ÇUKUROVA ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ (İNG. PR.)

ETERNA TEKNOLOJİ

AD SOYAD: ÖMER AKDOĞAN

KONU: Cihazdan gelen koordinat verilerini Google Maps üzerinde gösterme

PROJE TESLİM TARİHİ: 15.08.2025

**1. Giriş**

**1.1 Projenin Amacı ……………………………………………………………**

**1.2 Genel Sistem Mimarisi …………………………………………………..**

**1.3 Kullanılan Teknolojiler …………………………………………………..**

**2. Veritabanı Yapısı ve MySQL Bağlantısı**

**2.1 MySQL Bağlantı Havuzu (Connection Pool) ………………………...**

**2.2 Veritabanı Tabloları 2.3 Bağlantı Testi ………………………………..**

**3. Telemetry Endpoint ve Veri Kaydı Süreci**

**3.1 Telemetry Endpoint Tanımı …………………………………………….**

**3.2 API Key Doğrulama ………………………………………………………**

**3.3 Veri Doğrulama …………………………………………………………...**

**3.4 Veritabanına Kaydetme ………………………………………………….**

**3.5 WebSocket Üzerinden Yayın …………………………………………...**

**4. WebSocket Sunucusu ve İstemci İletişimi**

**4.1 WebSocket Sunucusunun Kurulumu …………………………………**

**4.2 Bağlantı Olayları ………………………………………………………….**

**4.3 Veri Yayını …………………………………………………………………**

**4.4 WebSocket Test İstemcisi (Node.js) ………………………………….**

**5. Simülasyon Sistemi (sim.js) ile Test Verisi Gönderimi**

**5.1 Amacı ……………………………………………………………………….**

**5.2 Çalışma Mantığı …………………………………………………………..**

**6. .env Dosyasının Rolü ve Konfigürasyon Yönetimi**

**6.1 Amacı 6.2 İçerik …………………………………………………………..**

**6.3 Önemli Parametreler ……………………………………………………..**

**6.4 Kodlarda Kullanımı ………………………………………………………**

**6.5 Avantajlar ………………………………………………………………….**

**7. Sonuç ve Gelecek Geliştirmeler**

**7.1 Projenin Başarı Durumu (Özet) ………………………………………..**

**7.2 İyileştirme Fikirleri (Yol Haritası) ………………………………………**

**A) Güvenlik & Dayanıklılık ……………………………………………**

**B) Veritabanı & Performans ………………………………………….**

**C) API ve Özellik Geliştirmeleri ……………………………………...**

**D) WebSocket & Yayın Mimarisini Güçlendirme …………………**

**E) Gözlemlenebilirlik & Operasyon …………………………………**

**F) Dağıtım & Altyapı ……………………………………………………**

**G) İstemci Entegrasyonu (Flutter/Web) ……………………………**

**8. PROJE TASK ‘I**

**1. Giriş**

**1.1 Projenin Amacı**

Bu projenin temel amacı, belirli cihazlardan (örneğin araç takip cihazı veya mobil uygulama) alınan konum verilerinin gerçek zamanlı olarak izlenebilmesini sağlamaktır.  
Sistem, cihazlardan gelen konum bilgilerini hem veritabanına kaydeder hem de WebSocket (Socket.IO) üzerinden anlık olarak bağlı tüm istemcilere iletir.  
Bu sayede kullanıcılar, herhangi bir web veya mobil istemciden cihazların güncel konumlarını harita üzerinde canlı olarak takip edebilir.

**1.2 Genel Sistem Mimarisi**

Sistem **3 ana bileşenden** oluşur:

1. **Backend Sunucusu (Node.js + Express + Socket.IO)**
   * HTTP API’ler aracılığıyla konum verilerini alır (/telemetry endpoint’i).
   * Verileri MySQL veritabanına kaydeder.
   * WebSocket üzerinden gerçek zamanlı yayın yapar.
2. **Simülasyon Scripti (Node.js)**
   * Cihaz gibi davranarak belirli aralıklarla rastgele konum verileri üretir.
   * Bu verileri API üzerinden backend’e gönderir.
3. **WebSocket İstemcileri (Test Uygulamaları / Flutter Uygulaması)**
   * Backend WebSocket kanalına bağlanarak anlık konum verilerini alır.
   * Harita veya listeler üzerinde canlı konum takibi yapılabilir.

