🚕 TaxiHub Mikroservis Backend Projesi

Teknik Tasarım ve Eğitim Dokümantasyonu

Proje Adı: Bitaksi Hackathon / TaxiHub Backend **Kısa Açıklama:** Sürücü yönetimi ve konum tabanlı en yakın taksi bulma işlemlerini gerçekleştiren, ölçeklenebilir mikroservis mimarisi. **Yazar:** [Adınız Soyadınız] **Tarih:** 7 Aralık 2025 **Statü:** Tamamlandı (v1.0)

---

2. İÇİNDEKİLER

---

3. GENEL BAKIŞ

Projenin Amacı

TaxiHub, taksi sürücülerinin sisteme kaydolmasını, bilgilerini güncellemesini ve kullanıcıların (müşterilerin) belirli bir konuma en yakın taksileri bulmasını sağlayan bir backend (arka uç) çözümüdür. Temel problem, coğrafi veriyi hızlı işleyerek kullanıcıya en uygun sonucu anlık olarak sunmaktır.

Bağlam

Proje, tek bir büyük yapı yerine **Mikroservis Mimarisi** ile tasarlanmıştır:

- **API Gateway:** Sistemin giriş kapısıdır, güvenlikten sorumludur.
- **Driver-Service:** İş mantığını yürüten ana servistir.

Hedef Kitle

Bu doküman; projeyi devralacak teknik ekip, Go diline yeni başlayan geliştiriciler ve kodu inceleyecek teknik liderler (Code Reviewers) için hazırlanmıştır.

Kapsam

- **Dahil Olanlar:** Sürücü CRUD işlemleri, Geolocation (konum) hesaplamaları, JWT kimlik doğrulama, Docker altyapısı.
- **Dahil Olmayanlar:** Mobil uygulama (Frontend), Ödeme sistemleri, Sürüş takibi (Ride tracking).

---

4. TEMEL KAVRAMLARA GİRİŞ

Bu proje modern backend teknolojilerini kullanır. Eğer bu kavramlara yabancıysanız, aşağıdaki özetler sizin için hazırlandı.

Go (Golang) Nedir?

Google tarafından geliştirilen, açık kaynaklı bir programlama dilidir.

- **Özelliği:** C dili kadar hızlıdır ama Python kadar kolay okunur. Özellikle sunucu (server) tarafında çok popülerdir.
- **Neden Tercih Ettik?:** Yüksek performanslıdır ve eş zamanlı işlemleri (concurrency) çok az kaynak harcayarak yapar.
- **Örnek:** Ekrana yazı yazan basit bir Go kodu:

```go
package main
import "fmt"
func main() {
    fmt.Println("Merhaba, TaxiHub!")
}
```

Backend Nedir?

Bir yazılımın "mutfağıdır". Kullanıcının ekranda (Frontend) görmediği veri kaydetme, hesaplama ve güvenlik işlemleri burada yapılır.

- **İstemci (Client):** Siparişi veren müşteri (Mobil Uygulama).
- **Sunucu (Server):** Yemeği hazırlayan mutfak (Backend).
- **API:** Garson. Siparişi alır, mutfağa iletir, yemeği getirir.

HTTP ve REST Nedir?

- **HTTP:** İnternetin iletişim dilidir. İstemci bir istek (**Request**) atar, sunucu bir cevap (**Response**) döner.
- **Methodlar:**
  - `GET`: Veri getir (Menüyü getir).
  - `POST`: Yeni veri oluştur (Sipariş ver).
  - `PUT`: Veriyi güncelle (Siparişi değiştir).
- **REST:** API'lerin düzenli ve standart bir yapıda olmasını sağlayan kurallar bütünüdür.

JSON Nedir?

Verilerin taşınma formatıdır. İnsanlar okuyabilir, bilgisayarlar kolayca işleyebilir.

- **Örnek:**

```
{
  "isim": "Ahmet",
  "yas": 30,
  "taksiTipi": "Sarı"
}
```

Mikroservis Nedir?

