**2024-2025 YILI BAHAR DÖNEMİ**

**BULUT BİLİŞİMİ FİNAL RAPORU**

****

**AD – SOYAD ÖĞRENCİ NUMARASI**

**HÜSEYİN TINAZTEPE: 21290360**

**YUSUF TUNÇ: 22290071**

**BORA KOCABIYIK: 21290270**

PROJE 1 GitHub Linki: **Çift Katmanlı Web Uygulaması (Web API + Frontend)**

<https://github.com/brckfrc/skynotes.git>

PROJE 2 GitHub Linki: **Akıllı Veri Analitiği ve Makine Öğrenmesi Uygulaması**

<https://github.com/hsyntinaztepe/mlproject>

PROJE 3 GitHub Linki: **E-Ticaret Uygulaması (Otomatik Ölçeklendirme ve Yönetim)**

<https://github.com/hsyntinaztepe/ecommerce-project>

PROJE 4 GitHub Linki: **Gerçek Zamanlı Veri Akışı ve İşleme (IoT Uygulaması)**

<https://github.com/brckfrc/sensora>

**PROJE 1: Çift Katmanlı Web Uygulaması (Web API + Frontend)**

**SkyNotes Tam Yığın Not Yönetim Uygulaması Proje Raporu**

**1. Proje Genel Bakış**

**1.1 Proje Amacı**

SkyNotes, kullanıcıların dijital not yönetimini kolaylaştırmak için geliştirilmiş bir web uygulamasıdır. Proje, modern web geliştirme teknolojilerini kullanarak kullanıcı dostu bir not yönetim sistemi oluşturmayı hedeflemektedir.

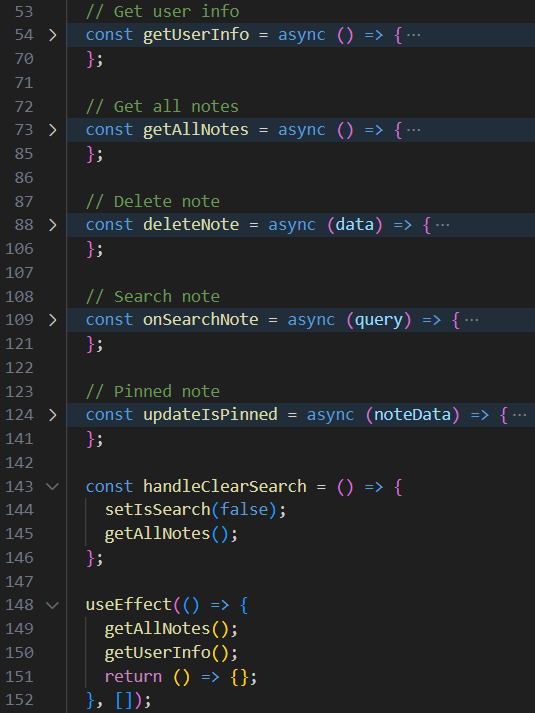
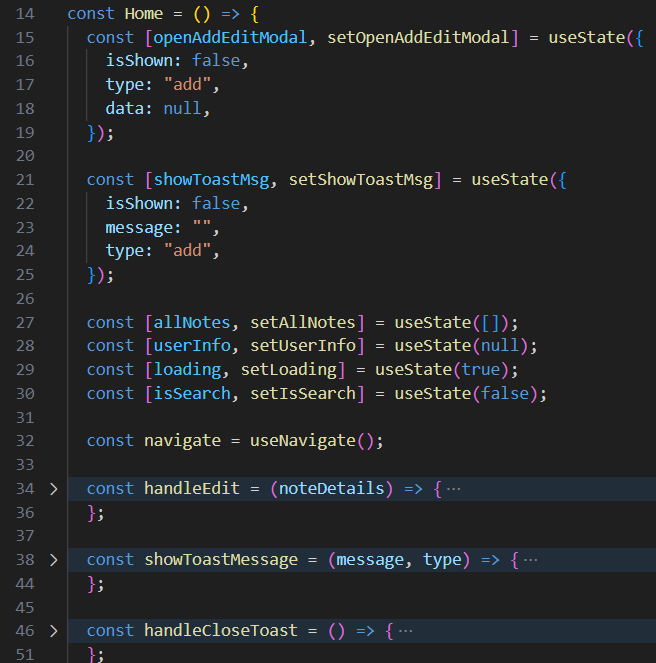
**2. Teknoloji Yığını**

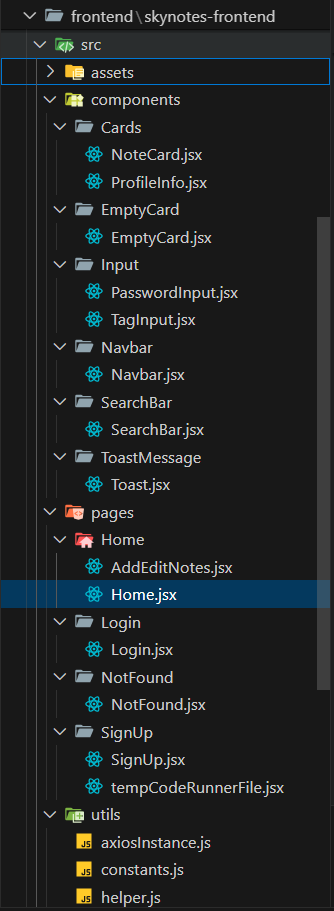
**2.1 Frontend Teknolojileri**

SkyNotes'un frontend geliştirme süreci, kullanıcı deneyimini merkeze alan bir yaklaşımla ilerletildi. Proje başlangıcında Vite ile hızlı ve modern bir React projesi kurularak temeller atıldı. İlk olarak, uygulamanın genel mimari yapısı ve bileşen hiyerarşisi planlandı. React Router DOM kullanılarak sayfa navigasyonu için sağlam bir altyapı oluşturuldu.

Tailwind CSS ile stil oluşturma süreci, hızlı ve responsive bir tasarım sağlamak üzere yapılandırıldı. Her bileşen, kullanıcı deneyimini optimize edecek şekilde detaylı olarak tasarlandı. Geliştirme sürecinde sürekli olarak bileşenlerin yeniden kullanılabilirliği ve kod tekrarından kaçınma prensipleri gözetildi.

* React: Kullanıcı arayüzü geliştirmek için modern JavaScript kütüphanesi
* React Router DOM: Sayfa yönlendirme ve navigasyon için
* Tailwind CSS: Hızlı ve esnek stil oluşturma için utility-first CSS framework'ü
* React Icons: Kullanıcı arayüzü için zengin icon seti



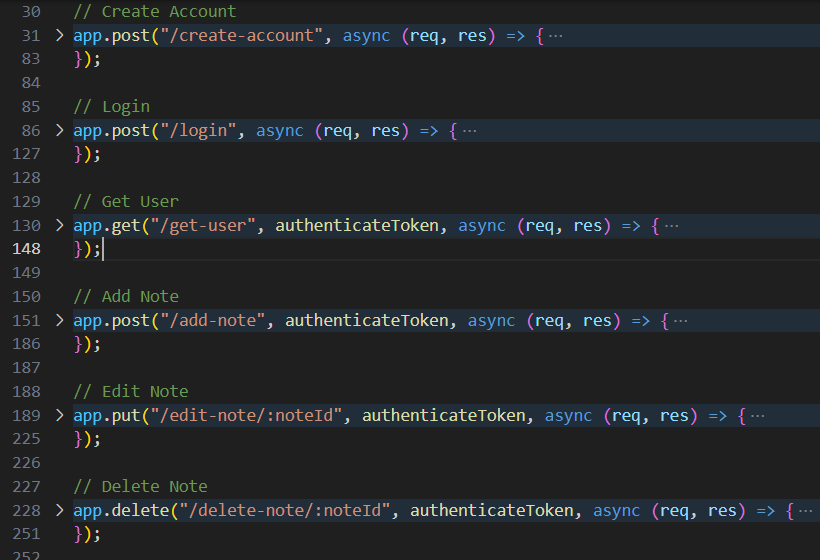


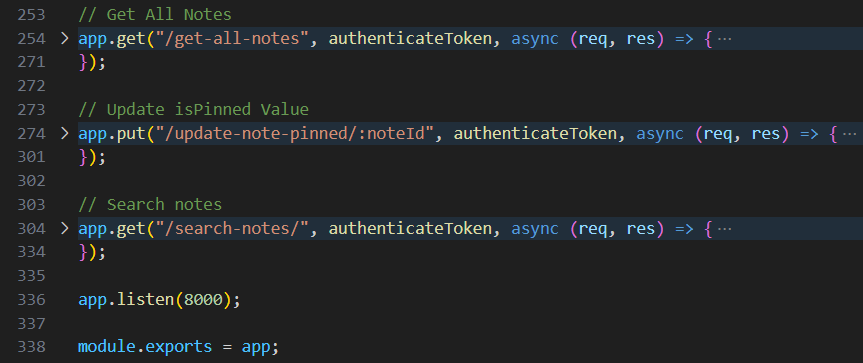
**2.2 Backend Teknolojileri**

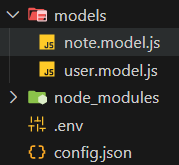
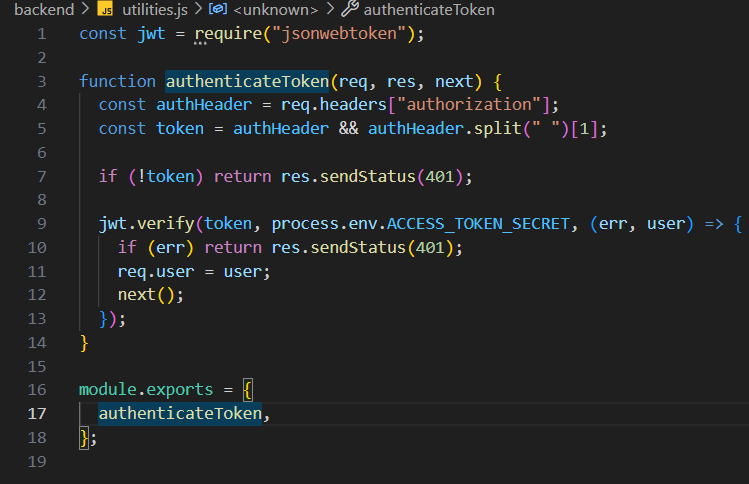
* Node.js: Sunucu tarafı JavaScript çalışma ortamı
* Express.js: Web uygulaması ve API geliştirme framework'ü
* Mongoose: MongoDB ile etkileşim için Object Data Modeling (ODM) kütüphanesi
* JSON Web Token (JWT): Güvenli kullanıcı kimlik doğrulama mekanizması
* dotenv: Ortam değişkenlerini yönetme
* cors: Cross-Origin Resource Sharing yönetimi

Backend geliştirme süreci, güvenli ve ölçeklenebilir bir API mimarisi oluşturmaya odaklandı. Node.js ve Express.js kombinasyonu, hızlı ve esnek bir sunucu tarafı çözümü sundu. Mongoose ORM kullanılarak MongoDB ile olan veri etkileşimi optimize edildi ve veri modellemesi için güçlü bir altyapı kuruldu.

Kullanıcı ve not modellerinin tasarımında, veri bütünlüğü ve güvenliği ön planda tutuldu. JWT (JSON Web Token) ile kimlik doğrulama mekanizması uygulandı, bu sayede kullanıcı oturumları güvenli bir şekilde yönetildi. Her API endpoint'i, hem güvenlik hem de performans açısından detaylı olarak test edildi.

****

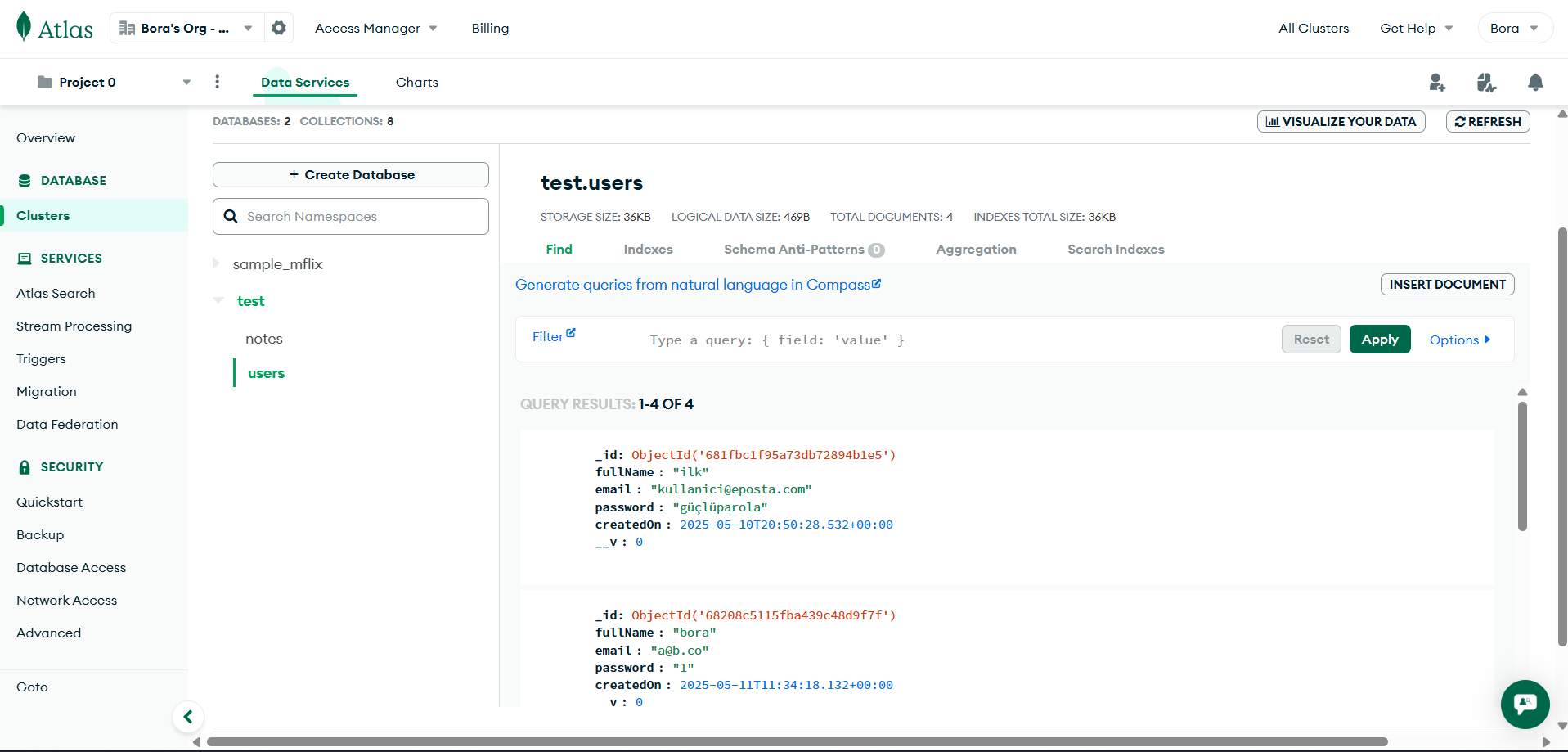
****

****

**2.3 Veritabanı**

* MongoDB: NoSQL, esnek ve ölçeklenebilir bulut tabanlı veritabanı





**3. Uygulama Özellikleri**

**3.1 Kullanıcı İşlevselliği**

* Kullanıcı kayıt ve giriş sistemi
* Not oluşturma, düzenleme ve silme
* Not arama fonksiyonu
* Notları sabitleme (pin) özelliği
* Okunabilir tarih formatı
* Bilgilendirme mesajları (Toast)
* Boş not listesi için özel kullanıcı arayüzü

**4. Geliştirme Süreci**

**4.1 Proje Kurulumu**

* Proje npm create vite@latest kullanılarak oluşturuldu
* Tailwind CSS resmi dokümantasyonuna göre entegre edildi
* Google Fonts'dan Noto Sans fontu kullanıldı
* Tarih biçimlendirme için moment kütüphanesi entegre edildi