Bu yapı, **gerçek zamanlı veri iletimi** ve **tarihsel veri kayıt** özelliklerini aynı anda sunar.

**1.3 Kullanılan Teknolojiler**

Proje, modern web ve backend teknolojilerinin bir kombinasyonunu kullanır:

* **Backend Framework**: Express.js
* **Gerçek Zamanlı İletişim**: [Socket.IO](https://socket.io/)
* **Veritabanı**: [MySQL](https://www.mysql.com/)
* **Veri Doğrulama**: Joi
* **HTTP İstemci (Simülasyon)**: Node.js fetch API
* **Çevre Değişkenleri Yönetimi**: [dotenv](https://www.npmjs.com/package/dotenv)
* **Test İstemcisi**: socket.io-client ve wscat CLI aracı
* **Ngrok**: Lokal backend’i internete açmak için tünelleme aracı

**2. Veritabanı Yapısı ve MySQL Bağlantısı**

Bu projede konum verilerini kalıcı olarak saklamak için **MySQL** veritabanı kullanıldı. Sunucu tarafında veri tabanına bağlanmak ve sorgu çalıştırmak için **mysql2/promise** modülü tercih edildi. Promise tabanlı kullanım, asenkron/await yapısı ile uyumlu çalışarak kod okunabilirliğini ve hata yönetimini kolaylaştırır.

**2.1 MySQL Bağlantı Havuzu (Connection Pool)**

Kod içerisinde veritabanı bağlantısı için **connection pool** oluşturuldu. Bu sayede her sorgu için yeni bir bağlantı açmak yerine, önceden oluşturulmuş bağlantılar tekrar kullanılabiliyor.  
Bağlantı bilgileri .env dosyasından okunuyor, böylece kodda herhangi bir hassas bilgi (kullanıcı adı, parola) tutulmuyor.

**Kod:**

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

🔹 **Önemli Satırlar:**

* process.env.DB\_HOST vb. değerler .env dosyasından geliyor.
* connectionLimit: 5 → Aynı anda maksimum 5 aktif bağlantı kullanılır.
* mysql.createPool → Bağlantı havuzu oluşturur, tek seferlik createConnection yerine sürekli kullanımda daha verimlidir.

**2.2 Veritabanı Tabloları**

Sistemde iki ana tablo vardı, ancak **searches** tablosu kullanılmadığı için tamamen kaldırıldı. Şu anda yalnızca **locations** tablosu aktif olarak kullanılıyor.

**locations tablosu yapısı:**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

🔹 **Açıklamalar:**

* device\_id: Konumu gönderen cihazın benzersiz ID’si.
* lat / lng: Enlem ve boylam değerleri.
* speed: Cihazın hız bilgisi (opsiyonel).
* heading: Cihazın yön bilgisi (opsiyonel).
* created\_at: Verinin kaydedildiği zaman.

**2.3 Bağlantı Testi**

Sunucu başlatılmadan önce MySQL bağlantısının çalıştığından emin olmak için bir test yapılıyor.  
Eğer bağlantı başarısız olursa sistem kapanıyor.

**Kod:**metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

🔹 **Önemli Satırlar:**

* pool.getConnection() → Bir bağlantı açar ve test eder.
* conn.release() → Bağlantıyı havuza geri bırakır.
* process.exit(1) → Bağlantı başarısızsa sunucu başlatılmaz.

**3. Telemetry Endpoint ve Veri Kaydı Süreci**

Bu başlık, cihazlardan gelen konum verilerinin **HTTP POST** isteği ile alınıp **veritabanına kaydedilmesi** ve **WebSocket üzerinden tüm istemcilere yayınlanması** sürecini kapsar.

**3.1 Telemetry Endpoint Tanımı**

Sunucu tarafında /telemetry endpoint’i oluşturuldu.  
Bu endpoint’e cihazlar konum verilerini **JSON** formatında gönderiyor.  
İstek, yalnızca doğru **API anahtarına (x-api-key)** sahip cihazlar tarafından kabul ediliyor.

**Kod:**metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**3.2 API Key Doğrulama**

Her cihaz, x-api-key başlığında .env dosyasında tanımlı olan **DEVICE\_API\_KEY** değerini göndermelidir.  
Bu kontrol authDevice middleware’i ile yapılır.

