

Yapılan İş: XML Fatura Verilerinin Temizlenmesi, Ayrıştırılması ve PDF Fatura Oluşturma		Tarih: 29/07/2024
<p>Açıklama:</p> <p>Stajımın 10. gününde, şirketin faturalama sürecini otomatikleştirmek amacıyla bir sistem geliştirdim. Bu sistem, XML formatında gelen fatura verilerini temizleyip işledikten sonra, bu verileri PDF fatura formatına dönüştürmeyi içeriyordu. Projede karşılaşılan sorunları çözüp, verimliliği artıracak adımlar atarak sistemi optimize ettim.</p> <p>1. XML Verilerinin Temizlenmesi</p> <p>Projenin ilk adımı, XML dosyalarındaki gereksiz ve hatalı verilerin temizlenmesiydi. Bu dosyalar bazen fazla detaylı, hatalı karakterler içeren ve büyük boyutlu olabiliyordu. Özellikle SignatureValue, X509Certificate, ve Attachment gibi alanlar XML dosyalarında belleği gereksiz yere şişiriyor ve işlenmesini zorlaştırıyordu. Bu alanları temizleyerek dosyaları sadeleştirdim ve performans sorunlarını çözdüm. Ayrıca, XML dosyalarının içerisinde eksik ya da yanlış karakterler de olabiliyordu. Bu karakterlerin temizlenmesi gerekti, böylece XML dosyası hatasız bir şekilde işlenebildi.</p> <p>Önemli Temizlik Adımları:</p> <p>XML dosyasındaki uzun ve gereksiz SignatureValue ve X509Certificate gibi alanların kaldırılması.</p> <p>Özel karakterlerin düzeltilmesi (örneğin &, <, > gibi semboller).</p> <p>Eksik alanların (tarih, ödeme bilgileri vb.) otomatik doldurulması.</p> <p>2. XML Dosyalarının Ayrıştırılması (Parsing)</p> <p>Temizlenen XML dosyalarını ayrıştırmak, yani içinde yer alan fatura numarası, müşteri bilgileri, ürün detayları, ve toplam tutar gibi önemli bilgileri almak projenin ikinci aşamasıydı. Bu aşamada XML'den gelen karmaşık veri yapıları, program tarafından işlenebilir bir formata dönüştürüldü. Örneğin, her faturada fatura numarası, tarih, müşteri adı, vergi numarası ve adres gibi bilgilerin bulunduğundan emin oldum.</p> <p>Ayrıca bazı dosyalarda eksik veri olabiliyordu. Örneğin, bazı faturalar tarih içermiyordu ya da ödeme bilgileri eksikti. Bu durumda sistem, eksik olan bilgileri doldurarak verinin kullanılabilir hale gelmesini sağladı. Örneğin, eksik olan fatura tarihleri, sistem tarafından otomatik olarak bugünün tarihi ile dolduruldu.</p>		
ÖĞRENCİNİN: İMZASI:		KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:

Yapılan İş: XML Fatura Verilerinin Temizlenmesi,
Ayrıştırılması ve PDF Fatura Oluşturma

Tarih: 29/07/2024

Açıklama:

5. Test Süreci ve Sonuçlar

Sistem, farklı XML fatura dosyaları ile test edildi. Her bir test sırasında XML dosyaları başarıyla temizlendi, ayrıştırıldı ve PDF formatına dönüştürüldü. Sonuçta, her fatura için otomatik olarak PDF oluşturulması işlemi kusursuz şekilde tamamlandı. Özellikle eksik verilerin otomatik olarak doldurulması ve performans iyileştirmeleri sayesinde sistem çok daha verimli hale getirildi.

Genel Sonuçlar:

XML verilerinin temizlenmesi ve düzenlenmesi, sistemin hatasız çalışmasını sağladı.

Ayrıştırma işlemi ile fatura bilgilerinin doğru şekilde alındığından emin olundu.

Performans iyileştirmeleri ile büyük dosyalar bile hızlıca işlenebilir hale getirildi.

PDF fatura oluşturma süreci, tüm bilgilerin eksiksiz ve düzenli şekilde sunulmasını sağladı.

Bu projeyle birlikte, manuel yapılan faturalama işlemlerini otomatik hale getirerek büyük bir zaman tasarrufu sağladım. Ayrıca, sistemin hataya açık alanlarını minimize ederek güvenilir bir çözüm oluşturulmuş oldu.



ÖĞRENCİNİN;
İMZASI:

KURUM SORUMLUSUNUN;

ADI VE SOYADI:

İMZASI:

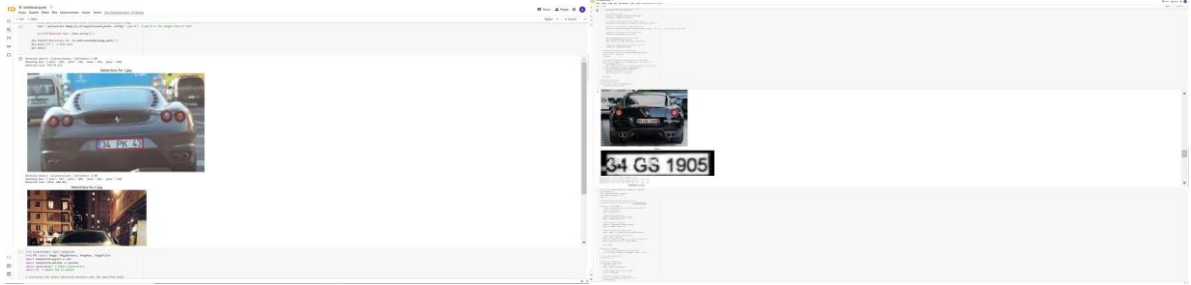
Açıklama:

Yapılan İşler: Bugün, otopark yönetim sistemine entegre edilebilecek bir plaka tanıma sistemi fikrini geliştirdim. Şirketin mevcut yapay zeka ekibi bulunmadığı için, bu fikri sunmak ve uygulanabilirliğini kanıtlamak amacıyla bir prototip üzerinde çalıştım. Plaka tanıma sisteminin araçların otoparka giriş ve çıkışlarını otomatik olarak kaydetmesi hedeflendi.

Plaka Tanıma Sistemi İçin Yapılanlar:

- **Görüntü İşleme ve Plaka Tanıma:** Araç plakalarının okunabilirliğini artırmak için görüntü işleme algoritmaları uygulandı. Düşük çözünürlüklü görüntülerden bile net plaka verileri elde etmek amacıyla çeşitli filtreleme ve netleştirme teknikleri kullanıldı.
- **OCR (Optical Character Recognition) Algoritması:** Görüntülerden plaka bilgilerini metin formatına dönüştüren OCR algoritması kullanıldı. Türk plaka formatına uygun karakterlerin doğru tespit edilmesi için OCR ayarları özelleştirildi.
- **Plaka Bilgilerinin Doğruluğunu Test Etme:** Farklı plaka numaralarına sahip araçların görüntüleri üzerinden testler yapılarak, plaka okuma algoritmasının doğruluğu ve hata oranları analiz edildi.
- **Geliştirme Süreci:** Proje için Tesseract OCR ve transformers kütüphanesi gibi yapay zeka temelli algoritmalar kullanıldı. Sistem, plaka görüntülerini okuyor ve bunları metin verisine dönüştürerek sunucuya gönderiyordu. Testlerde, çeşitli araçların plakaları üzerinde denemeler yapıldı ve başarı oranı değerlendirildi. Hataların minimize edilmesi için plaka okunabilirliğini artıran filtreler üzerinde çalışmalar yapıldı.

Sonuçlar: Yapılan çalışmalar sonucunda, plaka tanıma sisteminin temel prototipi başarıyla geliştirildi. Bu prototip, araç plakalarını yüksek doğrulukla okuma kapasitesine sahip olacak şekilde tasarlandı ve çeşitli testler yapıldı. Testler sırasında sistemin plaka verilerini doğru bir şekilde okuma yeteneği değerlendirildi ve başarılı sonuçlar elde edildi. Prototip, gerçek otopark yönetim sistemine entegrasyonu gerçekleştirilmeden önce, böyle bir sistemin uygulanabilirliğini ve potansiyelini ortaya koydu. Şirket içindeki yapay zeka ekibinin eksikliği nedeniyle, bu çalışmalar prototip aşamasında kalmış olup, gelecekte daha kapsamlı bir entegrasyon için bir temel oluşturdu.



ÖĞRENCİNİN:
İMzası:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMzası:

Yapılan İş: Proje Analizi ve Teknoloji Seçimi		Tarih: 31 /08/2024
<p>Açıklama:</p> <p>Bugün, İSPARK'taki mevcut tarif yönetim sistemindeki problemlerin analizine odaklandım. Sistem karmaşıklıklarının artması, veritabanına doğrudan müdahaleler gerektirmesi ve performans düşüklüğü gibi sorunlar, işletmenin iş süreçlerini olumsuz etkiliyordu. Bu nedenle, daha dinamik ve otomatik bir sistem geliştirme ihtiyacı ortaya çıktı.</p> <p>Yapılan Analiz</p> <p>Öncelikle, bu projenin neyi başarmayı hedeflediğini belirlemek üzere kapsamlı bir analiz gerçekleştirdim. Araç giriş-çıkış işlemlerinin, park yeri yönetiminin, ödeme süreçlerinin ve abonelik yönetiminin otomatik hale getirilmesi, kullanıcıların minimum müdahale ile maksimum verim elde etmeleri gibi başlıca hedefler tanımlandı. Bu analiz sonucunda, mevcut sistemin elle müdahaleye dayalı olması nedeniyle zamanla performansının düştüğü ve karmaşık bir yapıya sahip olduğu ortaya çıktı. Veritabanı işlemleri manuel olarak gerçekleştirildiği için tarife güncellemeleri zorlaşıyor ve bu da işlerin aksamasına neden oluyordu. Yeni sistemin gereksinimlerini belirlerken şu kriterler ön planda tutuldu:</p> <p>Dinamik Tarife Yönetimi: Kullanıcı profillerine, araç tiplerine, otoparklara ve bölgelere göre tarifelerin dinamik olarak belirlenebilmesi gerektiği belirlendi.</p> <p>Yüksek Ölçeklenebilirlik ve Modülerlik: Sistem, artan kullanıcı taleplerine yanıt verecek şekilde kolayca ölçeklenebilmeliydi. Mikroservis mimarisi bu amaç için en uygun çözüm olarak seçildi.</p> <p>Kullanıcı Dostu Arayüzler: Kullanıcıların ve park yeri çalışanlarının sistemi kolayca kullanabilmeleri için kullanıcı dostu arayüzler tasarlanmalıydı.</p> <p>Otomatik Ödeme ve Abonelik Yönetimi: Ödeme ve abonelik işlemlerinin kullanıcı müdahalesine gerek kalmadan otomatik olarak gerçekleşmesi önemliydi.</p> <p>Teknoloji Seçimi</p> <p>Bu hedefleri gerçekleştirmek için kullanılacak teknolojileri belirledik. Mikroservis mimarisi, yüksek modülerlik ve ölçeklenebilirlik sağlayacağı için tercih edildi. Her servisin bağımsız olarak geliştirilip yönetilebileceği bu mimaride, şu teknolojiler kullanıldı:</p> <p>Spring Boot ile mikroservislerin geliştirilmesi, Netflix Eureka ile hizmet keşfi ve servisler arası iletişimin kolaylaştırılması, Kafka ile mesajlaşma sistemi kurularak servislerin birbirleriyle asenkron iletişim kurabilmesi,</p> <p>MongoDB ve MySQL veritabanlarının farklı veri yönetimi ihtiyaçları için kullanılması, Docker ve Kubernetes ile sistemin konteyner bazlı dağıtımı ve yönetimi, Prometheus ve Grafana ile sistem izleme ve performans takibi.</p>		
<u>ÖĞRENCİNİN:</u> İMZASI:	<u>KURUM SORUMLUSUNUN:</u> ADI VE SOYADI: İMZASI:	

