ANKARA ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM3522

BULUT BİLİŞİM VE UYGULAMALARI

BÜT RAPORU

Ömer Asaf DEMİR

20290240

GitHub: AsafDemir/BulutBilisimVeUygulamalariBut

https://github.com/AsafDemir/BulutBilisimVeUygulamalariBut

Proje 1: Çift Katmanlı Web Uygulaması (Web API + Frontend)

1. Proje Genel Bakış

Çay Ocağı Yönetim Sistemi, kurumsal ortamlarda çay ve içecek sipariş süreçlerini dijitalleştiren, kullanıcı ve yönetici rolleri ile erişim kontrollü modern bir web uygulamasıdır. Azure bulut altyapısı üzerinde çalışarak yüksek erişilebilirlik, ölçeklenebilirlik ve güvenlik sağlamaktadır.

2. Kullanılan Teknolojiler ve Uygulamalar

Backend Teknolojileri

- .NET Core Web API
- Entity Framework Core (ORM)
- JWT Authentication

Projenin backend tarafı için yüksek performans, düşük kaynak tüketimi ve mikroservis mimarilerine uygun olması nedeniyle .NET Core Web API tercih edildi. RESTful prensiplerine uygun API endpoint'leri oluşturuldu ve HTTP metodları (GET, POST, PUT, DELETE) etkin kullanıldı.

Entity Framework Core ile ORM aracılığıyla veritabanı işlemleri nesne yönelimli olarak yönetildi. Code First yaklaşımı ile veritabanı tasarımı doğrudan C# sınıflarından oluşturuldu, migration kullanımıyla versiyonlama sağlandı ve veri erişim performansı Lazy Loading ve Eager Loading yöntemleriyle optimize edildi.

JWT Authentication kullanılarak güvenli ve stateless kimlik doğrulama gerçekleştirildi. Token'ların içerisinde kullanıcı bilgileri ve roller taşınarak yetkilendirme işlemleri hızlandırıldı.

Frontend Teknolojileri

- Angular 19.0.0
- Angular Material ve CDK
- RxJS
- TypeScript 5.6.2
- Reactive Forms

Frontend kısmında Angular framework’ü kullanıldı. Angular'ın modüler ve komponent tabanlı mimarisi sayesinde sürdürülebilir bir yapı sağlandı. Angular Material kullanılarak kullanıcı arayüzünde modern ve profesyonel bir görünüm oluşturuldu.

RxJS ile reaktif programlama uygulandı, gerçek zamanlı veri akışları yönetildi. Reactive Forms ile kullanıcı girişlerinin anlık doğrulaması ve hata yönetimi gerçekleştirildi.

Veritabanı Teknolojileri

- PostgreSQL

Projenin veri katmanında PostgreSQL kullanıldı. PostgreSQL'in ilişkisel veri yönetimi, sorgu performansı ve JSON desteği tercih sebebi oldu. Entity Framework Core aracılığıyla PostgreSQL üzerinde veritabanı yapıları oluşturuldu ve Azure servisleri ile otomatik yedekleme ve güvenlik yönetimi sağlandı.

Bulut Platformları

- Azure App Services
- Azure Database for PostgreSQL
- Azure Application Insights
- Azure DevOps (CI/CD)

Microsoft Azure platformu, uygulamanın barındırılması için kullanıldı. Azure App Services ile otomatik ölçeklendirme ve yönetim kolaylığı sağlandı. Azure Database for PostgreSQL ile veri güvenliği artırıldı ve performans izleme Azure Application Insights üzerinden gerçekleştirildi. Azure DevOps ile CI/CD süreçleri otomatize edildi, kod kalitesi sürekli kontrol edildi.

3. Sistem Mimari Yapısı

Katmanlı Mimari

- Sunum Katmanı (Angular Components, API Controllers)
- İş Mantığı Katmanı (Services)
- Veri Erişim Katmanı (Entity Framework, Data Layer)

Katmanlı mimari ile kod tabanı modüler hale getirildi. Bu mimari sayesinde bakım ve geliştirme süreçleri basitleştirildi.

RESTful API Tasarımı

RESTful API prensiplerine uygun olarak, açık, anlaşılır ve tutarlı endpoint'ler geliştirildi. HTTP metodlarıyla kullanıcı ve yönetici işlemleri net bir şekilde tanımlandı.

4. Güvenlik ve Kullanıcı Yönetimi

- JWT Authentication
- Rol tabanlı erişim kontrolü
- CORS Yapılandırması

JWT Authentication ile kullanıcı ve yetki yönetimi gerçekleştirildi. Kullanıcı rolleri tanımlanarak, rol bazlı erişim kontrolleri uygulandı ve CORS yapılandırmasıyla frontend güvenliği sağlandı.

