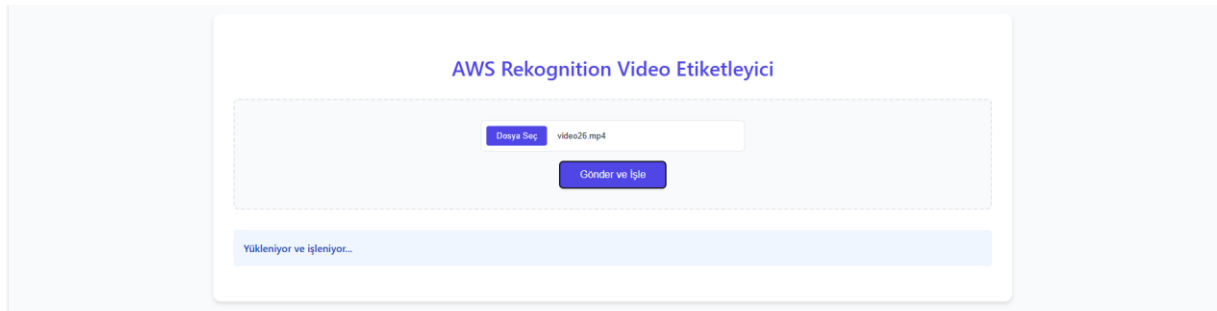
```
PS C:\Users\ezgis\Desktop\bulut bilişim\proje3> python -u "c:\Users\ezgis\Desktop\bulut bilişim\proje3\app.py"  
* Serving Flask app 'app'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on all addresses (0.0.0.0)  
* Running on http://127.0.0.1:5000  
* Running on http://192.168.1.3:5000  
Press CTRL+C to quit  
█
```



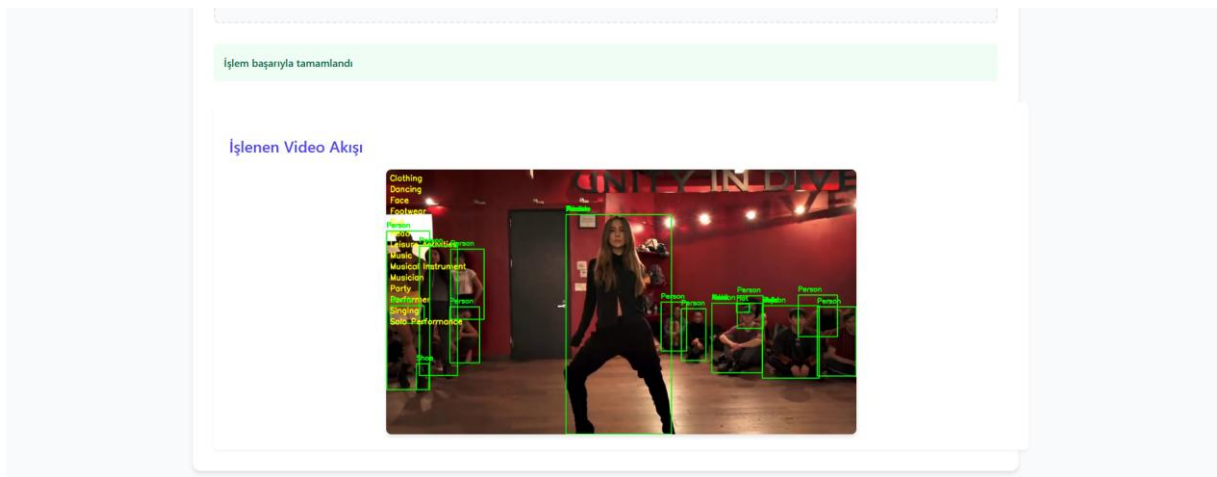
Dosya Seç butonundan dosya seçilir.



Gönder ve İşle butonuna basılıp videonun işlenmesi beklenir.



İşlenen video akışı ekranda gösterilir.



Ctrl+C ile uygulama kapatılır.

```
Press CTRL+C to quit
127.0.0.1 - - [12/May/2025 00:13:35] "GET / HTTP/1.1" 200 -
uploads\video26.mp4 dosyası video-akisi-bucket/video26.mp4 olarak yüklendi.
AWS Rekognition sonucu bekleniyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Bekleniyor... İşlem devam ediyor...
Etiketleme tamamlandı ✓
Etiketler uploads\video26_labels.json dosyasına kaydedildi.
127.0.0.1 - - [12/May/2025 00:16:10] "POST /upload HTTP/1.1" 200 -
127.0.0.1 - - [12/May/2025 00:16:10] "GET /video_feed HTTP/1.1" 200 -

Kaynaklar temizlendi, uygulama kapatılıyor...

Kaynaklar temizlendi, uygulama kapatılıyor...

Kaynaklar temizlendi, uygulama kapatılıyor...
PS C:\Users\ezgis\Desktop\bulut bilişim\proj3>
```

Oluşturduğumuz bucket’de uygulamaya yüklediğimiz video26.mp4 videosunu göreceğiz.

video-akisi-bucket [Info](#)

Objects	Metadata	Properties	Permissions	Metrics	Management	Access Points
<b>Objects (8)</b> <a href="#">Copy S3 URI</a> <a href="#">Copy URL</a> <a href="#">Download</a> <a href="#">Open</a> <a href="#">Delete</a> <a href="#">Actions</a> <a href="#">Create folder</a> <a href="#">Upload</a>						
Objects are the fundamental entities stored in Amazon S3. You can use <a href="#">Amazon S3 inventory</a> to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. <a href="#">Learn more</a>						
<input type="text" value="Find objects by prefix"/>						
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class	
<input type="checkbox"/>	<a href="#">video21.mp4</a>	mp4	May 11, 2025, 00:18:44 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	<a href="#">video22.mp4</a>	mp4	May 11, 2025, 00:41:26 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	<a href="#">video23.mp4</a>	mp4	May 11, 2025, 00:49:10 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	<a href="#">video24.mp4</a>	mp4	May 11, 2025, 14:48:00 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	<a href="#">video25.mp4</a>	mp4	May 11, 2025, 15:36:06 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	<a href="#">video26.mp4</a>	mp4	May 12, 2025, 00:15:20 (UTC+03:00)	8.1 MB	Standard	
<input type="checkbox"/>	<a href="#">video28.mp4</a>	mp4	May 11, 2025, 00:24:50 (UTC+03:00)	13.7 MB	Standard	
<input type="checkbox"/>	<a href="#">video32.mp4</a>	mp4	May 11, 2025, 00:30:36 (UTC+03:00)	5.4 MB	Standard	

Aws Rekognition’un video26.mp4 üzerinde yaptığı tespitleri Json formatında kaydetmiştik. Bu verileri MongoDB’ye kaydettik.

Create Database

Search Namespaces

projel2

sample\_mflix

projel2.projel2

STORAGE SIZE: 504KB LOGICAL DATA SIZE: 2.04MB TOTAL DOCUMENTS: 7 INDEXES TOTAL SIZE: 34KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

Filter Type a query: { field: 'value' } Reset Apply Options

QUERY RESULTS: 8-8 OF 8

```
{
  "_id": "68211397810608e1d6f8e7bb",
  "video_name": "video26.mp4",
  "analysis_date": "2025-05-12T00:16:07.738+00:00",
  "Labels": {
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
      "Codec": "h264",
      "DurationMillis": 103480,
      "Format": "QuickTime / MOV",
      "FrameRate": 25,
      "FrameHeight": 720,
      "FrameWidth": 1280,
      "ColorRange": "LIMITED",
      "NextToken": "rhk9KX3Rnw7N0LTgp/Wbpctpy0h0AwZokwXBW4Bq8fpr7Kue3BRZLK3yWYJC+UR7xT/_",
      "Labels": [
        {
          "Timestamp": 40,
          "Label": {
            "Name": "Accessories"
          }
        }
      ]
    }
  }
}
```

PREVIOUS 8 of 8 results NEXT



```
_id: ObjectId('68211397810608e1d6f8e7bb')
video_name: "video26.mp4"
analysis_date: 2025-05-12T00:16:07.738+00:00
Labels: Object
  JobStatus: "SUCCEEDED"
  VideoMetadata: Object
    Codec: "h264"
    DurationMillis: 103480
    Format: "QuickTime / MOV"
    FrameRate: 25
    FrameHeight: 720
    FrameWidth: 1280
    ColorRange: "LIMITED"
    NextToken: "RhKFWXK3Rnw7NOLTgp/WbpctpbyDhDAwIoW+X8M4Bqb8FpR7Kue3BRZLX3yWY3E+UR7xf/..."
  Labels: Array (1000)
    0: Object
      Timestamp: 40
      Label: Object
        Name: "Accessories"
```

 PREVIOUS

8 of 8 results

NEXT 

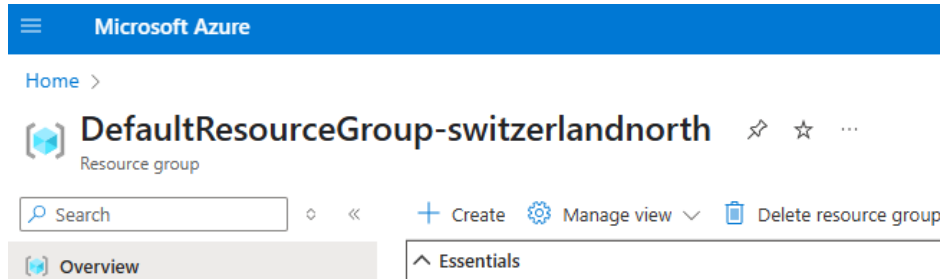
## Proje: Diyabet Tahmin Uygulaması

**Projenin Amacı:** Bu projenin amacı, kullanıcıdan alınan sağlık verilerine göre diyabet hastalığına yakalanma riskini tahmin eden bir uygulama geliştirmektir. Model, Microsoft Azure üzerinde eğitilmiş ve deploy edilmiştir. Uygulama, kullanıcı verilerini bu modele göndererek gerçek zamanlı tahminler sunar.

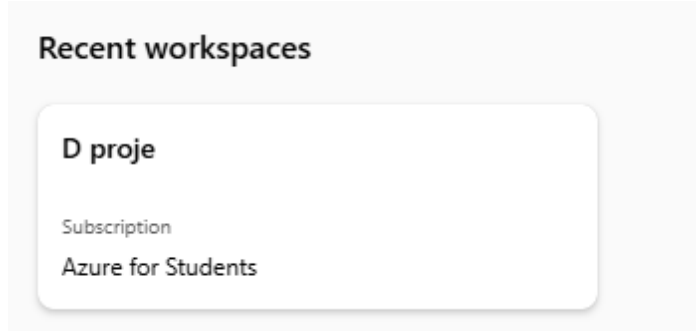
## GELİŞTİRME SÜRECİ

### Azure

Microsoft Azure hesabı açıldı. Ana ekrandaki Azure Services kısmından “Create a resource” denilerek resource group oluşturuldu.



Azure AI Machine Learning Studio'ya girilip Azure Portal bilgileri ile giriş yapıldı. Burada “Create Workspace” diyerek çalışma alanı oluşturuyoruz. Oluştururken “Resource Group” seçeneğini daha önce oluşturduğumuz “DefaultResourceGroup-switzerlandnorth” olarak seçeceğiz.



Oluşturduğumuz Workspace'e giriş yapıyoruz. Manage kısmından Compute' a tıklayarak sanal makine oluşturuyoruz.

Name	☆	State	Idle shutdown ⓘ
sankirezgi163		Running 🔄	1 hour

Ardından Assets kısmından Data'ya giriyoruz. Create dataset diyerek diabetes.csv dosyamızı buraya yüklüyoruz.