Mobil İhtiyaç Analizi ve Teknolojik Uyumun Değerlendirilmesi	Tarih: 01 /08/2024
<p>Açıklama:</p> <p>Bugün, bir önceki gün gerçekleştirdiğim ihtiyaç analizine dayanarak, projenin yeterliliğini kontrol ettim. Bu aşamada, kullanıcı profilleri, senaryolar, iş akışları ve sistemin fonksiyonel ve fonksiyonel olmayan gereksinimlerini detaylandırdım. Aynı zamanda seçilen teknolojilerin, belirlenen ihtiyaçlarla uyumlu olup olmadığını değerlendirdim.</p> <p>Sistem Yöneticileri ve Park Yeri Çalışanları: Park yeri yönetimi, raporlama, ve manuel müdahale gereken durumlar için sistem yöneticisi ve park yeri görevlisi profilleri tanımlandı.</p> <p>Her kullanıcı tipinin görev ve sorumlulukları net bir şekilde belirlendi. Bu görevler, günlük iş akışlarında minimum müdahale ile maksimum verim sağlayacak şekilde optimize edildi.</p> <p>Kullanıcı Senaryoları ve İş Akışları Kullanıcı senaryoları detaylandırıldı ve sistemde nasıl etkileşim kuracaklarına dair iş akışları belirlendi:</p> <p>Park Yeri Sorgulama ve Rezervasyon: Kullanıcılar mevcut park yerlerini ve doluluk durumunu sorgulayabilecek, uygun park yerlerini rezerve edebilecek.</p> <p>Araç Giriş ve Çıkış: Plaka okuma sistemi ile otomatik giriş/çıkış işlemleri tanımlandı. Manuel müdahale gerektiren durumlar için park yeri görevlilerinin sorumlulukları belirlendi.</p> <p>Ödeme ve Faturalandırma: Farklı ödeme yöntemleriyle kullanıcıların ödeme yapabileceği ve otomatik olarak fatura oluşturulup gönderileceği bir sistem kurgulandı.</p> <p>Fonksiyonel Gereksinimler Her mikroservis için fonksiyonel gereksinimler netleştirildi: Araç Yönetim Servisi: Araç ekleme, güncelleme ve silme işlemleri için gerekli API'ler tasarlandı.</p> <p>Park Yeri Yönetim Servisi: Park yeri durumu sorgulama, tahsis ve rezervasyon işlemleri için işlevsel gereksinimler tanımlandı. Giriş/Çıkış Yönetim Servisi: Plaka okuma sistemi ile otomatik giriş ve çıkış işlemleri için detaylar belirlendi.</p> <p>Ödeme ve Abonelik Yönetimi: Kullanıcıların farklı ödeme yöntemleriyle ödeme yapabileceği ve aboneliklerin otomatik olarak yenileneceği bir yapı oluşturuldu. Raporlama Servisi: Günlük, haftalık ve aylık raporlar oluşturularak, yöneticilerin performansı izleyebileceği bir dashboard geliştirilmesi planlandı.</p>	
<u>ÖĞRENCİNİN:</u> İMZASI:	<u>KURUM SORUMLUSUNUN:</u> ADI VE SOYADI: İMZASI:

Yapılan İş: Veritabanı Tasarımı ve Modüler Yapı		Tarih: 02 /08/2024
<p>Açıklama:</p> <p>Fonksiyonel Olmayan Gereksinimler</p> <p>Fonksiyonel olmayan gereksinimlerde de performans, güvenlik, ölçeklenebilirlik ve dayanıklılık gibi unsurlar göz önüne alındı:</p> <p>Performans: Sistem aynı anda 1000'den fazla kullanıcıyı destekleyebilecek şekilde yapılandırıldı.</p> <p>Güvenlik: OAuth2 ve JWT kullanılarak güvenli kimlik doğrulama ve yetkilendirme mekanizmaları oluşturuldu. Kullanıcı verilerinin şifreli olarak saklanması planlandı.</p> <p>Ölçeklenebilirlik: Mikroservislerin bağımsız olarak ölçeklenebilmesi için Kubernetes gibi bir platform kullanıldı.</p> <p>Dayanıklılık: Chaos Monkey ile sistemin hata toleransı test edilecek ve yüksek dayanıklılık sağlanacak.</p> <p>Proje Planı ve Zaman Çizelgesi</p> <p>Son olarak, proje planı ve zaman çizelgesini oluşturup analiz ettim. Her aşama için belirli adımlar ve hedefler net bir şekilde belirlendi. Bu aşamalar şunları kapsıyordu:</p> <p>Gereksinim Analizi ve Planlama Mimari Tasarım Mikroservis Geliştirme API Gateway ve Service Discovery Entegrasyonu Veri Yönetimi ve Depolama Güvenlik ve Kimlik Doğrulama İzleme ve Loglama Dağıtım ve Sürekli Entegrasyon Dokümantasyon ve Eğitim Yedekleme ve Felaket Kurtarma Test ve Hata Ayıklama</p> <p>Bugün, projenin ihtiyaçlarının yeterliliğini ve seçilen teknolojilerin gereksinimlerle uyumunu değerlendirdim. Yapılan bu kapsamlı analizle, projenin tüm adımlarını sistematik bir şekilde ele aldım. Sistem kapasitesi, güvenlik önlemleri ve ölçeklenebilirlik stratejileri belirlendi. Ayrıca, proje için seçilen teknolojilerin uygunluğunu test ettim ve geliştirme sürecinde kullanılacak araçları netleştirdim.</p>		
ÖĞRENCİNİN: İMZASI:		KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:

Yapılan İş: Veritabanı Tasarımı ve Modüler Yapı	Tarih: 02/08/2024
<p>Açıklama:</p> <p>Bugün, otopark yönetim sistemi için oldukça esnek ve modüler bir veritabanı yapısını tamamladım. Bu yapı, kullanıcı profillerinden araç tiplerine, park alanlarından tarifelere kadar her bileşeni özgürce yönetebilecek şekilde tasarlandı. Sistem, farklı kullanıcı grupları ve araç türleri için özel tarifeler ve park paketleri oluşturulmasına olanak sağlıyor.</p> <p>Esnek Veritabanı Yapısı</p> <p>Veritabanı tasarımı, dinamik ve ölçeklenebilir bir yapı kurmayı hedeflemektedir. Bu yapı, tarifeleri, park alanlarını, araç ve kullanıcı tiplerini birbirine bağlayarak, her bileşeni ayrı ayrı yönetme ve değiştirme özgürlüğü sunmaktadır. İstediğim tarifeyi, istediğim araç tipine, istediğim kullanıcı profiline ve park alanına tam anlamıyla özgürce tanımlayabildiğim bir yapı oluşturuldu.</p> <p>Başlıca Bileşenler:</p> <p>Tarifeler (Tariffs): Her tarifeye bağlı olarak zaman dilimleri ve fiyatlandırmalar detaylı bir şekilde yönetiliyor. Tarifeler araç tipine, zaman aralığına ve müşteri türüne göre dinamik bir şekilde atanabiliyor.</p> <p>Müşteri Tipleri (CustomerTypes): Standart, engelli, askeri gibi farklı müşteri grupları tanımlandı. Her müşteri grubu için farklı tarifeler uygulanabiliyor.</p> <p>Araç Tipleri (VehicleTypes): Otomobil, motosiklet, otobüs gibi farklı araç türleri için özel tarifeler ve park paketleri oluşturuldu. Her araç tipi, sistemde ayrı olarak ele alınarak, tarifeler ve müşteri tipleri ile ilişkilendirildi.</p> <p>Park Alanları (ParkingAreas): Farklı lokasyonlar ve özelliklerdeki park alanları, sistemdeki tüm tarifeler ve paketlerle entegre edilerek yönetiliyor. Park alanları, park paketleri ve tarifelerle ilişkilendirilerek dinamik bir yapıda yönetilebiliyor.</p> <p>Park Paketleri (ParkingPackages): Kullanıcıların belirli sürelerde ve belirli alanlarda kullanabileceği park paketleri tasarlandı. Her paket, belirli araç tipleri ve müşteri profilleri için uygun tarifelerle yapılandırıldı.</p> <p>Modüler ve Dinamik Yapı</p> <p>Bu veritabanı tasarımı, modülerliği ve esnekliği ön planda tutuyor. Her bileşen (tarife, araç, müşteri, park alanı vb.) ayrı ayrı yönetilebiliyor ve gerektiğinde değiştirilebiliyor. Bu sayede sistem, yeni tarifeler, paketler veya müşteri grupları eklemek gerektiğinde kolayca genişletilebiliyor. Ayrıca, her park alanı ve müşteri türü için farklı tarifeler atanarak, özelleştirilmiş bir kullanıcı deneyimi sağlanabiliyor.</p> <p>Sonuç</p> <p>Bu veritabanı tasarımı, otopark yönetim sisteminde karşılaşılan sorunların çözülmesini sağladı. Esnek yapı sayesinde tarifeler, araç tipleri ve müşteri profilleri arasında tam anlamıyla özgür bir ilişki kurulabiliyor. Sistem modüler yapısı ile, gelecekte ortaya çıkabilecek değişikliklere ve yeni gereksinimlere hızla adapte olabilecek bir altyapı sunuyor.</p>	
<p><u>ÖĞRENCİNİN:</u> İMZASI:</p>	<p><u>KURUM SORUMLUSUNUN:</u> ADI VE SOYADI: İMZASI:</p>

Açıklama:

```
{
  "id": 1,
  "tarifeNo": "TARIFE-001",
  "name": "Standard Weekday Car Tariff",
  "details": [
    {
      "id": 1,
      "startTime": "00",
      "endTime": "60",
      "price": 10
    },
    {
      "id": 2,
      "startTime": "61",
      "endTime": "120",
      "price": 20
    },
    {
      "id": 3,
      "startTime": "121",
      "endTime": "240",
      "price": 30
    }
  ],
  "lastUpdatedBy": "Auto-generated",
  "createDate": "2024-09-15T08:00:00",
  "updateDate": "2024-09-15T08:00:00"
}

{
  "id": 1,
  "parkingLotNo": "PLT-01",
  "name": "Central Parking",
  "capacity": 100,
  "availableSpaces": 80,
  "numberOfEmployees": 5,
  "numberOfEntrances": 2,
  "entranceNumbers": [
    "E1",
    "E2"
  ],
  "parkingLotType": "OUTD00R",
  "cityCode": "34",
  "districtCode": "37",
  "streetCode": "01",
  "address": "Main Street, Istanbul",
  "upperPackageIds": [
    "UP1"
  ],
  "activeUpperPackageId": "UP1",
  "operatingHours": "08:00-22:00",
  "contactNumber": "555-555-5555",
  "email": "info@parking.com",
  "description": "Main parking lot near the city center",
  "createdAt": "2024-09-15T11:16:23.179328",
  "updatedAt": "2024-09-15T11:16:23.179328",
  "active": true
}

{
  "upperPackageId": "UP1",
  "upperPackageName": "Main City Parking Package",
  "upperPackageDescription": "Comprehensive package for all central parking lots in the city.",
  "upperPackageNo": "UP1",
  "upperSubPackages": [
    {
      "subPackageId": "66e5f66007ffd85b0b0cfd8",
      "subPackageName": "Standard Parking Package",
      "subPackageDescription": "Standard tariff for standard users",
      "subCustomerType": "Standart"
    },
    {
      "subPackageId": "66e5f70207ffd85b0b0cfd9",
      "subPackageName": "Disabled Parking Package",
      "subPackageDescription": "Disabled tariff for disabled users",
      "subCustomerType": "Disabled"
    }
  ]
}

{
  "id": 1,
  "customerTypeNo": "c1",
  "name": "Standart",
  "createdAt": "2024-09-14T15:30:33.42809",
  "updatedAt": "2024-09-14T15:30:33.42809"
}

{
  "id": "66e5f66007ffd85b0b0cfd8",
  "packageName": "Standard Parking Package",
  "packageDescription": "This package offers standard parking for regular customers.",
  "customerTypeName": "Standart",
  "customerTypeNo": "c1",
  "vehicleTypes": [
    {
      "vehicleTypeName": "Car",
      "vehicleTypeNo": "v1",
      "tariffs": [
        {
          "tariffNo": "TARIFE-001",
          "tariffName": "Standard Weekday Car Tariff"
        }
      ],
      "activeTariffNo": "TARIFE-001"
    },
    {
      "vehicleTypeName": "Motorcycle",
      "vehicleTypeNo": "v2",
      "tariffs": [
        {
          "tariffNo": "TARIFE-002",
          "tariffName": "Standard Weekday Motorcycle Tariff"
        }
      ],
      "activeTariffNo": null
    }
  ]
}
```