**4.2 Frontend Geliştirme Aşamaları**

* Sayfa ve bileşen tasarımı
* React Router DOM ile navigasyon
* Tailwind CSS ile stil oluşturma
* Özel bileşenler geliştirildi:
* Navbar
* Giriş ekranı
* Şifre girişi bileşeni
* Arama çubuğu
* Not ekleme/düzenleme modalı
* Etiket girişi bileşeni

**4.3 Backend Geliştirme Aşamaları**

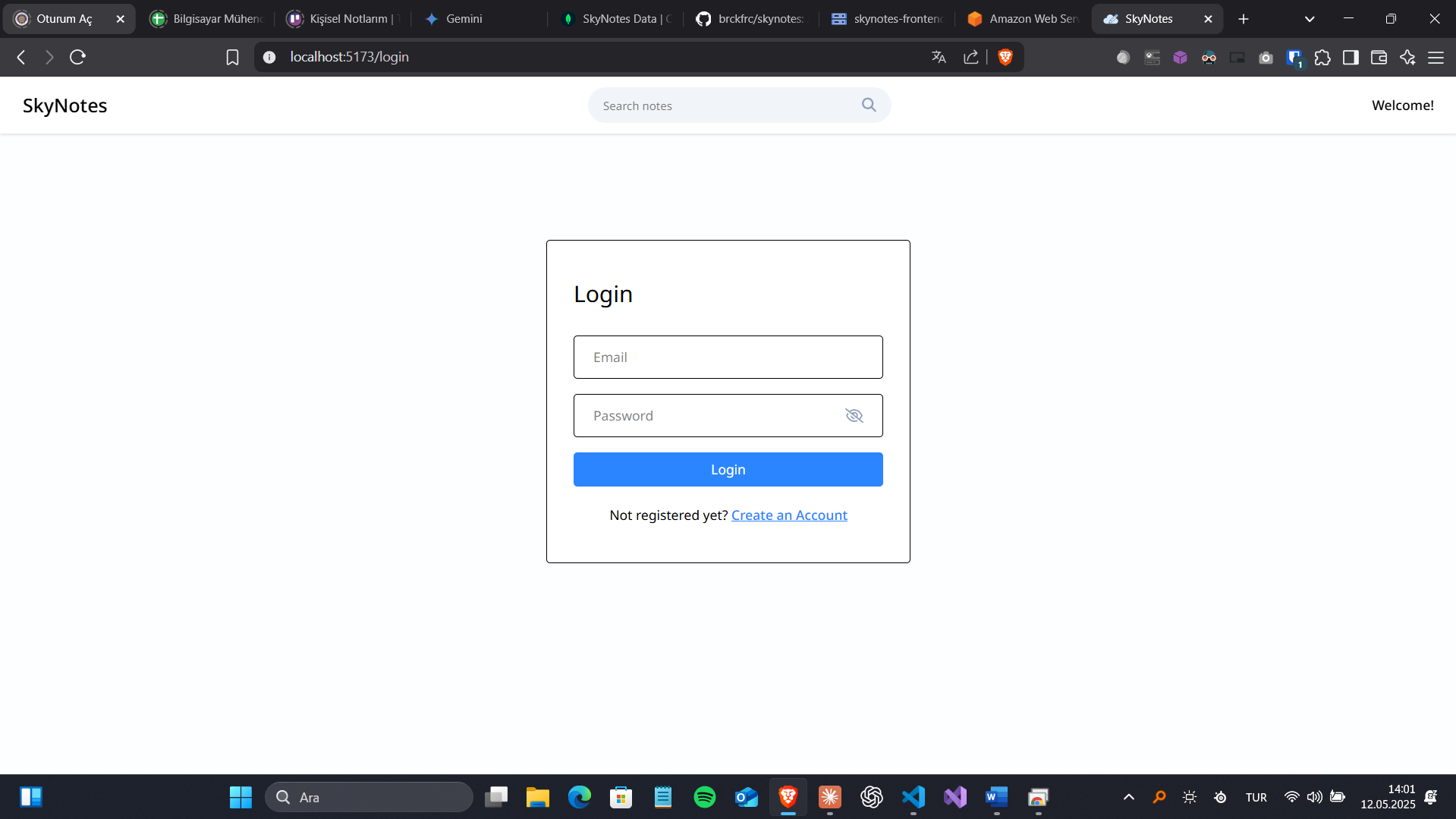
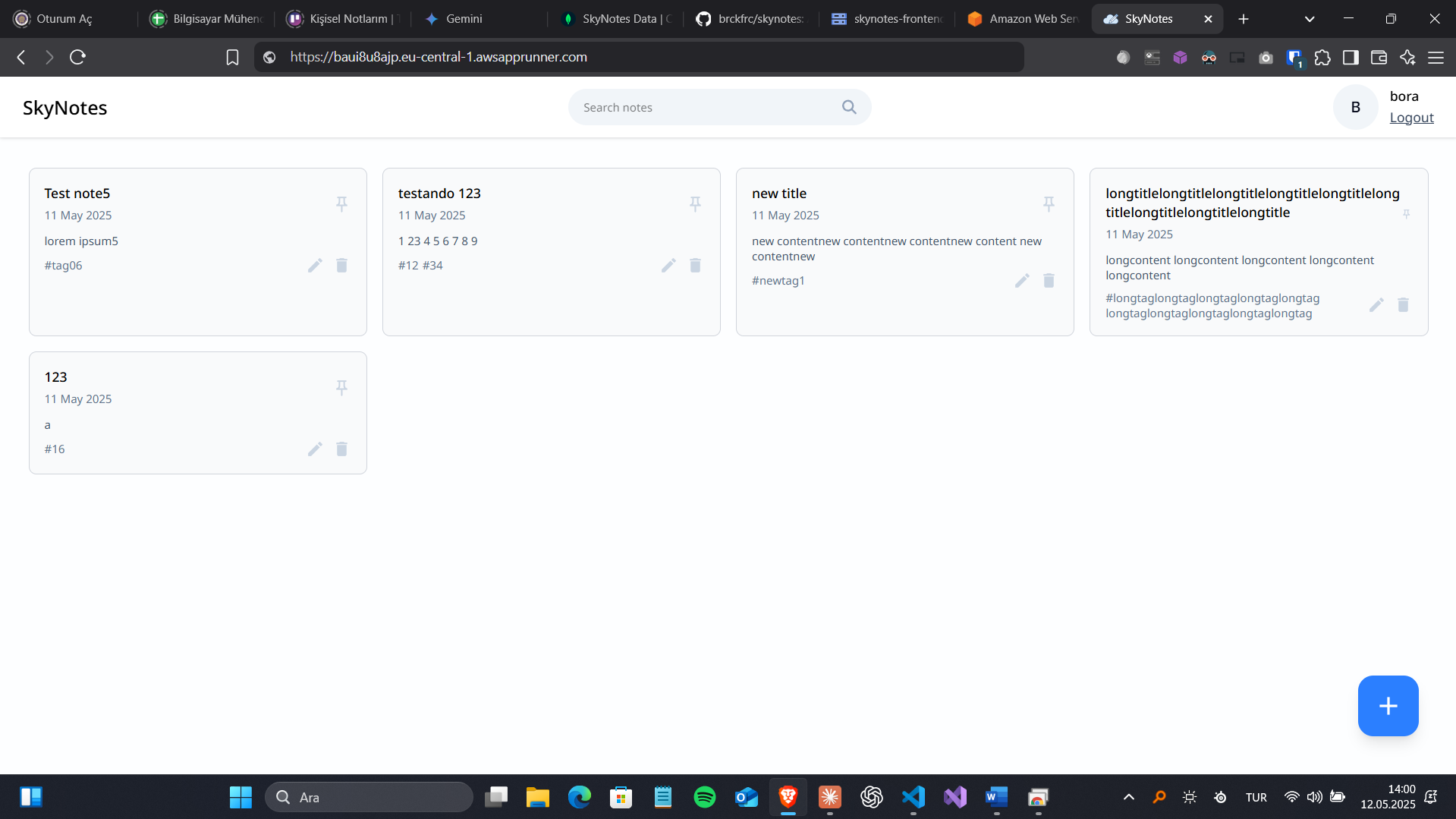
* Node.js, Express.js ve Mongoose ile API geliştirme
* Kullanıcı ve Not modelleri oluşturma
* API endpoint'leri hazırlama:
* Kullanıcı kayıt
* Kullanıcı girişi
* Not oluşturma
* Not düzenleme
* Not silme
* Not getirme
* Not sabitleme

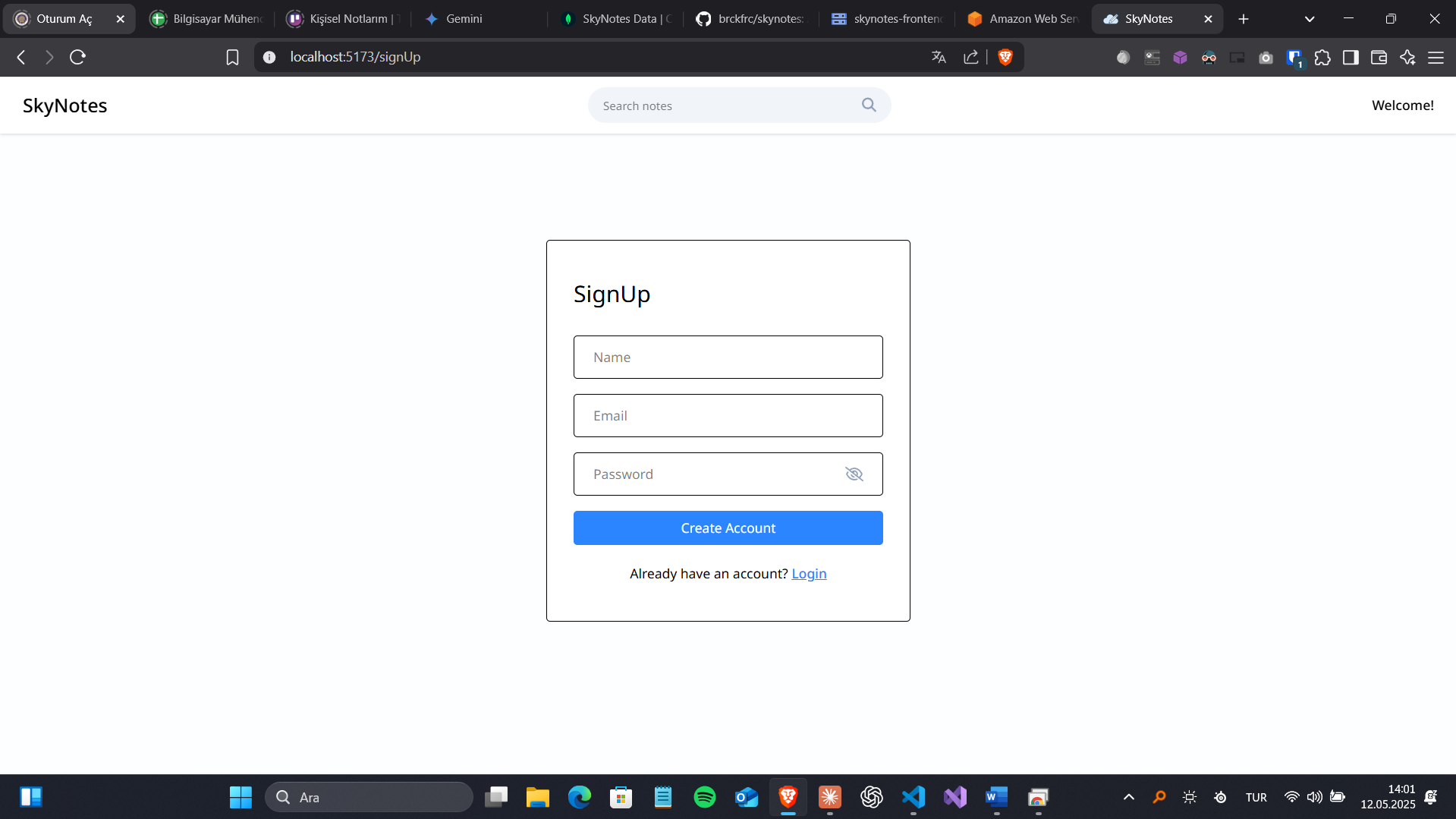
**4.4 Entegrasyon**

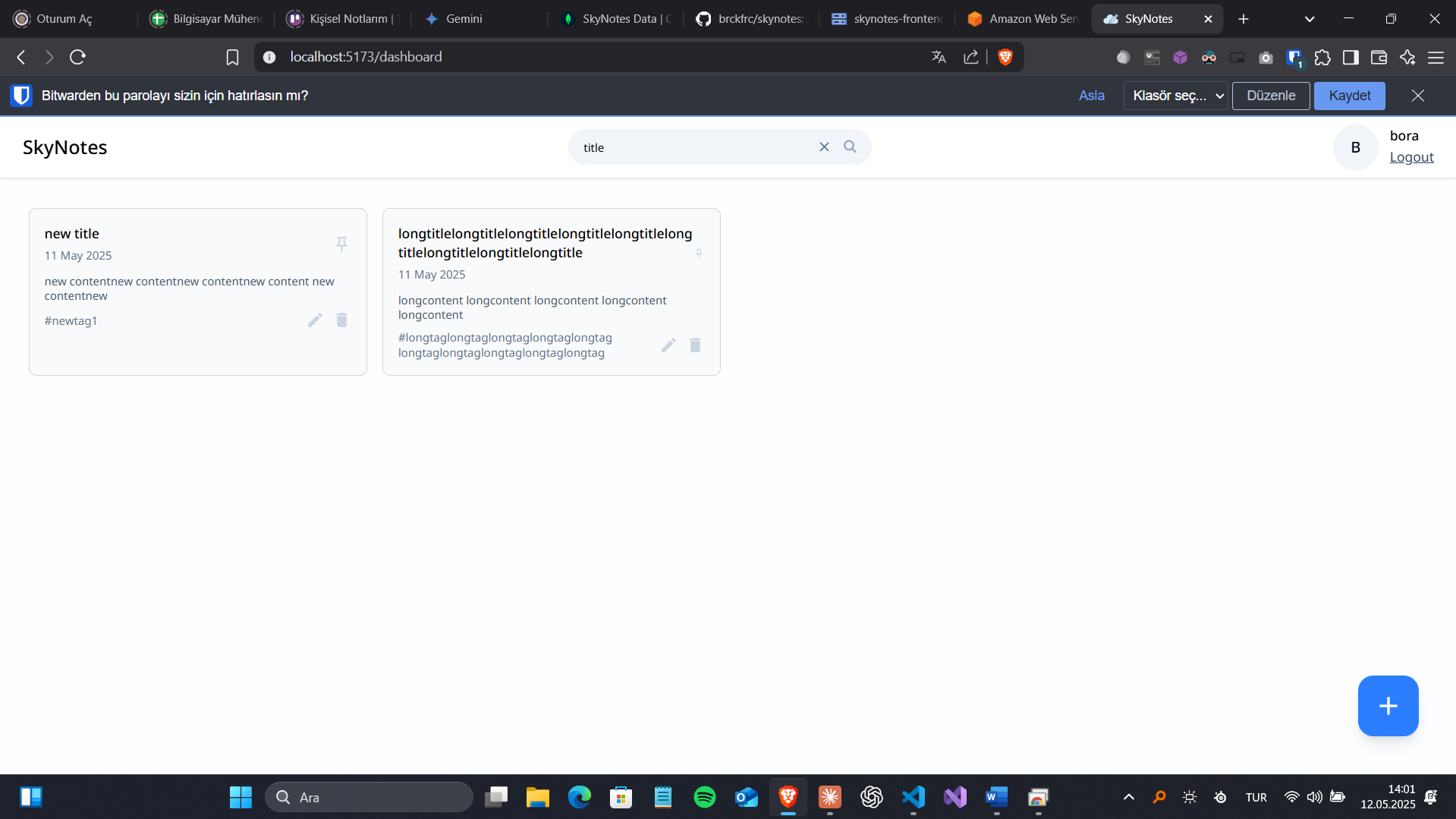
* Frontend ve backend arasında API bağlantıları kuruldu
* Kullanıcı bilgileri backend'den çekilip görüntülendi
* Kullanıcıya özel notlar listelendi
* Arama fonksiyonu entegre edildi
* Bilgilendirme mesajları (toast) eklendi

**5. Sonuç**

SkyNotes projesi, modern web geliştirme teknolojilerini kullanarak kullanıcı merkezli, esnek bir not yönetim uygulaması oluşturmayı başarmıştır. Proje, frontend ve backend teknolojilerinin entegrasyonunu gösteren kapsamlı bir örnek teşkil etmektedir.