5. Sipariş ve Veri Yönetimi

Kullanıcılar için kolaylaştırılmış sipariş arayüzleri oluşturuldu. Reactive Forms sayesinde kullanıcı verileri anlık doğrulanarak hatalı veri girişleri önlendi. RxJS kütüphanesinden BehaviorSubject kullanılarak siparişlerin gerçek zamanlı takibi mümkün hale getirildi.

Yöneticiler için kapsamlı sipariş yönetimi sağlandı. Yönetim panelinde sipariş durumları değiştirilebilir, filtrelenebilir ve iptal edilebilir. Ayrıca detaylı raporlama ve görsel analiz araçları ile yöneticilere kapsamlı veri analizi imkânı sunuldu. Yönetici panelinde içecek stokları, fiyatlandırma ve oda yönetimi için CRUD işlemleri sağlandı.

6. Kullanıcı Arayüzü Tasarımı

- Material Design
- Responsive ve erişilebilir tasarım

Material Design ilkeleri doğrultusunda kullanıcı dostu, responsive ve erişilebilir arayüz geliştirildi.

7. Performans Optimizasyonu

- Lazy loading
- Database Indexing
- Query Optimization

Performans iyileştirmeleri için lazy loading uygulandı, sık sorgulanan verilere indeksleme yapıldı ve sorgular optimize edildi.

8. Test ve Kalite Güvence

- Jasmine ve Karma (birim testleri)
- Protractor (end-to-end testleri)
- ESLint ve Prettier (kod kalitesi)
- Azure DevOps (CI/CD pipeline)

Test ve kalite süreçleri otomatize edilerek kod kalitesi yüksek seviyede tutuldu. CI/CD süreçleri ile sürekli entegrasyon ve dağıtım süreçleri yönetildi.

9. Proje Sonuçları

- Sipariş sürecinde %40 hızlanma
- Hata oranlarında %70 düşüş
- Kullanıcı memnuniyetinde artış

Projenin hedeflerine ulaşılmış olup dijital dönüşüm sağlanarak verimlilik, doğruluk ve kullanıcı memnuniyeti artırıldı.

10. Gelecek Geliştirmeler

- Mobil uygulama
- Gerçek zamanlı bildirimler
- Yapay zeka önerileri
- Otomasyon ve IoT entegrasyonu

Mobil uygulama, gerçek zamanlı bildirimler, yapay zeka destekli öneriler ve IoT entegrasyonu ile projenin kapsamı ve işlevselliği genişletilmesi planlanmaktadır.

Proje 4: Otomatik Ölçeklendirme ve Yük Yönetimi

Otomatik Ölçeklendirme ve Yük Yönetimi

Çay Ocağı Yönetim Sistemi, Microsoft Azure platformu üzerinde barındırılmakta olup, yüksek erişilebilirlik ve performans hedefleri doğrultusunda **otomatik ölçeklendirme** destekli bir yapıya geçirilmiştir. Uygulama, Azure App Services hizmetinde **Standard S1** planında çalışacak şekilde yapılandırılmış ve Azure portal üzerinden özel autoscale kuralları tanımlanmıştır.

Bu yapılandırma kapsamında:

- Uygulamanın CPU kullanım oranı %70'in üzerine çıktığında, Azure tarafından otomatik olarak yeni bir instance başlatılmakta,
- CPU kullanımı %30'un altına düştüğünde ise fazla instance'lar otomatik olarak kapatılmaktadır.

Bu sayede sistem dinamik olarak **yatay ölçeklenebilir** hale gelmiştir. Azure Load Balancer aracılığıyla yük dengelemesi sağlanmış, ani trafik artışlarında sistem performans kaybı yaşamadan hizmet verebilir duruma getirilmiştir.

Bununla birlikte:

- Azure Application Insights ile sistem performansı ve hata izleme gerçekleştirilmektedir.
- Azure Database for PostgreSQL ile ilişkisel veri yönetimi ve otomatik yedekleme süreçleri güvenli biçimde sürdürülmektedir.
- Azure DevOps entegrasyonu sayesinde, CI/CD süreçleriyle sürekli teslimat ve güncelleme operasyonları sağlanmıştır.

Bu çalışmalar neticesinde uygulama, sadece fonksiyonel olarak değil, **altyapı düzeyinde de modern, güvenli ve sürdürülebilir bir yapıya kavuşmuştur.** Projenin bulut mimarisi sayesinde, ileriye dönük ölçekleme, entegrasyon ve bakım süreçleri kolaylaştırılmıştır.