ÖĞRENCİNİN:**İMzası:****KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMzası:**

31/07/2023 Tarihinden 04/08/2023 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
05/08/2024	Pazartesi	Tarife Servisi Geliştirme (1. Bölüm)	21	8 Saat
06/08/2024	Salı	Tarife Servisi Geliştirme (2. Bölüm)	22/23	8 Saat
07/08/2024	Çarşamba	Tarife Operasyon Servisi Geliştirme	24/25	8 Saat
08/08/2024	Perşembe	Müşteri Tipi Servisi Geliştirme	26/27	8 Saat
09/08/2024	Cuma	Araç Tipi Servisi Geliştirme	28	8 Saat
TOPLAM SÜRE (Saat)				40 Saat
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Alican ÇAĞDAŞ		Adı SOYADI:		
İmzası:		Unvanı:		
Çalıştığı Bölüm: Bilgi Sistemleri Müdürlüğü		İmza/Kaşesi:		

Açıklama:

Bugün, tarife yönetim servisi için temel yapı taşlarını oluşturmaya başladım. Bu servis, tarifelerin dinamik olarak yönetilmesine olanak tanıyan, Kafka ile entegre çalışan ve MySQL ile MongoDB üzerinde veri depolayan bir yapı içeriyor. İlk olarak, Kafka entegrasyonu ve tarife yönetimi için gerekli konfigürasyonları ayarlayarak başladım.

• Kafka Entegrasyonu

Kafka entegrasyonu, sistemde tarifelerin güncellenmesi veya oluşturulması gibi işlemleri kolaylaştırıyor. Özellikle, bir tarifeye Kafka üzerinden gelen bir mesajla yeni bir tarife oluşturulması veya var olan bir tarifenin kopyalanması işlemleri için kullanıldı. Bu kapsamda:

- Producer ve Consumer yapılarını oluşturdum.
- ProducerFactory ve ConsumerFactory ayarlarını Kafka'nın Spring Boot entegrasyonu ile gerçekleştirdim.
- Tarife servisinde Kafka ile gelen mesajları dinleyip, gelen verileri işleyen bir KafkaListener oluşturdum.

Kafka ile tarife yönetimi sayesinde, sistem bir Kafka mesajı aldığında, bu mesajın içeriğine göre yeni tarifeler oluşturulabiliyor ya da mevcut tarifeler kopyalanarak isimleri değiştirilerek yeniden kullanılabilir. Bu işlem hem sistemin esnekliğini artırıyor hem de manuel işlem gereksinimlerini ortadan kaldırıyor.

• Veritabanı Konfigürasyonu

Bu projede MySQL, tarife bilgilerini yönetmek için ana veritabanı olarak kullanıldı. MongoDB ise log yönetimi için kullanılıyor. Veritabanı konfigürasyonları çevresel değişkenlerle kontrol edilebilir şekilde ayarlandı. Aşağıdaki adımları uyguladım:

- MySQL için Hibernate yapılandırması, spring.datasource ayarları ile sağlandı. Tarife detayları ve oluşturulan tarifeler JPA kullanılarak veritabanına kaydedildi.
- MongoDB ise Kafka olayları ve log yönetimi için entegre edildi. Loglar MongoDB'de saklanarak, sistemin olay geçmişi kaydedildi.

```
#!/bin/bash\n# Kafka application configuration\n\n# MySQL Configuration\nspring.datasource.url=${SPRING_DATASOURCE_URL}\nspring.datasource.username=${SPRING_DATASOURCE_USERNAME}\nspring.datasource.password=${SPRING_DATASOURCE_PASSWORD}\nspring.jpa.hibernate.ddl-auto=update\nspring.jpa.show-sql=true\nspring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect\n\n# MongoDB Configuration\nspring.data.mongodb.uri=${SPRING_DATA_MONGODB_URI}\n\n# Kafka Configuration\nspring.kafka.bootstrap.servers=kafka:9092\nspring.kafka.consumer.group-id=tariff-group\nspring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer\nspring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer\nspring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer\nspring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer\nspring.kafka.consumer.properties.spring.json.trusted.packages=*\n\n# Swagger Configuration\nspringdoc.swagger-ui.path=/api/docs\nspringdoc.swagger-ui.path=/swagger-ui.html\n\n# Actuator for Monitoring\nmanagement.endpoint.health.show-details=always\nmanagement.endpoint.metrics.enabled=true
```

ÖĞRENCİNİN:**İMZA:****KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMZA:**

Yapılan İş: Tarife Servisi Geliştirme (2. Bölüm)	Tarih: 06/08/2024
<p>Açıklama:</p> <p>Bugün, dün başladığım tarife yönetim servisi geliştirmesine devam ettim. Bu süreçte tarife servisinin ana bileşenlerini tamamladım ve Docker yapılandırmasını ekleyerek servisin konteyner ortamında çalışabilmesini sağladım.</p> <ul style="list-style-type: none">• Tarife CRUD Operasyonları <p>Tarifeleri yönetmek için CRUD (Create, Read, Update, Delete) işlemleri tamamlandı. Bu işlemler, sistemdeki tarifelerin kullanıcılar tarafından yönetilmesini sağlıyor. Öne çıkan özellikler:</p> <ul style="list-style-type: none">• GET, POST, PUT, DELETE metodlarıyla tarifelerin eklenmesi, güncellenmesi, silinmesi ve listelenmesi.• Tarife numarasına göre kopyalama özelliği. Bu işlemde bir tarife kopyalanarak yeni bir isimle tekrar kullanılabilir hale getiriliyor.• Servisin her bir işlemi loglanarak MongoDB'ye kaydediliyor.• Docker ve CI/CD Entegrasyonu <p>Servis geliştirilip test edildikten sonra, Docker ortamında çalışması için bir Dockerfile yazdım. Bu Dockerfile, Maven kullanarak projenin bağımlılıklarını indiriyor ve ardından uygulamayı Eclipse Temurin JRE üzerinde çalıştırıyor.</p> <ul style="list-style-type: none">• Dockerfile iki aşamalı bir yapı kullanarak Maven bağımlılıklarını indirip projeyi build ediyor, ardından sadece runtime için gerekli dosyaları içeriyor.• Uygulama konteyner bazlı çalıştığından, diğer mikroservislerle entegre şekilde Kubernetes ortamına kolayca dağıtılabilir. <p>• Sonuç</p> <p>Bu iki günlük süreçte, tarife servisi Kafka ve veritabanlarıyla entegre edilerek mikroservis mimarisine uygun hale getirildi. CRUD operasyonları, Kafka üzerinden gelen mesajlarla tarifelerin yönetimi ve veritabanı loglama gibi fonksiyonlar tamamlandı. Docker ile yapılandırılarak konteyner ortamında çalışabilir duruma getirildi.</p> <pre># Base image olarak Maven'i kullanıyoruz. Bu aşamada, projeyi derlemek için gerekli. FROM maven:3.8.8-eclipse-temurin-17 AS build # Çalışma dizinini belirliyoruz. WORKDIR /app # Maven bağımlılıklarını optimize etmek için önce pom.xml dosyasını kopyalıyoruz. COPY pom.xml . # Maven bağımlılıklarını indiriyoruz. Bu, gelecekteki yapıları hızlandırır. RUN mvn dependency:go-offline # Kaynak kodları kopyalıyoruz. COPY src ./src # Maven ile projeyi derliyoruz ve testleri atlıyoruz. RUN mvn package -DskipTests # Runtime için daha küçük bir JRE image kullanıyoruz. FROM eclipse-temurin:17-jre # Derlenmiş JAR dosyasını önceki aşamadan kopyalıyoruz. COPY --from=build /app/target/tariff-service-0.0.1-SNAPSHOT.jar app.jar # Uygulamayı başlatıyoruz. ENTRYPOINT ["java", "-jar", "/app.jar"]</pre>	
ÖĞRENCİNİN: İMZASI:	KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:

Yapılan İş: Tarife Operasyon Servisi Geliştirme	Tarih: 07/08/2024
<p>Açıklama:</p> <p>Bugün, tarife operasyon servisi üzerinde çalıştım. Bu servis, mevcut tarifelerin fiyatlarının dinamik olarak güncellenmesini sağlıyor. Bu süreçte Kafka entegrasyonu, fiyat değiştirme operasyonları ve veritabanı işlemlerini ele aldım.</p> <ul style="list-style-type: none"> • 1. Kafka Entegrasyonu Kafka, mesajlaşma altyapısı olarak kullanıldı. Tarife değişiklikleri yapıldığında bu bilgiler Kafka aracılığıyla diğer servislerle paylaşılabilir. Geliştirme sürecinde aşağıdaki adımları izledim: <ul style="list-style-type: none"> • Producer ve Consumer yapılarını oluşturdum. Servisin, güncellenen tarifeleri Kafka'ya göndermesi için bir Kafka producer oluşturuldu. • ProducerFactory ve ConsumerFactory ayarlarını Kafka'nın Spring Boot entegrasyonu ile yapılandırdım. Özellikle JsonSerializer ve JsonDeserializer kullanılarak tarifelerin JSON formatında seri hale getirilmesi sağlandı. • Tarifeler üzerinde yapılan her değişiklik, Kafka aracılığıyla sistemin diğer bileşenlerine iletiliyor. Bu sayede, tarifelerde yapılan her değişiklik merkezi bir şekilde takip edilebiliyor. • 2. Fiyat Değiştirme Operasyonu Tarife fiyatlarını dinamik olarak değiştirmek için bir REST API oluşturuldu. Bu API, kullanıcıdan alınan istek doğrultusunda tarifenin fiyatını artırma, azaltma, çarpma veya yüzdelik olarak güncelleme işlemlerini gerçekleştiriyor. <ul style="list-style-type: none"> • Fiyat Değiştirme Operasyonu: Kullanıcıdan gelen isteğe bağlı olarak tarifenin her bir detayındaki fiyatlar belirli bir oran veya değere göre güncelleniyor. Örneğin, bir tarifenin fiyatları yüzde 10 artırılabilir veya belirli bir sabit değerle azaltılabilir. • REST API ile Güncelleme: /modify-price endpoint'i ile bir tarife numarası ve güncellenmesi istenen değer girildiğinde, tarifiedeki her bir fiyat detayını güncelleyen bir yapı kuruldu. • Kafka'ya Gönderim: Güncellenen tarifeler, değişiklik yapıldıktan sonra Kafka üzerinden diğer servislere iletiliyor. • 3. Veritabanı ve Fiyat Değişikliği İşlemleri Tarifeler, MySQL veritabanında saklanıyor. Güncellemeler yapıldığında, bu değişiklikler veritabanında güncelleniyor ve ardından Kafka'ya gönderiliyor. <ul style="list-style-type: none"> • JPA ile Veritabanı Entegrasyonu: Tarife ve tarife detayları JPA kullanılarak veritabanında yönetiliyor. Her güncelleme işleminde veritabanı otomatik olarak güncelleniyor. • Fiyat Değiştirme: Fiyat değiştirme işlemleri için çeşitli matematiksel operasyonlar (arttırma, azaltma, çarpma, yüzdelik) tanımlandı. Kullanıcı hangi operasyonu seçerse, o operasyon tarifenin fiyatlarına uygulanıyor. 	
<p>ÖĞRENCİNİN: İMZASI:</p>	<p>KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:</p>

Açıklama:

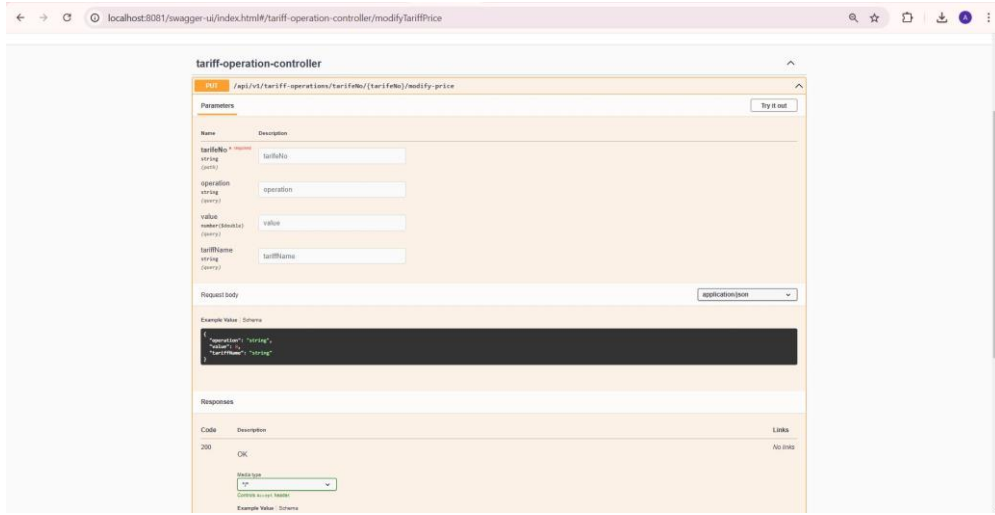
• 4. Docker ile Dağıtım

Servisin Docker ortamında çalıştırılması için gerekli yapılandırmalar yapıldı:

- Dockerfile oluşturuldu ve Maven kullanılarak bağımlılıklar indirildi. Proje, Docker konteyneri içinde çalışacak şekilde yapılandırıldı.
- Temurin JRE kullanarak daha küçük bir runtime imajı oluşturuldu. Bu sayede konteyner boyutu küçültülerek daha hızlı dağıtım ve performans elde edildi.
- Sonuç

Bu süreçte tarife operasyon servisi tamamlandı. Servisin, fiyat değiştirme ve Kafka entegrasyonu sayesinde dinamik bir şekilde çalışabilmesi sağlandı. Ayrıca Docker çalıştırılabilir hale getirilerek konteyner tabanlı mikroservis mimarisi ile uyumlu hale getirildi.

. Kullanıcı hangi operasyonu seçerse, o operasyon tarifenin fiyatlarına uygulanıyor.



ÖĞRENCİNİN:
İMZASI:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Yapılan İş: Müşteri Tipi Servisi Geliştirme	Tarih: 08 /08/2024
<p>Açıklama:</p> <p>Bugün, müşteri tipi servisi üzerinde çalıştım. Bu servis, sistemdeki farklı müşteri tiplerinin (örneğin, standart kullanıcı, engelli kullanıcı, polis-asker gibi) dinamik olarak yönetilmesini sağlıyor. Bu süreçte CRUD işlemleri, operasyon loglama ve Aspect-Oriented Programming (AOP) tabanlı loglama mekanizmalarını ele aldım.</p> <ul style="list-style-type: none"> • 1. Müşteri Tipi CRUD Operasyonları <p>Müşteri tiplerinin eklenmesi, güncellenmesi, silinmesi ve listelenmesi için bir REST API geliştirildi. Aşağıdaki işlemleri gerçekleştirdim:</p> <ul style="list-style-type: none"> • Get, Post, Put, Delete metodları ile müşteri tipleri üzerinde CRUD işlemleri yapıldı. Her müşteri tipinin benzersiz bir customerTypeNo ile yönetilmesi sağlandı. • Müşteri tipi bilgilerini güncellemek için /update endpoint'i üzerinden JSON formatında veriler alınıyor ve veritabanında güncelleniyor. <p>Öne çıkan CRUD işlemleri:</p> <ul style="list-style-type: none"> o GET /api/customer-types: Tüm müşteri tiplerini getirir. o GET /api/customer-types/{customerTypeNo}: Belirli bir customerTypeNo ile müşteri tipini getirir. o POST /api/customer-types: Yeni müşteri tipini oluşturur veya mevcut olanı günceller. o PUT /api/customer-types/{customerTypeNo}: Belirli bir customerTypeNo ile müşteri tipini günceller. o DELETE /api/customer-types/{customerTypeNo}: Belirli bir customerTypeNo ile müşteri tipini siler. <ul style="list-style-type: none"> • 2. Operasyon Loglama <p>Operasyon loglama, sistemde yapılan her değişikliğin kaydedilmesi için oluşturuldu. Bu, hangi işlemlerin ne zaman ve nasıl yapıldığını izlemek için kullanılıyor.</p> <ul style="list-style-type: none"> • OperationLog modeli, yapılan işlemleri (örneğin, bir müşteri tipi oluşturma veya silme) ve detaylarını veritabanında saklıyor. • Serviste yapılan her CRUD işlemi, loglanarak MongoDB'de saklanıyor. Bu loglar, ilerleyen süreçte sistemin izlenmesi ve hata ayıklama için kritik önem taşıyor. 	
<p><u>ÖĞRENCİNİN:</u></p> <p>İMZASI:</p>	<p><u>KURUM SORUMLUSUNUN:</u></p> <p>ADI VE SOYADI:</p> <p>İMZASI:</p>

Açıklama:

• 3. AOP ile Loglama

Sistemde her CRUD işlemi öncesinde ve sonrasında loglama işlemlerini otomatik hale getirmek için Aspect-Oriented Programming (AOP) kullandım.

• Before, AfterReturning, AfterThrowing olmak üzere, her CRUD operasyonunda loglama mekanizmasını otomatikleştirdim.

• LoggingAspect sınıfı, AOP tabanlı bir yapı kullanarak, müşteri tipi servisindeki tüm metod çağrılarının giriş ve çıkış noktalarını logluyor.

o Before: Herhangi bir metodun çağrılmadan önceki durumu loglanır.

o AfterReturning: Metod başarılı bir şekilde sonuçlandıktan sonra loglanır.

o AfterThrowing: Metotta bir hata oluştuğunda bu hata loglanır.

• 4. Veritabanı Yönetimi

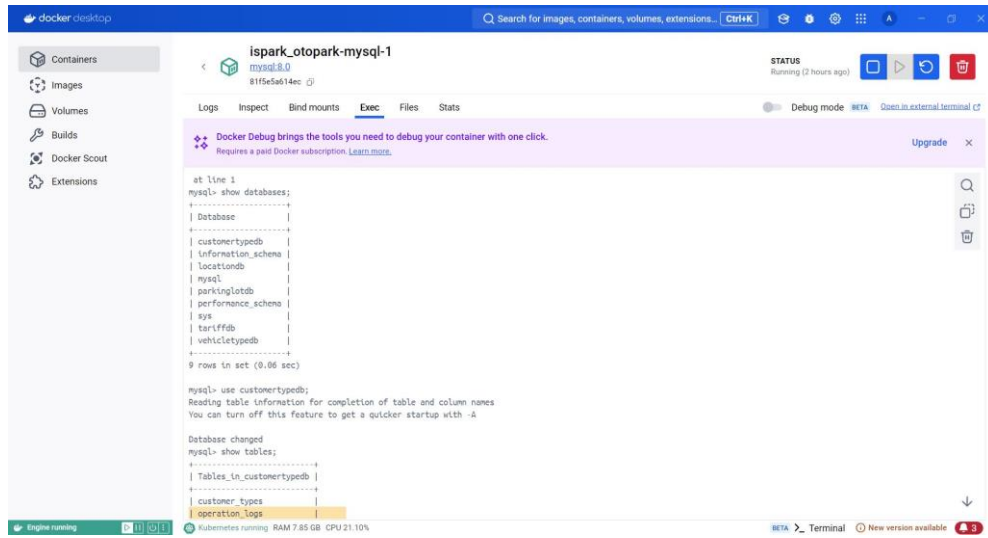
Müşteri tipleri, MySQL veritabanında saklanıyor. Her CRUD işlemi bu veritabanı üzerinden gerçekleştiriliyor.

• CustomerType modeli ile her müşteri tipine ait customerTypeNo, name, createdAt, ve updatedAt alanları tanımlandı.

• Veritabanındaki tüm işlemler, JPA kullanılarak yönetiliyor. JPA, hem müşteri tipi verilerini hem de log kayıtlarını yönetmek için kullanıldı.

• Sonuç

Bugün, müşteri tipi servisinin tam anlamıyla çalışır hale getirilmesi sağlandı. CRUD operasyonları başarıyla tamamlandı ve AOP tabanlı loglama ile sistemdeki her adım kayıt altına alınarak izlenebilirlik sağlandı. Bu yapı, müşteri tiplerinin kolayca yönetilmesine ve sistemde yapılan değişikliklerin izlenmesine olanak tanıyor.



```
at line 1
mysql> show databases;
+-----+
| Database |
+-----+
| customer_typedb |
| information_schema |
| locationdb |
| mysql |
| parkinglotdb |
| performance_schema |
| sys |
| tariffdb |
| vehicletypedb |
+-----+
9 rows in set (0.06 sec)

mysql> use customer_typedb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_customer_typedb |
+-----+
| customer_types |
| operation_logs |
+-----+
```

ÖĞRENCİNİN;
İMZASI:

KURUM SORUMLUSUNUN;

ADI VE SOYADI:

İMZASI:

Açıklama:

Bugün, Vehicle Type (Araç Tipi) Servisi üzerine yoğunlaştım. Bu servis, sistemdeki farklı araç tiplerini (otomobil, motosiklet, otobüs gibi) yönetmek için geliştirildi. CRUD işlemleriyle birlikte, araç tiplerinin dinamik bir şekilde yönetilebilmesi sağlandı.

• 1. CRUD Operasyonları

Araç tipleri üzerinde CRUD (Create, Read, Update, Delete) işlemleri gerçekleştirilebilecek şekilde REST API endpoint'leri oluşturdum. Her araç tipi için bir vehicleTypeNo (benzersiz bir araç tipi numarası) kullanarak işlemler yapılabilir.

Öne çıkan CRUD işlemleri:

- GET /api/vehicle-types: Tüm araç tiplerini listeler.
- GET /api/vehicle-types/{vehicleTypeNo}: Belirli bir vehicleTypeNo ile ilgili araç tipini getirir.
- POST /api/vehicle-types: Yeni bir araç tipi oluşturur veya mevcut olanı günceller.
- PUT /api/vehicle-types/{vehicleTypeNo}: Belirli bir vehicleTypeNo ile mevcut araç tipini günceller.
- DELETE /api/vehicle-types/{vehicleTypeNo}: Belirli bir vehicleTypeNo ile ilgili araç tipini siler.

• 2. Veritabanı Yönetimi

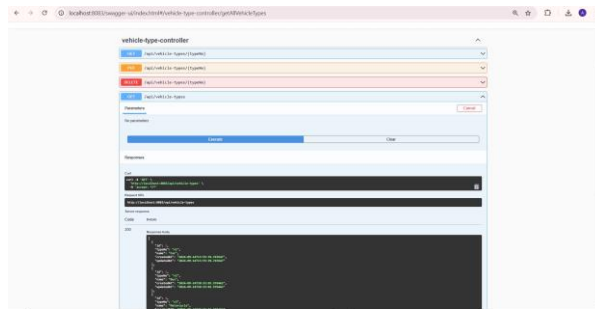
Vehicle Type verileri MySQL veritabanında saklanıyor. Araç tiplerinin her bir kaydı, veritabanındaki vehicle_types tablosunda yer alıyor. JPA kullanarak veritabanı ile entegre bir şekilde işlemleri gerçekleştirdim:

- VehicleType modeli, her araç tipi kaydını yönetiyor.
- Veritabanı işlemleri otomatik olarak güncelleniyor ve her CRUD işlemi veritabanına yansıtılıyor.
- Before: Metot çağrılmadan önce loglama yapılıyor.
- AfterReturning: Metot başarılı bir şekilde tamamlandıktan sonra loglama yapılıyor.
- AfterThrowing: Metotta bir hata oluştuğunda loglanıyor.