****

****

****

**PROJE 2: Akıllı Veri Analitiği ve Makine Öğrenmesi Uygulaması**

**Proje Konusu: AWS kullanarak telefon özelliklerini bulunduran veri setini eğiterek fiyat aralık tahmini yapan modeli buluta deploy ettik ve buradan API oluşturarak veritabanı uygulamasında veri analizi gerçekleştirdik.**

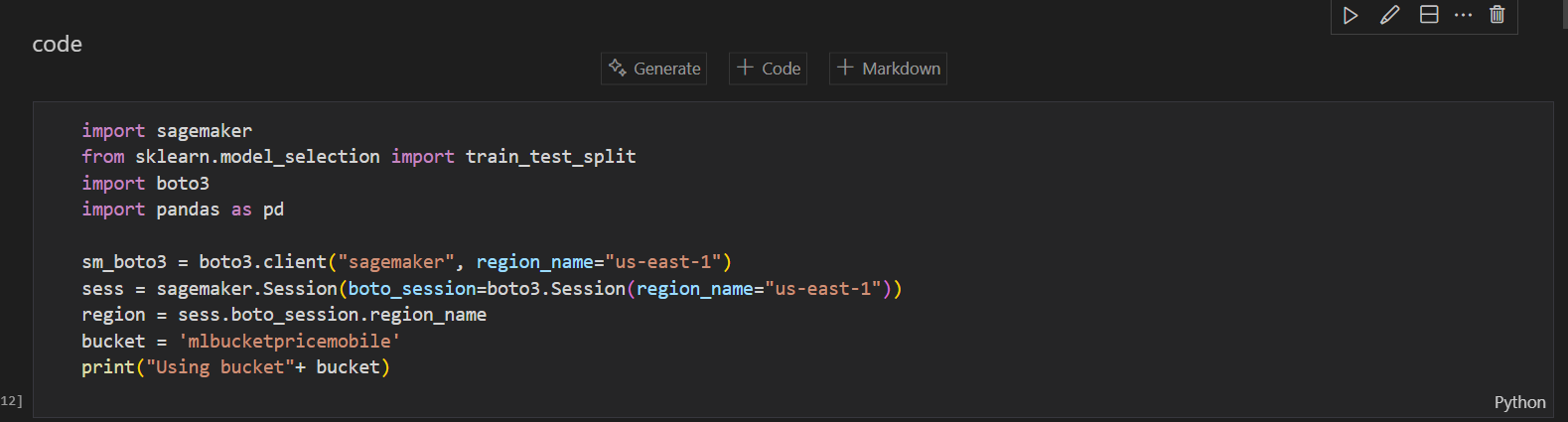
**Bu projede;**

**Backend dili: Python, Node.js**

**Makine Öğrenmesi Kütüphaneleri,**

**Veri Tabanı: MongoDB,**

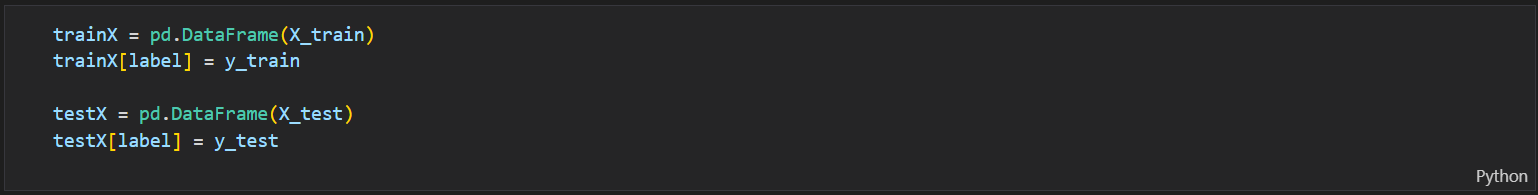
**Bulut Platformu: AWS (S3 Bucket, Sagemaker, Lambda, API Gateway)**

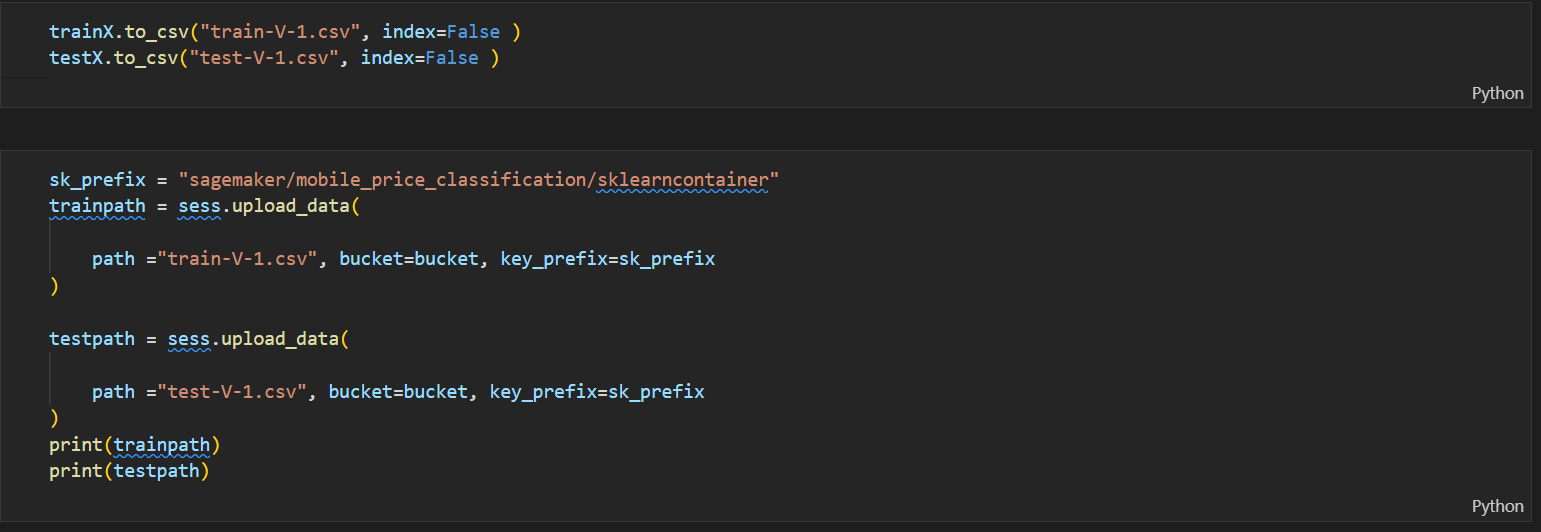
****

S3 Bucket oluşturarak localdeki kod ile bağlantı sağlandı.

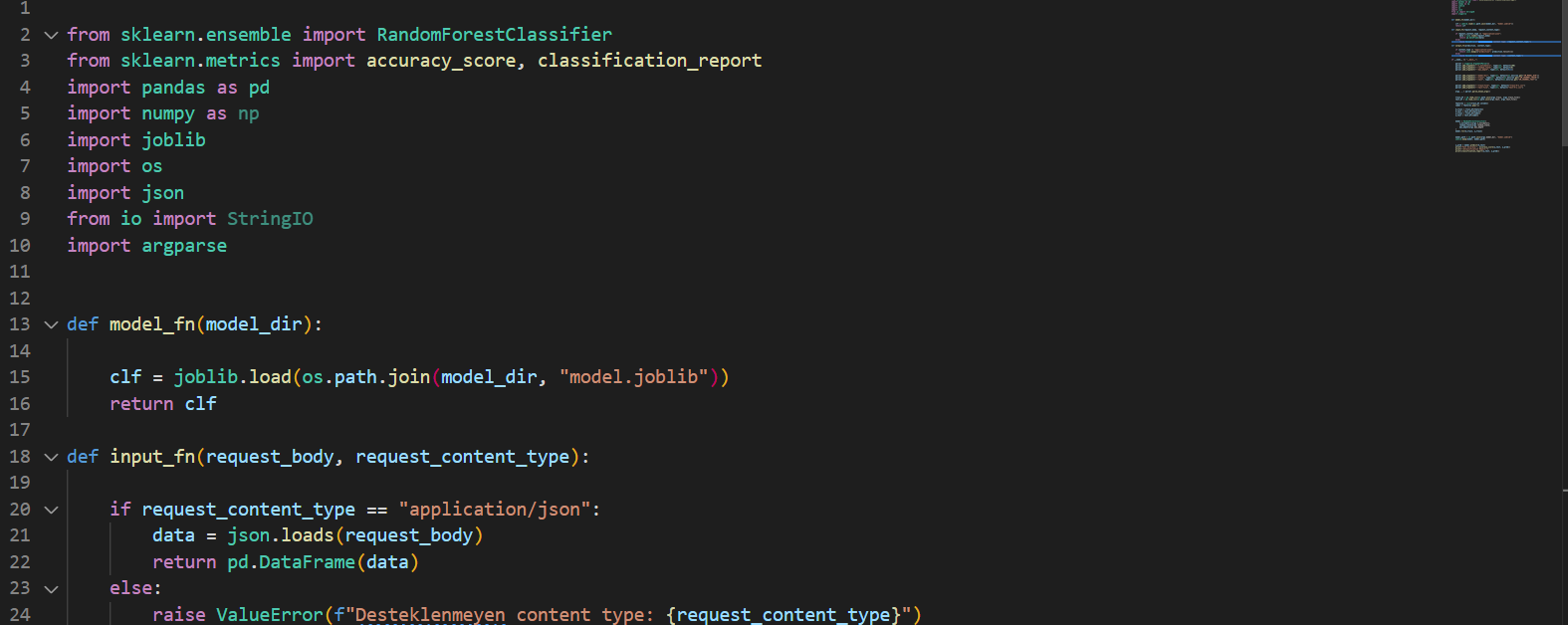


Elimizde bulunan dataset’i analiz edilerek eksiklikler gözlemlendi.

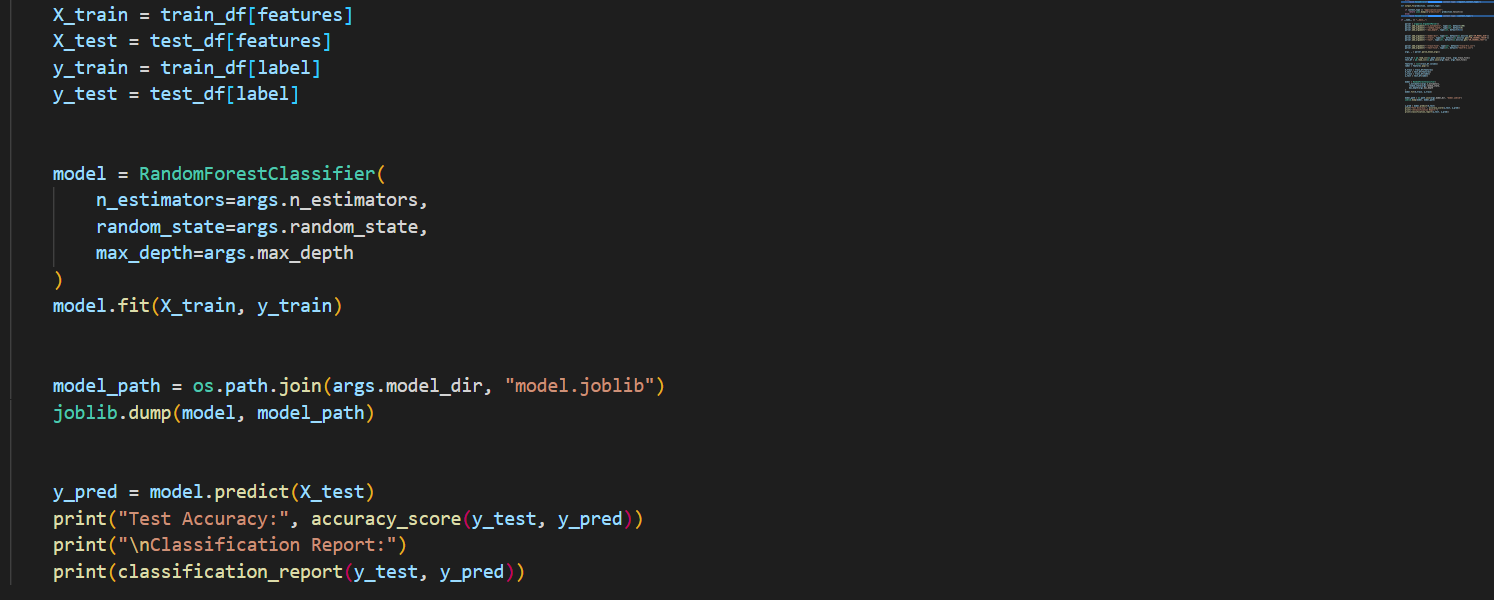




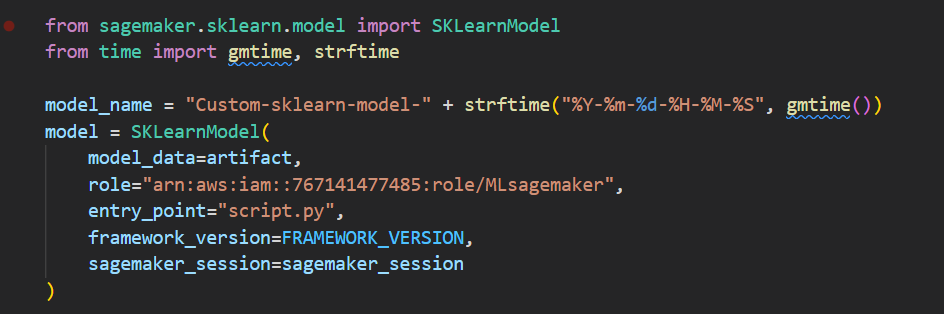
Bucket’ a var olan train.csv ile test.csv dosyaları yüklendi.



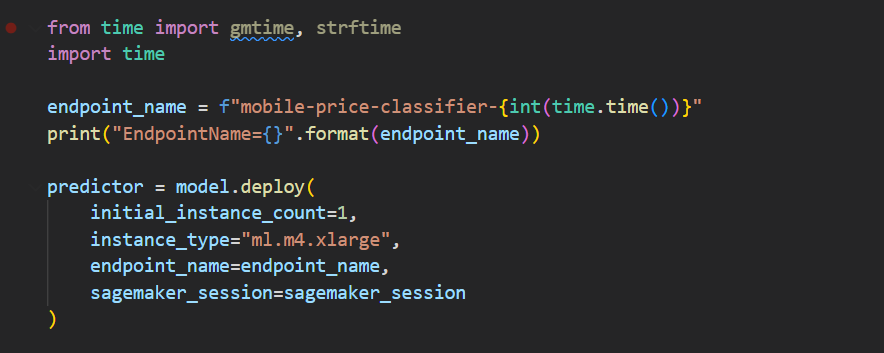




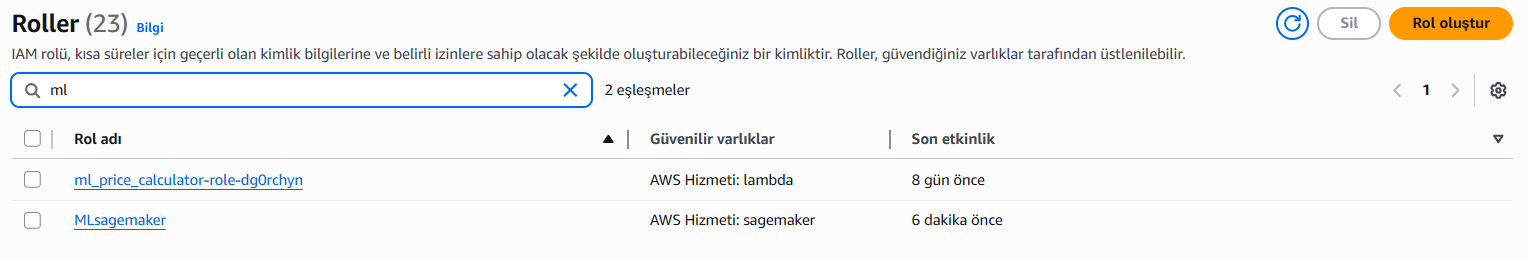
Randomforest ile elimizde bulunan dataseti eğitmek için ve daha sonra model eğitiminde kullanılması için script.py üzerine overwrite edildi.