DiabetesData	1	workspaceblobstore	Jun 27, 2025 5:32 PM
--------------	---	--------------------	----------------------

Authoring kısmından Notebook açıyoruz. Klasör oluşturuyoruz. Bu dosyanın içerisine Python dosyası oluşturuyoruz. Bu Python dosyasında Data kısmında oluşturduğumuz veri setine bağlanarak makine öğrenmesi gerçekleştireceğiz.

```
1 import pandas as pd
2 import mlflow
3 import mlflow.sklearn
4 from sklearn.model_selection import train_test_split
5 from sklearn.tree import DecisionTreeClassifier
6 from sklearn.metrics import accuracy_score
7
8 df = pd.read_csv("azureml://subscriptions/7b8751c6-1750-45ea-96b0-a951a2b76215/resourcegroups/defaultresourcegroup-switzerlandnorth/workspaces/D-proje/datasto
9
10 df.fillna(df.mean(), inplace=True)
11
12 X = df.drop("Outcome", axis=1).astype("float64")
13 y = df["Outcome"]
14
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
16
17 mlflow.set_experiment("Demo-Publish-Endpoint")
18 mlflow.sklearn.autolog() # run bloğunun dışına taşıdık
19
20 with mlflow.start_run():
21     model = DecisionTreeClassifier(random_state=42)
22     model.fit(X_train, y_train)
23
24     y_pred = model.predict(X_test)
25     acc = accuracy_score(y_test, y_pred)
26     print("Doğruluk (Accuracy):", acc)
27
28
```

Assets >Jobs kısmından erişiyoruz.

Display name (1 visualized)	Parent job name	Status	Created on ↓
joyful_truck_vy10rxw7		Completed	Jun 27, 2025 5:50 PM

Aynı sayfadan Register model kısmından modelimizi oluşturuyoruz. Oluşturduğumuz Model Assets>Models kısmında gözüküyor.

Name	☆	Version	Type	Source
DiabetesModel		1	MLFLOW	This workspace

Bu modele tıklayıp Real-time endpoint seçeneğine tıklıyoruz ve endpoint oluşturuyoruz.

## DiabetesModel:1

Details Versions Artifacts Endpoints Jobs Data Feature sets Responsible AI

Refresh Archive Use this model Download all Share model

Attributes

Name

DiabetesModel

Version

1

Created on

Jun 27, 2025 5:47 PM

Real-time endpoint

Deploy the model using the real-time endpoint wizard

Batch endpoint

Deploy the model using the batch endpoint wizard

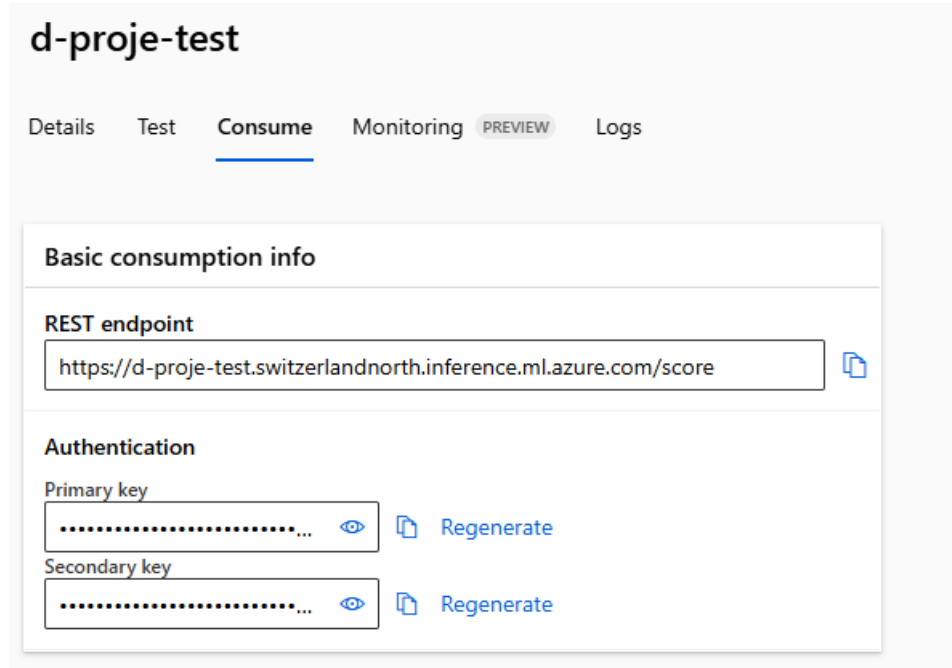
Web service

Deploy to a web service (only for models based on frameworks)

Oluşturduğumuz endpoint'i Assets>Endpoints kısmından görebiliriz.

Name	☆	Description	Quota type	Created on
d-proje-test			Dedicated	Jun 27, 2025 5:48 PM

Kodumuzda kullanacağımız key'leri alıyoruz.

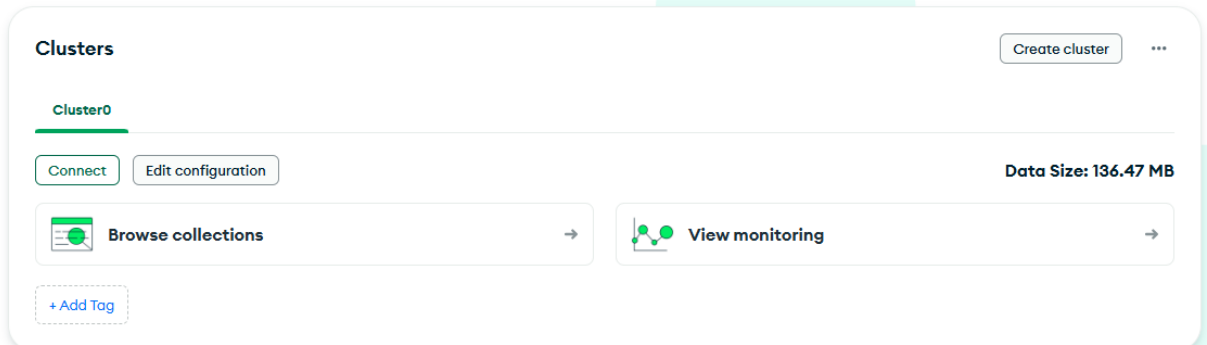


## MongoDb

MongoDB Atlas hesabı açıldı. **Proje** oluşturuldu. Cluster oluşturuldu. (Cluster 0)

ANKARA UNIVERSITY > PROJECT 0

### Overview



Connect kısmına girip, ardından Drivers kısmına girip Cluster'a bağlanabilmek için URI alındı.

```
mongodb+srv://<db_username>:<db_password>@cluster0.18atnrk.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
```

Collection ve Database oluşturuldu. Verilerimiz buraya kaydedilecek.

+ Create Database

Q Search Namespaces

projel2

projel3

projel3

sample\_mflix

### projel3

LOGICAL DATA	STORAGE	INDEX SIZE:	TOTAL
SIZE:	SIZE:		COLLECTIONS:
123B	36KB	36KB	1

CREATE

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
projel3	1	123B	123B	36KB	1	36KB	36KB

## KODLAR

### app.py

Uygulamamızı bu dosyayı çalıştırarak başlatacağız.

Gerekli kütüphaneler içe aktarılır.

```
app.py > predict
1 from flask import Flask, request, jsonify, render_template
2 import requests
3 import os
4 from pymongo import MongoClient
```

- Flask: Web uygulamasını oluşturmak için kullanılır.
- request: HTTP isteklerini almak için kullanılır.
- jsonify: JSON formatında cevap döndürmek için.
- render\_template: HTML dosyalarını yüklemek için.
- requests: Azure API'sine HTTP isteği göndermek için.
- pymongo: MongoDB ile bağlantı kurmak için.

MongoDB bağlantısı kurulur.

```
# MongoDB bağlantısı
MONGO_URI = "mongodb+srv://user11:test11@cluster0.18atnrk.mongodb.net/?retrywrites=true&w=majority&appName=Cluster0"
client = MongoClient(MONGO_URI)

# Veritabanı ve koleksiyon seç
db = client["projel3"]
collection = db["projel3"]
```

Flask uygulaması başlatılır.

```
app = Flask(__name__)
```

Azure üzerinde eğitilen makine öğrenmesi modelinin endpoint'i ve API anahtarı yazılır.

```
AZURE_ENDPOINT = "https://d-proje-test.switzerlandnorth.inference.ml.azure.com/score" # örnek: https://  
AZURE_API_KEY = "5twuxkYoXwITTs7aJVNLjKsArIwzL9ECgpzM2PhgH51H6oG6CTHJQQJ99BFAAAAAAAAAAAAINFRAZML4Cxz"
```

Azure API'sine yapılacak isteklerde gerekli olan header bilgileri belirlenir. JSON formatı ve kimlik doğrulama için API anahtarı kullanılır.

```
headers = {  
    "Content-Type": "application/json",  
    "Authorization": f"Bearer {AZURE_API_KEY}"  
}
```

Anasayfa tanımlanır. Kullanıcı siteye girdiğinde index.html dosyası tarayıcıda görüntülenir.

```
@app.route("/")  
def index():  
    return render_template("index.html")
```

/predict adresine POST isteği gönderildiğinde çalışacak olan predict() fonksiyonu tanımlanır.

```
@app.route("/predict", methods=["POST"])  
def predict():  
    # form
```

Kullanıcıdan gelen veriler JSON formatında alınır. Azure modeli için gerekli giriş formatı oluşturulur. Bu kısımda kullanıcıdan alınan veriler belirli bir sıraya göre bir liste içine yerleştirilir.

```

try:
    data = request.get_json()

    payload = {
        "input_data": {
            "columns": [
                "Pregnancies",
                "Glucose",
                "BloodPressure",
                "SkinThickness",
                "Insulin",
                "BMI",
                "DiabetesPedigreeFunction",
                "Age"
            ],
            "index": [0],
            "data": [[
                data["pregnancies"],
                data["glucose"],
                data["bloodPressure"],
                data["skinThickness"],
                data["insulin"],
                data["bmi"],
                data["diabetesPedigreeFunction"],
                data["age"]
            ]]
        }
    }

```

Azure modeline POST isteği gönderilir. `Raise_for_status()` hata varsa istisna ortaya çıkartır. Gelen sonuç JSON formatına dönüştürülür.

```

response = requests.post(AZURE_ENDPOINT, headers=headers, json=payload)
response.raise_for_status()
result = response.json()

```

Azure'dan gelen yanıtın biçimi kontrol edilir. Eğer liste biçimindeyse sadece tahmin değeri alınır. Eğer sözlük (dict) biçimindeyse `predicted_label` ve `probability` (olasılık) değerleri alınır. Beklenmeyen formatta yanıt alınırsa hata döndürülür.

```

if isinstance(result, list):
    prediction = result[0]
    probability = None
elif isinstance(result, dict) and "result" in result:
    prediction = result["result"][0].get("predicted_label")
    probability = result["result"][0].get("probability")
else:
    return jsonify({"error": "Beklenmeyen yanıt formatı", "received_response": result}), 500

```

Kullanıcının girdiği veriler ve tahmin sonucu `save_data` sözlüğüne aktarılır. Bu veri MongoDB veritabanına kaydedilir.



```
# MongoDB'ye veriyi kaydet
save_data = {
    "input": payload["input_data"]["data"][0],
    "prediction": prediction,
    "probability": probability
}
collection.insert_one(save_data)
```

Sonuçlar (tahmin ve varsa olasılık) JSON formatında kullanıcıya döndürülür.

```
return jsonify({
    "prediction": prediction,
    "probability": probability
})
```

Azure API isteği sırasında veya başka bir hata oluşursa uygun hata mesajı verilir ve 500 hata kodu döndürülür.

```
except requests.exceptions.RequestException as e:
    print("API İsteği Hatası:", str(e))
    return jsonify({"error": f"API isteği başarısız: {str(e)}"}), 500
except Exception as e:
    print("Hata oluştu:", str(e))
    return jsonify({"error": str(e)}), 500
```

Uygulama doğrudan çalıştırıldığında Flask sunucusu başlatılır. debug=True, geliştirme aşamasında hataların kolayca görülmesini sağlar.

```
if __name__ == "__main__":
    app.run(debug=True)
```

## index.html

Bu dosya, kullanıcıların diyabet riskini hesaplayabileceği bir web arayüzüdür. Temiz ve anlaşılır bir form tasarımıyla 8 farklı sağlık parametresini sorgular. Arka planda çalışan JavaScript kodu, kullanıcı girdilerini alıp sunucuya göndererek tahmin sonucunu ekranda gösterir. Form gönderildiğinde JavaScript ile /predict endpoint'ine AJAX isteği gönderir ve sonucu dinamik olarak görüntüler.