Bu mekanizmayı keşfettikten sonra Customer Type Servisine geri dönerek, aynı loglama yapısını oraya da ekledim. Bu sayede, her iki serviste de tüm işlemler izlenebilir hale getirildi.

• 4. Sonuç

Vehicle Type servisi, sistemdeki araç tiplerini dinamik olarak yönetmeyi mümkün hale getirdi. CRUD işlemleri eksiksiz olarak tamamlandı ve loglama mekanizması eklenerek her işlem adım adım izlenebilir hale getirildi. Loglama mekanizmasının keşfi, projenin diğer servislerinde de kullanılmak üzere önemli bir gelişme sağladı.

**ÖĞRENCİNİN:****İMZASI:****KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMZASI:**

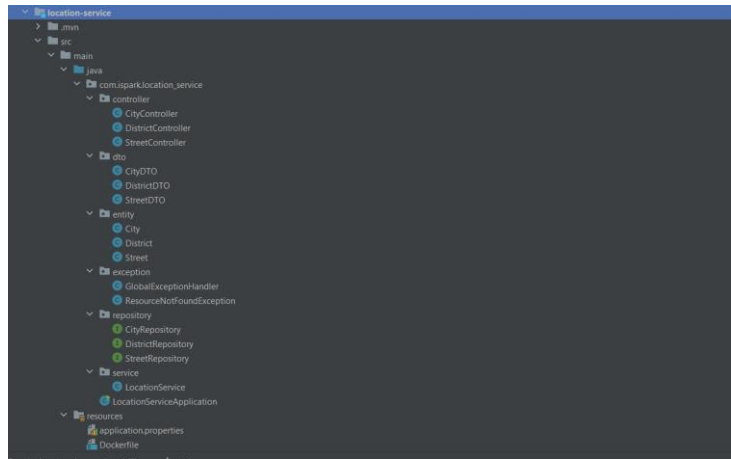
07/08/2023 Tarihinden 11/08/2023 Tarihine Kadar Bir Haftalık Çalışma Tablosu

Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
12/08/2024	Pazartesi	Location Servisi Geliştirme (1. Bölüm)	30	8 Saat
13/08/2024	Salı	Location Servisi Geliştirme (2. Bölüm)	31	8 Saat
14/08/2024	Çarşamba	React Tariff (Tarife), Vehicle Type (Araç Tipi) ve Customer Type (Müşteri Tipi) Bileşenlerinin Geliştirilmesi	32/34	8 Saat
15/08/2024	Perşembe	Paket Oluşturma Servisinin Backend Geliştirilmesi	35	8 Saat
16/08/2024	Cuma	Paket Yönetim Arayüzünün React ile Geliştirilmesi	36	8 Saat
TOPLAM SÜRE (Saat)				40 Saat
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Alican ÇAĞDAŞ		Adı SOYADI:		
İmzası:		Unvanı:		
Çalıştığı Bölüm: Bilgi Sistemleri Müdürlüğü		İmza/Kaşesi:		

Açıklama:

Bugün Location Service adlı mikroservis üzerine çalıştım. Bu servis, otoparkların yerleşeceği Türkiye'deki şehirler, ilçeler ve caddeleri yönetmek amacıyla geliştirildi. Servisin modüler ve esnek yapısı sayesinde ileride toplu değişiklikler yapabilmek mümkün olacak. Aşağıda gerçekleştirdiğim işlemler detaylandırılmıştır:

- 1. Şehir (City) Yönetimi
 - CRUD işlemleri: Şehirleri veritabanında yönetmek için Create, Read, Update ve Delete işlemlerini gerçekleştirdim. Şehir eklerken, her şehir için benzersiz cityCode ve cityName tanımlandı.
 - Swagger dokümantasyonu: Şehir yönetimi için oluşturduğum API'yi Swagger ile dokümente ettim. Böylece şehir sorgulama ve güncelleme işlemleri dış API'ler aracılığıyla yapılabilir hale getirildi.
- 2. İlçe (District) Yönetimi
 - İlçelerin şehir ile ilişkisi: Her bir ilçenin belirli bir şehre ait olacağını garanti etmek için ilçeleri cityCode ile ilişkilendirdim. İlçe ekleme, güncelleme ve silme işlemleri şehir bazında yapılacak şekilde yapılandırıldı.
 - Benzersizlik kontrolleri: İlçe kodu ve isminin şehir içerisinde benzersiz olması için gerekli validasyonları ekledim.
- 3. Cadde (Street) Yönetimi
 - Caddelerin ilçe ile ilişkisi: Caddeleri, bir ilçeye bağlı olacak şekilde yapılandırdım. Böylece her cadde bir ilçe ve şehre bağlı olacak.
 - CRUD işlemleri: Caddeler için de Create, Read, Update ve Delete işlemleri yapıldı. Ayrıca, streetCode ve streetName bilgilerinin ilçede benzersiz olmasını sağladım.
- 4. Hata Yönetimi
 - Sistemde olmayan şehir, ilçe veya cadde sorgulandığında kullanıcıya anlamlı hata mesajları verecek şekilde ResourceNotFoundException ve IllegalArgumentException gibi hataları yönettim. Böylece sistemdeki hatalar kullanıcı dostu mesajlarla bildirildi.



ÖĞRENCİNİN:
İMzası:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

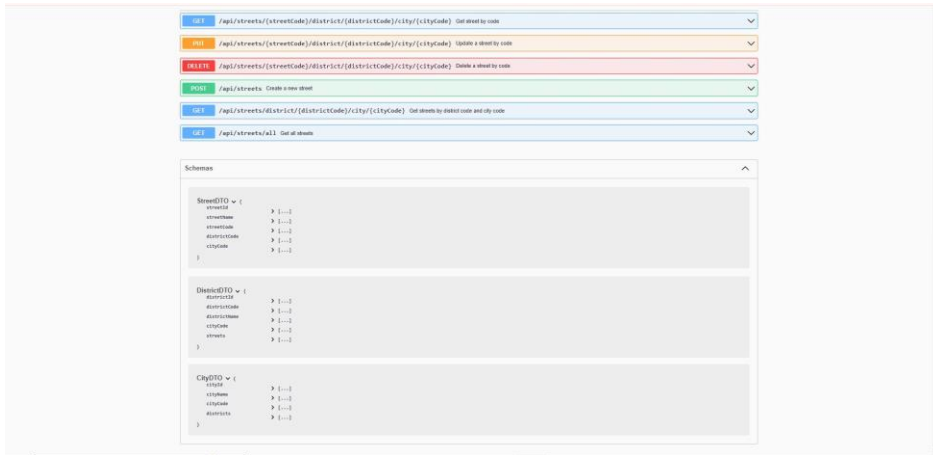
İMzası:

Açıklama:

. Bugün, Location Service üzerinde daha fazla detaylandırma ve geliştirme çalışmaları yaptım. Şehir, ilçe ve cadde yönetimi için geliştirdiğim sistemin modüler yapısını tamamladım. Ayrıca veri transferleri ve exception handling (hata yönetimi) gibi konularda çalıştım.

- 1. Veri Transferi (DTO)
- DTO yapıları: Şehir, ilçe ve cadde verilerini yönetmek için DTO (Data Transfer Object) yapıları oluşturup kullandım. Bu yapılar, veritabanı modellerini dış API'lerden ayırarak veri transferini optimize etti.
- 2. Toplu Değişiklikler ve Esneklik
- Lokasyon verileri toplu olarak güncellenebilecek şekilde modüler bir yapı kurdum. Bu yapı, şehir ve ilçeler üzerinde yapılacak toplu değişiklikleri kolaylaştırıyor. Ayrıca, gelecekte bu yapı kullanılarak otoparkların yerleşimi, ücret tarifeleri gibi konularda kolayca düzenlemeler yapılabilecek.
- 3. Swagger ve Actuator Entegrasyonu
- Swagger: API'nin kullanımı kolaylaştırmak için Swagger dokümantasyonu oluşturdum ve API üzerinden işlemleri test edilebilir hale getirdim.
- Actuator: Uygulamanın sağlık durumu ve performansını izlemek için Spring Actuator ile izleme ve gözlemlene özelliği ekledim.
- 4. Kazanımlar
- Veritabanı yönetimi, DTO yapıları ve Swagger entegrasyonu gibi konularda deneyim kazandım.
- Hata yönetimi konusunda önemli bilgiler edindim ve sistemin nasıl daha kullanıcı dostu hale getirileceğini öğrendim.

Bu iki günlük çalışmada, Türkiye'deki şehir, ilçe ve caddeleri yönetmek üzere esnek ve sürdürülebilir bir servis yapısı geliştirdim.



ÖĞRENCİNİN:
İMZASI:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:

Bu staj günümde, otopark yönetim sisteminde kullanılan Tariff (Tarife), Vehicle Type (Araç Tipi) ve Customer Type (Müşteri Tipi) bileşenlerinin React ile geliştirilmesi üzerine çalıştım. Bu bileşenler, otopark sistemindeki farklı verileri yönetmek için kullanıcıya bir arayüz sağlıyor ve gerekli API entegrasyonları ile backend'den gelen verileri dinamik olarak işleyebiliyor.

1. Tariff (Tarife) Yönetim Bileşeni

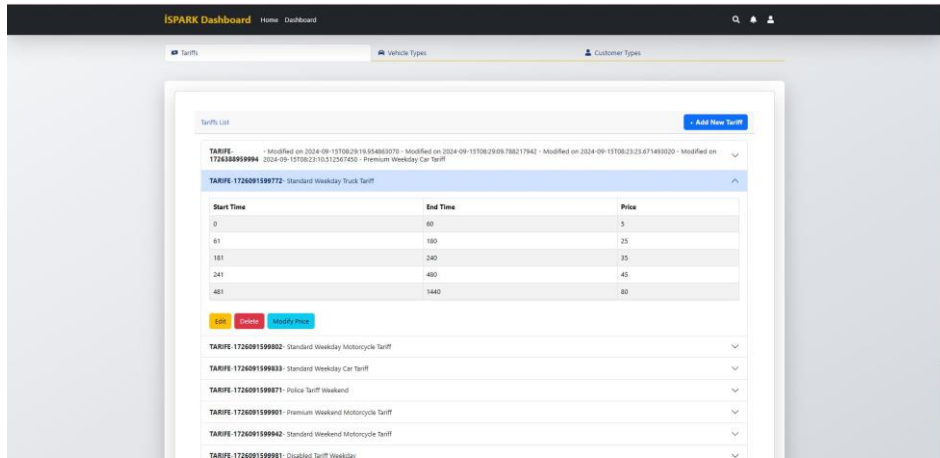
Tariff bileşeni, otoparkta uygulanan fiyatlandırma yapısının yönetilmesini sağlıyor.

Kullanıcıların otoparkta belirli zaman aralıklarına göre farklı fiyatlar belirlemesine olanak tanıyor. Bu bileşeni geliştirirken yaptığım işlemler şunlardır:

Tarife Listeleme: Otopark sistemine kayıtlı olan tüm tarifeleri dinamik olarak listeledim. React Bootstrap Accordion bileşenini kullanarak her bir tarife için detaylı bilgi (başlangıç ve bitiş saatleri, fiyat) gösteriliyor.

Yeni Tarife Ekleme ve Düzenleme: Kullanıcılar yeni bir tarife ekleyebiliyor ya da mevcut bir tarifeyi düzenleyebiliyor. Bunun için bir modal bileşeni kullandım. Modal üzerinde tarife numarası, isim ve zaman dilimleri ile fiyat bilgilerini kullanıcıdan alıp API'ye göndererek veritabanına kaydediyorum.