Proje 2: Gerçek Zamanlı Veri Akışı ve İşleme (IoT veya WebSocket Uygulaması)

1. Proje Genel Bakış

IoT Sensör Verisi İşleme Sistemi; sıcaklık ve nem gibi çevresel verileri gerçek zamanlı toplayan, işleyen ve görselleştiren modern bir bulut tabanlı IoT uygulamasıdır. MQTT protokolü ile sensörden gelen veriler Google Cloud Pub/Sub üzerinden Flask uygulamasına yönlendirilir ve MongoDB Atlas veritabanında saklanır. Sistem, ölçeklenebilirlik, güvenlik ve esneklik gibi bulut bilişim ilkelerini temel alarak tasarlanmıştır. Web tabanlı arayüz ile kullanıcılar verileri canlı olarak izleyebilir ve analiz edebilir.

2. Kullanılan Teknolojiler ve Uygulamalar

Backend ve IoT Teknolojileri

- Python 3
- Flask Framework
- Sensor Simulator (sensor_simulator.py)
- Data Consumer (data_consumer.py)

Python dili, sistemin tüm backend altyapısının geliştirilmesinde kullanılmıştır. Flask framework'ü ile RESTful API'ler tasarlanmış ve kullanıcıların veri erişimi sağlanmıştır. Sensor Simulator bileşeni ile gerçekçi sıcaklık ve nem verileri üretilmiş, Data Consumer modülü ise bu verileri alıp MongoDB'ye kaydetmiştir.

Mesajlaşma Katmanı

- MQTT Protokolü (Paho-MQTT Library)
- Eclipse Mosquitto (Lokal Broker)
- HiveMQ Public Broker (Test Ortamı)
- Google Cloud Pub/Sub

MQTT protokolü, IoT cihazları arasında hafif ve hızlı veri iletimi sağlamak amacıyla tercih edilmiştir. Paho-MQTT kütüphanesi ile Python ortamında MQTT işlemleri yürütülmüştür. Lokal testlerde Mosquitto kullanılmış, Pub/Sub ise üretim ortamında asenkron mesajlaşma ve servisler arası veri akışı için kullanılmıştır. Pub/Sub topic ve subscription

yapıları, sensör verilerinin Flask sunucusuna HTTP push yöntemiyle iletilmesini sağlamıştır.

Veritabanı Teknolojileri

- MongoDB Atlas (Cloud NoSQL Database)
- MongoDB Compass

MongoDB Atlas, yapılandırılmamış sensör verilerinin JSON formatında depolanması için kullanılmıştır. NoSQL yapısı sayesinde sistem yatayda ölçeklenebilir bir veritabanına sahip olmuştur. MongoDB Compass aracı, verilerin okunabilirliğini artırmış ve canlı kontrol süreçlerinde kullanılmıştır.

Bulut Servisleri ve Deployment

- Google Cloud Run
- Google IAM & Service Account
- Google Cloud Container Registry
- Docker (Multi-stage Dockerfile)
- .env Ortam Dosyası

Flask uygulaması Docker ile container haline getirilmiş ve Google Cloud Run üzerinde çalıştırılmıştır. Cloud Run'ın otomatik ölçeklenebilir ve serverless yapısı sayesinde maliyet ve kaynak kullanımı optimize edilmiştir. GCP kimlik doğrulama için IAM ve service account yapıları yapılandırılmış, JSON kimlik dosyaları ortam değişkeni olarak .env içinde yönetilmiştir.

Frontend Teknolojileri

- Vanilla JavaScript
- CSS Grid Layout
- Font Awesome
- RESTful API ile Gerçek Zamanlı Veri Görselleştirme

Web arayüzünde kullanıcıya görsel olarak sade ama işlevsel bir panel sunulmuştur. Vanilla JS ile yazılmış dinamik içerik yapısı 30 saniyelik otomatik veri yenileme ile canlı güncellenmekte, CSS Grid ile responsive yapı sağlanmaktadır. İkon desteği için Font Awesome kullanılmıştır.

Güvenlik ve İzleme

- Google IAM ve Rol Bazlı Erişim
- Environment Variable İzolasyonu
- MongoDB Encryption at Rest / In Transit
- Google Cloud Monitoring & Logging

Uygulamada kimlik bilgilerinin korunması için environment dosyaları kullanılmış ve erişim kontrolü IAM aracılığıyla sağlanmıştır. MongoDB tarafında hem at rest hem de in transit şifreleme aktif hale getirilmiştir. Google Cloud Operations ile sistemin gecikme süresi, hata oranları ve mesaj throughput gibi metrikleri izlenmiştir.