Uygulamayı tek bir devasa parça (Monolit) yerine, lego parçaları gibi küçük, bağımsız çalışan servislere bölmektir.

- **Farkı:** Monolit bir bina gibidir, tadilat zordur. Mikroservisler ise konteyner evler gibidir, biri değişirken diğeri etkilenmez.
- **Bizim Servisler:** `Gateway` (Kapı) ve `Driver-Service` (İşçi).

Veritabanı ve MongoDB Nedir?

Verilerin kalıcı olarak saklandığı dijital depodur.

- **SQL vs NoSQL:** SQL (Excel gibi satır/sütun) katı kurallıdır. **MongoDB (NoSQL)** ise esnektir, verileri yukarıdaki JSON formatına benzer "Dokümanlar" halinde saklar. Bu projede sürücü verileri değişkendir, bu yüzden Mongo seçildi.

Docker Nedir?

"Benim bilgisayarımda çalışıyordu ama sunucuda çalışmıyor" sorununu çözen teknolojidir.

- **Container:** Uygulamayı ve çalışması için gereken her şeyi (kütüphaneler, ayarlar) bir kutunun içine hapseder. Bu kutu her yerde aynı çalışır.

JWT (JSON Web Token) Nedir?

Dijital bir giriş kartıdır. Kullanıcı şifresini girince sistem ona şifreli bir **Token** verir. Kullanıcı sonraki isteklerinde şifresini değil, bu kartı gösterir. Sistem kartın geçerli olup olmadığını kontrol eder.

---

5. GEREKSİNİMLER

Fonksiyonel Gereksinimler (Sistemin Yapması Gerekenler)

1. **Sürücü Yönetimi:**
   - Yeni sürücü kaydı (`Create`).
   - Sürücü bilgilerini güncelleme (`Update`).
   - Tüm sürücüleri listeleme (`List`).
2. **Konum Bazlı Arama (Geospatial):**
   - Kullanıcının Enlem/Boylam bilgisine göre en yakın taksileri bulma.
   - Maksimum 6 km yarıçapında arama yapma.

    - Taksi tipine (sarı, siyah) göre filtreleme.
3. **Routing ve Proxy:**
    - Tüm istekler API Gateway üzerinden geçmeli ve ilgili servise yönlendirilmeli.

Fonksiyonel Olmayan Gereksinimler (Kalite)

1. **Güvenlik:** API uçları yetkisiz erişime kapatılmalı (JWT).
2. **Performans:** Mesafe hesaplamaları veritabanını yormadan yapılmalı.
3. **Bakım:** Kod yapısı "Clean Architecture" prensiplerine uymalı.

---

6. MİMARİ GENEL BAKIŞ

Sistem, sorumlulukların ayrıldığı katmanlı bir yapıya sahiptir.

Yüksek Seviye Mimari

1. **API Gateway:** Dış dünya ile iletişim kuran tek noktadır. Güvenlik kontrolü yapar.
2. **Driver-Service:** İş mantığını yürütür (Örn: Mesafe hesaplama).
3. **MongoDB:** Verileri saklar.

Akış Diyagramı

Kullanıcı bir istek attığında veri şu yolu izler:

```
graph LR
    User(Kullanıcı) -- 1. İstek --> Gateway[API Gateway :8000]
    Gateway -- 2. Token Kontrolü --> Gateway
    Gateway -- 3. İsteği İletir --> Service[Driver Service :8080]
    Service -- 4. Veri İşleme --> Repo[Repository]
    Repo -- 5. Kayıt/Okuma --> Mongo[(MongoDB)]
    Mongo -- 6. Veri --> Repo
    Repo -- 7. Veri --> Service
    Service -- 8. Cevap --> Gateway
    Gateway -- 9. Sonuç --> User
```

Repo Yapısı

Bu proje **Multi-repo** mantığıyla (servislerin ayrı klasörlerde olması) tasarlanmıştır ancak kolay yönetim için tek bir ana klasörde (Monorepo benzeri) toplanmıştır.