## style.css

Form alanlarına odaklanma efektleri ve tahmin sonuçları için özel tasarlanmış görsel vurgular bulunur. Özellikle diyabet risk sonuçları için "pozitif/negatif" durumlara özel renk kodlaması yapılmıştır.

## UYGULAMANIN ÇALIŞMASI

app.py dosyası çalıştırılır. <http://127.0.0.1:5000> adresine tıklanır.

```
PS C:\Users\ezgis\Desktop\bulut bilişim\proje6> python -u "c:\Users\ezgis\Desktop\bulut bilişim\proje6\app.py"
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 852-342-102
```

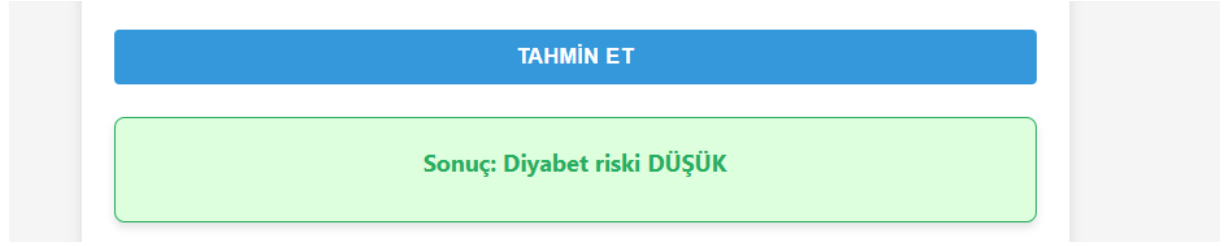
Uygulama açılır ve bilgiler girilir.

### Diyabet Risk Tahmin Uygulaması

- Hamilelik Sayısı:
- Glukoz Seviyesi (mg/dL):
- Kan Basıncı (mm Hg):
- Cilt Kalınlığı (mm):
- İnsulin Seviyesi (mu U/ml):
- Vücut Kitle İndeksi (kg/m<sup>2</sup>):
- Diyabet Soy Ağacı Fonksiyonu:
- Yaş:

TAHMİN ET

TAHMİN ET butonuna basılır. Eğittiğimiz modele göre bu verilere sahip birinin Diyabet riskinin olup olmadığı sonucuna varırız.



Girdiğimiz bilgiler ve sonuç veritabanımızda gözükür. Prediction 0 ise diyabet riski düşüktür,1 ise yüksektir.

#### QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('68640a71ccf2f7c7841c2a00')
input: Array (8)
  0: 13
  1: 106
  2: 72
  3: 54
  4: 125
  5: 36.6
  6: 0.178
  7: 45
prediction: 0
probability: null
```

# E-Ticaret Projesi Teknik Raporu

## 1. Proje Özeti

- **Proje Adı:** E-Ticaret Web Uygulaması
- **Kullanılan Teknolojiler:** Django, PostgreSQL, AWS
- **Amaç:** Django tabanlı bir e-ticaret sitesi geliştirdik ve AWS üzerinde yayınladık. Projenin amacı, ölçeklenebilir ve yüksek erişilebilirliğe sahip bir e-ticaret platformu oluşturmaktır.

## 2. Kullanılan Teknolojiler

### 2.1 Backend Geliştirme

- **Django Framework** kullanarak e-ticaret sitesi geliştirdik
- Ürün listeleme, detay görüntüleme ve sepet işlemleri için gerekli view'ları yazdık.
- PostgreSQL veritabanı ile ürün ve kullanıcı bilgilerini yönettik.
- Admin paneli oluşturduk ve superuser hesabı ile yönetimi sağladık

### 2.2 AWS Servisleri

- **EC2:** Web sunucusu olarak EC2 instance kullandık.
- **RDS:** PostgreSQL veritabanı için RDS kullandık.
- **S3:** Ürün resimleri için S3 bucket oluşturduk ve Django'ya entegre ettik.
- **Application Load Balancer:** Trafiği yönetmek için Application Load Balancer kurduk. (Yük dengeleme.)
- **Auto Scaling Group:** Yük arttığında otomatik olarak yeni instance'lar başlatacak şekilde ayarladık. (Otomatik ölçeklendirme.)
- **CloudWatch:** Metrik izleme sağladık.
- **IAM:** Güvenlik ve yetkilendirme sağladık.

### 3. Mimari Yapı

#### 3.1 Uygulama Katmanı

- Django web uygulaması.
- Statik ve medya dosyaları için S3 entegrasyonu.
- Veritabanı bağlantısı için RDS kullanımı.

#### 3.2 Altyapı Katmanı

- Load Balancer ile trafik yönetimi.
- Auto Scaling ile otomatik ölçeklendirme.

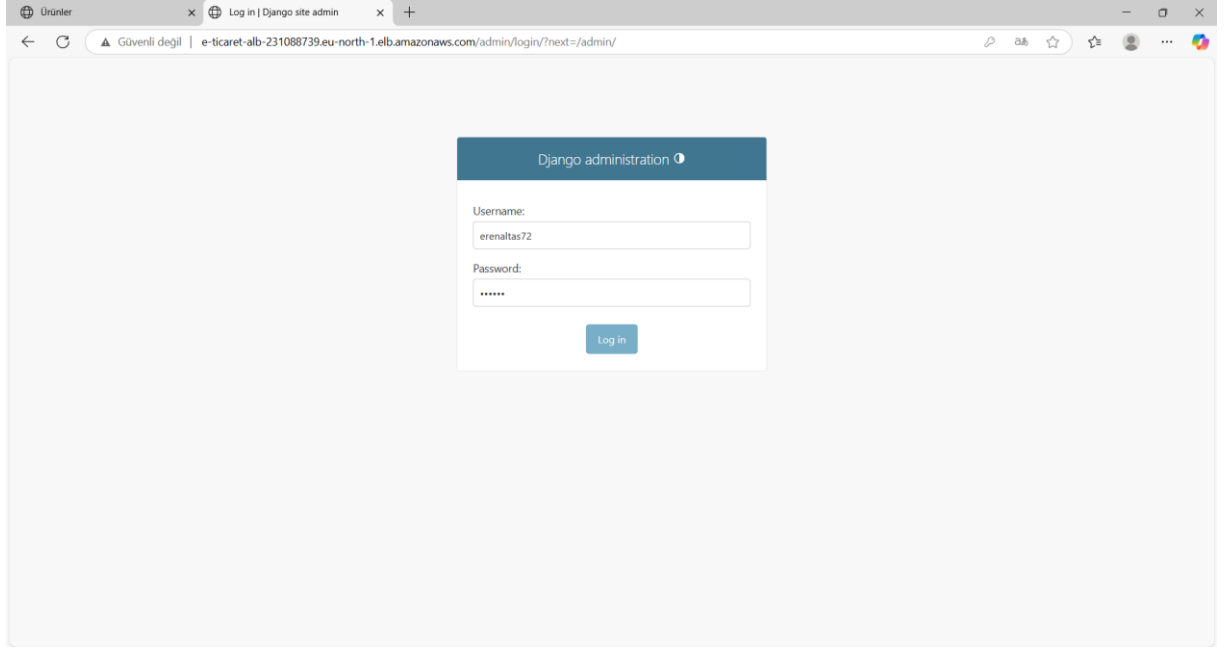
### 4. Öğrenilen Kavramlar ve Teknolojiler

- **Ölçeklendirme:** Sistemin yüke göre otomatik olarak büyüüp küçülmesi.
- **Yük Dengeleme:** Trafiğin instance'lar arasında dengeli dağıtılması.
- **Health Check:** Instance'ların sağlık durumunun kontrol edilmesi.
- **Cooldown Period:** Ölçeklendirme işlemleri arasındaki bekleme süresi.
- **IAM:** Güvenlik yönetimi. IAM rolleri ve politikaları.
- Security Group yönetimi.
- Veritabanı güvenliği.
- S3 bucket güvenliği.

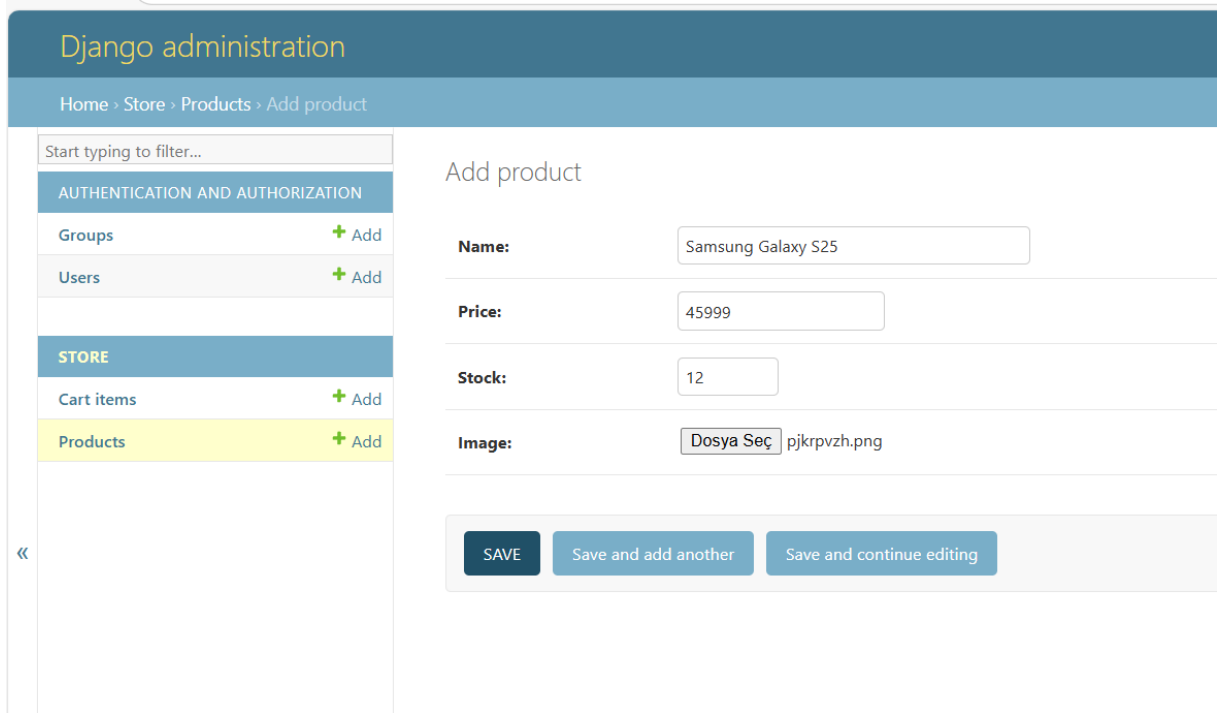
### 5. Proje Kazanımları

- AWS servislerinin pratik kullanımı.
- Ölçeklenebilir sistem tasarımı.
- DevOps pratiklerinin uygulanması.
- Cloud mimarisi deneyimi.
- Sistem yönetimi deneyimi.
- Yük testi ve performans optimizasyonu.
- Güvenlik yönetimi.

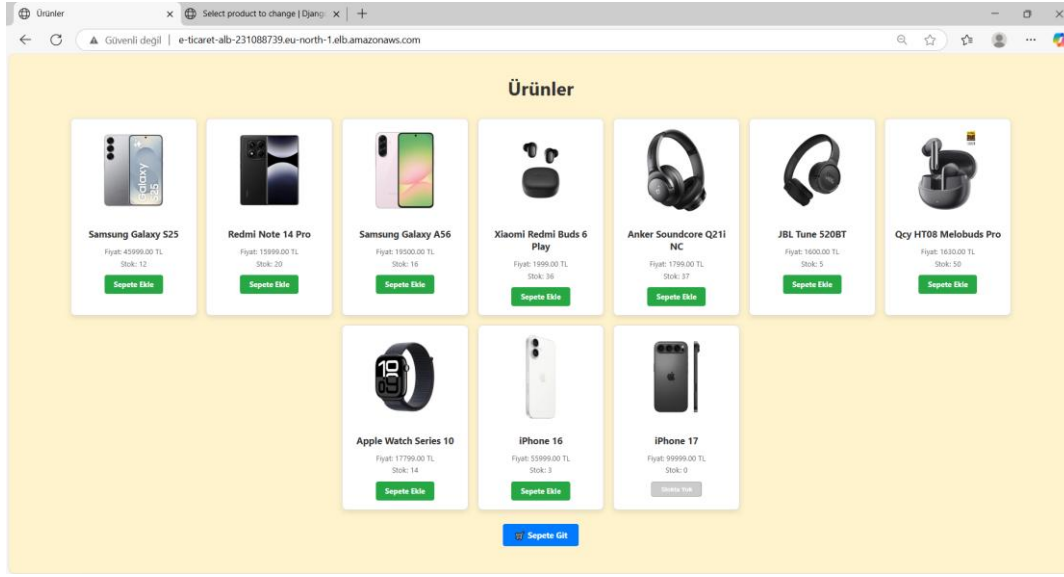
Şekil 1-Admin paneli giriş ekranı



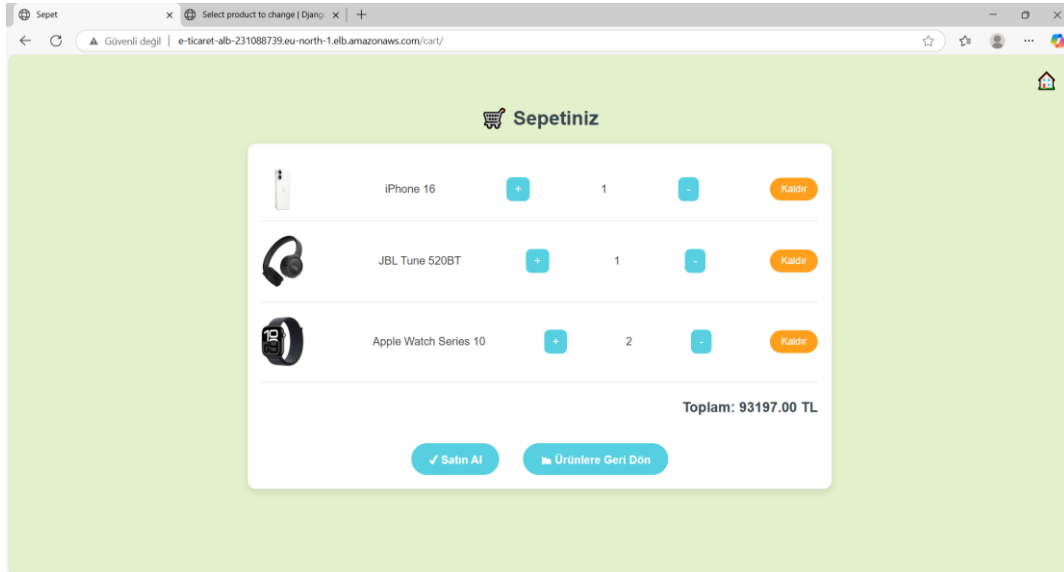
Şekil 2- Admin Paneli Ürün ekleme ekranı.



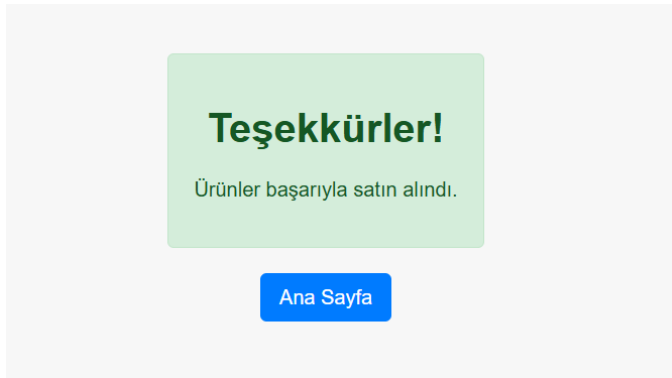
Şekil 3- Ürünlerin yer aldığı ana sayfa.



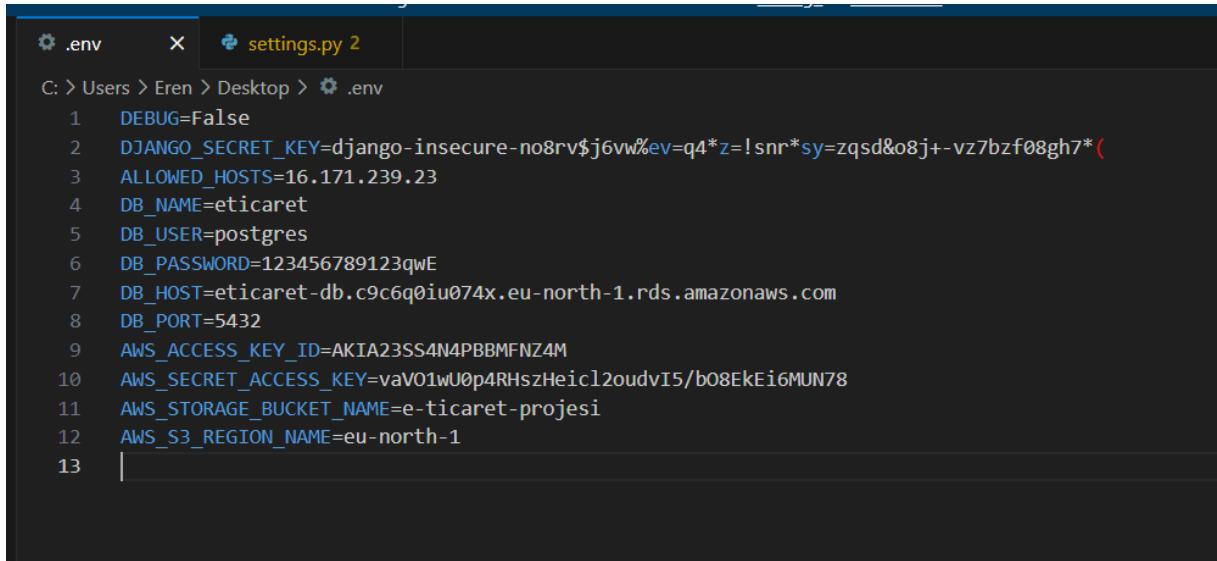
Şekil 4- Sepet sayfası.



Şekil 5- Satın alma ekranı.

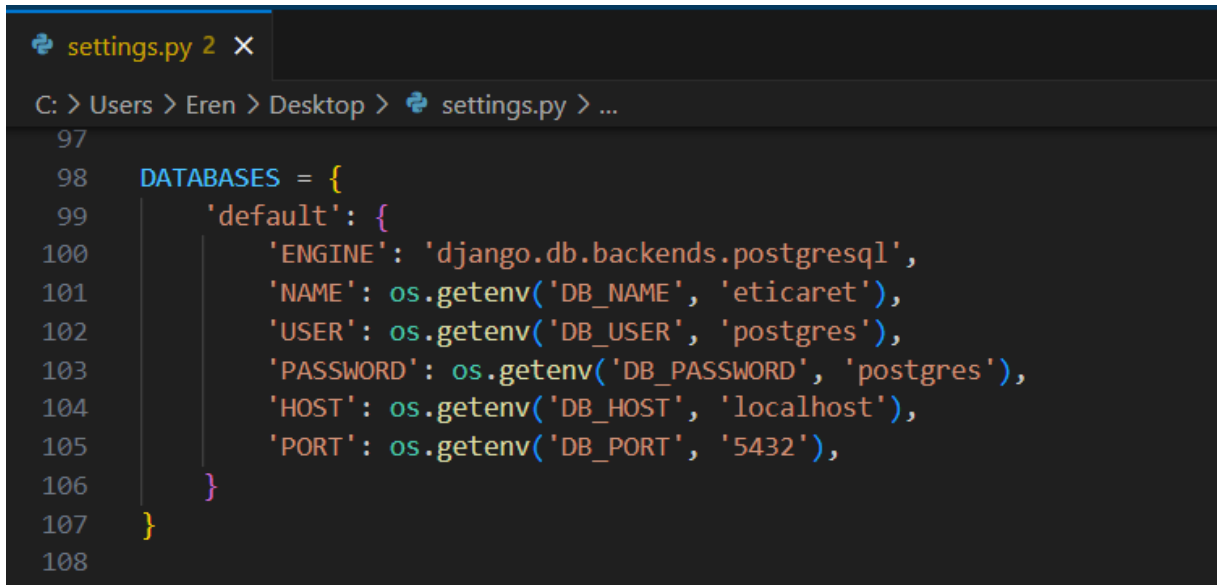


Şekil 6- .env dosyasının içeriği.