3. Sistem Mimarisi

Kullanılan Yapılar ve Bileşenler

- MQTT + Google Pub/Sub hibrit veri akışı
- Mosquitto MQTT Broker
- Flask Web Servisi (Cloud Run)
- MongoDB Atlas Veritabanı
- REST API Endpoint'leri
- Web Tabanlı Arayüz (JS + API)

Sensörden gelen veriler önce MQTT ile Mosquitto broker'a iletilir. Bu veriler ya doğrudan veri tüketici modülü ile MongoDB'ye yazılır ya da Google Pub/Sub ile Cloud Run'daki Flask API'ye gönderilir. Flask uygulaması gelen veriyi işler ve MongoDB'ye kaydeder. Web arayüzü, bu verileri sorgulayarak kullanıcılara canlı olarak sunar.

4. Güvenlik ve Hata Toleransı

Uygulanan Önlemler

- MQTT TLS veri şifreleme
- MongoDB şifreli veri aktarımı ve saklama
- Google IAM ile rol tabanlı erişim
- .env ile gizli bilgilerin korunması
- Cloud Run health check endpoint
- MQTT QoS ve yeniden bağlanma desteği
- GCP log takibi

MQTT ile veri iletiminde TLS kullanılarak güvenlik sağlanmıştır. MongoDB, verileri hem aktarımda hem saklamada şifreli tutar. IAM sayesinde yalnızca yetkili servisler erişim sağlar. Kimlik bilgilerinin sızmaması için .env dosyaları kullanılmıştır. Cloud Run, sistemin çalışır durumda olup olmadığını otomatik kontrol eder. MQTT'de QoS 1 ile veri kaybı engellenmiş, bağlantı kesilirse yeniden bağlanma sağlanmıştır.

5. Performans ve Ölçeklenebilirlik

Kullanılan Yöntemler

- Cloud Run autoscaling (0–10 instance)
- MongoDB indexing (timestamp, sensor_id)
- Query optimizasyonu
- GCP Monitoring ile izleme
- 1000 msg/s MQTT veri akışı
- <150ms dashboard yanıt süresi

Uygulama trafik yoğunluğuna göre otomatik olarak yeni instance başlatır. MongoDB'de yapılan indekslemeler sorgu hızını artırır. Cloud Monitoring ile sistem performansı düzenli takip edilir. Testlerde saniyede 1000 mesaj işlenmiş, arayüzde 150ms altında cevap süreleri elde edilmiştir.

6. Test ve Kalite Güvencesi

Uygulanan Testler

- Sensor Simulator → MQTT → MongoDB testi
- Pub/Sub → Flask → Veritabanı girişi testi
- Dashboard → API → Arayüz güncelleme testi
- Bağlantı kesintisi ve veri hatası senaryoları
- Yük altında sistem kararlılığı testi

Sistem uçtan uca testlerle doğrulanmıştır. Sensörden MongoDB'ye kadar olan veri akışı ve arayüzdeki veri güncellemeleri test edilmiştir. Hatalı veri girişleri, bağlantı sorunları ve yoğun trafik altında sistemin dayanıklılığı gözlemlenmiş ve güvenilirliği sağlanmıştır.

7. Proje Sonuçları

Elde Edilen Kazanımlar

- Gerçek zamanlı veri işleme başarıyla sağlandı
- Otomatik ölçeklenebilir mimari oluşturuldu
- MQTT ve Pub/Sub hibrit yapısı uygulandı
- IoT sistemlerinin temel ihtiyaçları karşılandı

Proje hedeflerine ulaşarak; veri akışının güvenli, hızlı ve ölçeklenebilir bir şekilde yönetilmesi sağlanmıştır. Hem geleneksel hem de bulut-native yöntemler desteklenerek esnek bir yapı kurulmuştur. Modern IoT uygulamaları için güçlü bir temel sunulmuştur.

Proje 7: IoT ve Akıllı Şehir Uygulaması

1. Proje Genel Tanıtım

Kullanılan Yapı

- Gerçek zamanlı otopark doluluk takibi
- AWS tabanlı sunucusuz mimari
- MQTT üzerinden IoT cihaz simülasyonu
- Web tabanlı veri görselleştirme

Bu proje, farklı konumlardaki otopark alanlarından gelen doluluk verilerini IoT cihazlar üzerinden toplayarak AWS servisleri aracılığıyla işler ve kullanıcıya sunar. Sistem, MQTT protokolü ile çalışan sensör simülatörleri, AWS IoT Core, Lambda, DynamoDB ve API Gateway gibi bileşenlerle serverless ve ölçeklenebilir bir mimariye sahiptir.