**Kod:**metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

🔹 **Önemli Noktalar:**

* API anahtarı yanlışsa veya eksikse **401 Unauthorized** hatası döner.
* Güvenlik için bu değer **.env** içinde saklanır.

**3.3 Veri Doğrulama**

Gönderilen verilerin formatı **Joi** kütüphanesi ile kontrol edilir. Eksik veya hatalı veri geldiğinde işlem yapılmaz.

**Kod:**metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

🔹 **Önemli Noktalar:**

* lat ve lng değerleri coğrafi sınırlarla kısıtlanır.
* ts değeri ISO formatında bir tarih olmalıdır, yoksa sunucu otomatik olarak **şu anki zamanı** kullanır.

**3.4 Veritabanına Kaydetme**

Veriler locations tablosuna eklenir. created\_at alanına **gelen zaman** veya **şu anki zaman** eklenir.

**Örnek SQL Sorgusu:**metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**3.5 WebSocket Üzerinden Yayın**

Veritabanına ekleme tamamlandıktan sonra, aynı veri tüm bağlı istemcilere **location** adlı WebSocket olayı ile gönderilir.

**Örnek Yayın Mesajı:**

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

🔹 **Önemli Noktalar:**

* Bu sayede harita uygulamaları, mobil uygulamalar veya başka sistemler **anlık konum** bilgilerini alabilir.
* Flutter, React, Node.js gibi farklı istemciler bu yayını dinleyebilir.

**4. WebSocket Sunucusu ve İstemci İletişimi**

Bu başlıkta, sunucunun WebSocket altyapısı ile nasıl çalıştığını ve istemcilerin bu sisteme nasıl bağlandığını detaylı olarak anlatacağım.

**4.1 WebSocket Sunucusunun Kurulumu**

Sunucu tarafında **Socket.IO** kütüphanesi kullanıldı. Socket.IO, hem **WebSocket** hem de **HTTP fallback** desteği sunar, fakat bu projede **yalnızca WebSocket** kullanılması için ayar yapıldı.

**Kod:**metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

🔹 **Önemli Noktalar:**

* path değeri **/socket.io** olarak sabitlendi, istemciler bağlanırken bunu kullanmalı.
* transports: ['websocket'] ile **polling** kapatıldı, sadece gerçek WebSocket bağlantısı kabul ediliyor.
* CORS ayarı .env dosyasındaki CORS\_ORIGIN ile kontrol ediliyor.

**4.2 Bağlantı Olayları**

Sunucuya bir istemci bağlandığında veya bağlantısı koptuğunda loglama yapılıyor.

**Kod:**metin, ekran görüntüsü, yazı tipi içeren bir resim

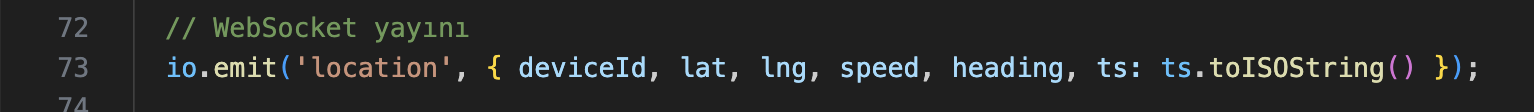
Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

🔹 **Önemli Noktalar:**

* socket.id her bağlantı için benzersizdir.
* Bağlantı koparsa tekrar bağlanmak için istemci tarafında yeniden bağlantı mekanizması kurulabilir.

**4.3 Veri Yayını**

/telemetry endpoint’i üzerinden gelen her yeni konum verisi, **io.emit()** kullanılarak tüm bağlı istemcilere iletilir.