---

7. TEKNOLOJİLER VE BAĞIMLILIKLAR

Teknoloji	Versiyon	Kullanım Amacı	Açıklama
Go (Golang)	1.24	Backend Dili	Yüksek performanslı sunucu dili.

Teknoloji	Versiyon	Kullanım Amacı	Açıklama
Echo	v4	Web Framework	Gateway tarafında hızlı yönlendirme (routing) yapan kütüphane.
Standard Lib	-	Web Framework	Driver-service tarafında harici kütüphane kullanmadan ("net/http") saf Go yapısı.
MongoDB Driver	Official	DB Sürücüsü	Go kodunun MongoDB ile konuşmasını sağlar.
JWT (Golang-jwt)	v5	Güvenlik	Token oluşturma ve doğrulama kütüphanesi.
Docker	Latest	Konteyner	Uygulamayı paketlemek için kullanılır.
Swagger	v1	Dokümantasyon	API kullanım kılavuzunu otomatik oluşturur.

8. PROJE YAPISI

Proje, Go topluluğunun kabul ettiği standartlara (Standard Go Project Layout) göre organize edilmiştir.

```
bitaksi-hackathon/
├── docker-compose.yaml      # Tüm sistemi tek tuşla başlatan dosya
├── README.md                # Proje giriş dokümanı
├── gateway/                 # --- API GATEWAY ---
│   ├── Dockerfile           # Gateway için kurulum tarifi
│   ├── go.mod               # Kütüphane listesi
│   └── cmd/
│       └── main.go          # Gateway'in beyni (Auth ve Proxy kodları)
└── driver-service/          # --- DRIVER MICROSERVICE ---
    ├── Dockerfile           # Service için kurulum tarifi
    ├── cmd/server/main.go   # Uygulamanın başlangıç noktası
    ├── internal/            # Dışarıya kapalı, özel kodlar
    │   ├── config/          # Ayarlar (.env okuma)
    │   ├── handler/         # HTTP isteklerini karşılayan garson
    │   ├── models/          # Veri yapıları (Structs)
    │   ├── repository/      # Veritabanı işlemleri (Depo görevlisi)
    │   ├── service/         # İş mantığı (Şef aşçı)
    │   └── utils/           # Yardımcı araçlar (Matematiksel hesaplar)
    └── pkg/database/        # Veritabanı bağlantı kodları
```

Katmanların Görevleri (Driver Service İçin)

- **Handler:** İsteği alır, JSON'ı çözer, `Service`'i çağırır. HTTP status (200, 400) döner.
- **Service:** İş kurallarını uygular (Örn: Mesafe 6km'den az mı?). `Repository`'i çağırır.
- **Repository:** Veritabanına "Kaydet", "Bul", "Güncelle" emirlerini verir. MongoDB detaylarını sadece burası bilir.

9. VERİ MODELİ VE MONGODB

MongoDB `drivers` koleksiyonunda saklanan verinin yapısı şöyledir:

Driver Şeması

Go dilindeki karşılığı (`struct`):

```
type Driver struct {
    ID        primitive.ObjectID `bson:"_id,omitempty" json:"id"` // Benzersiz
Kimlik
    FirstName string             `bson:"firstName" json:"firstName"`
    LastName  string             `bson:"lastName" json:"lastName"`
    Plate     string             `bson:"plate" json:"plate"`       // Plaka
    TaxiType  string             `bson:"taxiType" json:"taxiType"` // Örn: yellow
    Location  Location           `bson:"location" json:"location"` // Konum verisi
    CreatedAt time.Time          `bson:"createdAt" json:"createdAt"`
    UpdatedAt time.Time          `bson:"updatedAt" json:"updatedAt"`
}

type Location struct {
    Lat float64 `bson:"lat" json:"lat"` // Enlem
    Lon float64 `bson:"lon" json:"lon"` // Boylam
}
```

- **Veri Tipleri:** Koordinatlar `float64` (ondalıklı sayı) olarak tutulur. Zaman bilgileri `time.Time` tipindedir.
- **Etiketler:** `json:"id"` etiketi, bu verinin API cevabında `id` olarak görüneceğini belirtir.