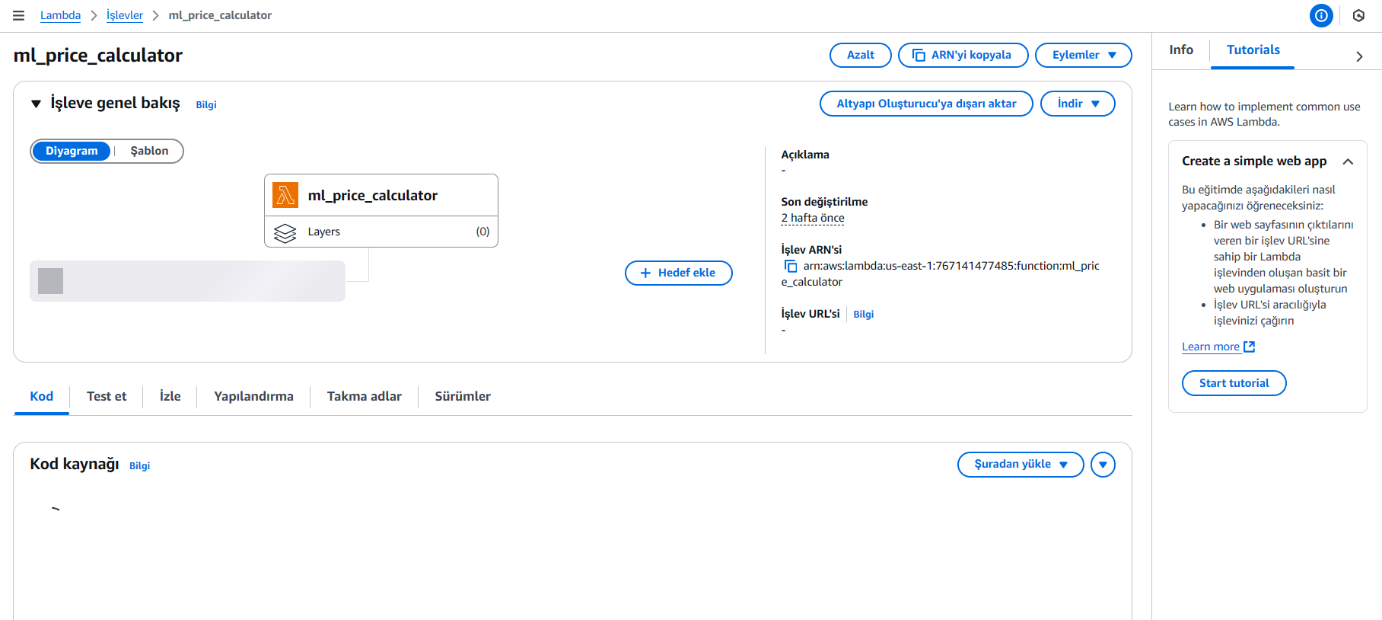


Aws üzerinde IAM rolü belirlendi ve bağlantı kurması için koda eklendi. Arından script.py ile model eğitildi.

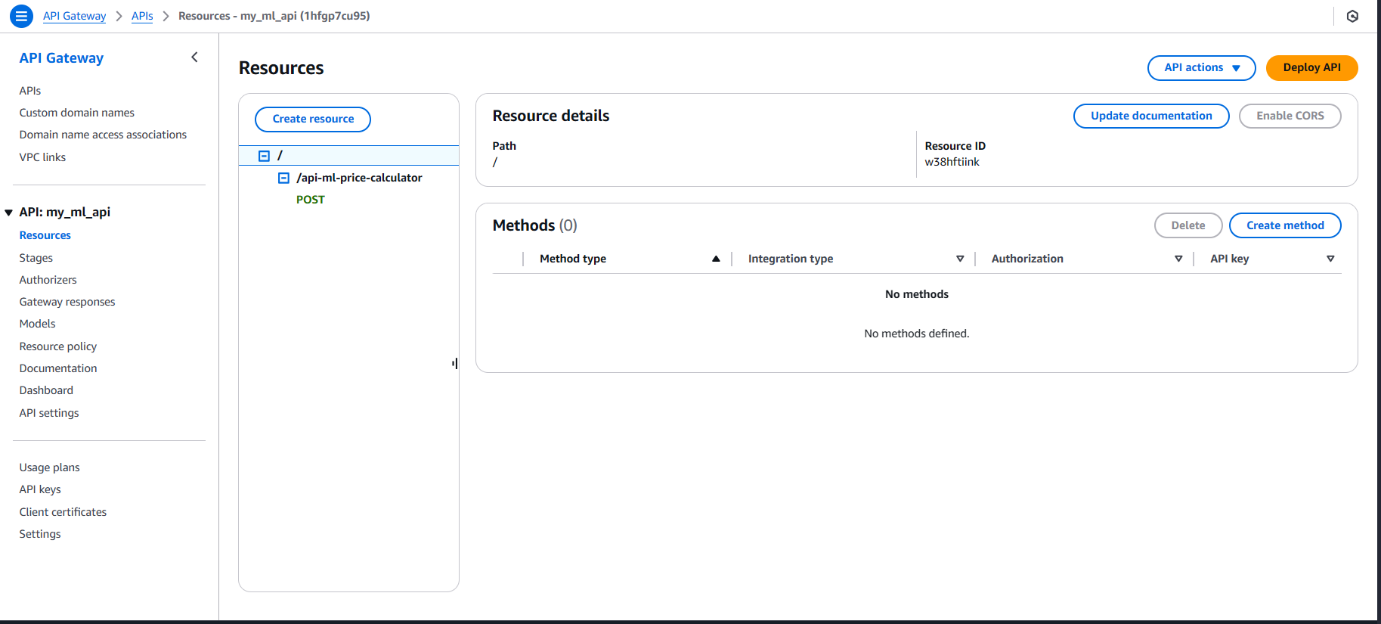


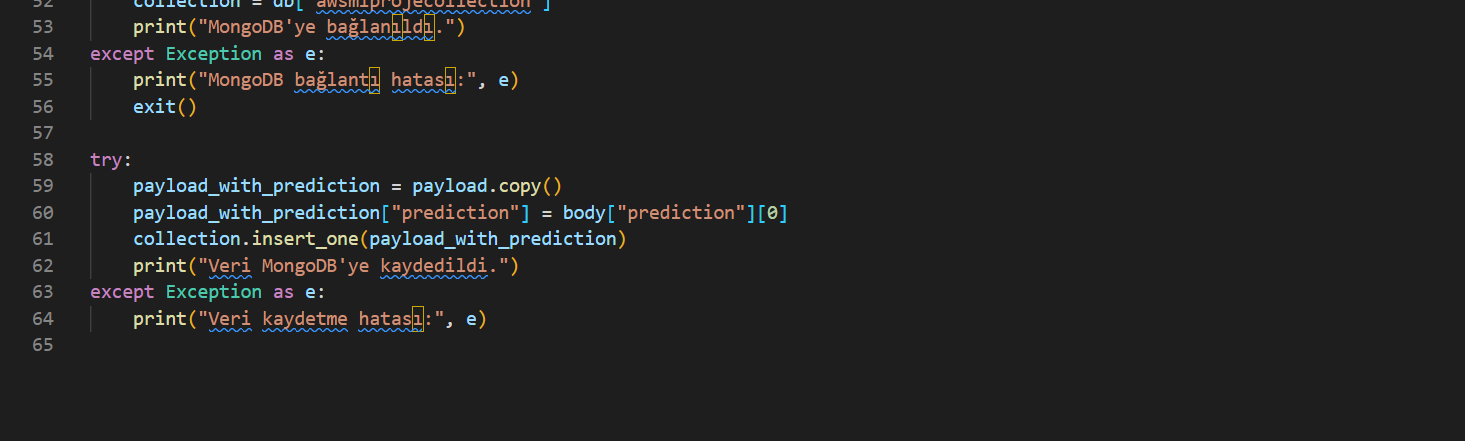
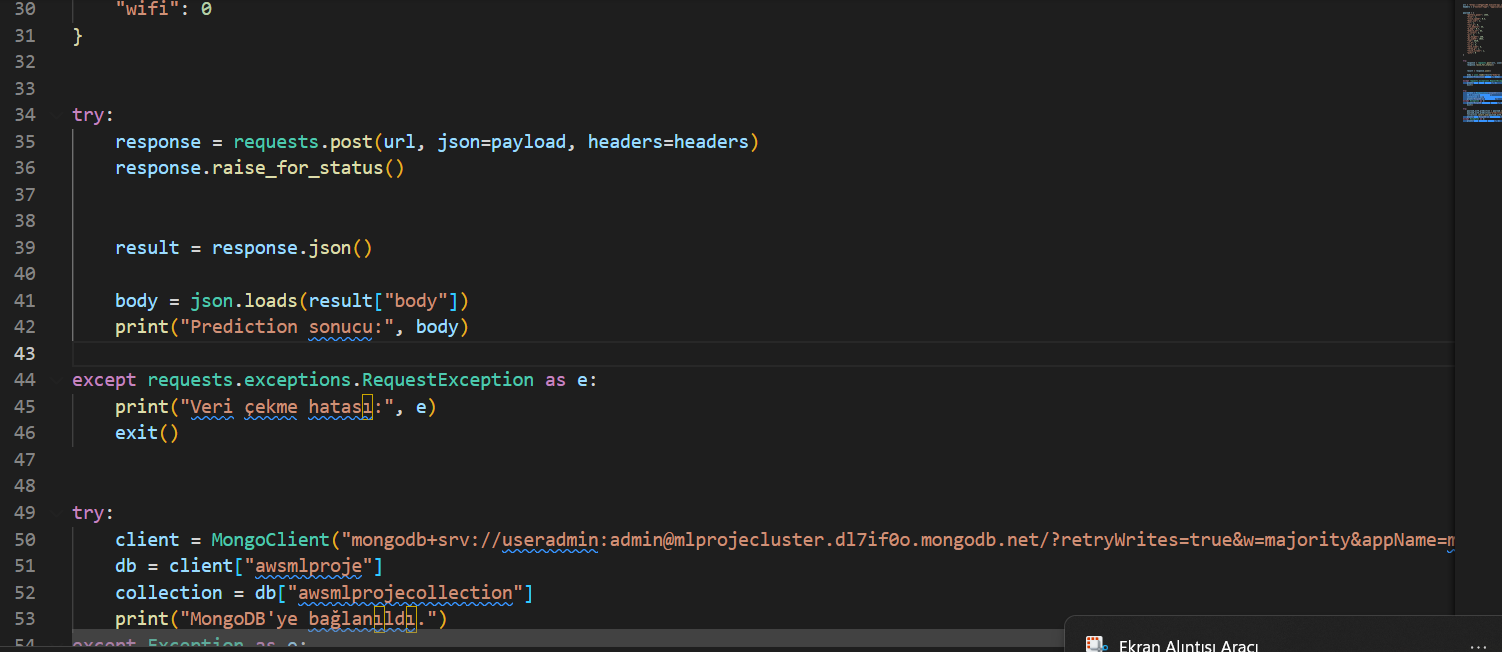
Bu kısımda Sagemaker için endpoint oluşturuldu.



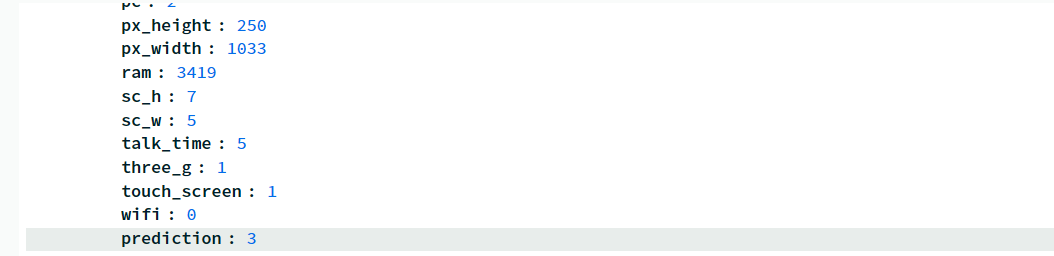
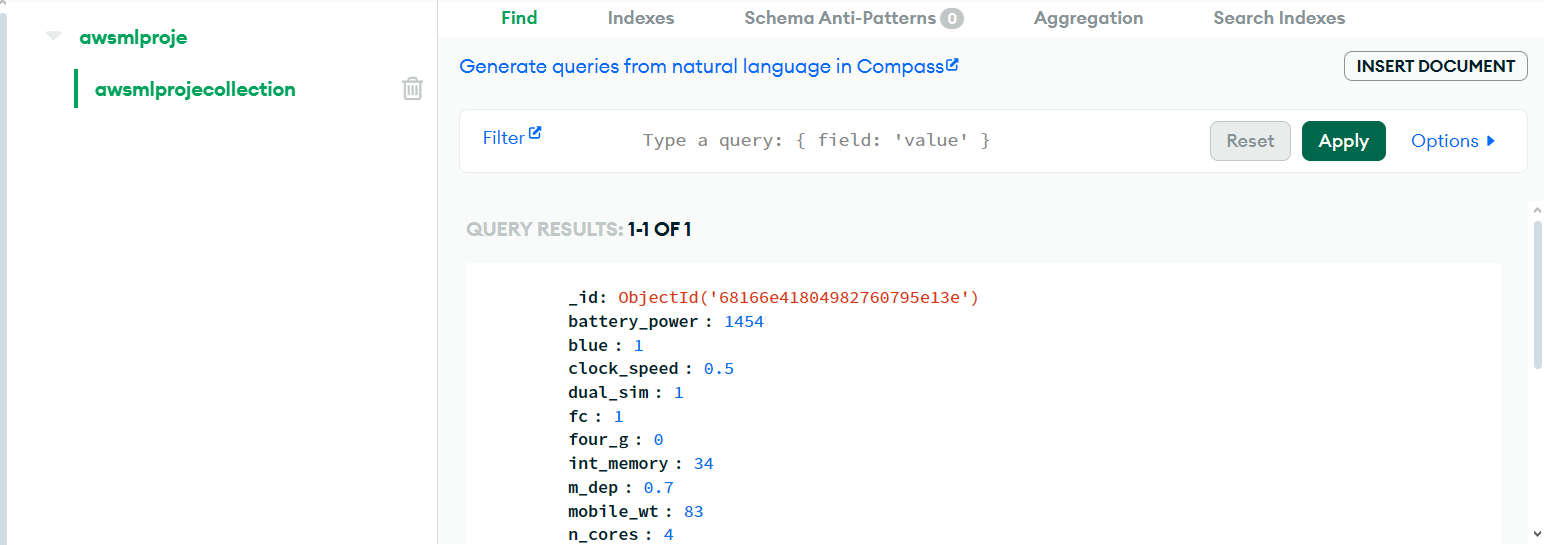


Ardından, Lambdaya IAM üzerinden sagemaker erişim rolü verildi. Ardından bu endpoint Lambda’ya bağlandı.



API Gateway kullanılarak MongoDB için RESTAPI oluşturuldu ve deploy edildi.

Bu kısımda MongoDB ile AWS API bağlantısı sağlandı ardından local kodda bu bağlantı gösterildi. Test verisi için veri yollandı ve ardından kod çalıştırılarak API kullanıldı



Sonuç olarak, MongoDB’ye Tahmin ve test verisi gönderildi ve gözlemlendi.