```
.env
1  DEBUG=False
2  DJANGO_SECRET_KEY=django-insecure-no8rv$j6vw%ev=q4*z=!snr*sy=zqsd&o8j+-vz7bzf08gh7*(
3  ALLOWED_HOSTS=16.171.239.23
4  DB_NAME=eticaret
5  DB_USER=postgres
6  DB_PASSWORD=123456789123qwE
7  DB_HOST=eticaret-db.c9c6q0iu074x.eu-north-1.rds.amazonaws.com
8  DB_PORT=5432
9  AWS_ACCESS_KEY_ID=AKIA23SS4N4PBBMFNZ4M
10 AWS_SECRET_ACCESS_KEY=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
11 AWS_STORAGE_BUCKET_NAME=e-ticaret-projesi
12 AWS_S3_REGION_NAME=eu-north-1
13
```

Şekil 7- settings.py dosyasındaki database tanımları.



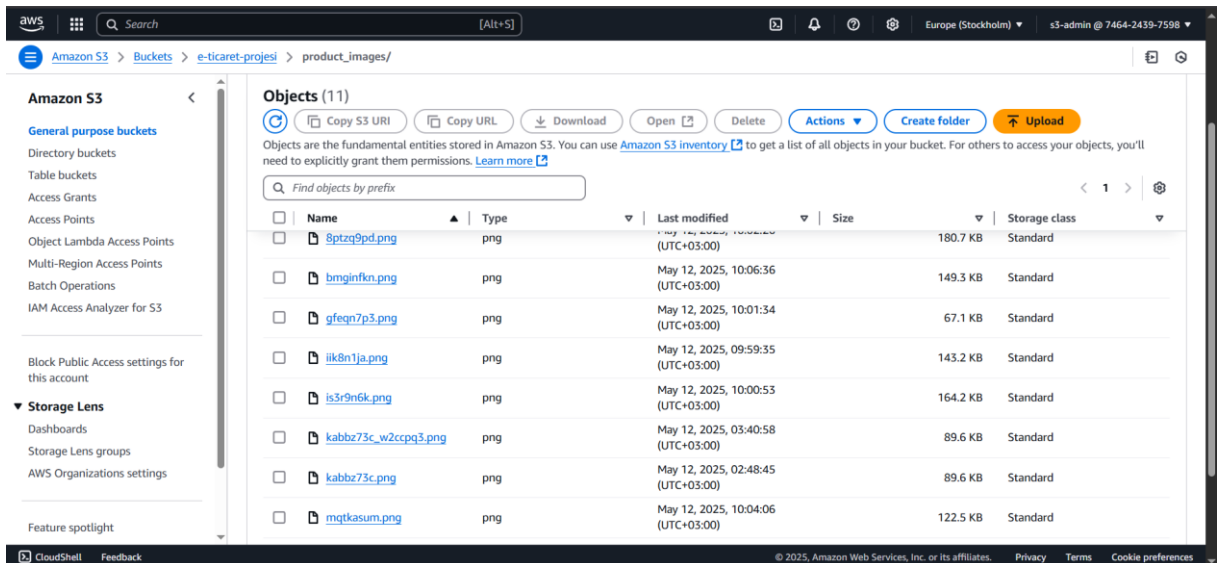
```
settings.py 2
C: > Users > Eren > Desktop > settings.py > ...
97
98  DATABASES = {
99      'default': {
100          'ENGINE': 'django.db.backends.postgresql',
101          'NAME': os.getenv('DB_NAME', 'eticaret'),
102          'USER': os.getenv('DB_USER', 'postgres'),
103          'PASSWORD': os.getenv('DB_PASSWORD', 'postgres'),
104          'HOST': os.getenv('DB_HOST', 'localhost'),
105          'PORT': os.getenv('DB_PORT', '5432'),
106      }
107  }
108
```



Şekil 8- settings.py dosyasındaki access\_key, S3 bucket ve media dosyalarının tanımları.