2. Sistem Mimarisi ve Çalışma Mantığı

Temel Bileşenler

- mqtt_simulator.py (Veri üretimi ve iletimi)
- AWS IoT Core (MQTT mesaj karşılayıcı)
- Lambda – SaveParkingData (Veri işleme ve kaydetme)
- DynamoDB – ParkingData tablosu
- Lambda – GetParkingData (Veri alma)
- API Gateway (/v1 endpoint)
- Web frontend (Gerçek zamanlı dashboard)

Simülatör, her 30 saniyede bir 10 farklı otopark cihazı adına veri üretip AWS IoT Core'daki parking/data topic'ine gönderir. IoT Rule bu veriyi yakalayarak SaveParkingData fonksiyonunu tetikler. Lambda fonksiyonu gelen JSON verisini kontrol eder ve DynamoDB'ye kaydeder. GetParkingData fonksiyonu en güncel verileri filtreleyip API Gateway üzerinden frontend'e iletir.

3. Kodlama ve Dosya Yapısı

Öne Çıkan Dosyalar

- mqtt_simulator.py
- lambda_function.py (2 mod: write & read)
- config.py (AWS ayarları ve cihaz bilgileri)
- aws_config klasörü (iot_policy.json, dynamodb_schema.json)
- frontend/index.html, script.js, style.css

Simülatör scripti cihaz başına ayrı thread açarak paralel veri iletimi yapar. Lambda kodları hem yazma hem okuma işlevi içerir. Config dosyaları merkezi ayar yönetimini sağlar. Frontend kısmı responsive ve modern tasarımla gerçek zamanlı görselleştirme sunar.

4. Güvenlik ve Erişim Kontrolleri

Uygulanan Önlemler

- MQTT SSL/TLS veri şifreleme
- IoT Policy: Minimum yetkili bağlantı izinleri
- IAM: Lambda ve DynamoDB için sınırlı roller
- API Gateway CORS ve API Key kontrolü

Tüm MQTT bağlantıları SSL ile güvence altına alınmıştır. IoT policy dosyasında sadece belirli client ID'lerin publish etmesine izin verilmiştir. IAM rolleri sadece gereken kaynaklara erişime sahiptir. API Gateway CORS desteğiyle frontend erişimlerine izin verirken, opsiyonel olarak API anahtarı da kullanılabilir.

5. Performans ve Optimizasyon

Sağlanan İyileştirmeler

- MQTT bağlantı havuzu (connection pooling)
- Lambda batch işleme desteği
- DynamoDB'de composite key kullanımı
- Local caching ile frontend performansı
- CloudWatch log analizi

Sistem, ölçeklenebilirlik ve gecikme azaltımı hedefiyle yapılandırılmıştır. Lambda fonksiyonları kısa süreli işlem süresiyle optimize edilmiştir. DynamoDB'de timestamp + device_id yapısı ile zaman bazlı sorgular hızlandırılmıştır. Frontend tarafında local cache ve async API çağrıları ile akıcı bir kullanıcı deneyimi sağlanmıştır.

6. Test ve Kalite Güvencesi

Uygulanan Testler

- MQTT bağlantı ve veri bütünlüğü testleri
- Lambda → DynamoDB yazma senaryoları
- API Gateway → JSON veri alma kontrolü
- UI güncelleme ve anlık yenileme doğrulaması

Sistem uçtan uca test senaryolarıyla doğrulanmıştır. Cihazdan veritabanına, kullanıcı arayüzüne kadar olan tüm veri akışı test edilmiştir. Gerçek zamanlı veri tazeleme, bağlantı kopması ve yeniden bağlanma durumları göz önüne alınarak sistem hata toleranslı şekilde çalıştırılmıştır.

7. Proje Sonuçları

Elde Edilen Başarılar

- 10 cihazdan 30 saniyede bir veri aktarımı
- Gerçek zamanlı dashboard güncellemeleri
- Serverless yapı sayesinde bakım gereksinimi az
- Düşük gecikmeli API erişimi ve görselleştirme

Proje başarıyla tamamlanmış, gerçek dünya koşullarında kullanılabilecek bir altyapı kurulmuştur. Sunucusuz ve otomatik ölçeklenebilir mimari sayesinde sistemin çalışabilirliği yüksek, maliyeti ise düşük tutulmuştur.