**PROJE 3: E-Ticaret Uygulaması (Otomatik Ölçeklendirme ve Yönetim)**

**Proje Konusu: Microsoft Azure ile E-Ticaret sitesi**

[**https://nice-sand-04de3bb0f.6.azurestaticapps.net**](https://nice-sand-04de3bb0f.6.azurestaticapps.net)

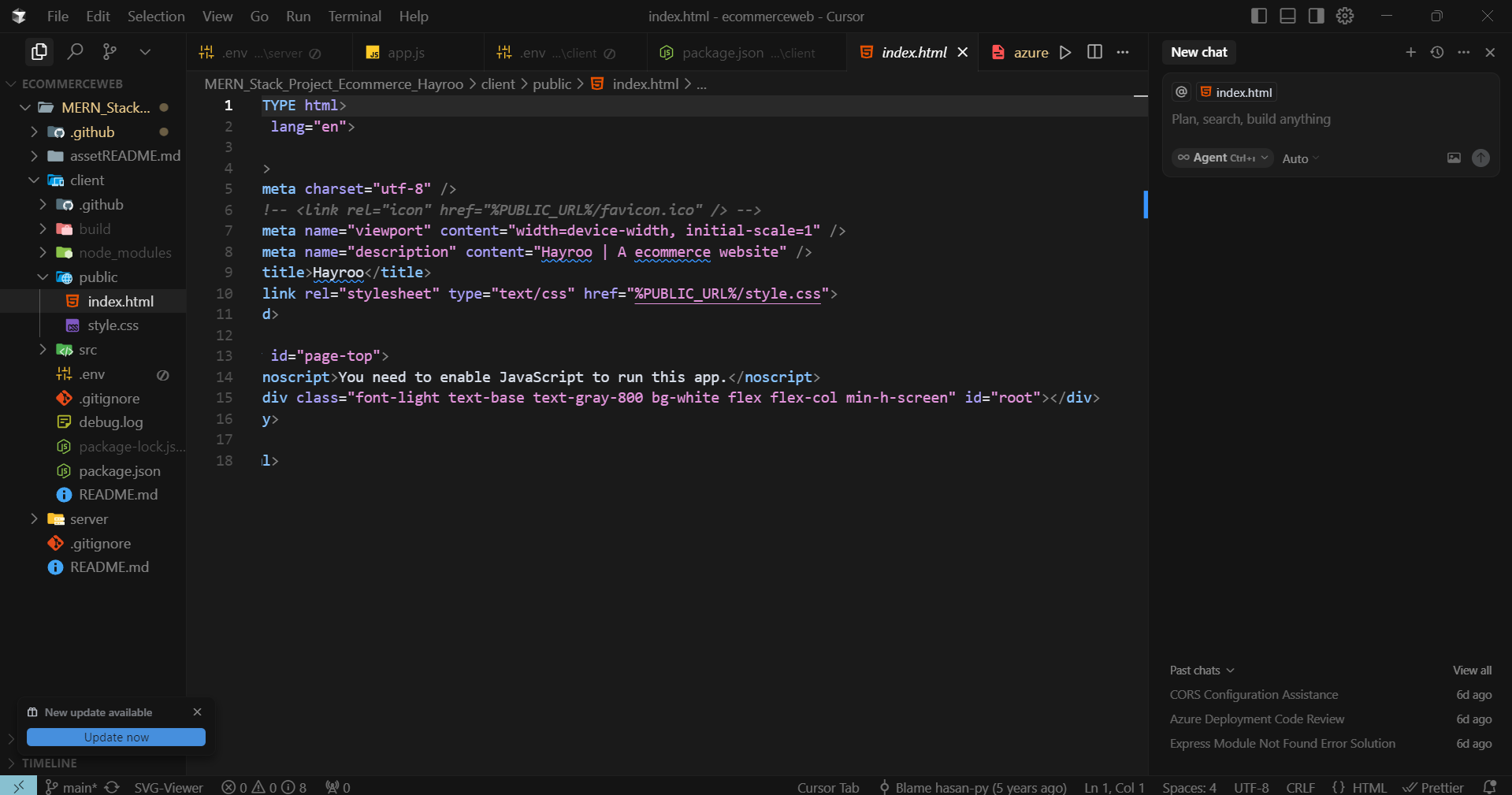
**Bu projede;**

**Frontend dili: JavaScript,**

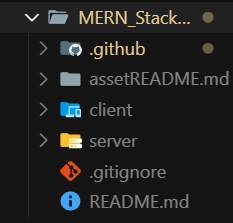
**Backend dili: Node.js,**

**Veri Tabanı: MongoDB,**

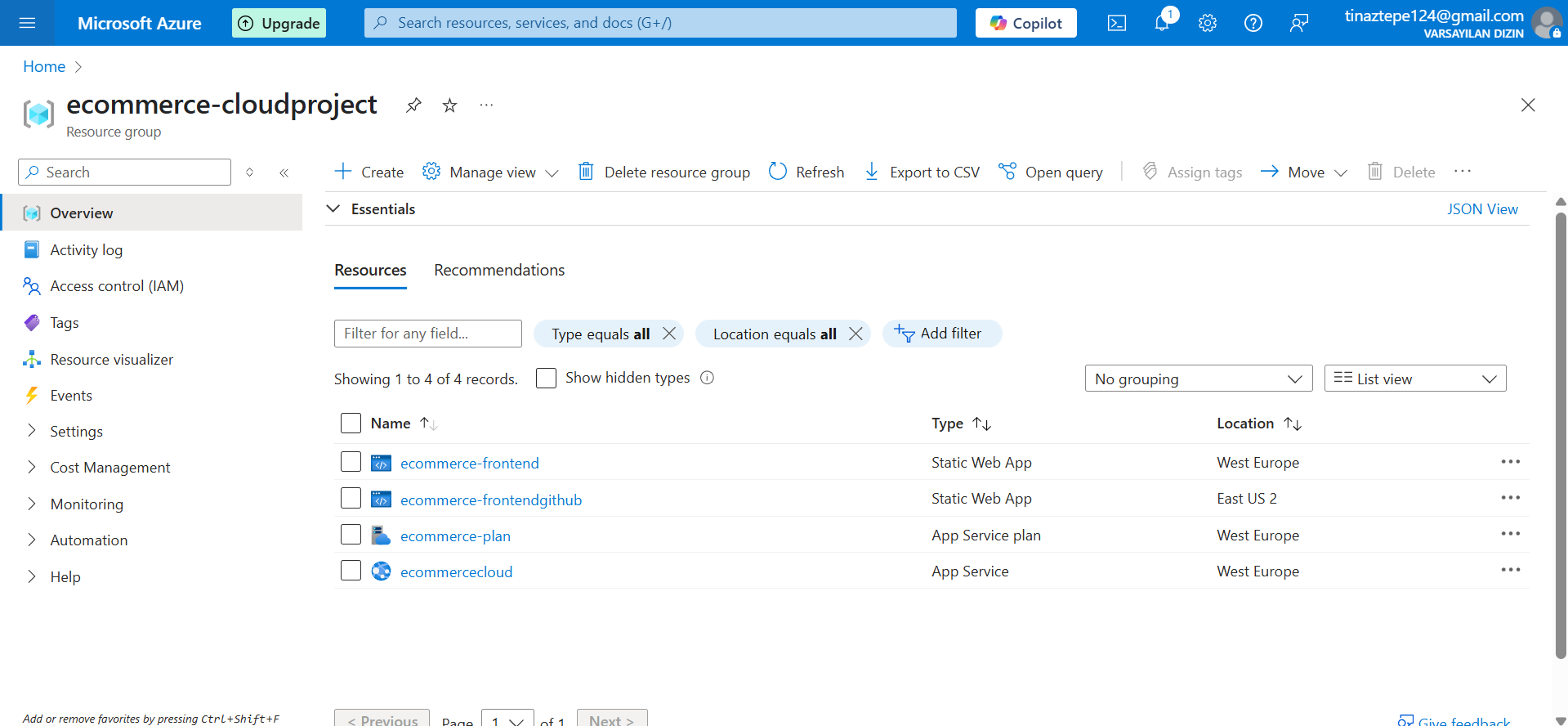
**Bulut Platformu: Microsoft Azure (Azure App Services)**



Bu projede, Cloud bağlantısı yapılanabilecek hazır website taslağı kullanıldı. Src: <https://github.com/hasan-py/MERN_Stack_Project_Ecommerce_Hayroo>

server dosyası backendi barındırırken (node.js),

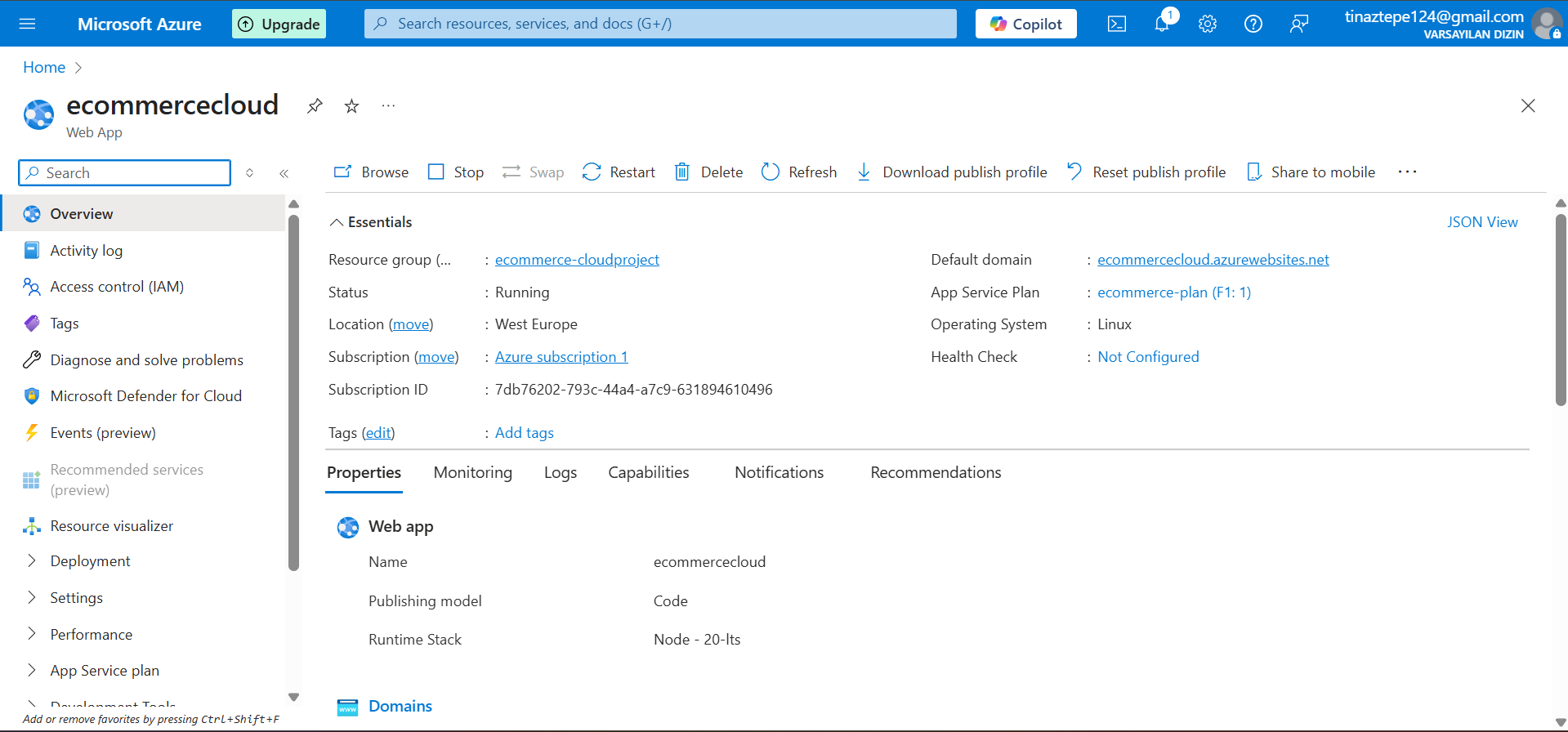
client dosyası frontendi barındırmaktadır. (Html, Css, Javascript)



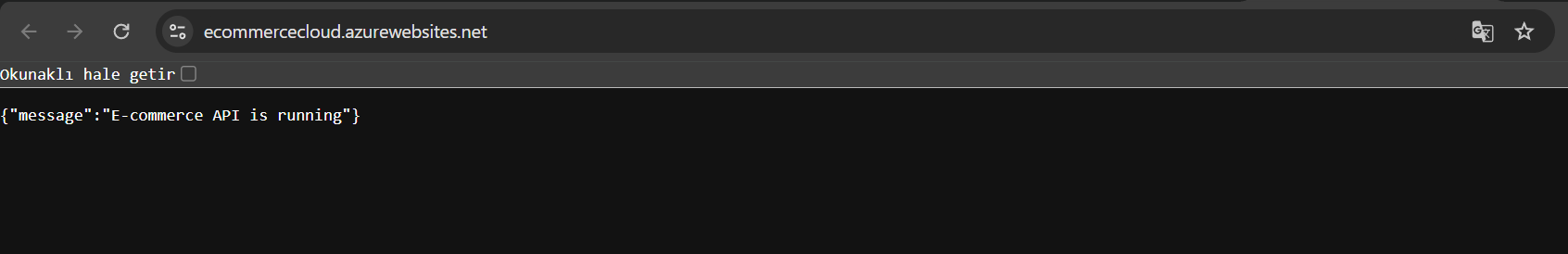
İlk olarak Microsoft Azure üzerinde dosyaların barınacağı Resource Group oluşturuldu.



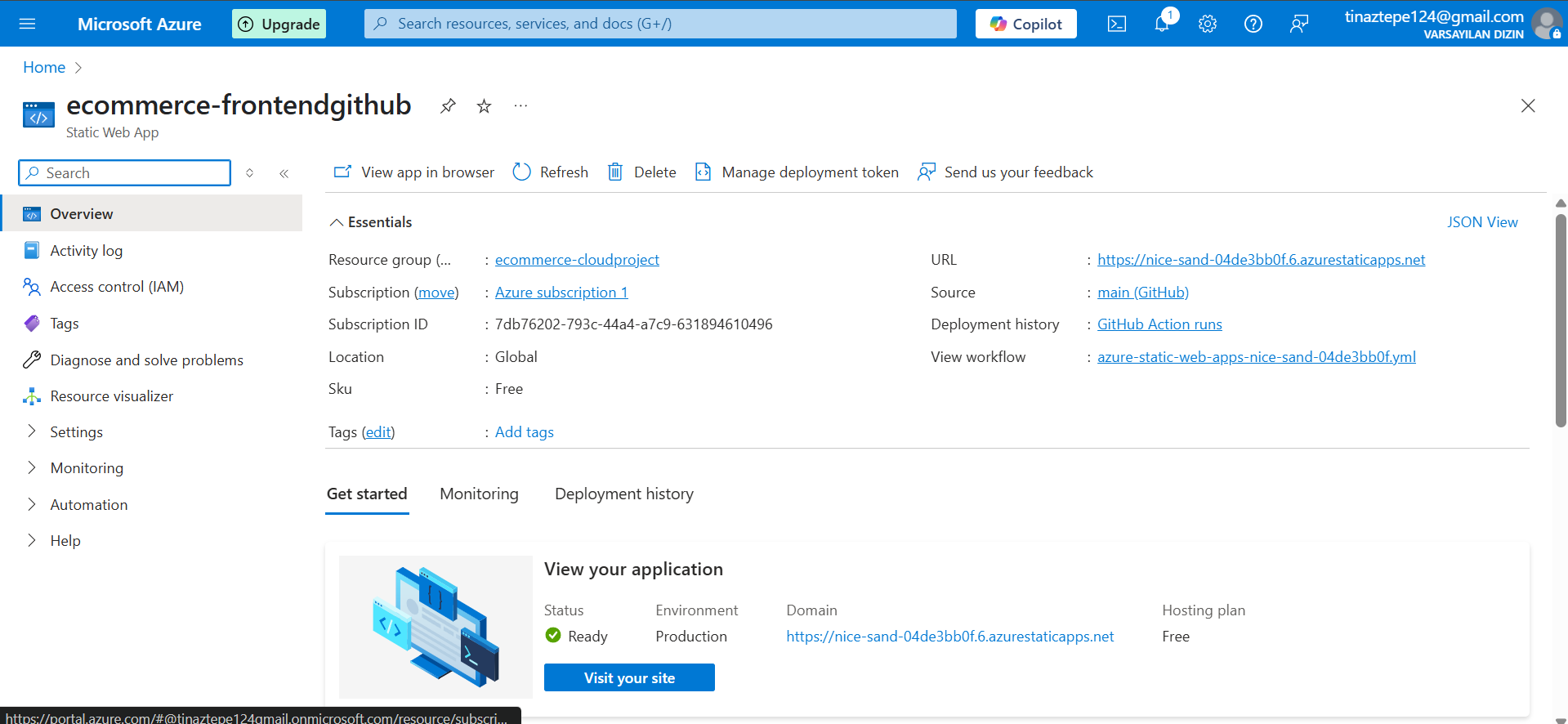
Devamında, Linux tabanlı plan hizmete ayrıldı ve resource group bağlantısı yapıldı.