```
settings.py 2 X
C: > Users > Eren > Desktop > settings.py > ...
144 # AWS S3 Configuration
145 AWS_ACCESS_KEY_ID = os.getenv('AWS_ACCESS_KEY_ID')
146 AWS_SECRET_ACCESS_KEY = os.getenv('AWS_SECRET_ACCESS_KEY')
147 AWS_STORAGE_BUCKET_NAME = 'e-ticaret-projesi'
148 AWS_S3_REGION_NAME = 'eu-north-1'
149 AWS_S3_FILE_OVERWRITE = False
150 AWS_DEFAULT_ACL = None
151 AWS_S3_VERIFY = True
152 AWS_S3_CUSTOM_DOMAIN = f'{AWS_STORAGE_BUCKET_NAME}.s3.{AWS_S3_REGION_NAME}.amazonaws.com'
153
154 # Static and Media Files Configuration
155 STATICFILES_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
156 DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
157 STATIC_URL = f'https://{AWS_S3_CUSTOM_DOMAIN}/static/'
158 MEDIA_URL = f'https://{AWS_S3_CUSTOM_DOMAIN}/media/'
159
160 # Static and Media Files
161 #STATIC_URL = '/static/'
162 #STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
163 #MEDIA_URL = '/media/'
164 #MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
165
166 # AWS S3 Configuration
167 AWS_ACCESS_KEY_ID = os.getenv('AWS_ACCESS_KEY_ID')
168 AWS_SECRET_ACCESS_KEY = os.getenv('AWS_SECRET_ACCESS_KEY')
169 AWS_STORAGE_BUCKET_NAME = os.getenv('AWS_STORAGE_BUCKET_NAME')
170 AWS_S3_REGION_NAME = os.getenv('AWS_S3_REGION_NAME', 'eu-north-1')
171 AWS_S3_FILE_OVERWRITE = False
172 AWS_DEFAULT_ACL = None
173 AWS_S3_VERIFY = True
174
```

Şekil 9- Admin panelinden upload edilen ürün görsellerinin S3 bucket içinde depolanması.



Amazon S3 > Buckets > e-ticaret-projesi > product\_images/

Objects (11)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	8ptzq9pd.png	png	May 12, 2025, 10:06:36 (UTC+03:00)	180.7 KB	Standard
<input type="checkbox"/>	bmgjnfkn.png	png	May 12, 2025, 10:06:36 (UTC+03:00)	149.3 KB	Standard
<input type="checkbox"/>	gfegq7p3.png	png	May 12, 2025, 10:01:34 (UTC+03:00)	67.1 KB	Standard
<input type="checkbox"/>	ilk8n1ja.png	png	May 12, 2025, 09:59:35 (UTC+03:00)	143.2 KB	Standard
<input type="checkbox"/>	is3e9n6k.png	png	May 12, 2025, 10:00:53 (UTC+03:00)	164.2 KB	Standard
<input type="checkbox"/>	kabbz73c_w2ccpq3.png	png	May 12, 2025, 03:40:58 (UTC+03:00)	89.6 KB	Standard
<input type="checkbox"/>	kabbz73c.png	png	May 12, 2025, 02:48:45 (UTC+03:00)	89.6 KB	Standard
<input type="checkbox"/>	mqtkasum.png	png	May 12, 2025, 10:04:06 (UTC+03:00)	122.5 KB	Standard

Şekil 10- Bucket Ayarı (Block all public access = Off)

## Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), but you can turn on Block all public access. These settings apply only to this bucket and its contents. If you turn on Block all public access, your applications will work correctly without public access. If you require some level of public access for your applications, you must use some level of storage use cases. [Learn more](#)

### Block *all* public access

⚠ Off

► Individual Block Public Access settings for this bucket

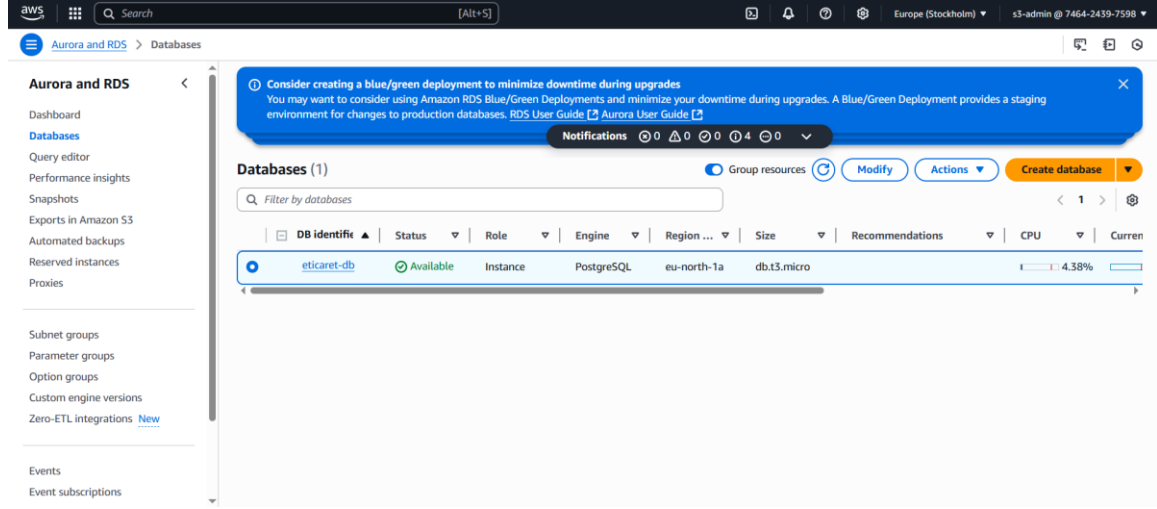
Şekil 11- CORS konfigürasyonu.

## Cross-origin resource sharing (CORS)

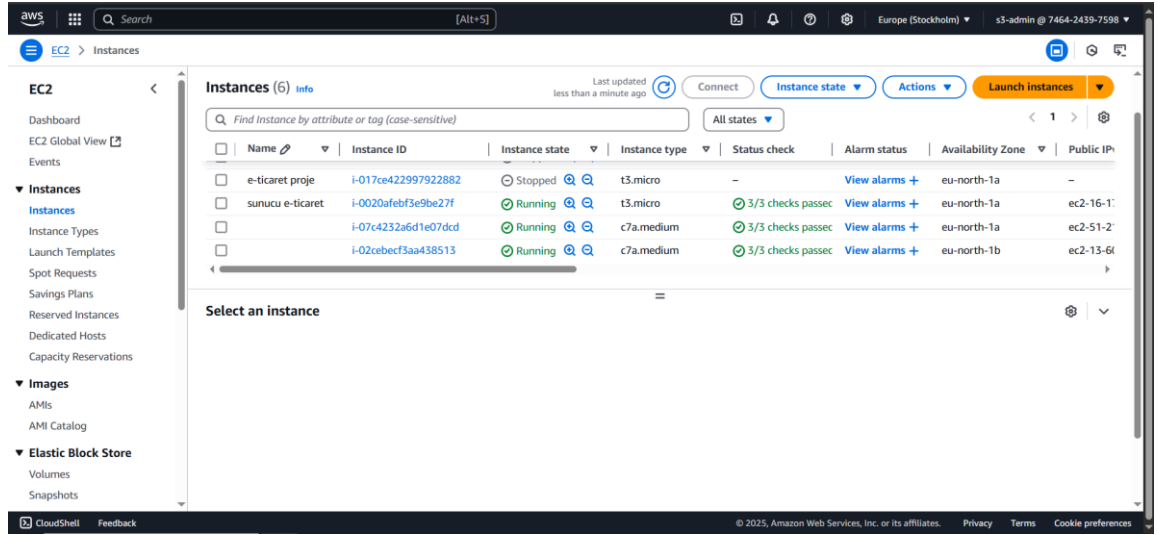
The CORS configuration, written in JSON, defines a way for client web

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
      "PUT",
      "POST",
      "DELETE"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

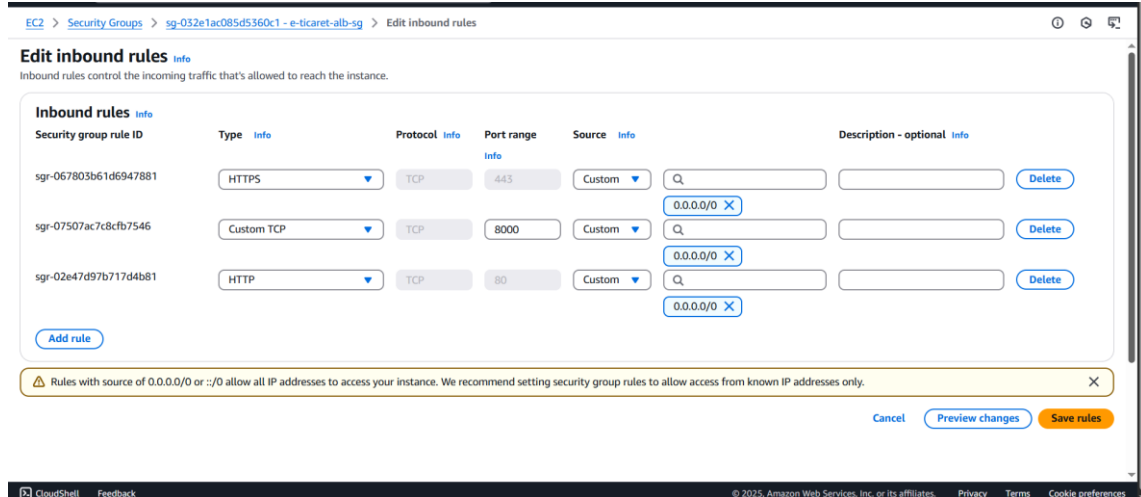
Şekil 12- RDS veritabanı ekranı.



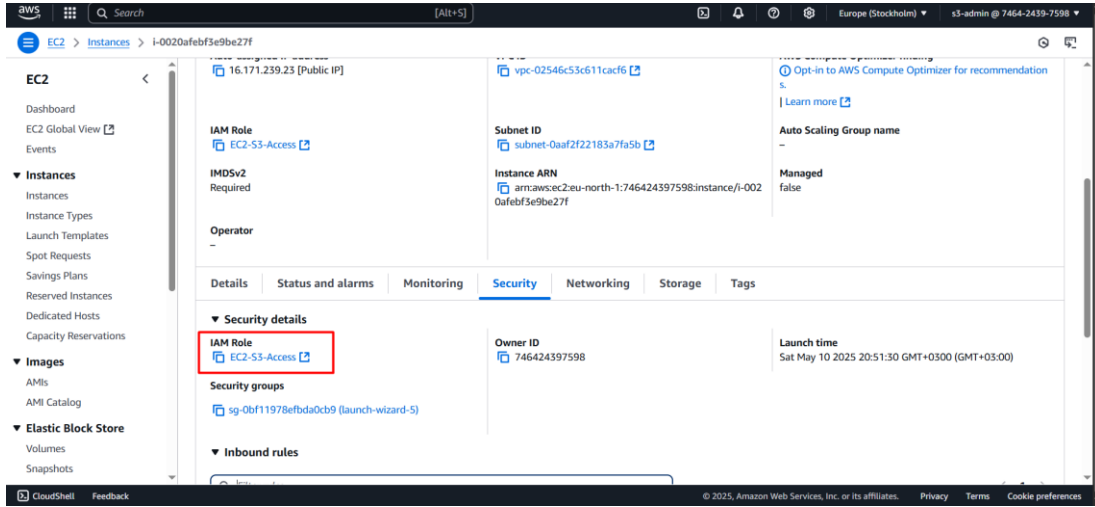
Şekil 13- Django uygulamasını barındırmak için bir veya birden fazla EC2 instance başlatılmıştır.



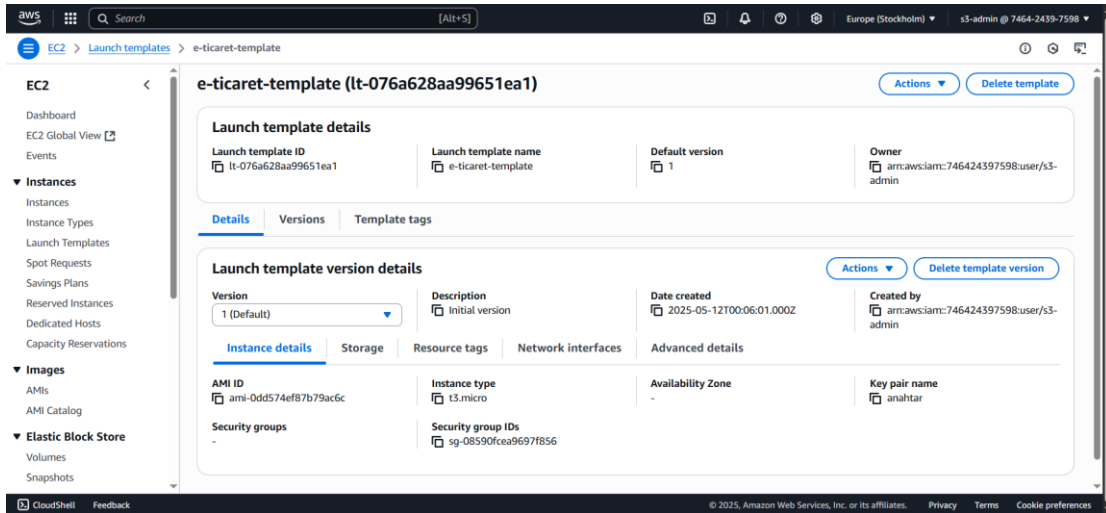
Şekil 14- Güvenlik Grupları Ayarları



Şekil 15-Instance'ların S3'ten dosya çekebilmesi için gerekli IAM rolü atanmıştır.



Şekil 16-Otomatik instance başlatmak için bir launch template hazırlanmıştır. (user data script ile).



Şekil 17- User data script

User data - optional | Info

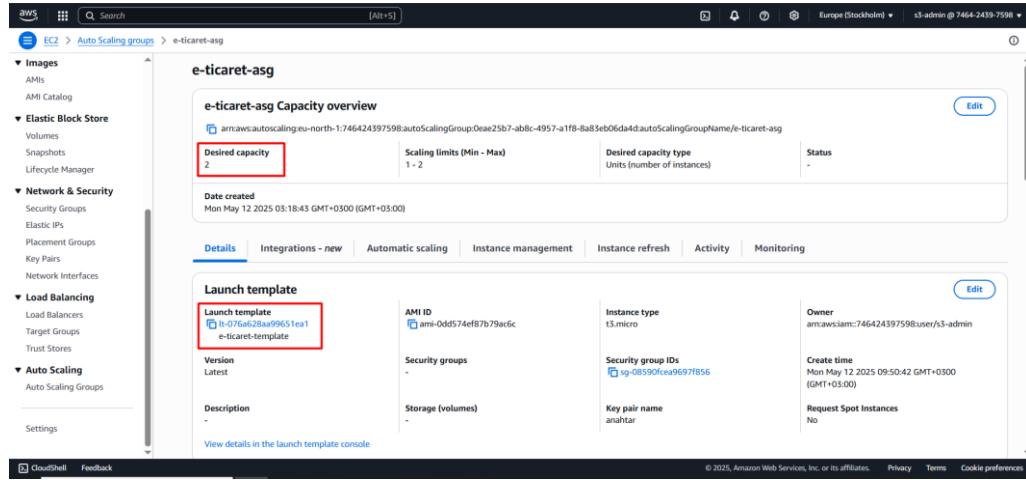
Upload a file with your user data or enter it in the field.

[Choose file](#)

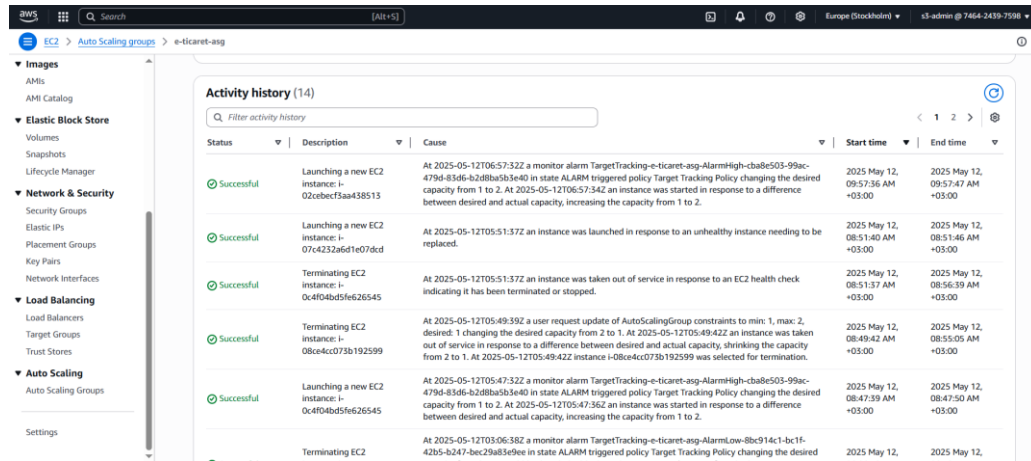
```
apt-get update
apt-get install -y python3-pip awscli
cd /home/ec2-user
aws s3 cp s3://e-ticaret-projesi/E-ticaret-Projesi.tar.gz .
tar -xzf E-ticaret-Projesi.tar.gz
cd E-ticaret-Projesi
pip3 install -r requirements.txt
nohup python3 manage.py runserver 0.0.0.0:8000 > django.log 2>&1 &
```

☐ User data has already been base64 encoded

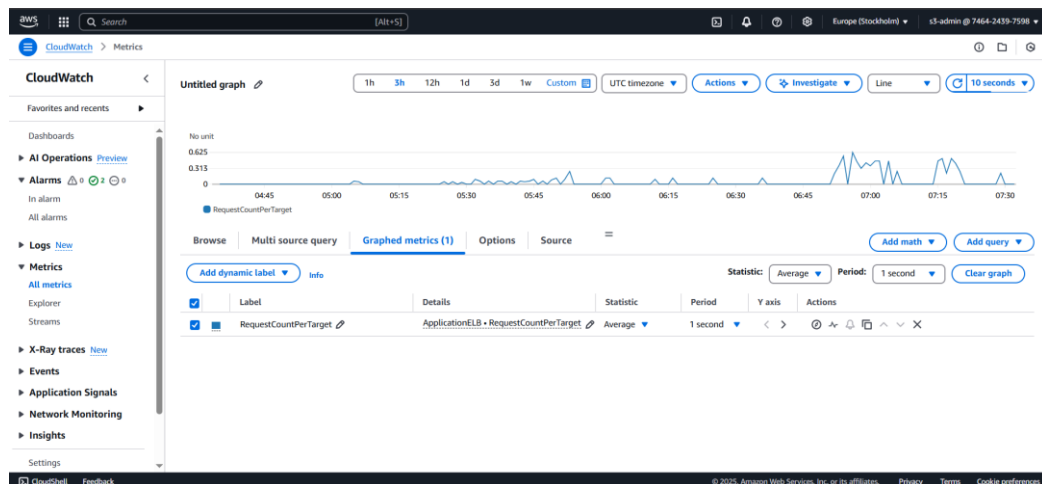
Şekil 18-Yük arttıkça yeni instance'lar otomatik başlatılıyor, azaldıkça kapatılıyor.



Şekil 19- Aktivite geçmişinde instance'ların gerektiğinde başlatılıp gerektiğinde durdurulduğu görülmektedir.



Şekil 20- EC2 instance'larının CPU, bellek, disk kullanımı gibi metrikleri CloudWatch'ta izlendi. Load Balancer'ın "RequestCountPerTarget" metriği takip edildi.