**Kod:**

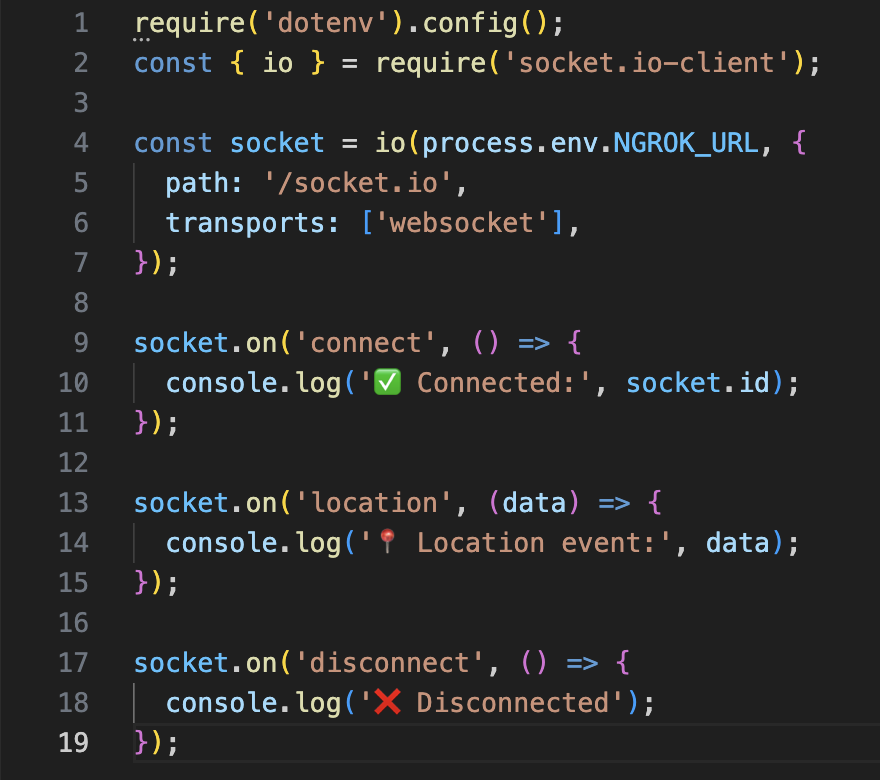
🔹 **Önemli Noktalar:**

* location olayı, istemcilerin dinlemesi gereken ana kanaldır.
* Bu sistem **broadcast** çalışır, yani veriyi gönderen cihaz dahil tüm istemcilere gönderilir.

**4.4 WebSocket Test İstemcisi (Node.js)**

Sunucunun yayını doğru şekilde yapıp yapmadığını test etmek için test-ws.js adında basit bir istemci hazırlandı.

**Kod:**



**5. Simülasyon Sistemi (sim.js) ile Test Verisi Gönderimi**

Bu başlıkta, gerçek cihaz olmadan **lokasyon verisini otomatik olarak sunucuya göndermek** için yazdığımız simülasyon kodunu detaylı anlatacağım.

**5.1 Amacı**

* Gerçek cihaz verisi gelmeden önce sistemin uçtan uca test edilebilmesini sağlamak.
* WebSocket yayınını ve veritabanı kayıt sürecini kontrol etmek.
* Konum verilerinin değişimini test etmek.

**5.2 Çalışma Mantığı**

1. .env dosyasındaki **NGROK\_URL**, **API\_KEY**, ve **DEVICE\_ID** bilgilerini okur.
2. Örnek olarak Diyarbakır koordinatlarından başlar (lat, lng).
3. Her 2.0 saniyede bir koordinatları **jitter** fonksiyonu ile biraz değiştirir.
4. /telemetry endpoint’ine POST isteği gönderir.
5. Sunucu bu veriyi **veritabanına kaydeder** ve **WebSocket üzerinden yayınlar**.

**Kod : metin, ekran görüntüsü, yazılım içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.**

**Önemli Kod Satırları**

* let lat = 37.9144; let lng = 40.2306; → Başlangıç konumu Diyarbakır.
* jitter(value, range = 0.2) → Konumu ±0.2 derece rastgele değiştirir.
* setInterval(..., 1500) → 1.5 saniyede bir veri gönderilir.
* headers: { 'x-api-key': apiKey } → Güvenlik kontrolü için API key eklenir.

**Örnek Çıktı**

**metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.**

**6. .env Dosyasının Rolü ve Konfigürasyon Yönetimi**

**6.1 Amacı**

* Tüm kritik ayarları ve hassas bilgileri kodun içine gömmek yerine **.env dosyasında saklamak**.
* NGROK bağlantı adresi, veritabanı bilgileri, API anahtarı gibi bilgilerin kolayca değiştirilmesini sağlamak.
* Gerektiğinde farklı ortamlar (development, test, production) için **farklı .env dosyaları** kullanabilmek.