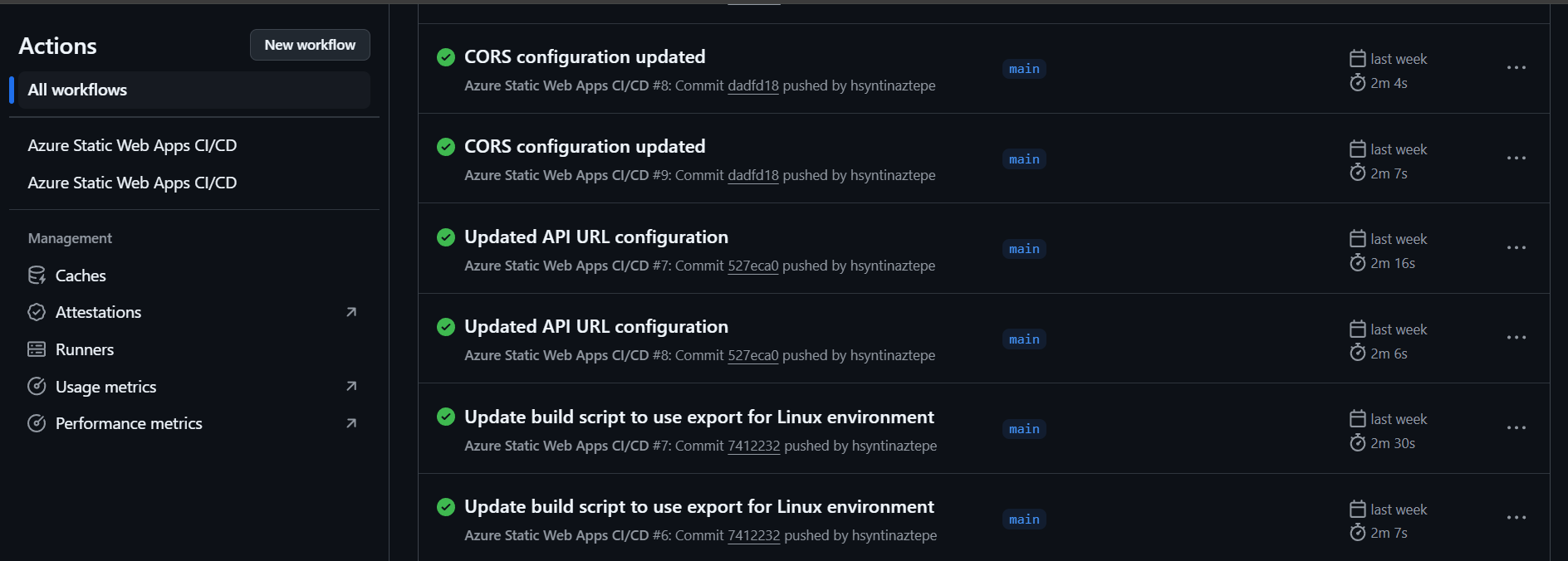
Bu kısımda, Azure Web App Service oluşturuldu.

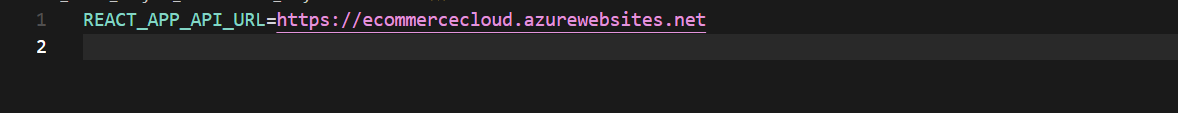
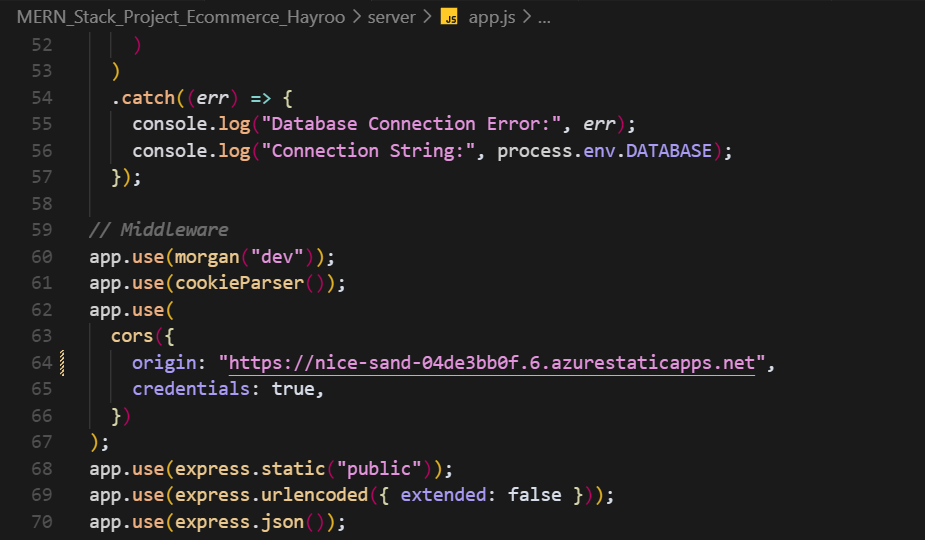


Azure Web App Service üzerinde localdeki kodun backend kısmı ile App Service API ile bağlandı. Backend sunucuya yüklendi.

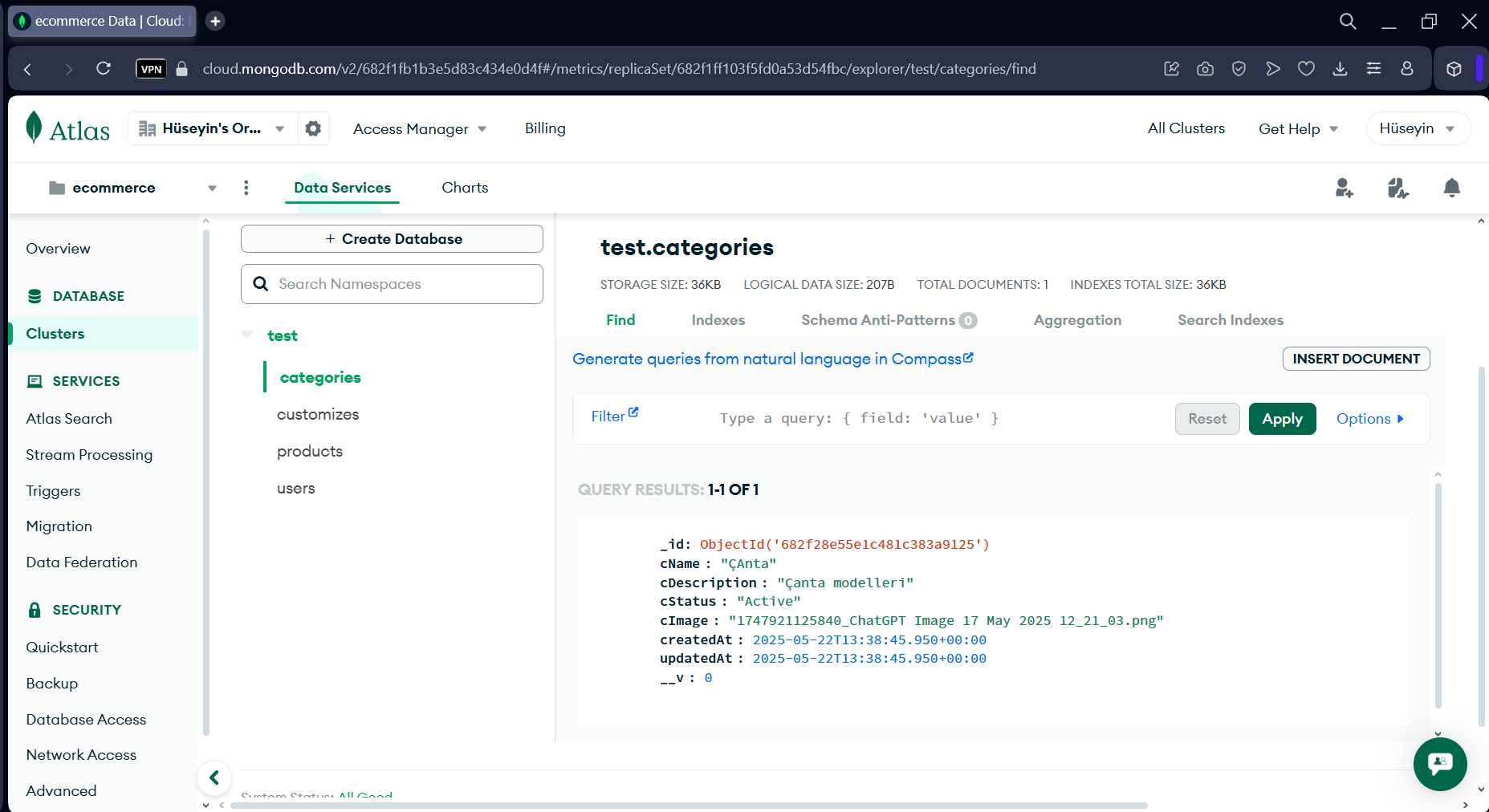


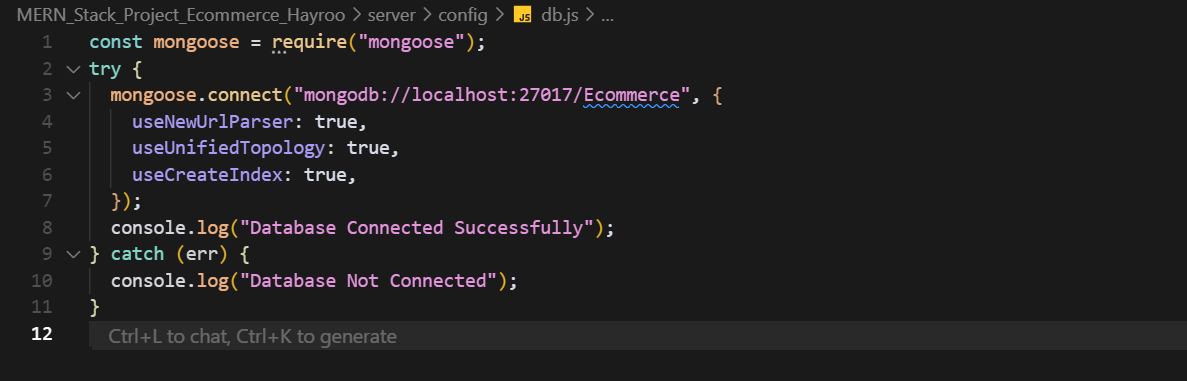
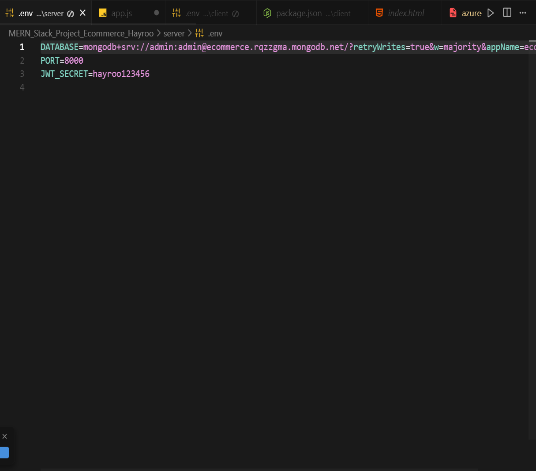
Microsoft Azure üzerinde Static Web App oluşturuldu. Localdeki frontend kodu Github aracılığıyla deploy edildi.