## 6. Sonuç

Bu proje, modern web uygulaması geliştirme ve bulut mimarisi alanlarında uygulamalı bir deneyim kazandırdı. Django ile geliştirilen e-ticaret platformu, AWS'nin RDS, S3, EC2, Load Balancer ve Auto Scaling servisleriyle entegre edilerek yüksek erişilebilirlik ve ölçeklenebilirlik sağlandı. Proje sürecinde, bulut kaynaklarının yönetimi, otomatik ölçeklendirme ve performans izleme gibi DevOps pratikleri başarıyla uygulandı. Sonuç olarak, hem teknik bilgi hem de bulut tabanlı sistem yönetimi konusunda önemli kazanımlar elde edildi.

# Çift Katmanlı Web Uygulaması Teknik Raporu

## 1. Proje Özeti

- **Proje Adı:** Todo Web Uygulaması
- **Kullanılan Teknolojiler:** Node.js, React, MongoDB, Google Cloud
- **Amaç:** Node.js ve React kullanarak çift katmanlı bir web uygulaması geliştirdik ve Google Cloud üzerinde yayınladık. Projenin amacı, modern ve ölçeklenebilir bir todo uygulaması oluşturmaktır.

## 2. Kullanılan Teknolojiler

### 2.1 Backend Geliştirme

- Node.js ve Express.js kullanarak RESTful API geliştirdik
- Görev ekleme, listeleme, güncelleme ve silme işlemleri için gerekli endpoint'leri yazdık
- MongoDB Atlas veritabanı ile görev bilgilerini yönettik
- CORS yapılandırması ile güvenli API erişimi sağladık

### 2.2 Frontend Geliştirme

- React kullanarak modern ve responsive bir kullanıcı arayüzü geliştirdik
- Axios ile backend API'sine bağlantı sağladık
- React Hooks ile state yönetimini gerçekleştirdik
- Modern CSS ile kullanıcı deneyimini iyileştirdik
- 2.3 Google Cloud Servisleri
- Cloud Run: Backend ve frontend uygulamalarını serverless olarak deploy ettik
- Cloud Build: Docker image'larını otomatik olarak build ettik
- Container Registry: Docker image'larını sakladık
- MongoDB Atlas: Cloud veritabanı hizmeti kullandık

### **3. Mimari Yapı**

#### **3.1 Uygulama Katmanı**

- React frontend uygulaması
- Express.js backend API'si
- MongoDB Atlas veritabanı
- Nginx web sunucusu (frontend için)

#### **3.2 Altyapı Katmanı**

- Google Cloud Run ile serverless deployment
- MongoDB Atlas ile cloud veritabanı
- Docker container'ları ile izolasyon

### **4. Öğrenilen Kavramlar ve Teknolojiler**

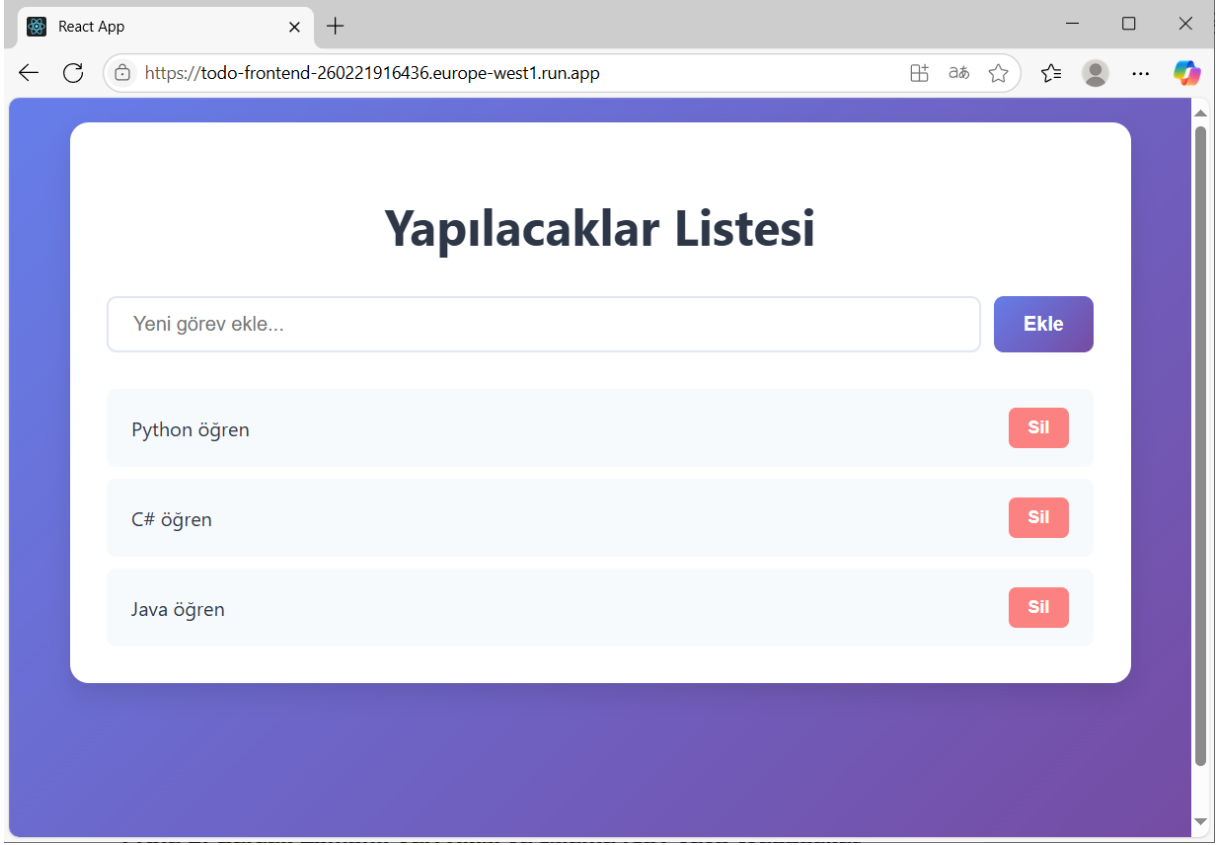
- Serverless Computing: Cloud Run ile serverless deployment
- Container Teknolojisi: Docker ve container yönetimi
- RESTful API Tasarımı: Backend API endpoint'lerinin tasarımı
- Modern Frontend Geliştirme: React ve modern JavaScript
- Cloud Deployment: Google Cloud servislerinin kullanımı
- Veritabanı Yönetimi: MongoDB Atlas kullanımı
- CORS ve Güvenlik: API güvenliği ve CORS yapılandırması

### **5. Proje Kazanımları**

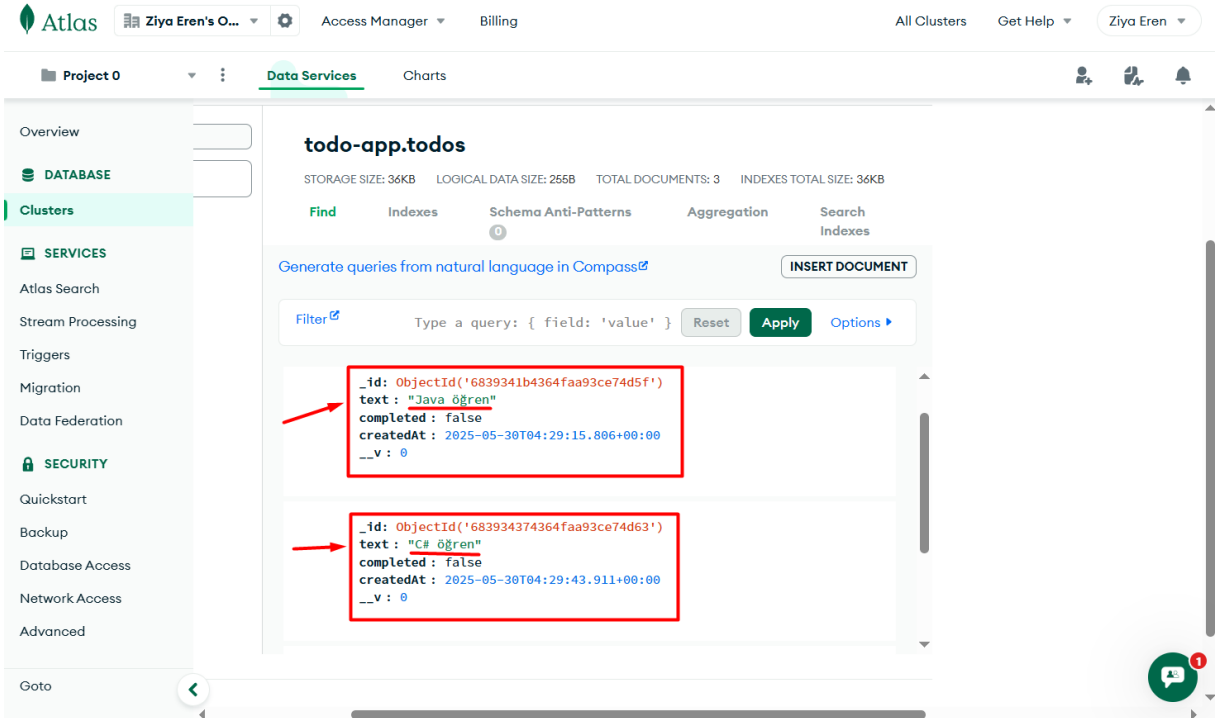
- Modern web teknolojilerinin pratik kullanımı
- Cloud platformlarında deployment deneyimi
- Full-stack web uygulaması geliştirme deneyimi
- RESTful API tasarımı ve implementasyonu
- Container teknolojileri kullanımı
- Cloud veritabanı yönetimi
- Modern frontend geliştirme pratikleri
- DevOps süreçlerinin uygulanması



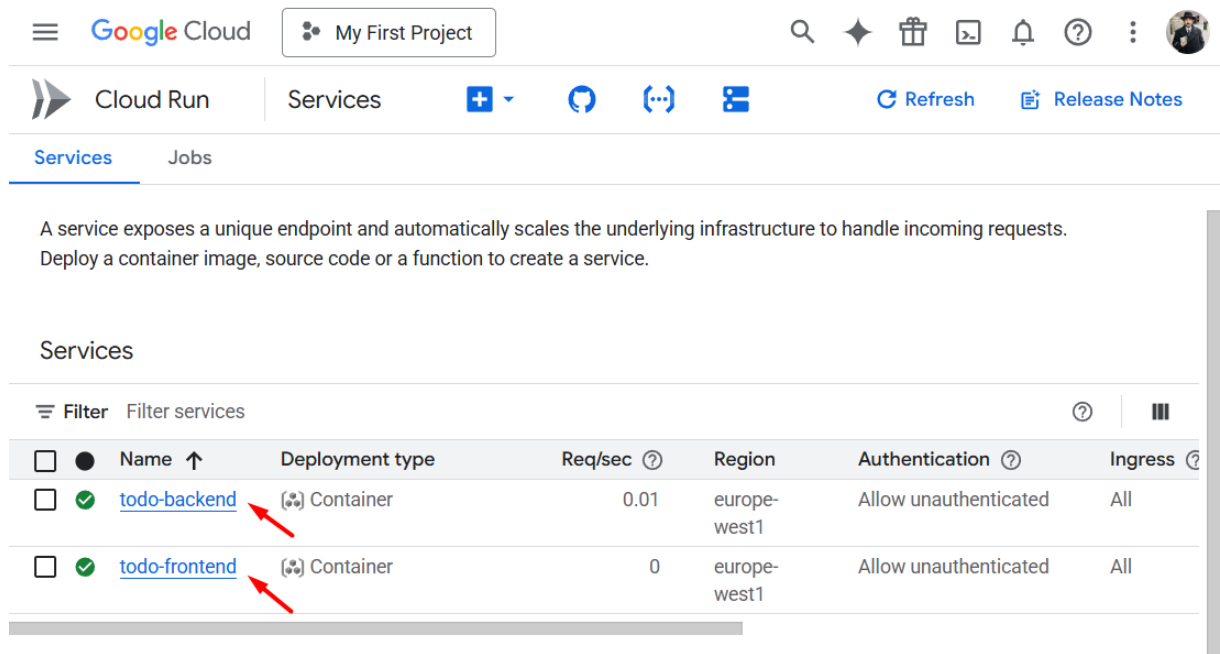
Şekil 21- Todo uygulamasının kullanıcı arayüzü ve örnek görevler



Şekil 22 - MongoDB Atlas'ta saklanan todo verileri



## Şekil 23 - Google Cloud Run'da deploy edilmiş servisler



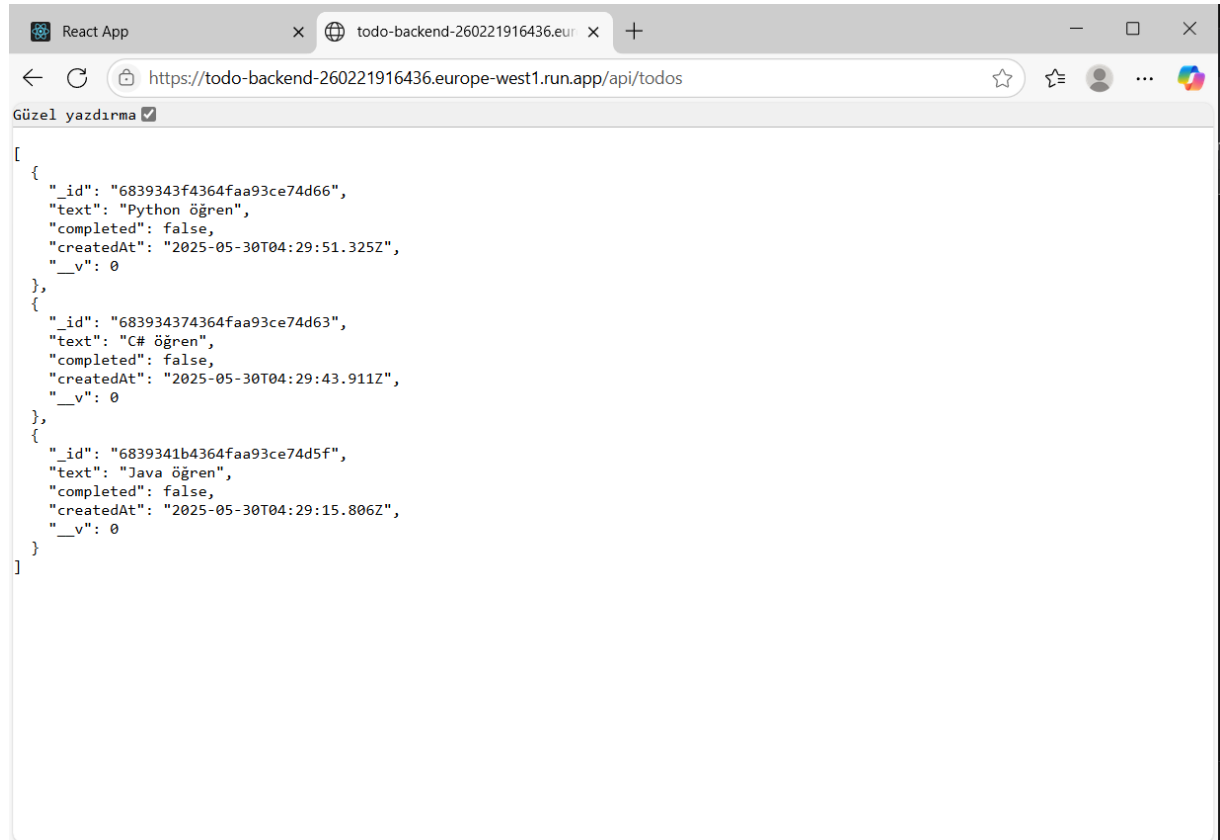
The screenshot shows the Google Cloud Run 'Services' page for a project named 'My First Project'. The page has a navigation bar with 'Cloud Run' and 'Services' tabs. Below the navigation bar, there's a description of a service: 'A service exposes a unique endpoint and automatically scales the underlying infrastructure to handle incoming requests. Deploy a container image, source code or a function to create a service.' Below this, there's a 'Services' section with a table of deployed services. The table has columns for 'Name', 'Deployment type', 'Req/sec', 'Region', 'Authentication', and 'Ingress'. Two services are listed: 'todo-backend' and 'todo-frontend', both with a 'Container' deployment type, '0.01' and '0' requests per second respectively, in the 'europe-west1' region, with 'Allow unauthenticated' authentication, and 'All' ingress. Red arrows point to the 'todo-backend' and 'todo-frontend' links in the table.

Services

A service exposes a unique endpoint and automatically scales the underlying infrastructure to handle incoming requests. Deploy a container image, source code or a function to create a service.

Name	Deployment type	Req/sec	Region	Authentication	Ingress
<a href="#">todo-backend</a>	Container	0.01	europe-west1	Allow unauthenticated	All
<a href="#">todo-frontend</a>	Container	0	europe-west1	Allow unauthenticated	All

## Şekil 24 - Backend API'den dönen todo verileri



The screenshot shows a web browser window with the address bar displaying 'https://todo-backend-260221916436.europe-west1.run.app/api/todos'. The page content shows a JSON array of three todo items. The first item has an id of '6839343f4364faa93ce74d66', text 'Python öğren', and is not completed. The second item has an id of '683934374364faa93ce74d63', text 'C# öğren', and is not completed. The third item has an id of '6839341b4364faa93ce74d5f', text 'Java öğren', and is not completed. The JSON is displayed in a code editor with a 'Güzel yazdırma' button.