**6.2 İçerik**

env dosyamız şu şekilde yapılandırılmıştır:

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**6.3 Önemli Parametreler**

1. **PORT** → Express.js sunucusunun çalışacağı port numarası.
2. **CORS\_ORIGIN** → WebSocket ve HTTP isteklerine hangi origin’lerden izin verileceği.
3. **DB\_HOST / DB\_USER / DB\_PASS / DB\_NAME** → MySQL bağlantı bilgileri.
4. **DEVICE\_API\_KEY** → /telemetry ve diğer güvenli endpoint’lere erişim için API anahtarı.
5. **NGROK\_URL** → Dış dünyadan erişilebilecek sunucu adresi.
6. **DEVICE\_ID** → Simülasyon yapan cihazın kimliği.
7. **API\_KEY** → Simülasyonun sunucuya veri göndermesi için gerekli anahtar.

**6.4 Kodlarda Kullanımı**

* require('dotenv').config() satırı ile .env dosyası yüklenir.
* Her yerde process.env.DEĞİŞKEN\_ADI ile erişim sağlanır.

**Örnek Kullanım (sim.js):**metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**6.5 Avantajlar**

* **Güvenlik:** API anahtarları ve DB şifreleri kodda tutulmaz.
* **Esneklik:** Ngrok URL’si değiştiğinde sadece .env güncellenir, kod değişmez.
* **Taşınabilirlik:** Kod farklı bir sunucuya taşındığında sadece .env düzenlenerek çalışır.

**7. Sonuç ve Gelecek Geliştirmeler**

**7.1 Projenin Başarı Durumu (Özet)**

Bu çalışma ile:

* Cihazlardan (veya simülasyondan) gelen **konum verileri** güvenli şekilde **/telemetry** ile alınıp **MySQL**’e yazılmış,
* Aynı veriler **Socket.IO** üzerinden **anlık** olarak tüm istemcilere **location** event’iyle yayınlanmış,
* **/latest** ile her cihazın en güncel kaydı API’dan servis edilmiş,
* **Ngrok** sayesinde yerel geliştirme ortamına dış dünyadan erişim sağlanmış,
* **test-ws.js** ile yayın zinciri uçtan uca doğrulanmıştır.

Sonuç: Mimarinin temel hedefleri (güvenli veri girişi, kalıcı kayıt, gerçek zamanlı yayın, test edilebilirlik) başarıyla gerçekleştirilmiştir.

**7.2 İyileştirme Fikirleri (Yol Haritası)**

Aşağıdaki maddeler, üretim ortamına geçişte kalite, güvenlik, ölçeklenebilirlik ve gözlemlenebilirliği arttırmak için önerilen geliştirmelerdir.

**A) Güvenlik & Dayanıklılık**

1. **API Key rotasyonu ve çoklu cihaz anahtarı**
   * Tek bir DEVICE\_API\_KEY yerine cihaz-bazlı anahtar tablosu: device(id, api\_key, status).
   * Middleware: Gelen deviceId ile api\_key eşleşmesi kontrolü.
2. **Oransal hız limitleme (Rate limiting)**
   * IP veya deviceId başına istek/saniye limiti (örn. 10 req/s).
   * Express-rate-limit ya da Nginx rate limit.
3. **İmzalı payload / Zaman damgalı nonce**
   * İsteğe HMAC imzası eklenerek içerik manipülasyonuna karşı koruma.
   * ts + nonce kontrolü ile tekrar oynatma (replay) önlenir.
4. **CORS kısıtlama**
   * CORS\_ORIGIN=\* yerine üretimde **spesifik domain listesi**.
5. **Girdi doğrulama seviyesini genişletme**
   * Joi mevcut—hatalı ts geleceği veya sıra dışı (ör. lat/lng outlier) durumları için ekstra mantık (ör. bounding box/ülke filtresi).