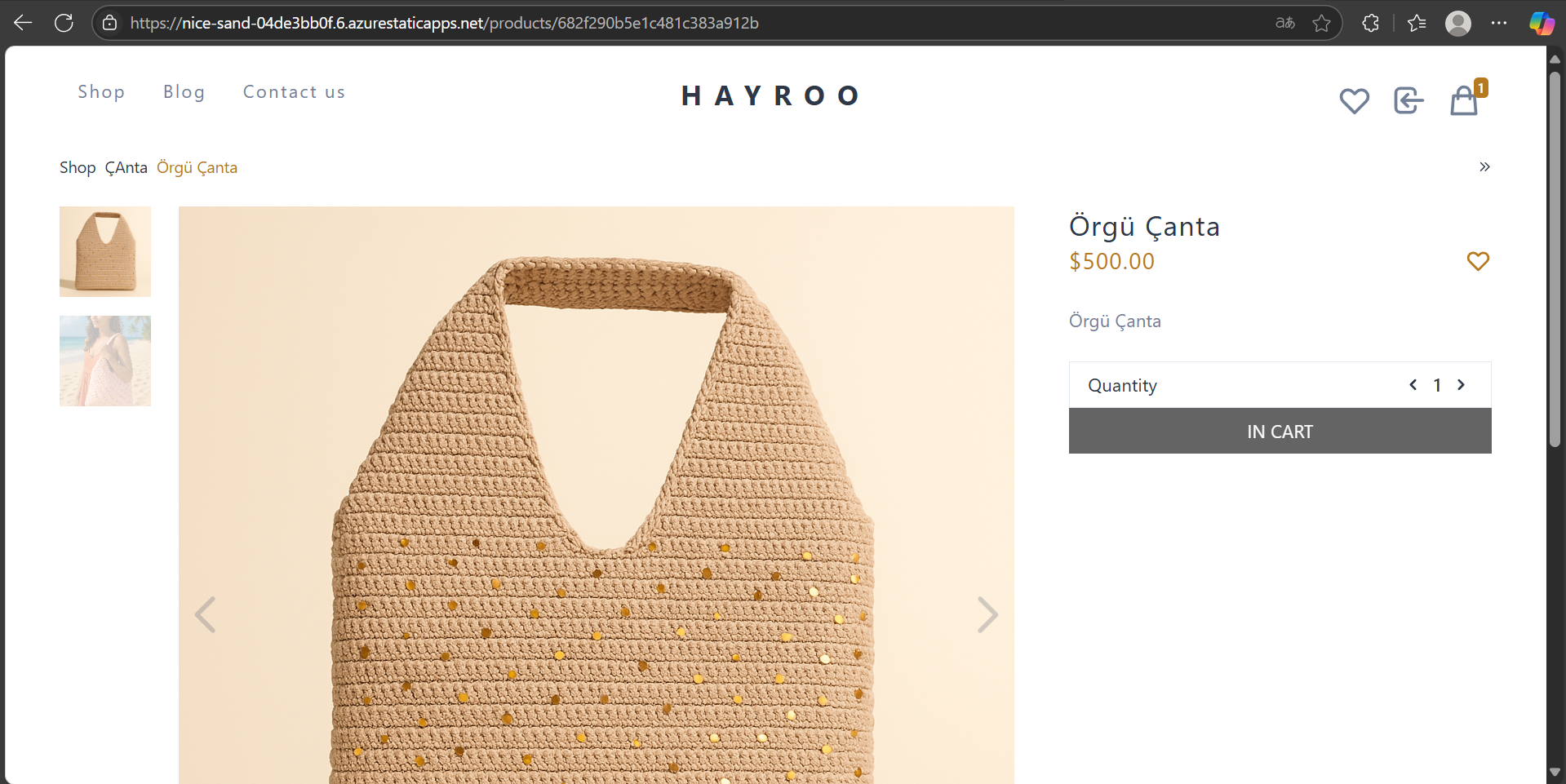


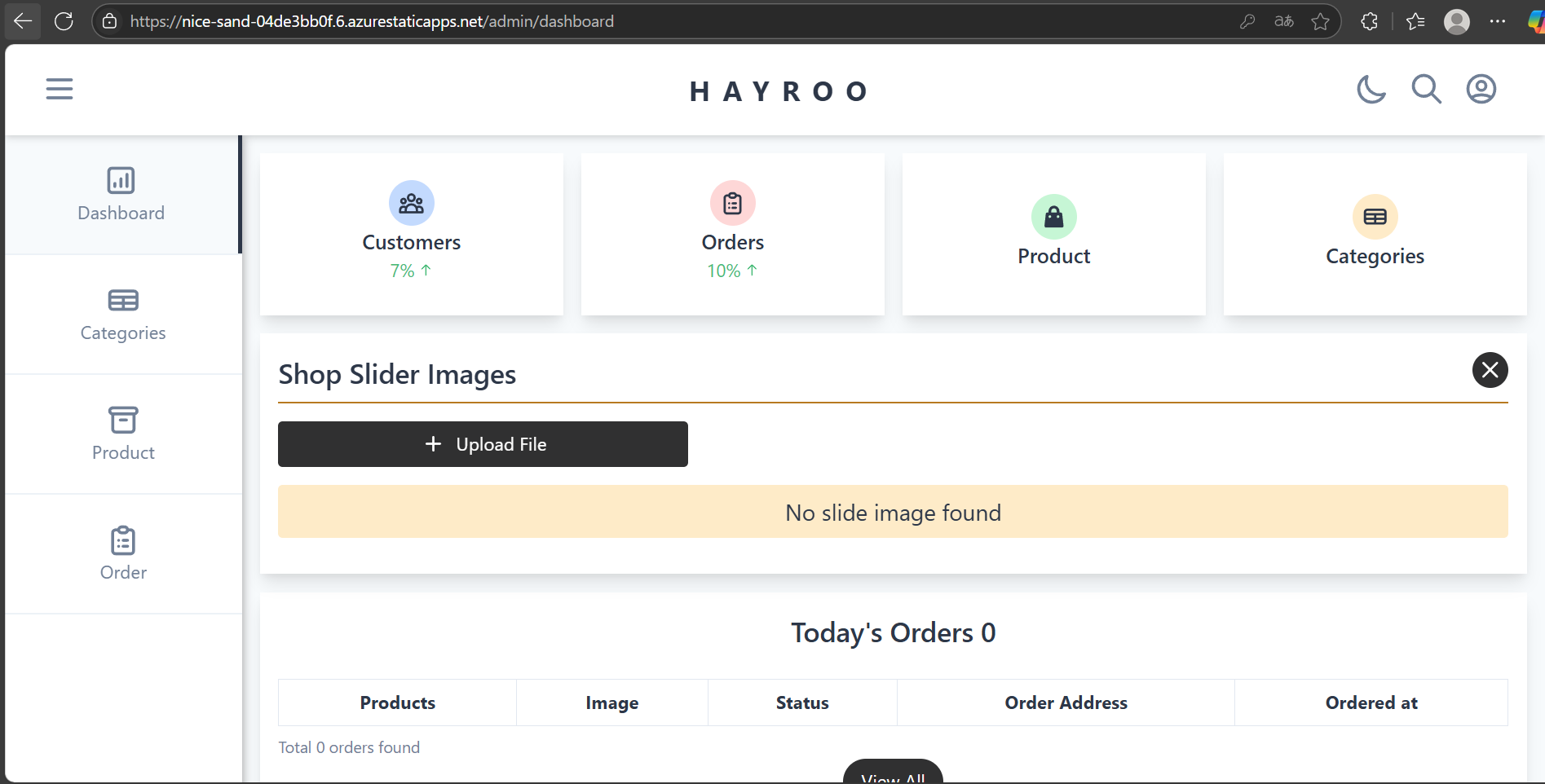
Projenin backend kısmı frontend kısmına Cloud üzerinde bağlanması için localde bağlantı sağlandı.



  
Database olarak kullanılması için MongoDB bağlantısı sağlandı.







Ve bulut üzerinde çalışan e-ticaret sitesi kullanıma açık.

**Sensora - IoT Sensör Verisi Gerçek Zamanlı İsleme Projesi**

**1. PROJE GENEL BAKIS**

***1.1 Proje Amacı***

Bu proje, Google Cloud Platform (GCP) üzerinde gerçek zamanlı IoT sensör verilerinin toplanması, işlenmesi ve saklanması için tam entegre bir sistem geliştirmeyi amaçlamaktadır. Sistem, Cloud Functions, Pub/Sub, Firestore ve BigQuery teknolojilerini kullanarak ölçeklenebilir bir veri işleme pipeline'ı oluşturmaktadır.

***1.2 Seçilen Proje***

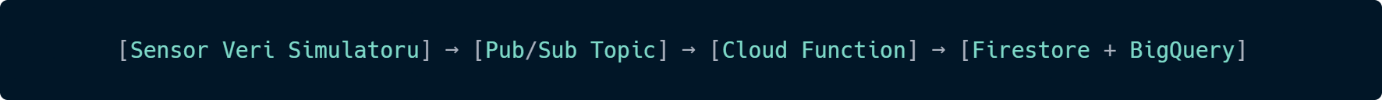
**Proje 2: Gerçek Zamanlı Veri Akışı ve İşleme (IoT veya WebSocket Uygulaması)**

Kullanılan teknolojiler:

* **Backend Dil**: Python
* **Protokol**: Pub/Sub (Google Cloud mesajlaşma servisi)
* **Veritabanları**: Firestore (NoSQL), BigQuery (Veri ambarı)
* **Bulut Platformu**: Google Cloud Platform

**2. SİSTEM MİMARİSİ**

***2.1 Genel Mimari***

****

***2.2 Bileşen Detayları***

**1. Veri Kaynağı (Sensör Simülatörü)**

* Sıcaklık ve nem verilerini simüle eder
* 5 saniye aralıklarla veri üretir
* JSON formatında Pub/Sub'a gönderir

**3. Cloud Function (process\_sensor\_data)**

* Serverless işlem birimi
* Pub/Sub trigger ile otomatik tetikleme
* Python 3.13 runtime

**2. Pub/Sub Topic (iot-sensor-data)**

* Asenkron mesaj kuyruğu
* Yüksek throughput desteği
* Güvenilir mesaj teslimi

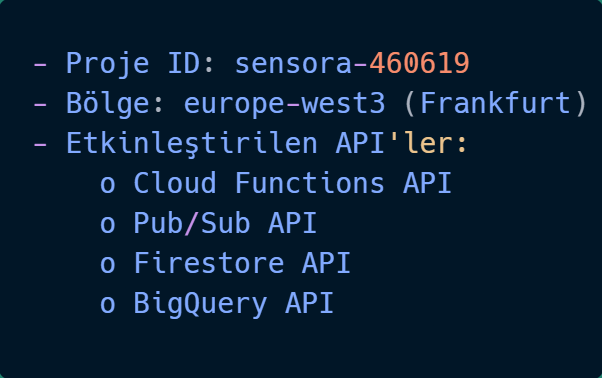
**4. Veri Depolama**

* **Firestore**: Gerçek zamanlı erişim için
* **BigQuery**: Analitik sorgular için

**3. TEKNIK UYGULAMA DETAYLARI**

***3.1 Proje Kurulumu ve Konfigürasyon***

**Google Cloud Proje Ayarları**

****

**Gerekli Servisler ve Kaynaklar**



Google Cloud projesi sensora-460619, Frankfurt bölgesinde (europe-west3) bulunuyor.

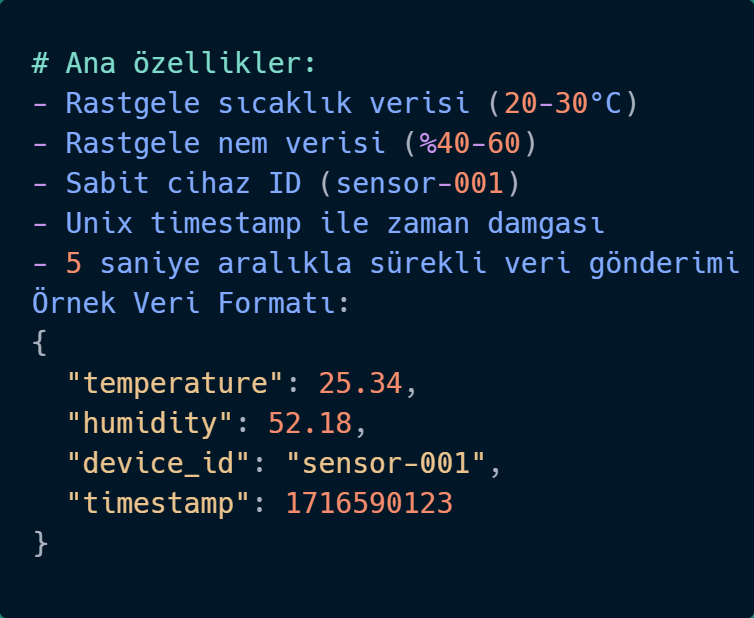
Cloud Functions, Pub/Sub, Firestore ve BigQuery API’leri etkinleştirildi.

Sagdaki görseldeki komutla proje seçilirken, gcloud functions deploy komutuyla Python 3.13 kullanarak process\_sensor\_data fonksiyonu, iot-sensor-data Pub/Sub konusu tetikleyicisiyle ilgili bölgede dağıtılıyor.

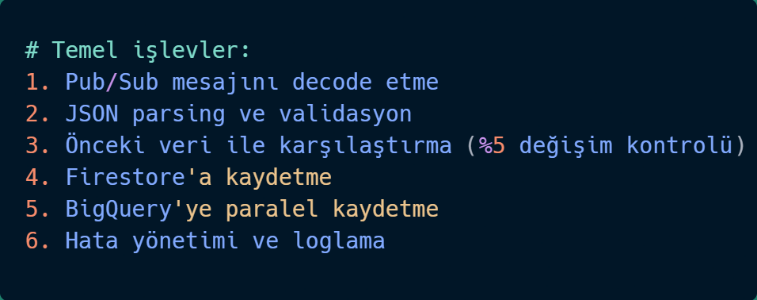
Bu yapılandırma, sensör verilerinin bulut ortamında işlenmesini sağlıyor.

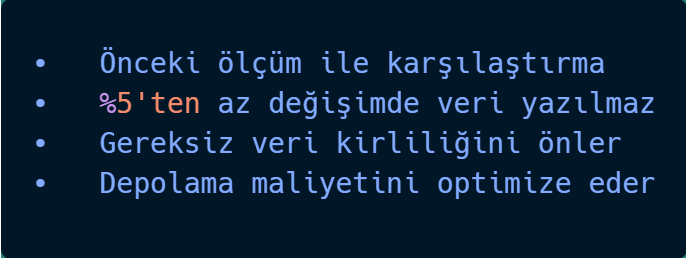
***3.2 Veri Akış Pipeline'ı***

**Sensör Veri Simülatörü (sensor\_data\_publisher.py)**



**Cloud Function İşleme Mantığı (main.py)**



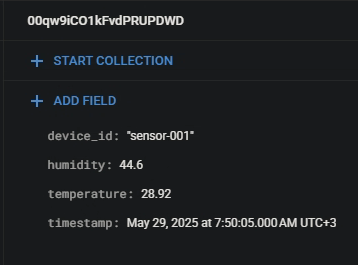
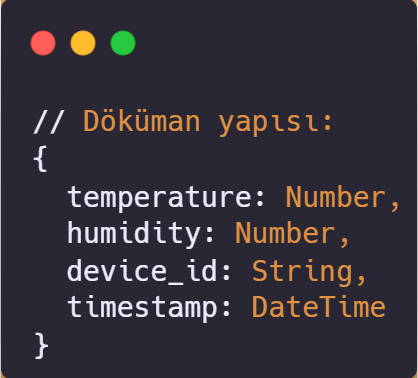
**Akıllı Filtreleme Algoritması:** 

Sensör Veri Simülatörü, cihaz kimliği ve zaman damgasıyla birlikte 5 saniyede bir rastgele sıcaklık ve nem verisi üretip JSON formatında gönderir.

Cloud Function, Pub/Sub’dan gelen veriyi alır, doğrular ve önceki veriye göre %5’ten az değişiklik varsa kaydetmez. Geçerli veriler Firestore ve BigQuery’ye kaydedilir, hatalar ise loglanarak yönetilir. Bu filtreleme yöntemi, gereksiz veri girişini engelleyip depolama maliyetini azaltır.

***3.3 Veri Depolama Stratejisi***

**Firestore Koleksiyonu (sensor\_readings)**



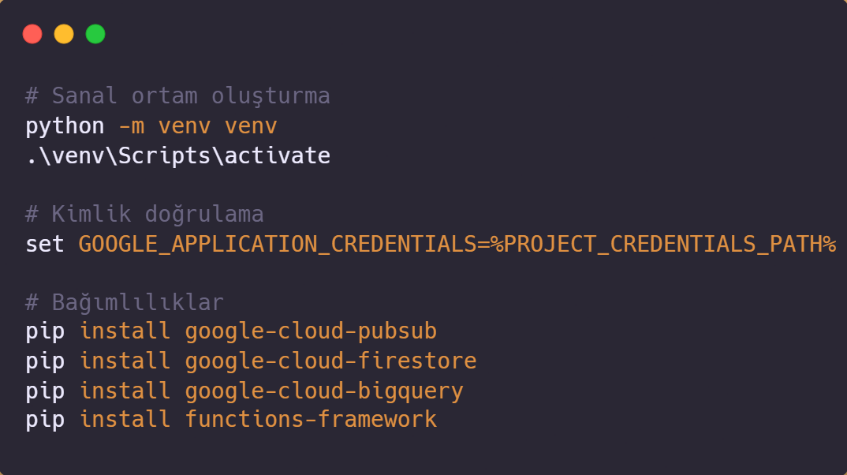
**BigQuery Tablosu (sensora\_dataset.sensor\_readings)**



Sensör verileri, hızlı erişim için Firestore koleksiyonunda saklanır. Aynı veriler, detaylı analiz için BigQuery’de tablo olarak tutulur. Böylece gerçek zamanlı işlem ve uzun dönemli analiz dengelenir.

**4. DEPLOYMENT VE TEST SÜREÇLERİ**

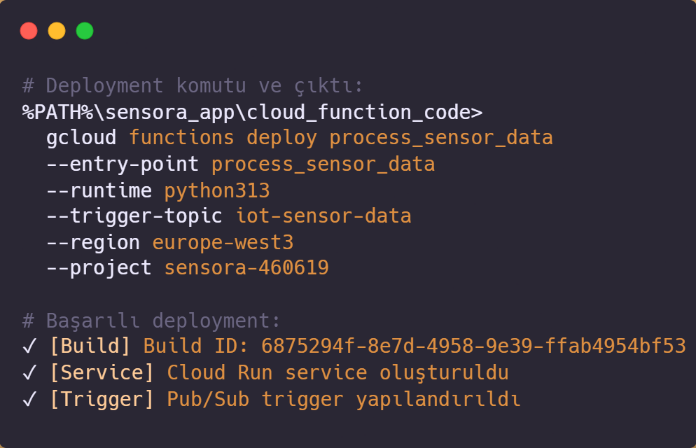
***4.1 Geliştirme Ortamı Kurulumu***



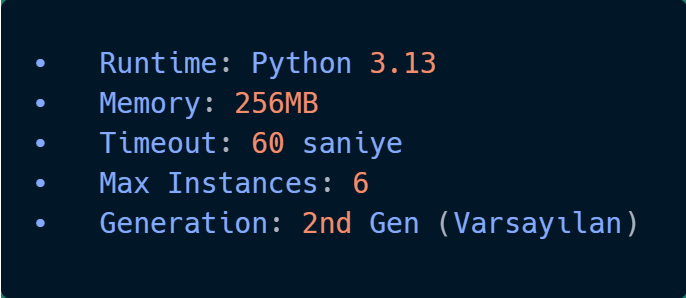
Geliştirme ortamı için Python sanal ortamı oluşturulur ve aktif hale getirilir. Gerekli Google Cloud kütüphaneleri (pubsub, firestore, bigquery) ile functions-framework paketi yüklenir.