```
[
  {
    "_id": "6839343f4364faa93ce74d66",
    "text": "Python öğren",
    "completed": false,
    "createdAt": "2025-05-30T04:29:51.325Z",
    "__v": 0
  },
  {
    "_id": "683934374364faa93ce74d63",
    "text": "C# öğren",
    "completed": false,
    "createdAt": "2025-05-30T04:29:43.911Z",
    "__v": 0
  },
  {
    "_id": "6839341b4364faa93ce74d5f",
    "text": "Java öğren",
    "completed": false,
    "createdAt": "2025-05-30T04:29:15.806Z",
    "__v": 0
  }
]
```

## Şekil 25 - Deployment logları

Google Cloud My First Project

Cloud Run Service details

todo-frontend Region: europe-west1 URL: <https://todo-frontend-260221916436.europe-west1.run.app>

Metrics SLOs **Logs** Revisions Triggers Networking Security YAML

Logs Severity: Default Filter Search all fields and values

Severity	Timestamp	Summary
> i	2025-05-30 07:29:33.864 EET	GET 200 3.98 KB 2 ms Chrome 136 https://todo-frontend-260221916436.europe-we...
> *	2025-05-30 07:29:33.868 EET	169.254.169.126 - - [30/May/2025:04:29:33 +0000] "GET /favicon.ico HTTP/1.1" 200 3...
> i	2025-05-30 07:29:33.898 EET	GET 200 696 B 2 ms Chrome 136 https://todo-frontend-260221916436.europe-west...
> *	2025-05-30 07:29:33.902 EET	169.254.169.126 - - [30/May/2025:04:29:33 +0000] "GET /manifest.json HTTP/1.1" 200...
> i	2025-05-30 07:29:33.987 EET	GET 200 5.42 KB 2 ms Chrome 136 https://todo-frontend-260221916436.europe-we...
> *	2025-05-30 07:29:33.990 EET	169.254.169.126 - - [30/May/2025:04:29:33 +0000] "GET /logo192.png HTTP/1.1" 200 5...
> *	2025-05-30 07:44:36.092 EET	2025/05/30 04:44:36 [notice] 1#1: signal 15 (SIGTERM) received, exiting
> *	2025-05-30 07:44:36.093 EET	2025/05/30 04:44:36 [notice] 25#25: exiting
> *	2025-05-30 07:44:36.093 EET	2025/05/30 04:44:36 [notice] 24#24: exiting
> *	2025-05-30 07:44:36.093 EET	2025/05/30 04:44:36 [notice] 24#24: exit
> *	2025-05-30 07:44:36.093 EET	2025/05/30 04:44:36 [notice] 1#1: signal 29 (SIGIO) received
> *	2025-05-30 07:44:36.093 EET	2025/05/30 04:44:36 [notice] 25#25: exit

## Şekil 26 - Google Container Registry'de saklanan Docker image'ları

Google Cloud My First Project

Artifact Registry / Project: prime-boulevard-461402-a4 / Location: us / Repository: gcr.io

Repositories Settings

Images for gc... Delete Refresh

Repository Details

Format Docker

Type Standard

Show more

Filter Enter property name or value

<input type="checkbox"/>	Name ↑	Connection	Created	Updated
<input type="checkbox"/>	<a href="#">todo-backend</a>	—	1 hour ago	23 minutes ago
<input type="checkbox"/>	<a href="#">todo-frontend</a>	—	1 hour ago	1 hour ago

Release Notes

<I

Şekil 27- Backend API kodları (server.js)