---

10. API TASARIMI VE ENDPOINT DOKÜMANTASYONU

Tüm endpoint'ler API Gateway (`localhost:8000`) üzerinden erişilir.

10.1. Kimlik Doğrulama (Auth)

Sistemi kullanmak için önce bilet (token) almalısınız.

- **Endpoint:** `POST /login`
- **İstek (Body):**

  ```
  { "username": "admin", "password": "password123" }
  ```

- **Cevap:**

  ```
  { "token": "eyJhbGciOiJIUzI1Ni..." }
  ```

10.2. Sürücü İşlemleri (Driver Service)

Bu isteklere `Authorization: Bearer <TOKEN>` başlığı (header) eklenmelidir.

A. Yeni Sürücü Oluştur

- **Endpoint:** `POST /drivers`
- **İstek:**

```
{
  "firstName": "Ali", "lastName": "Yilmaz",
  "plate": "34TAX01", "taxiType": "yellow",
  "location": { "lat": 41.0, "lon": 29.0 }
}
```

- **Cevap:** `201 Created`

B. Yakındaki Taksileri Bul

- **Endpoint:** `GET /drivers/nearby`
- **Parametreler:**
  - `lat`: 41.0 (Enlem)
  - `lon`: 29.0 (Boylam)
  - `taxiType`: yellow (Opsiyonel)
- **Örnek İstek (Curl):**

```
curl "http://localhost:8000/drivers/nearby?
lat=41.0&lon=29.0&taxiType=yellow"
```

- **Mantık:** Haversine formülü ile mesafesi 6km ve altı olanlar listelenir.

C. Sürücüleri Listele

- **Endpoint:** `GET /drivers?page=1&pageSize=10`
- **Amaç:** Sayfalı listeleme yapar.

---

11. HATA YÖNETİMİ VE RESPONSE FORMATI

Sistem, hataları standart HTTP kodları ile bildirir.

HTTP Durum Kodları

- **200 OK:** İşlem başarılı.
- **201 Created:** Yeni kayıt başarıyla oluşturuldu.
- **400 Bad Request:** Hatalı veri gönderdiniz (Örn: Eksik parametre, hatalı JSON).
- **401 Unauthorized:** Token yok veya geçersiz (Giriş yapmalısınız).

- **404 Not Found:** İstenen veri (Sürücü) bulunamadı.
- **500 Internal Server Error:** Sunucuda beklenmeyen bir hata oluştu (Örn: Veritabanı çöktü).

Hata Örneği (JSON)

```
{
  "error": "invalid request body",
  "details": "Latitude must be a number"
}
```

---

12. GÜVENLİK VE MIDDLEWARE'LER

Middleware Nedir?

İstek ile asıl kod arasına giren "Ara Yazılım"dır. Güvenlik kontrolü burada yapılır.

JWT Akışı

1. Kullanıcı `/login` endpoint'ine doğru şifreyi gönderir.
2. Gateway, içinde kullanıcı bilgisi ve son kullanma tarihi olan şifreli bir **Token** üretir.
3. Kullanıcı sonraki isteklerde bu Token'ı yollar.
4. **Middleware**, her istekte Token'ın imzasını (`JWT_SECRET` anahtarı ile) kontrol eder.
5. İmza doğruysa ve süre dolmadıysa geçişe izin verir.

Config Yönetimi

Şifreler (DB şifresi, JWT Secret) kodun içine gömülmez. `.env` (ortam değişkenleri) dosyasından okunur. Bu, kodun güvenliği için kritiktir.

---

13. ÇALIŞTIRMA VE DAĞITIM (DEPLOYMENT)

Proje Dockerize edilmiştir. Bilgisayarınızda Go veya MongoDB kurulu olmasa bile çalışır.

Gereksinimler

- Docker Desktop yüklü olmalıdır.