Ayrıca, kimlik doğrulama için servis hesabı anahtarı ortam değişkeni olarak ayarlanır. Bu adımlar, uygulamanın bulut servislerine güvenli ve sorunsuz bağlanmasını sağlar.

***4.2 Cloud Function Deployment***



**Deployment Özellikleri:**

****

Cloud Function, Python 3.13 runtime ile belirtilen bölgede ve iot-sensor-data Pub/Sub konusu tetikleyicisiyle dağıtılır.

Başarılı deploy sonrası Cloud Run servisi oluşturulur ve trigger aktif hale getirilir. Fonksiyon, 256MB bellek, 60 saniye zaman aşımı ve maksimum 6 instance ile çalışır.

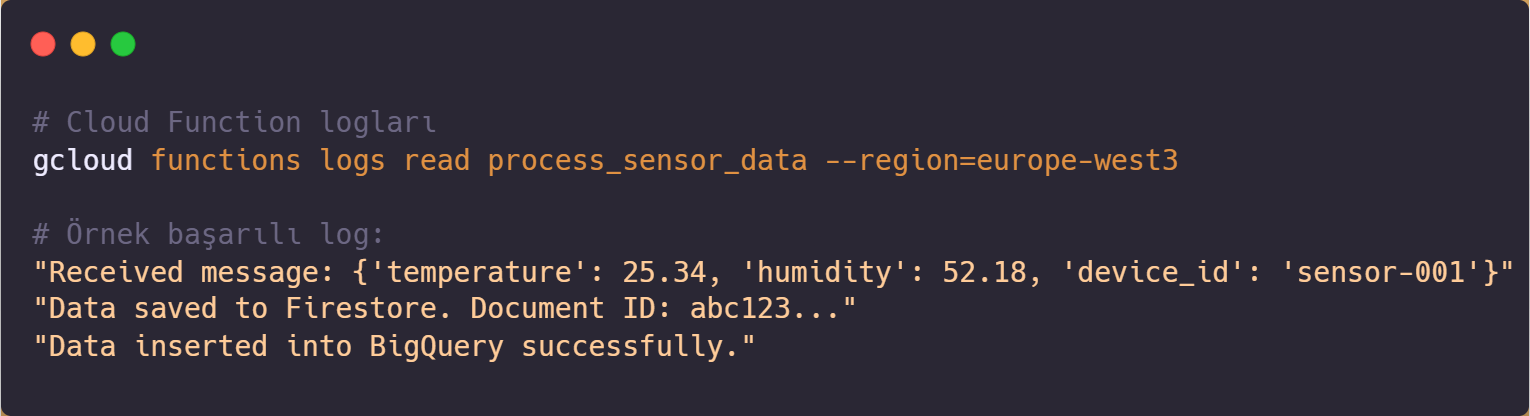
Bu yapılandırma, güvenilir ve ölçeklenebilir veri işleme sağlar.

***4.3 Test ve Doğrulama***

**Fonksiyon Testleri**

1. **Local Test**: Functions Framework ile yerel test
2. **Integration Test**: Pub/Sub mesaj gönderimi
3. **End-to-End Test**: Veri akışının tamamı

**Log Analizi**



Fonksiyonun doğruluğu üç aşamada test edilir. Öncelikle, Functions Framework kullanılarak yerelde çalışması kontrol edilir. Ardından, Pub/Sub’a mesaj gönderilerek entegrasyon testi yapılır.

Son olarak, veri akışının tamamı test edilerek uçtan uca doğrulama sağlanır.

Testler sırasında oluşan loglar, gcloud functions logs read komutuyla incelenir. Başarılı örnek loglar, mesaj alındığını ve verinin Firestore ile BigQuery’ye başarılı şekilde kaydedildiğini gösterir.

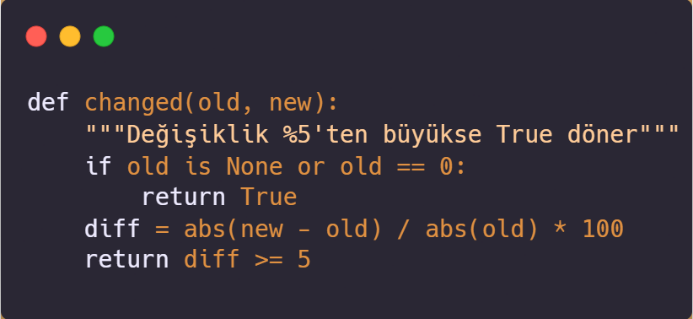
**5. PERFORMANS VE OPTİMİZASYON**

***5.1 Sistem Performansı***

Test ortamında mesaj işleme süresi yaklaşık 200-500 ms arasında değişir ve saniyede 5-10 mesaj işlenebilir. Hata oranı testlerde %1 olarak gözlemlenmiş olup, Cloud Functions SLA kapsamında %99.9 üzeri erişilebilirlik sağlanır.

***5.2 Optimizasyon Stratejileri***

**Akıllı Veri Filtreleme**



**Hata Yönetimi**

* Try-catch blokları ile exception handling
* Detaylı loglama ve monitoring
* Pub/Sub retry mekanizması (RETRY\_POLICY\_DO\_NOT\_RETRY)

Sistemde akıllı veri filtreleme kullanılarak, önceki veriye göre %5’ten az değişiklik gösteren veriler kaydedilmez.

Hata yönetimi için try-catch bloklarıyla istisnalar yakalanır, detaylı loglama yapılır ve Pub/Sub mesajlarının tekrar denemeleri kontrol edilir.

Bu sayede performans artarken hata yönetimi ve izleme güvence altına alınır.

**6. GÜVENLİK VE YETKI YÖNETİMİ**

Proje, minimal yetki prensibine uygun olarak IAM rolleri ve servis hesaplarıyla yönetilir;

Cloud Functions, Pub/Sub, Firestore ve BigQuery erişimleri ayrı ayrı yetkilendirilir ve kimlik doğrulama JSON anahtar dosyasıyla sağlanır.

***6.1 IAM Rolleri***

* **Cloud Functions Invoker**: Pub/Sub trigger için
* **Pub/Sub Publisher**: Veri gönderimi için
* **Firestore User**: Veritabanı erişimi
* **BigQuery Data Editor**: Veri ambarı erişimi

***6.2 Servis Hesabı***

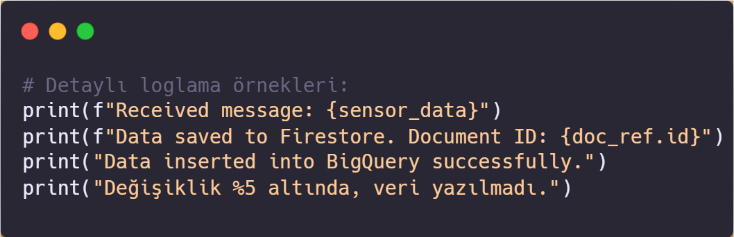
* Compute Engine default service account kullanımı
* Minimal yetki prensibi (principle of least privilege)
* JSON key file ile kimlik doğrulama

**7. MONİTORİNG VE LOGLAMA**

***7.1 Cloud Function Monitoring***

* **Invocation Count**: Fonksiyon çağrı sayısı
* **Execution Time**: Ortalama çalışma süresi
* **Error Rate**: Hata oranı izleme

***7.2 Loglama Stratejisi***



**8. VERİ GÖRSELLEŞTİRME VE DASHBOARD**

***8.1 Looker Studio Entegrasyonu***

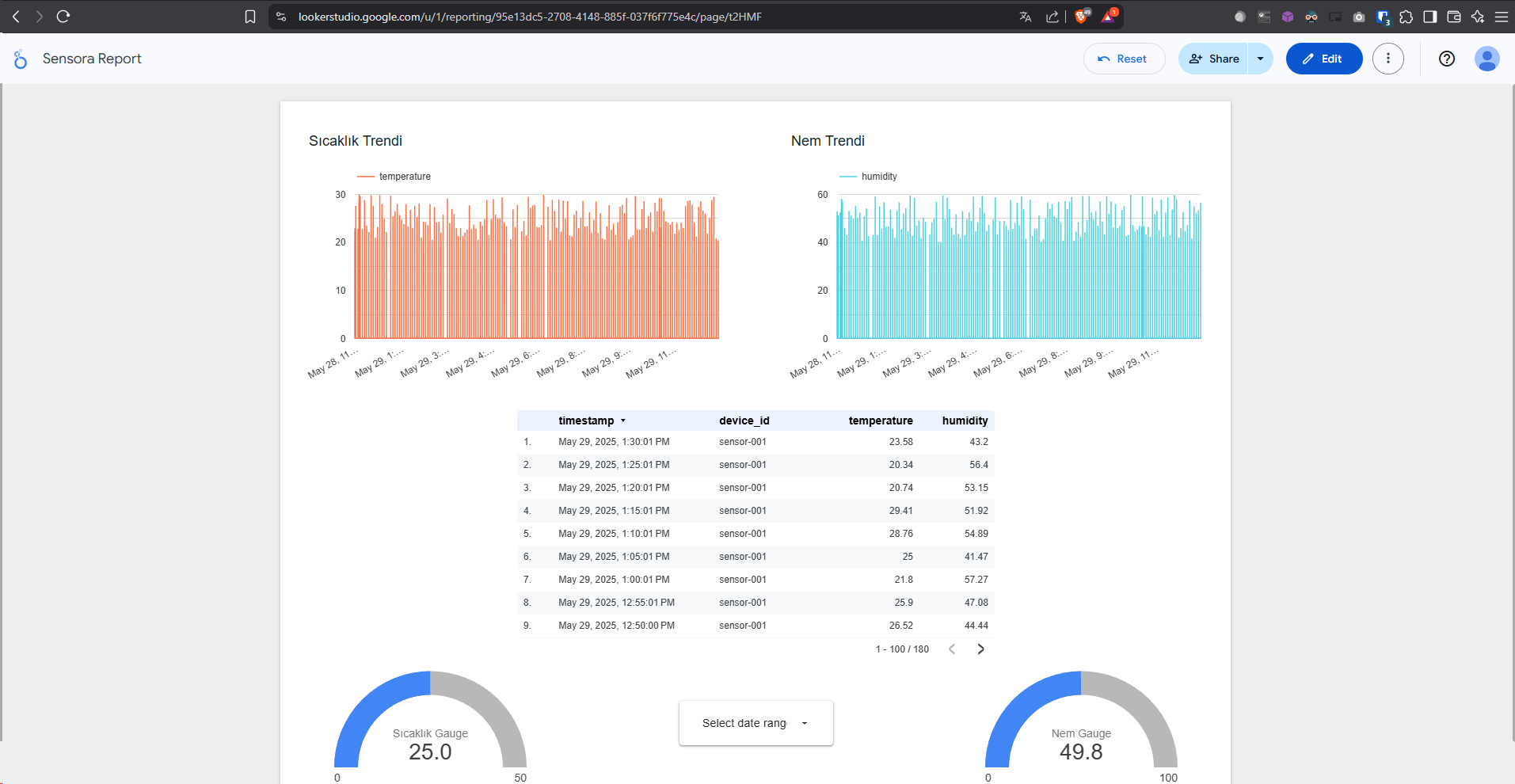
Proje kapsamında toplanan sensör verilerinin gerçek zamanlı izlenmesi ve analiz edilmesi için Google Looker Studio ile entegre bir dashboard oluşturulmuştur.

**Dashboard Özellikleri**

* **Gerçek Zamanlı Veri**: BigQuery veri kaynağından canlı veri akışı
* **Zaman Serisi Grafikleri**: Sıcaklık ve nem değerlerinin zaman içindeki değişimi
* **Cihaz Bazlı Filtreleme**: Device ID'ye göre veri filtreleme imkanı
* **İstatistiksel Özetler**: Ortalama, minimum, maksimum değerler
* **Trend Analizi**: Günlük, saatlik trend görünümleri

**Görselleştirme Faydaları**

* **Operasyonel İzleme**: Sensör durumlarının anlık takibi
* **Trend Analizi**: Uzun dönemli veri eğilimlerinin belirlenmesi
* **Anomali Tespiti**: Normal dışı değerlerin görsel tespiti
* **Raporlama**: Periyodik raporların otomatik oluşturulması



**9. PROJE DOSYA YAPISI**

****

**10. ÖGRENILEN DERSLER VE ÇÖZÜLEN SORUNLAR**

***10.1 Teknik Zorluklar***

1. **Cloud Functions Gen2 Geçişi**: Otomatik olarak 2nd gen'e geçiş
2. **IAM Yetkilendirme**: Servis hesabı rollerinin doğru yapılandırılması
3. **Timestamp Conversion**: Unix timestamp → DateTime dönüşümü
4. **BigQuery Schema**: Tablo şemasının doğru tanımlanması

***10.2 Çözüm Yaklaşımları***

1. **Dokümantasyon Takibi**: GCP dökümanlarından aktif yararlanma
2. **Yapay Zeka Destekli Yardım:** Dokümantasyon yorumu, hata çözümü ve yapılandırma
3. **Log-Driven Development**: Detaylı loglama ile hata ayıklama
4. **Incremental Testing**: Adım adım test ve doğrulama
5. **Error Handling**: Kapsamlı exception yönetimi

**11. SONUÇ VE GELECEK PLANLAR**

***11.1 Başarılan Hedefler***

* Google Cloud üzerinde tam fonksiyonel IoT pipeline
* Gerçek zamanlı veri işleme ve depolama
* Kapsamlı hata yönetimi ve loglama
* Ölçeklenebilir serverless mimari
* Çoklu veri deposu entegrasyonu (Firestore + BigQuery)

***11.2 Sistem Avantajları***

* **Serverless**: Otomatik ölçekleme, yönetim gerekmez
* **Cost-Effective**: Kullanım bazlı ücretlendirme
* **High Availability**: Cloud provider SLA garantileri
* **Real-time**: Düşük latency ile veri işleme
* **Scalable**: Yüksek throughput desteği

***11.3 Gelecek Geliştirmeler (Roadmap)***

* **Cloud Scheduler**: Otomatik veri gönderimi
* **Anomaly Detection**: ML tabanlı anormallik tespiti
* **Multi-Device Support**: Çoklu sensör desteği
* **Real-time Alerts**: Threshold bazlı uyarı sistemi

**12. REFERANSLAR VE KAYNAKLAR**

***12.1 Kullanılan Teknolojiler***

* **Google Cloud Functions:** Sunucusuz (serverless) hesaplama hizmeti
* **Google Cloud Pub/Sub:** Mesajlaşma servisi
* **Google Cloud Firestore:** NoSQL veritabanı
* **Google Cloud BigQuery:** Veri ambarı (data warehouse)
* **Google Cloud IAM:** Yetkilendirme ve erişim kontrolü
* **Cloud Logging (Stackdriver):** Loglama ve izleme hizmeti
* **Docker:** Uygulamaların konteyner içinde taşınabilir şekilde çalışması için
* **Python 3.13:** Backend geliştirme dili

***12.2 Önemli Linkler***

* [Cloud Run Functions - Resmi Dökümantasyon](https://cloud.google.com/functions/docs)
* [Pub/Sub - Resmi Dökümantasyon](https://cloud.google.com/pubsub/docs)
* [Firestore - Resmi Dökümantasyon](https://cloud.google.com/firestore/native/docs)
* [BigQuery - Resmi Dökümantasyon](https://cloud.google.com/bigquery/docs)
* [IoT ve Pub/Sub Entegrasyonu - ChirpStack](https://www.chirpstack.io/docs/chirpstack/integrations/gcp-pub-sub.html)
* [Firestore ve BigQuery Entegrasyonu - GCP Rehberi](https://gcloud.readthedocs.io/en/latest/pubsub-client.html)