```
app.use(cors({
  origin: ['https://todo-frontend-260221916436.europe-west1.run.app', 'http://localhost:3000'],
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  credentials: true
}));
app.use(express.json());

// MongoDB bağlantısı
mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/todo-app', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

// Todo modeli
const Todo = mongoose.model('Todo', {
  text: String,
  completed: Boolean,
  createdAt: { type: Date, default: Date.now }
});

// Routes
app.get('/api/todos', async (req, res) => {
  try {
    const todos = await Todo.find().sort({ createdAt: -1 });
    res.json(todos);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});
```

Şekil 28 - Frontend ana bileşeni (App.js)

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import './App.css';

function App() {
  const [todos, setTodos] = useState([]);
  const [newTodo, setNewTodo] = useState('');
  const API_URL = 'https://todo-backend-260221916436.europe-west1.run.app';

  useEffect(() => {
    fetchTodos();
  }, []);

  const fetchTodos = async () => {
    try {
      const response = await axios.get(`${API_URL}/api/todos`);
      setTodos(response.data);
    } catch (error) {
      console.error('Görevler yüklenirken hata oluştu:', error);
    }
  };

  const addTodo = async (e) => {
    e.preventDefault();
    if (!newTodo.trim()) return;

    try {
      await axios.post(`${API_URL}/api/todos`, { text: newTodo });
      setNewTodo('');
      fetchTodos();
    } catch (error) {
      console.error('Görev eklenirken hata oluştu:', error);
    }
  };
}
```

```
return (
  <div className="App">
    <div className="container">
      <h1>Yapılacaklar Listesi</h1>
      <div className="todo-form">
        <form onSubmit={addTodo}>
          <input
            type="text"
            value={newTodo}
            onChange={(e) => setNewTodo(e.target.value)}
            placeholder="Yeni görev ekle..."
            className="todo-input"
          />
          <button type="submit" className="add-button">
            <span>Ekle</span>
          </button>
        </form>
      </div>

      <div className="todo-list">
        {todos.length === 0 ? (
          <p className="empty-message">Henüz görev eklenmemiş</p>
        ) : (
          todos.map((todo) => (
            <div key={todo._id} className="todo-item">
              <span
                className={`todo-text ${todo.completed ? 'completed' : ''}`}
                onClick={() => toggleTodo(todo._id)}
              >
                {todo.text}
              </span>
              <button
                type="button"
                className="toggle-button"
                onClick={() => toggleTodo(todo._id)}
              >
                {todo.completed ? 'İptal' : 'Tamamla'}
              </button>
            </div>
          ))
        )}
      </div>
    </div>
  </div>
);
```

## Şekil 29 - Frontend stil tanımlamaları (App.css)

```
1 .App {
2   min-height: 100vh;
3   background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
4   padding: 20px;
5 }
6
7 .container {
8   max-width: 800px;
9   margin: 0 auto;
10  background: #fff;
11  border-radius: 15px;
12  padding: 30px;
13  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
14 }
15
16 h1 {
17   text-align: center;
18   color: #2d3748;
19   font-size: 2.5rem;
20   margin-bottom: 30px;
21   font-weight: 700;
22 }
23
24 .todo-form {
25   margin-bottom: 30px;
26 }
27
28 form {
29   display: flex;
30   gap: 10px;
31 }
32
33 .todo-input {
34   flex: 1;
35   padding: 12px 20px;
36   font-size: 16px;
```

```
37   border: 2px solid #e2e8f0;
38   border-radius: 8px;
39   transition: all 0.3s ease;
40 }
41
42 .todo-input:focus {
43   outline: none;
44   border-color: #667eea;
45   box-shadow: 0 0 3px rgba(102, 126, 234, 0.1);
46 }
47
48 .add-button {
49   padding: 12px 24px;
50   background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
51   color: #fff;
52   border: none;
53   border-radius: 8px;
54   font-size: 16px;
55   font-weight: 600;
56   cursor: pointer;
57   transition: all 0.3s ease;
58 }
59
60 .add-button:hover {
61   transform: translateY(-2px);
62   box-shadow: 0 5px 15px rgba(102, 126, 234, 0.4);
63 }
64
65 .todo-list {
66   display: flex;
67   flex-direction: column;
68   gap: 10px;
69 }
70
71 .todo-item {
```

```
72   display: flex;
73   justify-content: space-between;
74   align-items: center;
75   padding: 15px 20px;
76   background: #f7fafc;
77   border-radius: 8px;
78   transition: all 0.3s ease;
79 }
80
81 .todo-item:hover {
82   transform: translateX(5px);
83   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.05);
84 }
85
86 .todo-text {
87   font-size: 16px;
88   color: #2d3748;
89   cursor: pointer;
90   transition: all 0.3s ease;
91 }
92
93 .todo-text.completed {
94   text-decoration: line-through;
95   color: #a0aec0;
96 }
97
98 .delete-button {
99   padding: 8px 16px;
100  background-color: #fc8181;
101  color: #fff;
102  border: none;
103  border-radius: 6px;
104  font-size: 14px;
105  font-weight: 600;
106  cursor: pointer;
```

```
107   transition: all 0.3s ease;
108 }
109
110 .delete-button:hover {
111   background-color: #f56565;
112   transform: translateY(-2px);
113 }
114
115 .empty-message {
116   text-align: center;
117   color: #a0aec0;
118   font-size: 16px;
119   padding: 20px;
120 }
121
122 @media (max-width: 600px) {
123   .container {
124     padding: 20px;
125   }
126
127   h1 {
128     font-size: 2rem;
129   }
130
131   form {
132     flex-direction: column;
133   }
134
135   .add-button {
136     width: 100%;
137   }
138 }
```

## 6. Sonuç

Bu proje, modern web uygulaması geliştirme ve bulut mimarisi alanlarında uygulamalı bir deneyim kazandırdı. Node.js ve React ile geliştirilen todo uygulaması, Google Cloud Run ve MongoDB Atlas servisleriyle entegre edilerek yüksek erişilebilirlik ve ölçeklenebilirlik sağlandı. Proje sürecinde, serverless mimari, container teknolojileri, RESTful API tasarımı ve modern frontend geliştirme pratikleri başarıyla uygulandı. Ayrıca, CORS yapılandırması ve güvenlik önlemleri ile güvenli bir uygulama geliştirildi. Sonuç olarak, hem full-stack web geliştirme hem de cloud tabanlı sistem yönetimi konusunda önemli kazanımlar elde edildi. Proje, modern web teknolojilerinin ve cloud servislerinin pratik uygulamasını göstererek, gerçek dünya senaryolarında kullanılabilecek bir deneyim sağladı.

## 7. Uygulama Bağlantıları

- Frontend URL: <https://todo-frontend-260221916436.europe-west1.run.app>
- Backend URL: <https://todo-backend-260221916436.europe-west1.run.app/api/todos>
- MongoDB Atlas: Cloud veritabanı
- Google Cloud Console: Proje yönetimi ve monitoring