Fiyat Değişikliği: Fiyatlar üzerinde toplu değişiklikler yapılabilir. Kullanıcı fiyatlara ekleme, çıkarma, çarpma ya da yüzde değişikliği uygulayabiliyor. Bu işlemler de modal üzerinden gerçekleşiyor.



Start Time	End Time	Price
0	60	5
61	180	25
181	240	35
241	480	45
481	1440	80

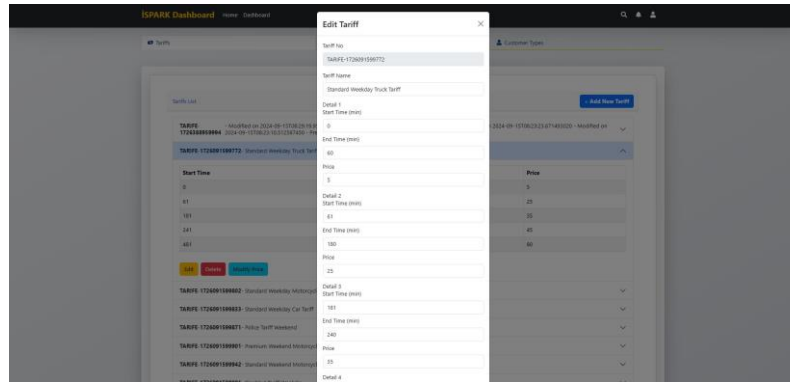
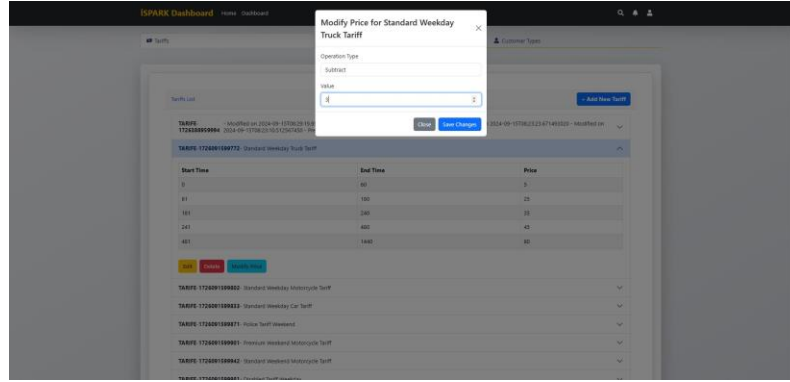
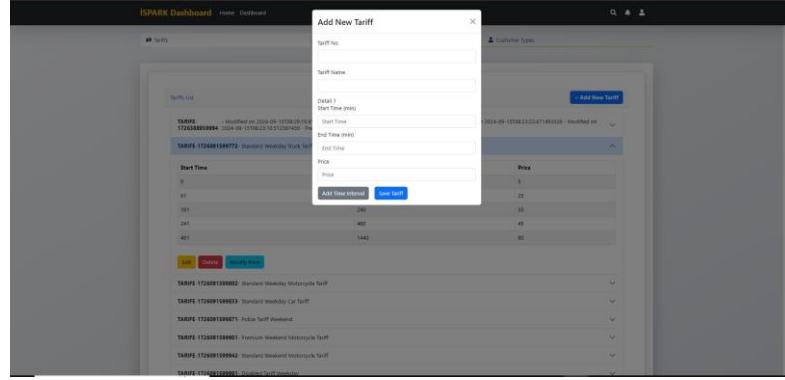
ÖĞRENCİNİN:
İMzası:

KURUM SORUMLUSUNUN:
ADI VE SOYADI:
İMzası:

Yapılan İş: React Tariff (Tarife), Vehicle Type (Araç Tipi) ve Customer Type (Müşteri Tipi) Bileşenlerinin Geliştirilmesi

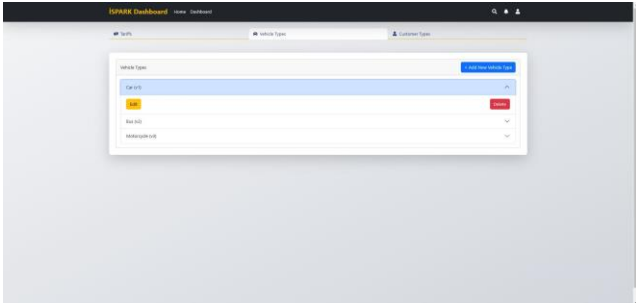
Tarih: 14 /08/2024

Açıklama:



ÖĞRENCİNİN:
İMZASI:

KURUM SORUMLUSUNUN:
ADI VE SOYADI:
İMZASI:

Yapılan İş: React Tariff (Tarife), Vehicle Type (Araç Tipi) ve Customer Type (Müşteri Tipi) Bileşenlerinin Geliştirilmesi	Tarih: 14/08/2024
<p>Açıklama:</p> <p>2. Vehicle Type (Araç Tipi) Yönetim Bileşeni Vehicle Type bileşeni, otoparkta park eden farklı araç türlerinin (örneğin, araba, motosiklet, kamyon) yönetimini sağlıyor. Bu bileşende gerçekleştirdiğim işlemler: Araç Tipi Listeleme: Tüm araç tipleri bir tablo içerisinde listeleniyor. Her araç tipi için düzenleme ve silme butonları ekledim. Yeni Araç Tipi Ekleme/Düzenleme: Kullanıcı, yeni bir araç tipi ekleyebiliyor ya da mevcut bir tipi düzenleyebiliyor. Bu işlem yine modal üzerinden yapılıyor. Araç tipine ait numara ve isim bilgilerini alıp API'ye gönderiyorum. Silme İşlemi: Kullanıcı herhangi bir araç tipini silebiliyor. Bu işlem, kullanıcıdan onay alındıktan sonra API'ye silme isteği göndererek gerçekleştiriliyor.</p> <p>3. Customer Type (Müşteri Tipi) Yönetim Bileşeni Customer Type bileşeni, otopark sisteminde farklı müşteri profillerinin (örneğin, "Standart", "Engelli", "Asker-Polis") yönetimini sağlıyor. Bu bileşende yaptığım işlemler: Müşteri Tipi Listeleme: Müşteri tipleri Accordion yapısıyla listeleniyor. Her müşteri tipi için düzenleme ve silme butonları ekleniyor. Yeni Müşteri Tipi Ekleme ve Düzenleme: Müşteri tipi eklemek veya var olanı düzenlemek için modal kullandım. Müşteri tipi numarası ve isim bilgilerini kullanıcıdan alıp API'ye kaydediyorum. Silme İşlemi: Kullanıcı, müşteri tipini silebiliyor. Silme işlemi API aracılığıyla gerçekleştiriliyor.</p> <p>API Entegrasyonu Bu üç bileşende de API ile veri alışverişi gerçekleştirdim. Her bir bileşen için: GET ile verileri listeleme, POST ile yeni veri ekleme, PUT ile var olan veriyi güncelleme, DELETE ile veriyi silme işlemlerini gerçekleştirdim.</p> <p>Bu staj gününde React ile hem frontend geliştirmeyi hem de API entegrasyonunu yaparak tam işlevsel bir kullanıcı arayüzü oluşturmayı öğrendim. Ayrıca Bootstrap ve Tailwind CSS kullanarak UI tasarımını daha kullanıcı dostu hale getirdim.</p> 	
<p>ÖĞRENCİNİN: İMZASI:</p>	<p>KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:</p>

<p>Yapılan İş: Paket Oluşturma Servisi Backendi Yazıldı UpperPackage Servis İçin Plan Yapıldı Yapının ön yüz etkileşimiyle Oluşmasına Karar Verildi ve Uygulandı</p>	<p>Tarih: 15 /08/2024</p>
<p>Açıklama:</p> <p>Bugün, otopark yönetim sistemi için paket oluşturma servisini geliştirdim. Bu servis, müşteriler için farklı araç tipleri ve tarifeler içeren otopark paketlerini yönetmeye olanak sağlıyor. MongoDB ile veritabanı işlemlerini gerçekleştiren ve Kafka aracılığıyla diğer mikroservislerle haberleşen bir mimari kullanıldı.</p> <p>Yapılan İşlemler: Parking Package Modeli: MongoDB'de saklanacak şekilde otopark paketleri için bir model oluşturuldu. Bu model, paket adı, müşteri tipi, araç tipleri ve tarifeleri içeriyor. Araç tipleri için ayrı bir liste tutuluyor ve her bir araç tipine ait farklı tarifeler kaydediliyor. Ayrıca, her araç tipi için aktif olan bir tarife seçilebiliyor. VehicleType: Paket içerisindeki araç tiplerini ve bu araç tiplerine ait tarifeleri tutan alt model. Tariff: Her araç tipi için kullanılabilen tarifeleri tutan alt model.</p> <p>Controller ve Servis Katmanı: ParkingPackageController sınıfı, RESTful API uç noktalarını yöneten kontrol katmanıdır. Paket oluşturma, güncelleme, silme ve paketleri listeleme gibi işlemleri gerçekleştirdim. ParkingPackageService sınıfı ise iş mantığını içeriyor. CRUD işlemlerini gerçekleştiren servis metodlarını tanımladım.</p> <p>Kafka Entegrasyonu: Araç tipleri ve tarifeleri Kafka aracılığıyla diğer mikroservislerden alabilecek şekilde Kafka consumer ve producer işlemlerini gerçekleştirdim. Kafka mesajlarını dinleyerek paketlere yeni araç tipleri ve tarifeler ekleyebiliyorum.</p> <p>Gerçekleştirilen API'ler:</p> <p>POST /api/parking-packages: Yeni bir otopark paketi oluşturur. GET /api/parking-packages: Mevcut tüm otopark paketlerini listeler. GET /api/parking-packages/{id}: Belirtilen paketi getirir. PUT /api/parking-packages/{id}: Belirtilen paketi günceller. DELETE /api/parking-packages/{id}: Belirtilen paketi siler. POST /api/parking-packages/duplicate/{id}: Mevcut bir paketin kopyasını oluşturur.</p> <p>Bugün Paketlerin oluşturulmasında backendin ve reactın nasıl rol izleyeceğini detaylı bir şekilde çalıştım ve karar vermem uzun sürdü.Feign client kullanmayı düşündüm fakat verimsizdi.</p>	
<p><u>ÖĞRENCİNİN:</u> İMZASI:</p>	<p><u>KURUM SORUMLUSUNUN:</u> ADI VE SOYADI: İMZASI:</p>

Açıklama:

Bugün, otopark paketlerini yönetmek amacıyla bir React tabanlı kullanıcı arayüzü geliştirdim. Bu arayüz, paketlerin dinamik olarak oluşturulmasına, araç tiplerinin ve tarifelerin seçilmesine ve yönetilmesine olanak sağlıyor.

• Yapılan İşlemler:

1. PackageCreator Bileşeni:

o Paket Oluşturma ve Düzenleme: Kullanıcılar, paket adı, açıklama ve müşteri tipi seçerek yeni otopark paketleri oluşturabiliyor. Ayrıca, mevcut paketler düzenlenebiliyor.

2. Araç Tipi ve Tarife Seçimi:

o Arayüzde, farklı araç tipleri ve bu tipler için çeşitli tarifeler seçilebiliyor. Tarifeler arasından biri aktif tarife olarak işaretlenebiliyor.