**B) Veritabanı & Performans**

1. **İndeksler**
   * locations(device\_id, created\_at) bileşik indeks (en çok yapılan sorgu: son kayıt).
   * Büyük hacimlerde ciddi hız kazandırır.
2. **Arşivleme & Yaşlandırma (Retention)**
   * Sıcak veri: Son 30–90 gün, soğuk veri: arşiv tablosu veya dış depolama (S3/Cold storage).
   * Bakım script’i: günlük/haftalık purge veya partitioning.
3. **Partitioning**
   * Zaman bazlı bölümlendirme (örn. aylık) ile sorgular hızlanır, temizlik kolaylaşır.
4. **Toplu yazma (batching) / Kuyruk**
   * Yük altındayken /telemetry → mesaj kuyruğu (Kafka/RabbitMQ) → DB worker.
   * Geri basınç (backpressure) kontrolü ve esneklik.

**C) API ve Özellik Geliştirmeleri**

1. **Zaman aralığına göre sorgu**
   * Yeni endpoint: /locations?deviceId=...&from=...&to=...&limit=...
   * Harita oynatma (trail) ve raporlama için gerekli.
2. **Cihaz listesi & durum uçları**
   * /devices, /devices/:id/status (son ping, batarya, versiyon vs. ileride eklenebilir).
3. **/latest için pagination/filtre**
   * Çok cihaz olduğunda (örn. 10k+) sayfalanmış veya deviceId ile filtrelenmiş sonuç.

**D) WebSocket & Yayın Mimarisini Güçlendirme**

1. **Oda (room) bazlı yayın**
   * io.to(deviceId) / io.to(companyId) gibi gruplama; istemci **sadece ilgilendiği** cihazı dinler → trafik azalır.
2. **Event şeması versiyonlama**
   * type, version, payload alanları; istemci/sunucu uyumluluğunu kolaylaştırır.
3. **Backpressure stratejisi**
   * Aşırı yayın yükünde kuyruk boyu/sıkıştırma veya örnekleme (throttle) mekanizması.

**E) Gözlemlenebilirlik & Operasyon**

1. **Yapılandırılmış loglama**
   * JSON log, request-id, deviceId, latency metrikleri.
   * Log rotasyonu ve merkezi log (ELK, Loki).
2. **Metri̇kler & Alarm**
   * Prometheus/Grafana: RPS, hata oranı, DB latency, ws bağlantı sayısı.
   * Alarm: 5xx artışı, DB’e yazma hatası, ws disconnect spike.
3. **Health & Readiness uçları**
   * /health var — ek olarak /ready (DB ve WS bağımlılık kontrolü) CI/CD’de kullanılabilir.

**F) Dağıtım & Altyapı**

1. **Dockerize & Compose**
   * Dockerfile + docker-compose.yml (API + MySQL).
   * Yerel/CI-CD aynı ortam.
2. **Reverse proxy**
   * Nginx ile SSL terminasyonu, WebSocket upgrade ayarları, gzip/brotli.
   * Üretim domain’i ve cert yönetimi (Let’s Encrypt).
3. **Kapsayıcı ölçekleme**
   * Yatay ölçek (Node çoklu replika) → **Socket.IO adapter (Redis)** ile sticky session/room paylaşımı.

**G) İstemci Entegrasyonu (Flutter/Web)**

1. **Reconnect ayarları**
   * socket.io-client’te exponential backoff + timeout + offline queue.
2. **Harita katmanları ve marker optimizasyonu**
   * Çok cihazda cluster; trail çizgisi için polyline çizimi.
3. **Kullanıcı yetkileri**
   * Hangi kullanıcı hangi device’ı görebilir → backend’de authz katmanı.

**8. Task 1 – Cihazdan gelen koordinat verilerini Google Maps üzerinde gösterme**

* **Amaç:** Gerçek zamanlı olarak cihazdan gelen enlem-boylam verilerini alıp Google Maps üzerinde işaretleme.
* **Kullanılacak Teknolojiler:**
  + **Backend:** Node.js ile API oluşturma
  + **Veri Alımı:** WebSocket (IoT cihazlardan veri almak için)
  + **Veritabanı:** MySQL (koordinatları kaydetmek için opsiyonel)
* **Nasıl Yapılır:**
  + Cihaz koordinatları belirli aralıklarla HTTP POST üzerinden sunucuya gönderir.
  + Backend bu verileri alır, gerekirse veritabanına yazar.
  + Google Maps JavaScript API ile harita sayfası oluşturulur.
  + Sunucudan gelen en son koordinat marker olarak harita üzerinde gösterilir.
  + WebSocket ile haritadaki konum anlık güncellenir.