Adım Adım Çalıştırma

1. Terminali açın ve proje klasörüne gidin.
2. Şu komutu çalıştırın:

   ```
   docker-compose up --build
   ```

   - `--build`: Kodda değişiklik yaptıysanız yeniden derler.

- `up`: Sistemi ayağa kaldırır.

Servis Adresleri

- **Gateway (API):** `http://localhost:8000`
- **Driver Service:** `http://localhost:8080` (Gateway arkasında gizlidir)
- **Swagger (Doküman):** `http://localhost:8080/swagger/index.html`
- **Mongo Express (DB Arayüzü):** `http://localhost:8081`

---

14. TESTLER VE KALİTE

Test Nedir?

Yazdığımız kodun doğru çalışıp çalışmadığını kontrol eden kodlardır. "Benim kodum çalışıyor" demek yetmez, kanıtlamak gerekir.

Test Türleri

- **Unit Test (Birim Testi):** Kodun en küçük parçasını (örneğin mesafe hesaplayan fonksiyonu) tek başına test etmektir. Gelecekte `go test ./...` komutu ile çalıştırılabilir.
- **Integration Test (Entegrasyon Testi):** Servisin veritabanı ile doğru konuşup konuşmadığını test etmektir.

---

15. ÖNEMLİ TEKNİK KARARLAR (DECISION LOG)

Geliştirme sürecinde alınan kararlar ve nedenleri:

1. Neden Go?

Karar: Backend dili olarak Go seçildi. **Gerekçe:** Mikroservisler için endüstri standardıdır. Çok az bellek (RAM) kullanır ve çok hızlı başlar. Node.js veya Java'ya göre daha az kaynak tüketir.

2. Neden API Gateway (Echo)?

Karar: İstemcilerin doğrudan servise gitmesi engellendi, araya Gateway konuldu. **Gerekçe:** Güvenliği (Auth) tek bir merkezden yönetmek istedik. Servis sayımız 10'a çıksa bile, kullanıcı sadece Gateway'i bilecek. Echo framework'ü çok hızlı ve kullanımı kolay olduğu için seçildi.

3. Neden MongoDB?

Karar: Veritabanı olarak NoSQL (Mongo) seçildi. **Gerekçe:** Taksi ve konum verileri ile çalışıyoruz. MongoDB'nin coğrafi sorgu yetenekleri (Geo-spatial queries) çok güçlüdür ve şema yapısı esnektir.

4. Neden Haversine Formülü?

Karar: Mesafe hesabı veritabanı yerine Go kodunda yapıldı. **Gerekçe:** Küçük veri setlerinde uygulama katmanında hesaplama yapmak, veritabanı üzerindeki yükü azaltır ve algoritmayı tamamen kontrol etmemizi sağlar.

---

16. GELECEK GELİŞTİRMELER

Proje şu an "MVP" (Minimum Uygulanabilir Ürün) aşamasındadır. İleride şunlar eklenebilir:

1. **Rate Limiting:** Bir kullanıcının saniyede atabileceği istek sayısını sınırlamak (Saldırılara karşı koruma).
2. **Geo-Index:** Veri sayısı milyonlara ulaştığında MongoDB tarafında `2dsphere` indeksi oluşturarak sorguları hızlandırmak.
3. **Logging (Loglama):** Logları sadece ekrana basmak yerine, analiz için bir sisteme (Elasticsearch) göndermek.
4. **CI/CD:** GitHub'a kod yüklendiğinde testlerin otomatik çalışması.

---

17. EKLER

Swagger Kullanımı

`http://localhost:8080/swagger/index.html` adresine giderek, kod yazmadan görsel arayüz üzerinden API'yi test edebilirsiniz. "Try it out" butonuna basarak istek gönderebilirsiniz.

Örnek Curl Komutu (Login)

```
curl -X POST http://localhost:8000/login \
   -H "Content-Type: application/x-www-form-urlencoded" \
   -d "username=admin" \
   -d "password=password123"
```