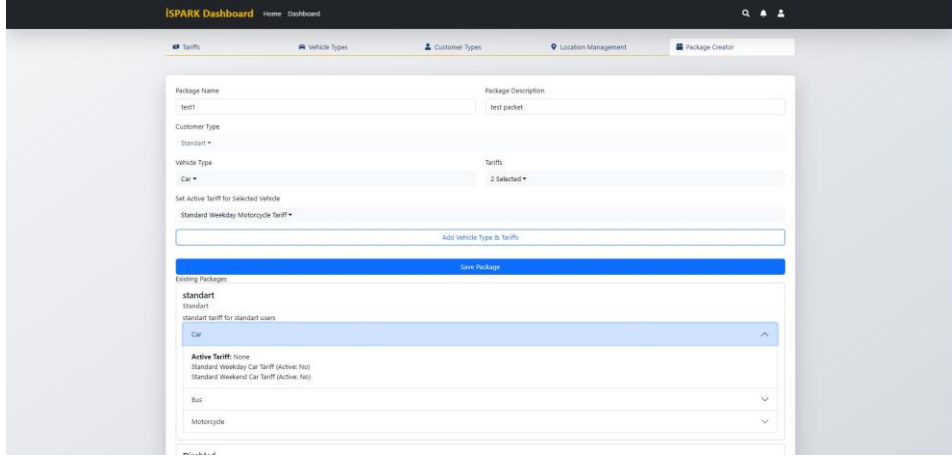
3. Paket Listeleme ve Silme:

o Mevcut otopark paketleri arayüzde listeleniyor ve her paket düzenlenip silinebiliyor.

• Gerçekleştirilen İşlemler:

- Araç tipleri ve tarifeler dinamik olarak kullanıcıya sunuldu.
- Seçilen araç tiplerine uygun tarifeler belirlenerek sistemde aktif hale getirildi.
- Yeni paketler oluşturulurken, kullanıcı dostu bir arayüz ile süreç kolaylaştırıldı.

Bu iki gün boyunca paket oluşturma ve yönetim sürecini, hem backend hem de frontend olarak tam anlamıyla geliştirdim ve Kafka ile mikroservisler arasında iletişim sağladım.



ÖĞRENCİNİN:
İMzası:

KURUM SORUMLUSUNUN:
ADI VE SOYADI:
İMzası:

19/08/2023 Tarihinden 18/08/2023 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
19/08/2024	Pazartesi	Üst Paket Servisinin Geliştirilmesi ve Yönetim Paneli	38/39	8 Saat
20/08/2024	Salı	Otopark Alanı Yönetim Sistemi	40/41	8 Saat
21/08/2024	Çarşamba	Konfigürasyonlar Yapılandırıldı Testler Yapıldı	42	8 Saat
22/08/2024	Perşembe	Prometheus ile İzleme Grafana Kurulumu ve İzleme	43/45	8 Saat
23/08/2024	Cuma	Proje Raporu ve Ayrılma İşlemleri	46	8 Saat
TOPLAM SÜRE (Saat)				40 Saat
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Alican ÇAĞDAŞ		Adı SOYADI:		
İmzası:		Unvanı:		
Çalıştığı Bölüm: Bilgi Sistemleri Müdürlüğü		İmza/Kaşesi:		

Açıklama:

Bugün, otopark yönetim sisteminin üst paket (upper package) işlevini geliştirdim. Bu fonksiyon, birden fazla alt paketi (sub-package) içeren üst paketlerin oluşturulması ve yönetilmesini sağlıyor. Üst paketler, birden fazla alt paketi barındırdığı için daha esnek ve modüler bir yapı sunuyor. Ayrıca, paketlerin Kafka ile sistem içi iletişimini ve React tabanlı bir yönetim panelini tamamladım.

• Backend Çalışmaları:

Üst paketler için bir RESTful API oluşturulması gerekti. Bu yüzden Spring Boot kullanarak aşağıdaki işlemleri gerçekleştirdim:

1. Model Tanımı:

o UpperPackage modelini oluşturdum. Bu model, bir üst paketle ilgili temel bilgileri ve alt paketlerin bilgilerini içeriyor. Alt paketler, müşteri tipleriyle ilişkilendiriliyor.

o SubPackageInfo sınıfı, alt paketlerin detaylarını içeriyor. Alt paketlerin ID, isim, açıklama ve müşteri tipi gibi bilgileri bu modelde yer aldı.

2. Controller ve Service Katmanı:

o UpperPackageController: Bu katmanda REST API uç noktalarını tanımladım. Üst paketlerin eklenmesi, silinmesi, güncellenmesi ve listelenmesi işlemlerini gerçekleştirdim. Ayrıca, alt paketlere müşteri tipi ile erişim sağlayan özel bir uç nokta (endpoint) ekledim.

o UpperPackageService: CRUD işlemlerini yöneten servis katmanını oluşturdum. Bu katmanda, alt paketlerle ilişkilendirilen işlemleri de yönettim.

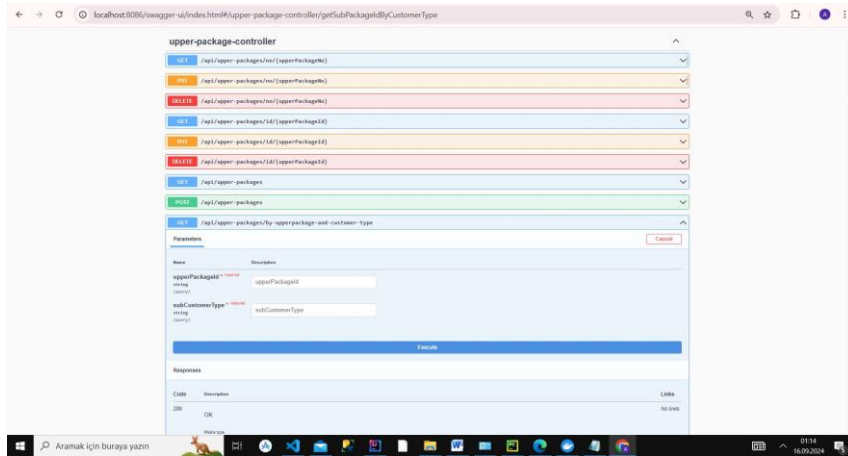
o Üst paket servisine Kafka ile mesaj gönderme ve alma işlemlerini ekledim. Kafka, sistemler arasında haberleşmeyi sağlamak için kullanıldı. Örneğin, bir üst paket talebi geldiğinde Kafka üzerinden diğer mikroservislerle iletişim sağlandı.

• Kafka Entegrasyonu:

Kafka'yı, üst paketlerin isteklerinin ve cevaplarının yönetilmesinde kullandım:

• UpperPackageProducer: Kafka üzerinden üst paket cevaplarını gönderiyor.

• UpperPackageConsumer: Kafka'dan gelen istekleri dinliyor ve ilgili paketi bulup yanıt gönderiyor.

**ÖĞRENCİNİN:****İMZASI:****KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMZASI:**

Açıklama:**• Frontend Çalışmaları:**

Üst paketlerin kullanıcı dostu bir arayüzle yönetilebilmesi için React kullanarak yönetim panelini oluşturdum. Bu panel, kullanıcıların üst paketleri kolayca oluşturmalarına, düzenlemesine ve görüntülenmesine olanak sağlıyor.

1. Üst Paket Yönetim Arayüzü:

- o UpperPackageCreator bileşeni: Kullanıcılar, üst paketleri oluşturabilir, açıklamalarını ekleyebilir ve alt paketleri seçebilir. Ayrıca, müşteri tipine göre alt paketler tanımlandı.
- o Üst paketlerin oluşturulması ve alt paketlerle ilişkilendirilmesi, kullanıcıların basit bir arayüz üzerinden gerçekleştirilebiliyor. Bu sayede paketlerin yönetimi kolaylaştı.

2. Üst Paket Kartları:

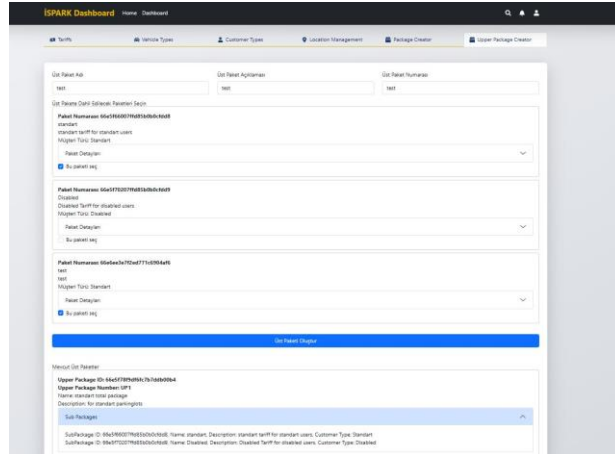
- o Kullanıcılar mevcut üst paketleri listeleyp düzenleyebiliyor veya silebiliyor. Her üst paket kartında, o pakete bağlı olan alt paketler listeleniyor.
- o UpperPackageCard bileşeni, her bir üst paket için bilgi kartı olarak kullanıldı.

3. Form Doğrulama ve Uyarılar:

- o Üst paketler oluşturulurken veya düzenlenirken form doğrulamaları eklendi. Ayrıca, hatalı girişler veya eksik bilgiler olduğunda uyarı mesajları görüntülendi.

• Günün Genel Özeti:

Bu staj günü boyunca, hem backend tarafında üst paket yönetimi için sağlam bir servis geliştirdim, hem de frontend tarafında kullanıcıların paketleri kolayca yönetebileceği bir arayüz oluşturdum. Özellikle Kafka entegrasyonu sayesinde mikroservisler arası haberleşmeyi daha etkin hale getirdim.

**ÖĞRENCİNİN:**
İMZASI:**KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMZASI:**

Açıklama:

Bugün, Otopark Yönetim Sistemi'nde önemli bir güncelleme üzerinde çalıştım. Bu güncelleme kapsamında otoparkların yönetimi için hem backend hem de frontend tarafında yeni özellikler ekledim.

Günün İlk Yarısı - Backend Geliştirme:

Otoparkların yönetimi için REST API endpoint'leri geliştirdim. Bu endpoint'ler, otoparkların eklenmesi, düzenlenmesi, silinmesi ve listelenmesi gibi temel CRUD işlemlerini kapsıyor.

Ayrıca her otopark için aktif olan üst paketlerin yönetimini de ekledim.

Otopark CRUD İşlemleri:

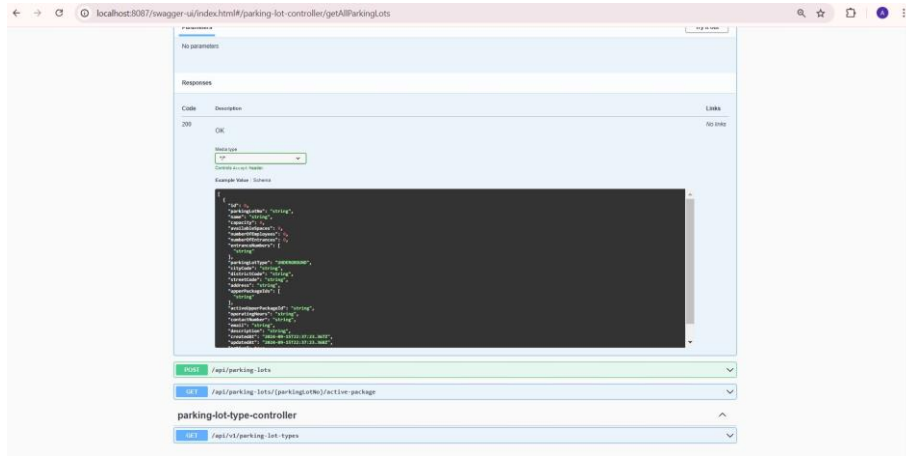
POST /api/parking-lots: Yeni bir otopark eklenmesi için gerekli fonksiyonu yazdım. Otopark bilgileri JSON formatında alınıyor ve veritabanına kaydediliyor.

GET /api/parking-lots: Tüm otoparkların listelenmesi için bir endpoint geliştirdim.

PUT /api/parking-lots/{parkingLotNo}: Mevcut bir otoparkın güncellenmesi için gerekli fonksiyonu yazdım.

DELETE /api/parking-lots/{parkingLotNo}: Otoparkın silinmesi işlemini gerçekleştirdim.

Aktif Üst Paket Yönetimi: Her otoparkın farklı üst paketlere (upper packages) sahip olabilmesi ve bir tanesinin aktif olarak seçilmesi sağlandı. Bu özellik, otoparklara özel fiyatlandırma ve hizmet paketlerinin atanmasını kolaylaştırıyor.

**ÖĞRENCİNİN:**

İMZASI:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:

Günün İkinci Yarısı - Frontend Geliştirme:

Backend'deki CRUD işlemlerini desteklemek için React kullanarak otopark yönetim arayüzünü geliştirdim. Arayüzde otoparkların eklenmesi, düzenlenmesi ve silinmesi işlemlerini kolaylaştıran bir yapı oluşturdum.

Otopark Listesi ve Düzenleme:

Tüm otoparkları listeleyen bir tablo oluşturdum. Bu tabloda otoparkların adı, kapasitesi, mevcut boş alanları, adresi gibi bilgiler gösteriliyor.

Her otopark için düzenleme ve silme butonları ekledim. Düzenleme işlemi için bir modal açılıyor ve kullanıcının otopark bilgilerini güncellemesine izin veriliyor.

Yeni Otopark Ekleme: Yeni bir otopark eklemek için bir form geliştirdim. Formda otopark numarası, adı, kapasitesi, giriş sayısı, adres ve üst paketler gibi bilgiler alınıyor. Kullanıcı, form üzerinden otoparkın sahip olduğu üst paketleri seçebiliyor ve birini aktif olarak işaretleyebiliyor.

API Entegrasyonu: Arayüzdeki her işlem, backend'deki REST API'lerle entegre edildi. Otopark ekleme, düzenleme ve silme işlemleri, fetch API kullanılarak backend ile haberleştirildi.

Sonuç olarak, bugünkü çalışmalarında otopark yönetim sistemini daha kullanıcı dostu ve yönetilebilir hale getirdim. Geliştirdiğim yeni özelliklerle, otopark yöneticileri sistem üzerinden otopark bilgilerini kolayca yönetebilecek ve farklı hizmet paketlerini tanımlayabilecek.

ÖĞRENCİNİN:
İMZASI:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:

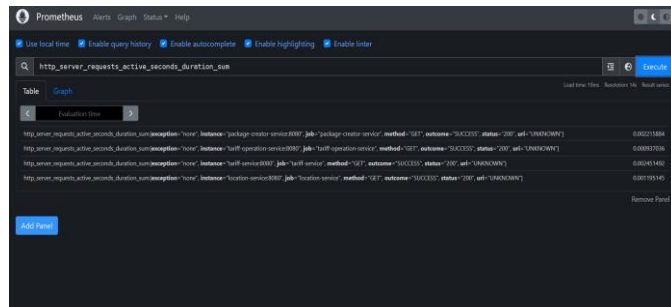
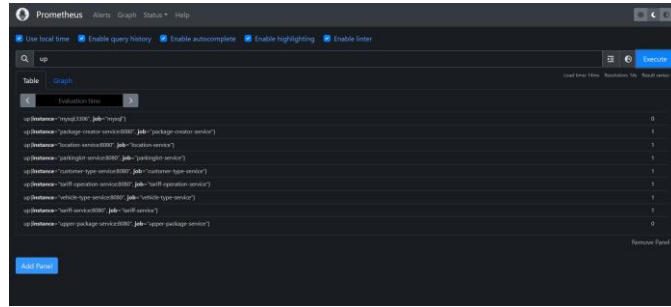
Bugünkü staj çalışmalarımnda Prometheus ve Grafana'yı kullanarak otopark yönetim sistemindeki mikroservislerin izlenmesi ve performans takibine yönelik monitöring sistemini kurdum. Bu sistem, mikroservislerin performansını gerçek zamanlı olarak takip etmeme ve sorunları tespit etmemi sağlıyor.

Prometheus İzleme Sistemi Kurulumu:

Prometheus, mikroservislerin durumunu ve sağlığını kontrol etmek amacıyla kullanıldı. Prometheus konfigürasyon dosyasına mikroservislerin "metrics" endpoint'leri eklendi. Örnek olarak:

tariff-service, package-creator-service, customer-type-service, parkinglot-service, location-service, gibi servisler monitör edilecek şekilde Prometheus ayarlarına eklendi.

Ekran görüntüsünde görüldüğü üzere, servislerin up durumu sorgulandı ve her bir servisin durumunu (up/down) kontrol ettik. Ayrıca, sorgulara dair sonuçlar Prometheus konsolunda başarılı bir şekilde görüntülendi.



ÖĞRENCİNİN:
İMzası:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMzası:

Açıklama:

Prometheus ile topladığımız metriklerin daha iyi bir şekilde analiz edilmesi ve raporlanması için Grafana'yı yapılandırdım.

Prometheus, Grafana'ya veri kaynağı olarak bağlandı (Grafana'dan görüldüğü gibi "Save & Test" butonuna tıklayarak Prometheus API'si başarıyla sorgulandı).

Mikroservislerin CPU kullanımını ve yanıt sürelerini izlemek için Grafana panelleri oluşturuldu. Grafikte, servislere dair CPU kullanım oranları belirli zaman dilimlerinde izlendi. Bu grafikte, örneğin package-creator-service, customer-type-service, location-service gibi servislere ait CPU kullanım eğrileri görülebiliyor.

Prometheus Query'leri ve Metod İstatistikleri:

Prometheus üzerinde sorgular yaparak HTTP server isteklerinin başarı durumu, yanıt süresi ve metod tiplerini izledim. Özellikle http_server_requests_active_seconds_duration_sum metriği kullanarak mikroservislerin yanıt süreleri hesaplandı.

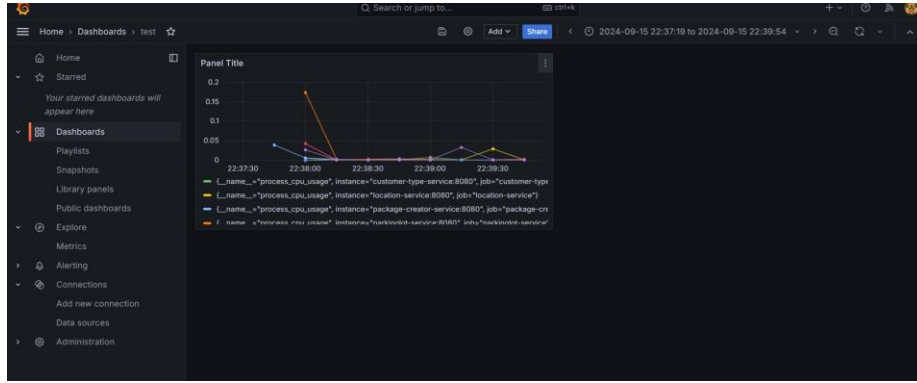
Bu metrikle, örneğin GET metoduna ait başarılı yanıtlar ve süreleri görüntülenerek servislerin yanıt verme süreleri analiz edildi. İkinci ekran görüntüsünde, bu sorguların başarılı sonuçlarını görebiliriz.

Scrape Konfigürasyonları:

Prometheus scrape interval ve scrape_configs ayarları yapıldı. Her 10 saniyede bir metrik toplama işlemi yapılacak şekilde yapılandırıldı.

Yüklenen ekran görüntülerinden de anlaşılaacağı üzere, Prometheus konfigürasyon dosyasındaki tariff-service, package-creator-service, customer-type-service gibi servislerin metrics endpoint'leri, actuator/prometheus yolunu kullanarak başarıyla Prometheus'a bağlandı.

Bu adımlarla mikroservislerimizin performansını gerçek zamanlı izleme ve analiz etme altyapısını kurmuş oldum. Otopark yönetim sisteminde yer alan her servisin durumu ve performansı artık sürekli takip edilebilir hale geldi. Olası hata ve performans sorunlarını tespit etmek ve daha hızlı müdahale edebilmek için önemli bir altyapı kuruldu.

**ÖĞRENCİNİN:**

İMZASI:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:

```
global:
  scrape_interval: 10s # How frequently to scrape targets by default

scrape_configs:
  # Scrape configuration for tariff-service
  - job_name: 'tariff-service'
    metrics_path: '/actuator/prometheus' # Assuming Spring Boot exposes metrics at this path
    static_configs:
      - targets: ['tariff-service:8080']

  # Scrape configuration for tariff-operation-service
  - job_name: 'tariff-operation-service'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['tariff-operation-service:8080']

  # Scrape configuration for package-creator-service
  - job_name: 'package-creator-service'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['package-creator-service:8080']

  # Scrape configuration for customer-type-service
  - job_name: 'customer-type-service'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['customer-type-service:8080']
```

ÖĞRENCİNİN:
İMZASI:

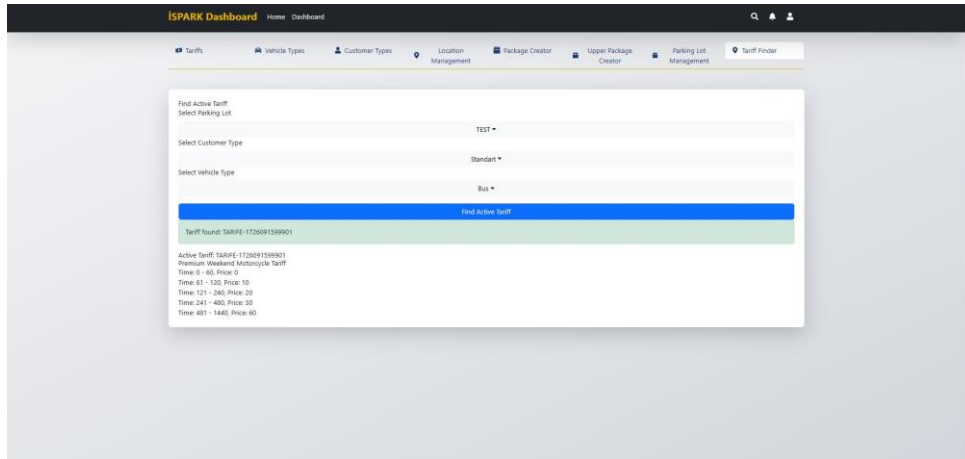
KURUM SORUMLUSUNUN:
ADI VE SOYADI:
İMZASI:

Açıklama:

Bugün stajımın son günü ve staj sonu raporumu oluşturdum ardından öncelikle ekip arkadaşlarımla ve departmanımda bana yardımda bulunan herkesle vedalaştım. Bilgisayar ve zimmet teslimlerimi yaptım.

Stajımın son gününde, projemin Kubernetes entegrasyonu üzerine yoğunlaştım ve projenin sunumunu gerçekleştirdim. Kubernetes'i kullanarak container'ların yönetimini ve ölçeklenmesini sağlamak için gerekli adımları attım. Özellikle tüm mikroservisleri Kubernetes pod'ları olarak çalıştırmaya yönelik çalışmalar yaptım. Projemin GitLab'e push edilmesi ve sürüm kontrolü GitLab üzerinden sağlandı. Bununla birlikte, Kubernetes altyapısı ile projenin verimli bir şekilde dağıtılması ve yönetilmesi amaçlandı. Sunum sırasında Kubernetes entegrasyonu ile ilgili elde edilen kazanımlar ve uygulamanın ölçeklenebilirliği üzerinde durdum. Ancak, Kubernetes üzerinde canlıya alma süreci devam ediyor ve bu sürecin tam anlamıyla çalışabilmesi için bazı ayarlamaların yapılması gerekiyor.

Ayrıca, stajımın son gününde "Tariff Finder" adında bir React tabanlı uygulama geliştirdim. Bu uygulama, otoparklarda geçerli olan aktif tarifeleri bulmayı ve müşteriye özel tarifelerin detaylarını göstermeyi hedefliyor. Kullanıcılar, otopark, müşteri türü ve araç türü seçerek aktif tarifeleri bulabiliyor. Bu süreçte, otoparkın aktif olduğu paketi bulma, müşteri türüne göre alt paketi seçme ve seçilen araç türüne göre aktif tarifeyi getirme gibi adımlar bulunuyor. Bu işlemleri API çağrıları ile gerçekleştiren uygulama, kullanıcıya doğru tarife detaylarını sunuyor ve sistemin tarifeleri doğru bir şekilde yönetmesine yardımcı oluyor



ÖĞRENCİNİN:
İMZASI:

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI: