BURSA TEKNİK ÜNİVESİTESİ

MÜHENDİSLİK ve DOĞA BİLİMLERİ FAKÜLTESİ

GENETİK ALGORİTMA İLE MOBİLİTE TABANLI YOL OPTİMİZASYONU UYGULAMASI

LİSANS BİTİRME ÇALIŞMASI

Güven TUNCAY

Erdoğan TURPCU

Mehmetcan DALMAZGİL

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

TEMMUZ, 2022

BURSA TEKNİK ÜNİVESİTESİ

MÜHENDİSLİK ve DOĞA BİLİMLERİ FAKÜLTESİ

GENETİK ALGORİTMA İLE MOBİLİTE TABANLI YOL OPTİMİZASYONU UYGULAMASI

LİSANS BİTİRME ÇALIŞMASI

Güven TUNCAY

18360859042

Erdoğan TURPCU

19360859076

Mehmetcan DALMAZGİL

18360869001

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

DANIŞMAN: Dr. Öğr. Üyesi İzzet Fatih ŞENTÜRK

TEMMUZ, 2022

BTÜ, Mühendislik ve Doğa Bilimleri Fakültesi Bilgisayar Mühendisliği Bölümü öğrencileri 18360859042 öğrenci numaralı Güven TUNCAY, 19360859076 öğrenci numaralı Erdoğan TUPRCU ve 18360859001 öğrenci numaralı Mehmetcan DALMAZGİL ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "Genetik Algorimta ile Mobilite Tabanlı Yol Optimizasyonu Uygulaması" başlıklı bitirme çalışmasını aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Danışman : **Dr. Öğr. Üyesi İzzet Fatih ŞENTÜRK** .....

Bursa Teknik Üniversitesi

Jüri Üyeleri : **Doç. Dr. Haydar ÖZKAN** .....

Bursa Teknik Üniversitesi

Dr. Öğr. Üyesi Volkan ALTUNTAŞ .....

Bursa Teknik Üniversitesi

Savunma Tarihi : 07 Temmuz 2022

BM Bölüm Başkanı : Prof. Dr. Turgay Tugay BİLGİN .....

Bursa Teknik Üniversitesi ...../...../.....

İNTİHAL BEYANI

“Bu bitirme çalışmasındaki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim”

Öğrencilerin Adı Soyadı: Güven TUNCAY Erdoğan TURPCU Mehmetcan DALMAZGİL

İmzası:

ÖZET

Son yollarda, çevrimiçi yemek siparişi uygulamalarının sayısı hızla artmıştır. Bu artış işletmelerin gün içinde aldıkları sipariş sayısının da artmasına sebep olmuş ve siparişlerin yönetilmesi oldukça zor bir hale gelmiştir. Temel sorun bir restoranın kısa süre içerisinde teslim etmesi gereken onlarca siparişi birden çok kurye arasında uygun bir şekilde paylaştıramamasıdır. Rastgele yapılan bir paylaştırma işlemi hem siparişlerin istenilen zamanda ulaştırılamamasına hem de kuryelerin gereksiz yakıt ve zaman kaybına uğramasına neden olabilmektedir. Bu da işletmenin müşteri karşısında zamanla kötü bir izlenim oluşturmasına sebep olmaktadır. Özellikle çevrimiçi yemek siparişi uygulamalarındaki puanlama sistemi ile müşterilerinin kötü yorumlarının adeta bir kartopu etkisi oluşturması işten bile değildir.

Günümüz teknolojileri ile, tüm dünyadaki yollar, bu yollardaki farklı araçların gidebilecekleri hız, bu yollardan harcanabilecek yakıt gibi bilgiler, meta-sezgisel algoritmalar ile birleştirilerek bu sorunun çözümüne ulaşılabilir. Gelen siparişlerin kuryeler arasına paylaştırılması ve kuryelerin en optimum rota ile yola çıkması; müşterinin siparişi olması gerektiği zaman teslim alması, gelen siparişleri gereğinden az ya da çok beklememesi, kuryelerin minimum yakıt harcaması, kuryelerin mümkün olduğunca az zaman kaybetmesi, birim zamanda teslim edilen sipariş miktarının artması gibi birçok fayda sağlanılabilir.

ABSTRACT

In recent years, the number of online food ordering apps has grown rapidly. This increase has led to an increase in the number of orders received by businesses during the day and it has become very difficult to manage the orders. The main problem is that a restaurant cannot properly allocate dozens of orders that need to be delivered in a short time, among multiple couriers. A random allocation process can cause both orders not to be delivered at the desired time and cause the couriers to lose unnecessary fuel and time. This causes the business to have a bad impression over time. Especially with the scoring system in online food ordering applications, bad reviews of customers easily create a snowball effect.

With today's technologies, information such as roads all over the world, the speed at which different vehicles on these roads can go, and the fuel that can be spent on these roads can be combined with meta-heuristic algorithms to solve this problem. Sharing the incoming orders among the couriers and the couriers set out with the most optimum route; many benefits can be provided, such as the customer receiving the order when it should be, not waiting for incoming orders more or less than necessary, minimum fuel consumption by the couriers, wasting as little time as possible by the couriers, and increasing the amount of orders delivered per unit time.

TEŞEKKÜR

Bitirme projesi geliştirim aşamasında çalışmamızı titizlikle takip eden değerli hocamız Dr. Öğr. Üyesi İzzet Fatih ŞENTÜRK' e, lisans dönemi boyunca derslerinde bulunduğumuz tüm saygıdeğer hocalarımıza ve çalışmamız sırasında bizden yardımlarını esirgemeyen herkese değerli katkıları için içten teşekkürlerimizi ve saygılarımızı sunuyoruz.

Ayrıca bizi bu günlere getiren sevgili ailelerimize de teşekkürü bir borç biliriz.

TEMMUZ 2022

Güven TUNCAY
Erdoğan TURPCU
Mehmetcan DALMAZGİL

İÇİNDEKİLER

ÖZET III
TEŞEKKÜR V
İÇİNDEKİLER VI
ŞEKİLLER DİZİNİ VII
KODLAR DİZİNİ IX
1. GİRİŞ 1
2. PROJE TANITIMI 1
2.1. Optimizasyon Tarafı Geliştirim Çalışmaları 3
2.2. Arka Uç Tarafı Geliştirim Çalışmaları 3
2.3. Mobil Tarafı Geliştirim Çalışmaları 3
3. KULLANILAN TEKNOLOJİLER 5
3.1. Optimizasyon Tarafında Kullanılan Teknolojiler 5
3.2. Arka Uç Geliştirme Tarafında Kullanılan Teknolojiler 11
3.3. Mobil Uygulama Geliştirme Tarafında Kullanılan Teknolojiler 16
4. YÖNTEM VE ÇALIŞMALAR 20
4.1. Optimizasyon Çalışmaları 20
4.2. Arka Uç Çalışmaları 36
4.3. Mobil Uygulama Çalışmaları 44
5. SONUÇLAR 56
6. KAYNAKLAR 58
7. ÖZGEÇMİŞ 59

ŞEKİLLER DİZİNİ

Şekil 2.1. Uygulama logosuna ait görsel....4
Şekil 2.2. Uygulama mobil ana ekranına ait görsel....5
Şekil 3.1. Genetik algoritma akış diyagramı....8
Şekil 3.2. Java Programlama Dili Logosu....11
Şekil 3.3. Spring Framework Logosu....12
Şekil 3.4. Spring Boot Logosu....13
Şekil 3.5. Spring Data Logosu....13
Şekil 3.6. Spring Security Logosu....14
Şekil 3.7. PostgreSQL Logosu....14
Şekil 3.8. Örnek bir JWT....15
Şekil 3.9. Swagger UI Logosu....16
Şekil 3.10. MVC Mimarisi Modeli....19
Şekil 4.1. TSP ve VRP problemine ait görsel....20
Şekil 4.2. Örneklem veri setine ait görsel....22
Şekil 4.3. İki boyutlu "Öklid" uzaklık formülü....22
Şekil 4.4. Örneklem veri setine ait uzaklık matrisi....23
Şekil 4.5. Örnek TSP genetik algoritması parametre değerleri....23
Şekil 4.6. Örneklem veri seti için olası bir çözüm kümesi(kromozom)....24
Şekil 4.7. Örneklem veri seti genetik algoritma sonucu....26
Şekil 4.8. Örneklem veri seti genetik algoritma sonuç grafiği....26
Şekil 4.9: Görükle Mahallesine ait graf yapısı....28
Şekil 4.10: 20 adrese ve depoya ait uzaklık matrisi kesiti(metre)....28
Şekil 4.11: Postman uygulaması üzerinden atılan istek sorgusu kesiti....30
Şekil 4.12: Genetik algoritma ve araç parametrelerine ait görsel....31
Şekil 4.13: Genetik algoritma adımlarına ait toplam alınan mesafe değişimi....31
Şekil 4.14: Genetik algoritma çıktı değeri ve çalışma süresine ait görsel....32
Şekil 4.15: Hiper parametre optimizasyonunda kullanılan parametre değerleri....32
Şekil 4.16: En iyi 10 çözüm değerine ait görsel....33
Şekil 4.17: En kötü 10 çözüm değerine ait görsel....33
Şekil 4.18: Manuel arama sonucu belirlenen en iyi çözüme ait hiper parametre değerleri....34

Şekil 4.19: Yeni hiper parametre değerlerine ait çıktı 34
Şekil 4.20: 1.kurye için arka uca iletilen cevap JSON içeriği.....35
Şekil 4.21: Geliştirilen Optimizasyon API'nin çalışmasına ait görsel.....36
Şekil 4.22: Arka uç kavramsalına ait görsel.....37
Şekil 4.23: IntelliJ ile Spring Initialize kullanımı 38
Şekil 4.24: Arka Yüz Paket Mimarisi 39
Şekil 4.25: OAuth2 Çalışma Mimarisi.....41
Şekil 4.26: Veri tabanı Mimarisi.....43
Şekil 4.27: Kayıt Ol ekranı model çalışması.....45
Şekil 4.28: Giriş ekranı model çalışması.....46
Şekil 4.29: Harita ekranı model çalışması.....47
Şekil 4.30: Yönlendirme ekranı model çalışması 48
Şekil 4.31: Profil ekranı model çalışması 48
Şekil 4.32: Giriş ekranı 49
Şekil 4.33: Giriş ekranı boş alanlar uyarı mesajı 50
Şekil 4.34: Kayıt Ol ekranı 51
Şekil 4.35: Kayıt Ol ekranı uyarı mesajları.....52
Şekil 4.36: Başarılı kayıt mesajı.....52
Şekil 4.37: Ana Sayfa ekranları 53
Şekil 4.38: Ana Sayfa ekranları 53
Şekil 4.39: Yönlendirme adımlarının başlaması 54
Şekil 4.40: Hedefe ve Merkeze varıldığına dair uyarı ekranları 55
Şekil 4.41: Hedefe ve Merkeze varıldığına dair uyarı ekranları 55
Şekil 4.42: Profil Sayfası 55

KODLAR DİZİNİ

Kod 4.1: /complete-order/{orderId} uç noktasının implementasyonu.....41
Kod 4.2: Siparişi tamamlama işlemini gerçekleştiren metodu.....42
Kod 4.3: Siparişi tamamlamak için veri tabanı güncelleme kodu 43
Kod 4.4: “Auth” uç noktası cevabını karşılamak için oluşturulmuş model.....50
Kod 4.5: E-posta kontrolü için oluşturulan düzenli ifade (regex) kodu.....51
Kod 4.6: “User” uç noktasının cevabı için oluşturulan model.....56

1. GİRİŞ

Son yıllarda pandemi dönemiyle beraber e-ticaret sektörü hızlı bir gelişim göstermiştir. E-ticaret sektöründe meydana gelen bu gelişmeler kargoculuk sektörünü de olumlu olarak etkilemiştir. İnternet alışverişine olan talebin artmasıyla kargo ve kurye sayısı günden güne artmıştır. İnternet üzerinden verilen sipariş sayısının önemli ölçüde artmasıyla kuryelerin siparişleri zamanında ve en az maliyetle teslim etmesi önem kazanmıştır. Ülke genelinde günlük internet alışverişi sayısının 6,2 milyona ulaştığını göz önüne alındığında siparişlerin maliyet ve zamandan kazançla teslim edilmesinin önemi görülebilmektedir. Literatürde büyük online alışveriş sitelerinin bu kapsamda geliştirdikleri uygulama ve algoritmalar ile önlem almaya çalıştığı görülmektedir. Orta ve küçük işletmelerin ise hala sipariş dağıtımında yaşadığı sorunlar devam etmektedir. Bu kapsamda geliştirilmiş olan mobilite tabanlı yol uygulamasıyla orta ve küçük işletmelerin yaşamış olduğu sipariş dağıtım problemine genetik algoritma yardımıyla çözüm sağlanması hedeflenmiştir. Uygulama geliştirim çalışmaları 3 ana başlık altında gerçekleştirilmiştir. Bu başlıklar aşağıdaki şekildedir.

- Optimizasyon Tarafı Geliştirim Çalışmaları
- Arka Uç Tarafı Geliştirim Çalışmaları
- Mobil Tarafı Geliştirim

3 ana başlık altında gerçekleştirilen çalışmalar neticesinde orta ve küçük işletmelerin yaşamış olduğu sipariş dağıtım sorununa çözüm sunabilecek mobilite tabanlı bir yol optimizasyonu uygulaması başarılı bir şekilde geliştirilmiştir. Rapor içerisinde uygulama geliştirim çalışmaları her bir ana başlık için ayrıntılı olarak açıklanmıştır.

2. PROJE TANITIMI

Mobilite tabanlı yol optimizasyonu uygulaması, ülke içerisinde bulunan küçük ve orta ölçekli işletmelere hitap ederek işletmelerin sipariş teslimi sırasında yaşamış olduğu sorunları ve maddi kaybı minimize etmeyi amaçlamaktadır.

Son yıllarda teknolojinin gelişmesi ve yaşanılan pandemi dönemiyle birlikte internet alışverişine olan ilgi dünya genelinde artmaktadır. Ülkemiz bünyesinde bulunan birçok e-ticaret firması da bu gelişime ayak uydurarak gerek yurt içi gerek yurt dışında internet alışverişi organizasyonuna yönelik birçok ticari çalışma gerçekleştirmekte ve hızla büyümektedir. Ülkemiz genelinde e-ticaret üzerine gerçekleştirilen atılım ve çalışmalara yönelik verilere bakıldığında pandemi döneminden önce 50 bin olan e- ticaret yapan işletme sayısının pandemi dönemiyle birlikte 4-5 yılda beklenen büyüme oranına ulaşarak 6 kat arttığı görülmektedir. Ülke içerisinde e-ticaret yapmaya başlayan işletme sayısının artmasında vatandaşların online alışverişe olan ilgi ve güveninin artması etkili olmuştur. Bu ilgi ve güvenin artışıyla ülkemizde internet üzerinden gerçekleştirilen günlük alışveriş sayısı 6 milyonu aşmıştır ve bu sayı günden güne artmaya devam etmektedir [1].

Online alışverişe olan talebin artmasıyla beraber siparişlerin teslim edilme süresinin ayarlanması ve sipariş tesliminin en az maliyetle gerçekleştirilmesi gibi çözülmesi gereken problemler de açığa çıkmıştır. Ülkemizde online satış yapan firma sayısının 320 bini aştığı ve günlük sipariş sayısının 6 milyonu geçtiği göz önüne alınırsa sipariş teslim aşamasının müşteri memnuniyeti sağlanarak en az maliyetle gerçekleştirilmesinin işletmelere ve doğal olarak ülke ekonomisine sağlayacağı katkının küçümsenmeyecek büyüklükte olduğu görülebilmektedir.

Ülkemizde olduğu gibi yurt dışında da online alışveriş pazarı hızla büyümekte ve diğer ülkelerde bulunan işletmelerinde aynı sorunlarla karşı karşıya kaldığı görülmektedir. Yurt dışında kurulmuş olan birçok firma işletmelerin karşılaşmış olduğu sipariş teslimi sorunlarına çözüm sunmaya çalışmaktadır. Bu şirketlere Almanya da bulunan Qixxit, Kanada da bulunan Bixi, Singapur da bulunan Beeline ve dünyanın daha birçok yerinden örnek verilebilmektedir [2]. Ülkemizde e-ticaret konusunda ilerleme kat etmiş ve yurt dışı pazara da açılmış olan Getir, Trendyol, Hepsiburada gibi milyar dolar değerlemeyi aşmış olan şirketlerin bünyesinde bulunan işletmelere bu konuda destek sağladığı görülmektedir. Fakat ülkemiz çapında kendi e-ticaret sitesi üzerinden satış gerçekleştiren, bünyesinde kurye yapısı bulunduran ve sipariş dağıtım konusunda aynı sorunu yaşayan birçok küçük ve orta ölçekli işletme bulunmaktadır.

Projenin çıkış noktasını ve hedeflediği kitleyi yukarıda belirtilen küçük ve orta ölçekli işletmeler oluşturmaktadır. Geliştirilmiş olan mobilite tabanlı yol optimizasyonu uygulamasıyla kuryeler tarafından dağıtımı gerçekleştirilecek olan siparişlerin hangi kurye tarafından hangi rotayla

dağıtılacağının belirlenmesi görevini işletme sahibi ve kurye çalışanı üzerinden alarak bu işlemin tamamen uygulama üzerinden yapılması hedeflenmiştir. Bu şekilde sipariş dağıtımı rotasının kişiler tarafından belirlenmesinden doğacak olan zaman ve maddi kayıpların önüne geçilerek işletmelerin sipariş dağıtımında yaşamış olduğu maddi kayıpların minimize edilmesi amaçlanmıştır. Bu maddi ve zaman kaybının azaltılmasıyla işletme tarafına olan müşteri memnuniyetinin pozitif yönde etkileneceği görülmektedir. Uygulamanın kullanımının yaygınlaşmasıyla beraber birçok işletme bünyesinde sağlanacak olan zaman ve yakıt tasarrufunun ülke ekonomisine önemli ölçüde katkı sunması beklenmektedir.

Proje çalışmaları üç ana başlık altında gerçekleştirilmiştir. Bu başlıklar şu şekildedir.

2.1. Optimizasyon Tarafı Geliştirim Çalışmaları

Bu başlık altında kuryelere ait sipariş dağıtımının gerçekleştirilmesi ve rotaların belirlenmesi için literatürde bulunan "Araç Rotalama Problemi" üzerine gerçekleştirilen çalışmalar incelenerek bu probleme yönelik Genetik Algoritma kullanılarak optimizasyon çalışması gerçekleştirilmiştir. Optimizasyon tarafının diğer alanlar ile haberleşmesi geliştirimiş olan Python Flask API ile sağlanmıştır. Optimizasyon tarafı çalışmalarında Python programlama dili kullanılmıştır. Gerçekleştirilen çalışmalara ait ayrıntılı bilgi Bölüm 3.1'de verilmiştir.

2.2. Arka Uç Tarafı Geliştirim Çalışmaları

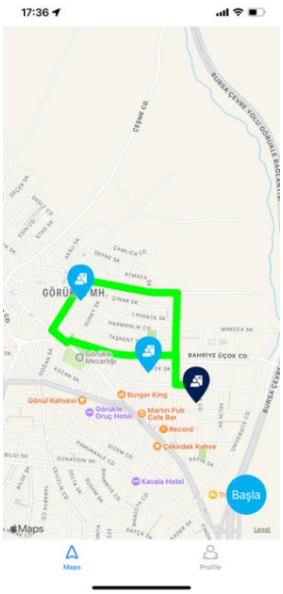
Bu başlık altında uygulamanın ana merkezi olan arka uç geliştirim çalışmaları gerçekleştirilmiştir. Uygulama ait veri tabanı kontrol ve veri yönlendirme işlemleri bu bölümde yapılmaktadır. Kurye ve sipariş bilgilerinin optimizasyon tarafına iletilmesi ve optimizasyon tarafından alınan sonuçların veri tabanına kaydedilerek uygun şekilde mobil tarafa sunulması, kurye personel bilgilerinin saklanması ve kontrolü vb. birçok kontrol ve veri iletimi işlemleri burada yapılmaktadır. Arka uç tarafı çalışmalarında Java programlama dili kullanılmıştır. Gerçekleştirilen çalışmalara ait ayrıntılı bilgi Bölüm 3.2'de verilmiştir.

2.3. Mobil Tarafı Geliştirim Çalışmaları

Bu başlık altında kurye personellerine yönelik rota ve adres bilgilerinin sunulacağı mobil uygulama çalışmaları gerçekleştirilmiştir. Her bir kuryenin bırakması gereken sipariş adresleri ve rotası mobil uygulama tarafından kurye personeline sağlanmaktadır. Kurye personeli mobil uygulama üzerinden kullanıcı adı ve şifresiyle giriş yaparak bırakması gereken siparişlere ait rota bilgisine harita üzerinden ulaşabilmektedir. Arka uç tarafından API ile sağlanan rota bilgisinin harita üzerinde görselleştirilmesi, kurye personeline gerekli olan bilgilerin kullanıcı dostu bir ara yüz ile sunulması, kullanıcı giriş ve kaydolma ekranının geliştirilmesi vb. birçok işlem bu tarafta yapılmıştır. Mobil tarafı geliştirim çalışmalarında Swift programlama dili kullanılmıştır. Gerçekleştirilen çalışmalara ait ayrıntılı bilgi Bölüm 3.3'te verilmiştir.

Üç ana başlık altında gerçekleştirilen çalışmalar neticesinde küçük ve orta ölçekli işletmelerin yaşamış olduğu sipariş dağıtım sorununa çözüm sunabilecek mobilite tabanlı bir yol optimizasyonu uygulaması başarılı bir şekilde geliştirilmiştir. Uygulama logosu ve mobil uygulama ana ekranına ait görseller Şekil 2.1. ve Şekil 2.2'de verilmektedir.

Şekil 2.1: Uygulama logosuna ait görsel



Şekil 2.2: Uygulama mobil ana ekranına ait görsel

1. KULLANILAN TEKNOLOJİLER

Uygulamanın ayrılmış olduğu her alt başlık için farklı programlama dili, kütüphaneler ve teknolojiler kullanılmıştır. Bu teknolojilerin seçiminde başlık altında gerçekleştirilmesi planlanan görev parçası etkili olmuştur. Her bir alt başlık için yapılan araştırma ve çalışmalar neticesinde kullanılması belirlenen programlama dilleri şu şekildedir;

- **Optimizasyon:** Python
- **Arka Uç Geliştirme:** Java
- **Mobil Geliştirme:** Swift

Belirlenen programlama dilleriyle beraber kullanılan teknolojilere ait ayrıntılı bilgi aşağıda verilmektedir.

3.1. Optimizasyon Tarafında Kullanılan Teknolojiler:

3.1.1 Genetik algoritma

Genetik Algoritma, canlılardaki genetik sürece benzer mekanizmalar kullanarak çalışan bir eniyileştirme yöntemidir. Çok boyutlu uzayda belirli bir maliyet fonksiyonuna göre en iyileştirme amacıyla iterasyonlar yapan ve her iterasyonda en iyi sonucu üreten kromozomun hayatta

kalması prensibine dayanan en iyi çözümü arama yöntemidir [3]. Genetik algoritma, NP-Hard yani çözümü imkânsız görünen problemlere yönelik üretilmiş olan bir meta sezgisel algoritmadır. Bu algoritma problem çözümü üretebilmek için evrimsel süreci bilgisayar ortamında taklit etmektedir.

Genetik algoritmaların çalışma prensibi ise şu şekildedir; doğal yaşamda popülasyonlar bireylerin bir arada bulunmasıyla oluşmaktadır. Genetik algoritma için oluşturulan popülasyonda çok sayıda bireyin bir araya gelmesiyle, başka bir deyişle çok sayıda olası çözüm adaylarının bir araya gelmesiyle oluşmaktadır. Bireylerin yani çözüm adaylarının uygunluk fonksiyonu işlemine tabi tutulması sonucunda, çözümün optimal çözüme ne kadar yaklaştığını değerlendiren uygunluk değeri belirlenmektedir [4]. Seçim işlemi, yeni popülasyonda çocukların oluşturulması için bir önceki popülasyon içerisinden ebeveyn seçilmesidir. Çaprazlama operatörü, seçim işleminden sonra uygulanır ve ebeveyn bireylere ait kromozomların belirli kısımlarının karşılıklı yer değiştirmesini ve böylece yeni özellikte bireylerin oluşmasını ifade eder [5]. Mutasyon işlemi ise yeni oluşan bireyin kromozomlarından herhangi birinin içindeki bir geni mutasyon olasılığına bağlı olarak değiştirme işlemidir. Genetik algoritmanın sonlanması için kullanılan farklı yöntemler mevcuttur. Bu yöntemler; algoritmanın çalışması esnasında istenen çözüm bulunduğunda, genetik algoritmanın başlangıcında tanımlanan toplam adım(iterasyon) sayısına ulaşıldığında veya uygunluk değeri sürekli olarak sabit kaldığında bulunan en iyi bireyin temsil ettiği çözüm, problem için bulunmuş en uygun çözüm olarak sunulmaktadır [6]. Genetik alanında yer alan bazı kavramlar algoritma içerisinde modellenerek kullanılmaktadır. Bu kavramlar şu şekildedir;

- **Gen:** Kendi başına anlamlı genetik bilgi taşıyan en küçük yapıdır.

- **Kromozom (Birey):** Gen yapılarının birleşimiyle oluşmaktadır. Genetik algoritmada bir kromozom problemin olası çözümlerinden birini ifade etmektedir.

- **Populayon (Nesil):** Kromozomlar kümesidir. Genetik algoritmada problemin olası çözümlerinin bir arada bulunduğu yapıdır. Algoritmada işletilen her jenerasyonda popülasyon büyüklüğü aynıdır.

- **Elitizm (Seçim):** Belirli kromozomların bir sonraki jenerasyona aktarılmasıdır. Genetik algoritmada amaç fonksiyonu değerine göre belirlenen, şu an ki jenerasyonda bulunan en iyi çözümlerin yani kromozomların bir sonraki jenerasyona değişmeden aktarılmasıdır.

- **Crossing Over (Çaprazlama):** Populasyon içerisinden rastgele seçilen iki bireyin bir veya birden çok geninin yer değiştirilmesi ile iki yeni bireyin oluşmasıdır. Bu işlem sonucunda popülasyon içerisinde olmayan yeni kromozom yapıları(birey) oluşmaktadır. Genetik algoritmada yeni çözümler oluşturmak için kullanılmaktadır.

- **Mutasyon:** Bir bireyin bir veya birden çok geninin değişmesiyle farklı bir birey haline gelmesidir. Genetik algoritmada yeni çözümler oluşturmak için kullanılmaktadır.

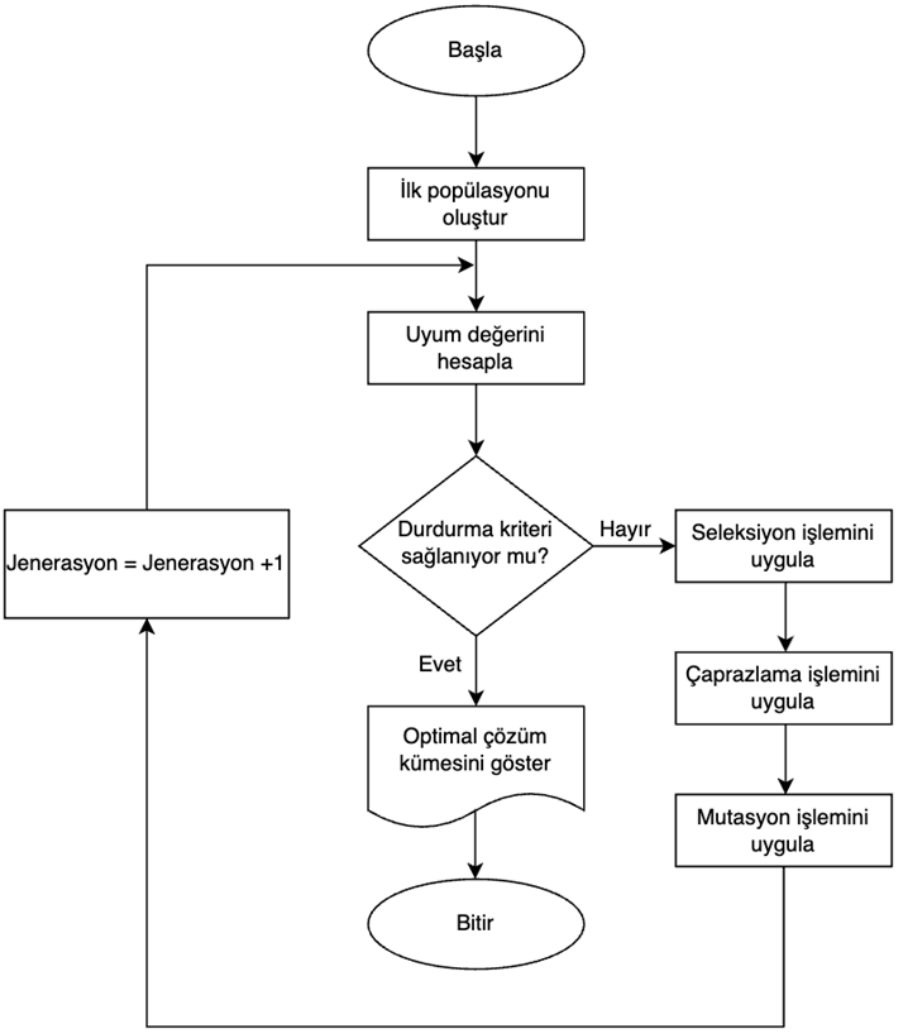
- **Amaç Fonksiyonu:** Problem çözümüne göre geliştirilen, bireyin hedeflenen sonuç değerine ne kadar yakınsadığını yani ne kadar sağlıklı olduğunu belirleyen fonksiyondur. Bir sonraki jenerasyona aktarılacak olan bireyler bu fonksiyon kullanılarak belirlenmektedir.

Genetik algoritmanın çözülmesi imkânsız olarak görülen problemlere uygulanmasında problem için hedeflenen sonuca bağlı olarak izlenen uygulama adımları mevcuttur. Problemin çözümüne yönelik algoritmanın geliştirilmesinde izlenen adımlar şu şekildedir.

1. Problem yapısı esas alınarak populasyon genişliği kadar rastgele kromozom yapıları oluşturulmaktadır. Oluşturulan her bir kromozom problem için bir olası çözüm kümesidir.
2. Problem yapısı esas alınarak oluşturulan amaç fonksiyonu kullanılarak populasyon içerisinde yer alan her bireyin uygunluk değeri hesaplanmaktadır. Uygunluk değeri en iyi olan birey durdurma kriterini sağlıyorsa algoritma sonlanmaktadır.
3. Durdurma kriteri sağlanmıyorsa şu an ki populasyon içerisinden rastgele seçilen bireylerin çaprazlanmasıyla yeni bireyler oluşturulmaktadır. Oluşturulan bireyler üzerinde mutasyon işlemi uygulanarak çeşitlilik artırılmaktadır.
4. Bir önceki jenerasyonda uygunluk değeri en iyi olan bireyler ile çaprazlama ve mutasyon işlemleri sonucunda üretilen yeni bireyler birleştirilerek şu an ki jenerasyon için yeni populasyon oluşturma işlemi tamamlanmaktadır.

5. Aynı işlemler durdurma kriteri sağlanana veya belirlenen jenerasyon sayısına ulaşılana kadar devam etmektedir. Algoritma sonlandıktan sonra amaç fonksiyonu yardımıyla hesaplanan uygunluk değerine göre şu ana kadar bulunmuş olan en iyi çözüm elde edilmektedir.

Genetik algoritmaya ait akış diyagramı Şekil 3.1'de verilmektedir;



Şekil 3.1: Genetik algoritma akış diyagramı [7]

Kullanım Nedeni: Literatürde yapılan çalışmalar incelendiğinde çözümü NP-Hard görülen birçok problem için genetik algoritma kullanılarak gerçekleştirilen çalışmalar mevcuttur. Literatürdeki çalışmalardan yola çıkılarak meta sezgisel algoritmalar içerisinden genetik

algoritmanın çözümü imkânsız olarak görülen problemlerden biri olan araç rotalama problemi için kullanılması kararlaştırılmıştır. Bu şekilde algoritmaya ait geliştirme ve parametre belirleme çalışmalarının, genetik algoritmanın sahip olmuş olduğu geniş araştırma geçmişinden faydalanılarak, problemin çözümüne yönelik optimum çalışacak şekilde gerçekleştirilmesi hedeflenmiştir.

3.1.2 Python programlama dili

Python, açık kaynak kodlu, nesne yönelimli özellik gösteren, üst düzey bir programlama dilidir. Söz dizimi insan mantığına çok yakın olduğundan "yüksek seviyeli" bir dildir. Yazımı ve anlaşılması kolaydır. Bulundurmuş olduğu geniş kütüphane yelpazesiyle birçok alanda kullanılmaktadır. Bu alanlara veri bilimi, web geliştirme, makine öğrenimi vb. alanlar örnek verilebilmektedir. Geniş kütüphane yelpazesi sayesinde akademik çalışmalarda da kullanımı en fazla tercih edilen dildir.

Kullanım Nedeni: Yapılan literatür taraması ve araştırmalar neticesinde optimizasyon çalışması gerçekleştirilecek araç rotalama problemi için;

- Açık kaynak harita (OpenStreetMap) üzerinden bir bölgeye ait yol bilgisinin alınabilmesi,
- Adres koordinatları üzerinde yapılan işlemlerin gelişmiş liste yapısı kullanılarak efektif ve hızlı bir şekilde gerçekleştirilebilmesi,
- Genetik algoritma kodunun anlaşılır ve daha az kodlamayla yazılabilmesi,
- Genetik algoritma hiperparametre optimizasyonu çalışmalarında elde edilen sonuçların görselleştirilebilmesi ve optimum parametrelerin daha doğru bir şekilde belirlenebilmesi,
- Optimizasyon tarafı RESTful API çalışmaların gerekli ihtiyacı karşılayacak şekilde geliştirilebilmesi vb.

gerekliliklerin Python programlama dili içerisinde bulunan kütüphaneler yardımıyla daha efektif, hızlı ve kolay bir şekilde karşılanabileceği kararına varılmıştır.

3.1.3 OpenStreetMap haritası ve OSMnx kütüphanesi

*OpenStreetMap (OSM)***,** açık kaynak kodlu bir harita oluşturma projesidir. Katılımcıların gönüllü olarak katkıda bulunduğu bir projedir. Haritalar kullanıcıların sunmuş olduğu GPS izleri, uydu fotoğrafları ve yerel yönetimler tarafından sağlanan çeşitli veriler doğrultusunda oluşturulmaktadır [8].

OSM ana sayfa linki: https://www.openstreetmap.org/

OSMnx, OSM içerisinden herhangi bir şehir, ilçe veya bölgeye ait cadde ve sokak yol bilgilerinin graf yapısı kullanılarak kod içerisine dahil edilmesine ve üzerinde işlemler gerçekleştirilmesine olanak sağlayan bir Python kütüphanesidir.

OSMnx doküman linki: https://geoffboeing.com/osmnx-python-street-networks/

Kullanım Nedeni: Uygulama çalışma alanı olarak belirlenen Bursa Nilüfer ilçesi Görükle Mahallesine ait sokak ve cadde yol bilgilerinin alınması ve kod içerisinde kullanılabilir hale getirilmesi gerekmektedir. OSM açık kaynaklı ve ücretsiz bir harita olduğundan istenilen bölgeye ait yol bilgilerinin alınması için tercih edilmiştir. OSMnx ise Python içerisinde OSM üzerinden alınan yol bilgilerine ait graf yapısının kullanımı için geliştirilmiş bir kütüphane olduğundan kullanılmıştır.

3.1.4 Flask kütüphanesi

Flask, bir web mikro çerçevesidir. Tamamen Python diliyle geliştirilmiştir. Arka uç geliştirme için içerisine birçok hazır modül bulunmaktadır. Flask kullanarak REST mimaride API geliştirmek oldukça basit ve kolaydır.

Kullanım Nedeni: Geliştirilen optimizasyon algoritması arka uç tarafından yollanan koordinat bilgilerini girdi olarak almakta ve bu bilgilere yönelik bir çıktı değeri üretmektedir. Üretilen çıktı değerinin arka uca gönderilmesi gerekmektedir. Bu işlem için bir API yapısı oluşturulmuştur. Bu yapının oluşturulması için Flask mikro çerçevesinden yararlanılmıştır.

3.2. Arka Uç Geliştirme Tarafında Kullanılan Teknolojiler:

3.2.1 Java programlama dili

Şekil 3.2: Java Programlama Dili Logosu [9]

Bu hizmetleri sağlamak adına arka-yüz uygulamasının Java programlama dili ile geliştirilmesi uygun görüldü. Java programlama dili arka-yüz geliştirmesinde kullanılan en popüler programlama dillerindendir. WORA (Write once, run anywhere / bir kere yaz, her yerde çalıştır) sloganını 1995 yılında ortaya atan Sun Microsystems, bu slogan ile, kodu bir kere yazdığınız fakat Java Runtime Environment (JRE) bulunan bütün makinelerde tekrardan derlemeden çalıştırabildiğiniz bir programlama dili olan Java'yı ortaya attı.

Oracle'ın satın alması ile Java, sürekli kendini geliştiren, düzenli olarak güncelleme alan (son yıllarda 6 ayda bir güncelleme alır) bir programlama dili haline geldi. Büyük bir topluluğa sahip olması yeni Java geliştiricileri için büyük bir avantajdır. İki yılı aşkın süredir Java ekosistemi birçok açık kaynak kodlu kütüphane ile desteklenmektedir.

Bütün bunlarında yanında Java, C programlama dilinin sözdizimini örnek aldığı için öğrenilmesi kolaydır, nesne yönelimli bir programlama dilidir, çoklu çekirdek kullanımını destekler, kendi koşturma ortamında çalıştığı için (JRE) güvenlidir [10].

Arka-yüz servisini geliştirmek için Java programlama dilinin bir framework'ü (yazılım iskeleti) kullanmak gerekmektedir. Çünkü geliştirme işini hiçbir framework ya da kütüphane kullanmadan yapmak çok büyük gereksiz efor gerektirir. Bunun yerine uygun bir framework seçerek yazılım geliştirmeyi hızlı, dinamik ve güvenilir hale getirmek amaçlandı.

3.2.2 Spring framework

Şekil 3.3: Spring Framework Logosu [11]

Framework seçimi Spring Framework'den yana yapıldı. Çünkü Spring Java geliştirmesini hızlı kolay ve güvenli yapar. İçinde birçok farklı framework ve kütüphane bulunan Spring, hız, basitlik ve üretkenlik üzerine odaklanır ve bu sayede düyanın en popüler Java framework'üdür.
Örneğin bu projede, Spring framework'ü hızlı bir şekilde inşa etmek adına Spring boot, Veritabanı bağlantılarını sağlamak ve veritabanı operasyonlarını gerçekleştirmek adına Spring Data, Uygulama güvenliği, kimlik doğrulama, yetkilendirme sağlamak adına Spring Security ve birçok farklı Spring kütüphanesi kullanılmıştır.

3.2.3 Spring framework inversion of control (IOC) prensibi

Spring Framework'ün en önemli iki özelliğinden biri Inversion Of Control'dür (IOC) (kontrolün tersine çevrilmesi), IOC, nesnelerin kontrolünü bir framework'e sağlamaktır. Yazılım mühendisliğinin temel prensiplerindendir ve genelde nesne yönelimli programlamada kullanılır [12].

Farklı uygulamalar arasında geçişi kolaylaştırır.Modülerlik sağlar. Biz uygulamamızda Spring Bean'leri kullanarak, sınıflarımızdan birer örnek oluşturmadan ve nullPointerException hatası almadan sınıflarımızın statik olmayan metodlarını kullanabiliriz. Bütun bunları Spring framework'ün IOC özelliğine borçluyuz.

3.2.4. Dependency injection

Dependency Injection kullanmak için değişkenlerimizi @Autowired anotasyonu ile ya da constructor kullanarak ya da getter ve setter'ları kullanarak gereklilikleri bağlamalıyız. Bu sayede Spring Framework'ün IOC özelliğini kullanabiliriz [12].

3.2.5 Spring boot

Şekil 3.4: Spring Boot Logosu [11]

Spring boot, Spring Configürasyonları ile zaman kaybetmeden direk uygulamayı ayağa kaldırmamızı sağlar. Gerekli konfigürasyonları kendisi yapar. Dilediğimiz Spring ekosistemi framework'ünün hangi versiyonunu kullanacağımızı düşünmemize gerek kalmaz. Sadece hangi spring boot versiyonunu kullanacağımıza karar vermemizle otomatik olarak diğer kütüphanelerin uyumlu versiyonlarını bize sağlar. Spring Data sayesinde, ilişkisel, ilişkisel olmayan, bulut tabanlı veritabanlarını rahat bir şekilde kullanabiliriz. Spring Data içerisinde her bir veritabanı için farklı kütüphaneler bulundurur. Bu sayede birçok veritabanı ile kolayca çalışılabilir kılar.

3.2.4 Spring data

Şekil 3.5: Spring Data Logosu [11]

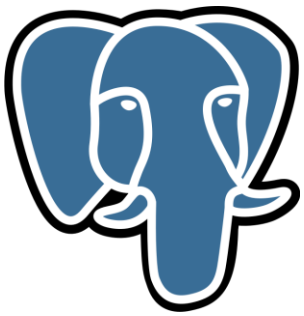
Spring Data sayesinde, ilişkisel, ilişkisel olmayan, bulut tabanlı veritabanlarını rahat bir şekilde kullanabiliriz. Spring Data içerisinde her bir veritabanı için farklı kütüphaneler bulundurur. Bu sayede birçok veritabanı ile kolayca çalışılabilir kılar.

3.2.5 Spring security

Şekil 3.6: Spring Security Logosu [11]

Spring Security, güçlü ve oldukça konfigüre edilebilir kimlik doğrulama ve erişim kontrol framework'üdür. Spring Security, Java uygulamalarına hem kimlik doğrulama hem de yetkilendirme özellikleri sunmaya odaklanır.

3.2.6 PostgreSQL



Şekil 3.7: PostgreSQL Logosu [13]

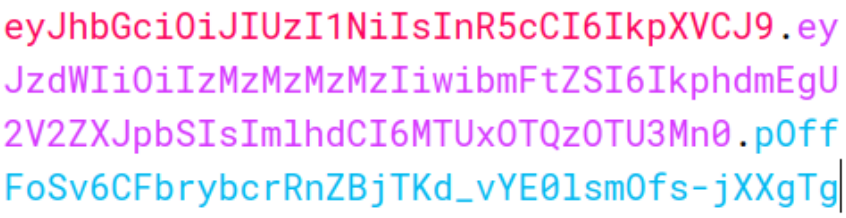
PostgreSQL, MySQL'den sonra en popüler ilişkisel veritabanıdır. Oldukça ölçeklendirilebilirdir. JSON, XML, H-Store gibi özel veritiplerini varsayılan olarak destekler. Üçüncü parti araçlar kolayca entegre edilebilir. Açık kaynak kodlu ve topluluk tarafından çokça desteklenir. Bu ve bu tip özellikleri PostgreSQL'i seçmemize neden olmuştur.

3.2.7 Project lombok

Lombok Java programcılarını her bir sınıfa yazmak zorunda olduktarı basmakalıp(boilerplate) kodlardan kurtarır. Getter, setter, toString, constructor, gibi metodları teker teker yazmak yerine Lombok ile kolayca oluşturabiliriz [14].

3.2.8 JWT (JSON Web Token)

JWT güvenli bir şekilde JSON objesi şeklinde veri transferi yapmayı sağlayan bir standarttır. Veri dijital olarak imzalı olduğu için güvenilirdir. HMAC algoritması ile özel anahtar kullanılarak dijital imzalanabilir. JWT kimlik doğrulama işlemlerinde sık sık kullanılır. Bir JWT sırasıyla başlık (header), yük (payload) ve imza(signature) kısımlarından oluşur. Başlık alanı kullanılan şifreleme algoritmasının tipini ve token'in tipini içerir. Yük alanı claim adı verilen jwt içerisindeki varlık (genelde kullanıcı olur) ile ilgili bilgiler içerir. İmza alanı ise JWT verisini korumak adına kullanılan imzayı içerir [15]. Kırmızı alan başlık(header) kısmını, mor alan yük(payload) kısmını, mavi olan kısım ise imza (signature) kısmını içerir.



Şekil 3.8: Örnek bir JWT

Kırmızı alan başlık(header) kısmını, mor alan yük(payload) kısmını, mavi olan kısım ise imza (signature) kısmını içerir.

3.2.9 Swagger UI

Şekil 3.9: Swagger UI Logosu [16]

Swagger, web servislerinin sağladıkları endpoint'lerin (uç nokta) tek seferde dökümante edilmesi ve bu doküman üzerinden manuel olarak test edilebilmesine imkân tanır. Bu projede arka-yüz uygulamasına swagger ui kütüphanesi eklenerek, diğer geliştiricilerin de arka-yüz servisini kolayca kullanabilmesine olanak tanınmıştır.

3.3. Mobil Uygulama Geliştirme Tarafında Kullanılan Teknolojiler:

3.3.1 Swift programlama dili

Swift, güvenlik, performans ve yazılım tasarım modellerine modern bir yaklaşım kullanılarak oluşturulmuş genel amaçlı bir programlama dilidir. Esas olarak iOS, macOS, iPadOS, watchOS gibi Apple ekosistemi ürünlerinin kullandığı işletim sistemlerine uygulama geliştirme için kullanılmaktadır. Swift ücretsiz ve açık bir programlama dilidir ve Apache 2.0 açık kaynak lisansı altında geliştiriciler, eğitimciler ve öğrenciler tarafından kullanılabilmektedir. Swift, öğrenmesi ve kullanması kolay olacak şekilde tasarlanmıştır.

Kullanım Nedeni: İyi bir mobil uygulama;

- Hızlı

- Güvenli
- Kullanıcı dostu
- Kaynak tüketimi konusunda cimri
- Kullanımı ve geliştirmesi kolay

gibi özelliklere sahip olmalıdır. Swift programlama dili geliştirme amacına uygun olarak bu özellikleri sağlamaktadır. Ayrıca geliştirilecek olan uygulama Apple ekosistemine ait iOS işletim sistemi kullanan mobil cihazlar üzerinde çalışacağı için Swift programlama dili kullanılmasına karar verilmiştir.

3.3.2 Alamofire kütüphanesi

Alamofire, Swift tabanlı bir HTTP ağ kütüphanesidir. Ortak ağ görevlerini basitleştiren "Apple Foundation" ağ sınıfının üzerinde kolay bir ara yüz ve kullanım sağlar. Özellikleri arasında istek/yanıt yöntemleri, JSON ve "Codable" kod çözme, kimlik doğrulama ve daha fazlası bulunur.

Kullanım Nedeni: Alamofire kütüphanesi uygulama içerisinde;

- RESTful API'sinden veri isteme.
- İstek parametreleri gönderme.
- Yanıtı JSON'a dönüştürme.
- Yanıtı, "Codable" protokolü aracılığıyla bir Swift veri modeline geri dönüştürme.

İşlemleri için kullanlmıştır.

3.3.3 KeyChainAccess kütüphanesi

KeychainAccess, iOS ve macOS işletim sistemlerinde çalışan Anahtar Zinciri uygulamasına erişim için kullanılan bir Swift kütüphanesidir. Keychain, Anahtar Zinciri uygulamasının özelliklerinin kullanımını Swift'de son derece kolay hale getirir.

Apple geliştiricileri için en önemli güvenlik unsurlarından biri, meta verileri ve hassas bilgileri depolamak için özel bir veritabanı olan iOS Anahtar Zinciridir. Anahtar Zinciri'ni kullanmak, token ve şifreler gibi uygulama için kritik olan küçük veri parçalarını depolamanın en iyi yoludur.

Özellikle Swift'de Anahtarlık ile doğrudan etkileşim kurmak karmaşıktır. Çoğunlukla C ile yazılmış olan Güvenlik çerçevesinin kullanılması gerekir.

Kullanım Nedeni: KeychainAccess kütüphanesi API'ler ile elde edilen hem genel hem de internet şifrelerinin ve tokenlerinin işletim sistemi veritabanında güvenli bir şekilde eklenebilmesi, değiştirilebilmesi, silinebilmesi için kullanılmıştır.

3.3.4 PromisedFuture kütüphanesi

PromisedFuture, Futures/Promises çerçevelerinin birer türevidir. PromisedFuture, okunabilir ve anlaşılabilir asenkron kod yazmaya yardımcı olur.

Kullanım Nedeni: Asenkron görevlerle çalışırken genellikle geri arama (callback) mekanizması kullanılır. Ancak genellikle birden fazla asenkron işlem gerçekleştirilmesi gerekir, bu nedenle ikinci işlem ilkinin tamamlama bloğunun içine yerleştirilir, ancak iç içe geri aramalar yapıldığında kod dağınık bir hal almaya başlar ve sürdürülebilirlik, okunabilirlik ve kontrol açısından karmaşık hale gelir. PromisedFuture Kütüphanesi, kodun okunabilirliğini ve sürdürülebilirliğini artırmak adına asenkron işlemlerin daha efektif yazılabilmesine olanak sağlamaktadır. Proje içerisinde de bu amaçla kullanılmıştır.

3.3.5 Lottie kütüphanesi

Lottie, vektör tabanlı animasyonları gerçek zamanlı olarak işleyen Android ve iOS için oluşturulmuş bir kütüphanedir. JSON formatında elde edilmiş animasyonların, Android ve iOS ortamında yerel olarak minimum kod ve kaynak tüketimi ile kullanılabilmesi sağlar.

Kullanım Nedeni: Uygulama içerisinde minimum kod ve maksimum kaynak tasarrufu ile animasyonların gösterilebilmesi için kullanılmıştır.

3.3.6 MVC mimarisi

MVC, üç ana nesneden oluşan bir yazılım geliştirme modelidir: Model, Görünüm ve Denetleyici.

Model Katmanı: Model katmanı, uygulamanın iş mantığından sorumludur. Uygulama durumunu yönetir. Bu aynı zamanda veri okuma ve yazma, kalıcı uygulama durumunu ve hatta ağ oluşturma ve veri doğrulama gibi veri yönetimi ile ilgili görevleri içerebilmektedir. Uygulama

içerisindeki Utils, Storage ve Networking sınıfları da bu katmanı temsil eder. Doğrudan Görünüm katmanı ile iletişim kurmamalı ve kullanıcı ara yüzü ile ilgilenmemelidir.

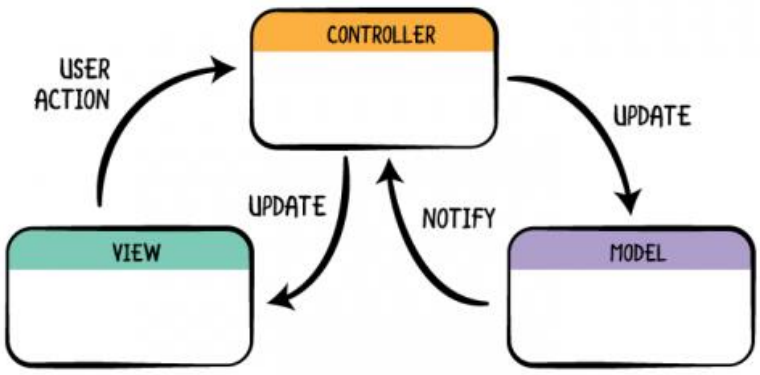
Görünüm Katmanı: Görünüm katmanının, kullanıcıya veri sunma ve kullanıcı etkileşimini yönetme olmak üzere iki önemli görevi vardır.

MVC modelinin temel ilkesi, görünüm katmanının model katmanına göre bilgi içermemesidir. Görünüm katmanı yalnızca verileri kullanıcıya nasıl sunacağı üzerinde çalışmaktadır. Ne sunduğunu bilmez veya anlamlandırmaz. Görünüm, kullanıcıların ekranda gördüklerinin bileşenlerini temsil eder. Temel Apple çerçeveleri olan UIKit ve AppKit bu katmanı temsil etmektedir.

Denetleyici: Bu katman, MVC'nin çekirdek katmanıdır. Görünüm ve model güncellemeleriyle ilgilenir ve Görünümü ve Modeli değiştirir/günceller. Denetleyiciler, diğer nesnelerin yaşam döngülerini de yönetmektedir.

Görünüm katmanı ve model katmanı, bir veya daha fazla denetleyici tarafından birbirine bağlanır. Örneğin iOS ve tvOS uygulamalarında bu bağlayıcı görünüm denetleyicisi, UIViewController sınıfının bir örneği veya bunun bir alt sınıfıdır.

MVC mimarisinin modellemesi şekil 3.10'daki gibidir.



Şekil 3.10: MVC Mimarisi Modeli [17].

Kullanım Nedeni: MVC kütüphanesinin kullanımı Apple tarafından tavsiye edilir ve yerel olarak sunulan UIKit ile desteklenmektedir. MVC mimarisi çoğu geliştirici tarafından bilinmekte hakkında çok fazla doküman bulunmaktadır. Ayrıca MVC mimarisi ile diğer mimarilerin

sunduğu okunabilirlik, erişilebilirlik, test edilebilirlik özelliklerinden ödün vermeden daha hızlı geliştirme yapılabilmektedir. Bu nedenlerden dolayı uygulamamızda MVC mimarisi kullanılmıştır.

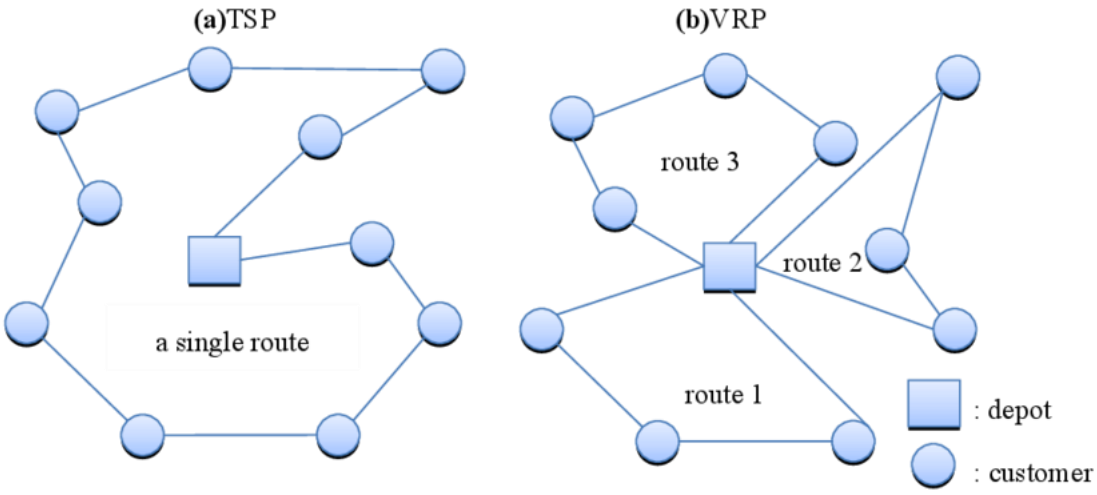
2. YÖNTEM VE ÇALIŞMALAR

4.1. Optimizasyon Çalışmaları

Bu başlık altında gerçekleştirilen çalışmalar neticesinde arka uç tarafından sağlanan sipariş ait koordinat bilgileri ve kurye bilgileri (filo sayısı, kapasite) kullanılarak siparişlerin teslim edilmesinde ortaya çıkan ve literatürde çözümü imkânsız olarak kabul edilen araç rotalama problemine meta sezgisel algoritma olan genetik algoritma kullanılarak çözüm sunulması amaçlanmıştır. Bu amaç doğrultusunda geliştirilen genetik algoritmayla kuryeler tarafından dağıtıma çıkarılacak olan siparişlerin en az yol maliyetiyle dağıtılması sağlanmıştır. Bu aşamada gerçekleştirilen çalışmalara ait ayrıntılı bilgi aşağıda alt başlıklar halinde verilmektedir.

2.1.1 Gereksinim analizi

Araç Rotalama Problemi (VRP), bir veya birkaç depodan müşterilere hizmet götürecek araçlar için en uygun rotaları belirlemeyi amaçlayan bir kombinatoryal eniyileme (optimizasyon) problemidir. Eniyileme literatürünün en iyi bilinen problemlerinden Gezgin Satıcı Problemi'nin (TSP) daha genel bir halidir [18]. Problemlere ait görsel Şekil 4.1'de verilmektedir.



Şekil 4.1: TSP ve VRP problemine ait görsel [19]

Bu iki problemde literatürde çözümü imkânsız anlamına gelen NP-Hard problem olarak sınıflandırılmaktadır. Bu durumdan dolayı problemlerin çözümü için meta sezgisel yaklaşımlar kullanılmaktadır. Bu yaklaşımla beraber çözümü imkânsız olan bu iki problem için sezgisel algoritmalar kullanılarak optimum çözüme yakın çözümler elde edilmektedir. Gün içinde kurye firmaları bu problemlerle birçok durumda karşılaşmaktadır. Literatürde yapılan çalışmalar incelendiğinde VRP ve TSP problemi için birçok çalışma gerçekleştirilmiş olsa da bu problem çözümünün günlük hayatta (kurye firmaları vb.) karşılaşılan problemlere çözüm olacak şekilde kullanılmasına yönelik pek fazla örnek bulunmamaktadır. Bu durumdan yola çıkılarak geliştirilecek olan bir meta sezgisel algoritmayla kurye firmaları tarafından karşılaşılan problemlere bir çözüm sağlanarak meta sezgisel yöntemlerin getirdiği artılardan sektör bazlı yararlanılması amaçlanmıştır. Literatürde yapılan VRP ve TSP problemlerine yönelik meta sezgisel çalışmalar incelenerek genetik algoritmanın problem çözümü için kullanılması kararlaştırılmıştır.

4.1.2 Genetik algoritma implementasyon yöntemi

VRP problemi TSP'nin daha genel bir hali olduğu için genetik algoritma TSP problemine cevap verebilecek şekilde geliştirilmektedir. Çünkü VRP temelinde birçok TSP problemi çözümü bulunmaktadır. Bu maddeden yola çıkarak genetik algoritmanın TSP problemine uygulanma yöntemini açıklayan temel örnek aşamaları aşağıda verilmektedir. Uygulama yönteminin örneklendirilmesi için 10 adet adres bilgisine sahip örneklem kullanılmıştır. Örneklem verisine ait görsel Şekil 4.2'de verilmektedir.

Adres Numarası	X	Y
1	565.0	575.0
2	25.0	185.0
3	345.0	750.0
4	945.0	685.0
5	845.0	655.0
6	880.0	660.0
7	25.0	230.0
8	525.0	1000.0
9	580.0	1175.0
10	650.0	1130.0

Şekil 4.2: Örneklem veri setine ait görsel

1. Adreslere ait koordinat bilgisinin alınması ve uzaklık matrisinin oluşturulması

Genetik algoritma üzerinde problem çözümü inşa edilebilmesi için adres bilgilerinden yola çıkılarak oluşturulan bir uzaklık matrisi gerekmektedir. Uzaklık matrisi için adresler arasındaki mesafe "Öklid" uzaklık formülü kullanılarak hesaplanmaktadır. "Manhattan" ve "Minkowski" uzaklık formülleri de bu aşamada kullanılabilmektedir. "Öklid" uzaklık formülü Şekil 4.3'te verilmektedir.

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Şekil 4.3: İki boyutlu "Öklid" uzaklık formülü [20]

Şekil 4.2'de verilen örneklem veri setine ait "Öklid" uzaklık formülü kullanılarak hesaplanan uzaklık matrisi Şekil 4.4'te verilmektedir.

	1	2	3	4	5	6	7	8	9	10
1	0.000000	666.108099	281.113856	395.600809	291.204396	326.266762	640.800281	426.878203	600.187471	561.471282
2	666.108099	0.000000	649.326574	1047.091209	945.145491	978.084863	45.000000	956.151139	1134.955946	1132.982789
3	281.113856	649.326574	0.000000	603.510563	508.944987	542.517281	610.573501	308.058436	485.643902	487.262763
4	395.600809	1047.091209	603.510563	0.000000	104.403065	69.641941	1026.364945	525.000000	611.003273	533.900740
5	291.204396	945.145491	508.944987	104.403065	0.000000	35.355339	923.593525	470.558179	583.630876	513.468597
6	326.266762	978.084863	542.517281	69.641941	35.355339	0.000000	957.039707	491.553659	596.007550	523.259018
7	640.800281	45.000000	610.573501	1026.364945	923.593525	957.039707	0.000000	918.095856	1095.924267	1095.730350
8	426.878203	956.151139	308.058436	525.000000	470.558179	491.553659	918.095856	0.000000	183.439363	180.346888
9	600.187471	1134.955946	485.643902	611.003273	583.630876	596.007550	1095.924267	183.439363	0.000000	83.216585
10	561.471282	1132.982789	487.262763	533.900740	513.468597	523.259018	1095.730350	180.346888	83.216585	0.000000

Şekil 4.4: Örneklem veri setine ait uzaklık matrisi

2. Genetik Algoritma parametre değerlerinin belirlenmesi

Bu aşamada örneklem olarak alınan veri setinden, literatürde yapılan benzer çalışma ve önerilerden yola çıkılarak parametre değerleri belirlenmektedir. Örneğin literatürde populasyon büyüklüğünün kromozom uzunluğunun 2 katı olması önerilmektedir. Bundan dolay örneklem veri seti için populasyon uzunluğu 20 olarak belirlenmiştir. Parametre değerleri aşağıda verilmektedir.

```
populationLength = 20 # Literatüre göre populasyon sayısı = 2 * kromozom sayısı
generationNumber = 50
crossoverRate = 0.7 # Çaprazlama olasılığı
mutationRate = 0.4 # Mutasyon olasılığı
tournamentLength = 4 # Populasyon içerisinden k adet birey ebeveyn olarak seçilmektedir.
elitizmLength = 2 # Her iterasyondan en iyi 2 çözüm bir sonraki iterasyona geçmektedir.
```

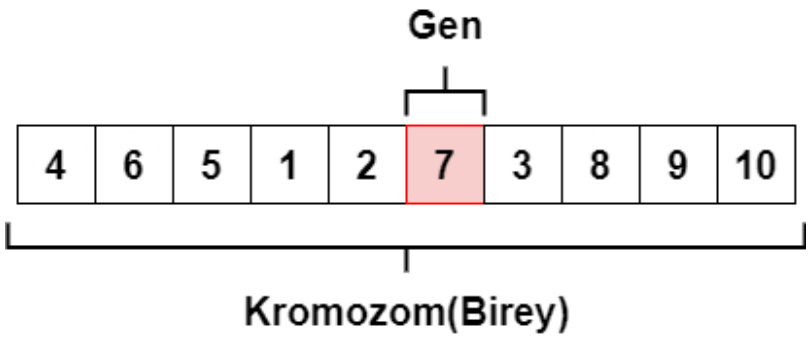
Şekil 4.5: Örnek TSP genetik algoritması parametre değerleri

“generationNumber” oluşturulacak jenerasyon sayısını, “crossoverRate” çaprazlama gerçekleşme olasılığını, “mutationRate” mutasyon gerçekleşme olasılığını, “tournamentLength” ebeveyn seçiminde turnuva yöntemi kullanılarak seçilecek kromozom sayısını ve “elitizmLenght” bir sonraki jenerasyona değiştirilmeden aktarılacak kromozom sayısını ifade etmektedir.

3. İlk popülasyonun oluşturulması

Bu aşamada belirlenen parametre değerlerinden yola çıkılarak örneklem veri setinde olan adreslerin sıralaması rastgele belirlenerek "populationLength" parametresiyle belirlenmiş olan populasyon büyüklüğü kadar rastgele bireyler oluşturulmaktadır. Bu işlem sonucunda genetik algoritmada "0.jenerasyon" olarak adlandırılan ilk populasyon oluşmaktadır. Bu populasyon içerisinde bulunan her kromozom(birey) için amaç fonksiyonu yardımıyla uygunluk değeri hesaplanarak "0.jenerasyon" için probleme ait en iyi çözüm kümesi belirlenmektedir.

Genetik algoritmada bir kromozom(birey) problem için olası bir çözüm kümesini ifade etmektedir. TSP problemi için oluşturulan bir kromozom problem çözümü için olası bir çözüm kümesidir yani adres bilgilerinin sıralamasıdır. Gen, bu kromozomu oluşturan probleme ait bilgi taşıyan en küçük yapıtaşıdır. TSP probleminde bir gen bir adres bilgisini ifade etmektedir. Bir kromozom içerisinde toplam adres bilgisi kadar gen bulunmaktadır. Genetik algoritma bünyesinde kullanılan terimlerin örneklem veri seti kullanılarak oluşturulan TSP probleminde karşılık geldiği değerler Şekil 4.6'da ifade edilmektedir.



Şekil 4.6: Örneklem veri seti için olası bir çözüm kümesi(kromozom)

4. Örneklem veri seti için genetik algoritmanın çalıştırılması

Bu aşamada genetik algoritma Şekil 3.1'de verilen genetik algoritma akış diyagramına uygun olacak şekilde işletilmektedir. Her jenerasyonda mevcut populasyon üzerinden ebeveyn seçimi yapılarak çocuk üretimi gerçekleştirilmektedir. Turnuva yöntemi kullanılarak seçilen ebeveyn kromozomları üzerinde çaprazlama uygulanarak yeni bireyler (olası çözüm kümesi) elde edilmektedir. Oluşturulan yeni bireyler üzerinde çeşitliliğin arttırılması amacıyla mutasyon işlemi uygulanmaktadır. Bu işlemden sonra oluşturulan yeni birey popülasyonuna elitizm işlemi

sonucunda bir önceki nesilden alınan en iyi çözüm değerleri de eklenerek mevcut jenerasyona ait populasyon yapısının oluşturulma işlemi tamamlanmaktadır. Oluşturulan populasyon içerisinden amaç fonksiyonu yardımıyla en iyi çözüm değeri seçilerek kaydedilmektedir. Bu işlem eğer varsa durdurma kriteri sağlanmadığı veya parametre seçiminde belirlenen jenerasyon sayısına ulaşılmadığı sürece devam etmektedir.

Genetik algoritmada ebeveyn seçimi, çaprazlama ve mutasyon adımlarında kullanılan birçok yöntem mevcuttur. Literatürde kullanımı en çok yaygın olan yöntemler şu şekildedir;

- Ebeveyn Şeçimi
  - Turnuva Yöntemi
  - Rulet Tekerleği Seçimi
  - Sıralama Seçimi
- Çaprazlama
  - Tek noktalı çaprazlama
  - Çift Noktalı Çaprazlama
  - Tekdüze çaprazlama
  - PMX çaprazlama
- Mutasyon
  - Insertion Mutasyon
  - Swap Mutasyon
  - 2-Opt Mutasyon
  - Shuffle Mutasyon

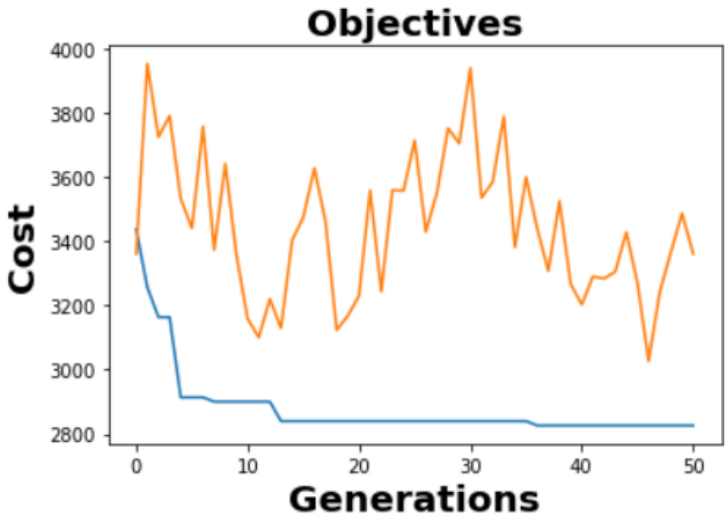
Genetik algoritmanın çalışması tamamlandıktan sonra algoritmanın çalıştığı jenerasyon süreci boyunca amaç fonksiyonuna göre elde edilmiş olan en iyi çözüm elde edilmektedir. “generationNumber” parametre değeri 20 olduğundan örneklem veri seti için genetik algoritma 20 jenerasyon çalıştırılmış ve amaç fonksiyonuna göre bir çözüm kümesi elde edilmiştir. Örneklem veri seti için gerçekleştirilen işlemler sonucunda elde edilen çıktı değerleri aşağıda verilmektedir.

```
##### Solution Output #####
Best Solution        : [4, 6, 5, 1, 2, 7, 3, 8, 9, 10]

Cost                 : 2826.4984002679575
Found at generation  : 50
############################

######## Parameters ########
Number of generation : 50
Population size      : 20
Probability of crossover : %70.0
Probability of mutation : %40.0
Tournament selection : 4
Elitism selection : 2
############################
```

Şekil 4.7: Örneklem veri seti genetik algoritma sonucu



Şekil 4.8. Örneklem veri seti genetik algoritma sonuç grafiği

Yukarıdaki çalışmada çözümü imkânsız olarak kabul edilen “Gezgin Satıcı Problemi” için meta sezgisel algoritma olan Genetik Algoritma kullanılarak optimizasyon işlemi gerçekleştirilmesine dair bir örnek verilmiştir. Araç rotalama problemi, gezgin satıcı probleminin genel bir hali olduğu için gerçekleştirilen çalışmalarda TSP için izlenen süreç adımları takip edilmiştir. VRP, TSP probleminin genel bir halidir ve farklı tipte problemleri mevcuttur. Bu çalışmada belirlenen amaç doğrultusunda VRP problemine ait “Homojen Kapasiteli Araç Problemi” incelenmiş olup genetik algoritma bu problemin optimizasyonu amacıyla geliştirilmiştir. Bu amaç doğrultusunda

belirtilen işlem adımları izlenerek proje özelinde gerçekleştirilen çalışmalara ait ayrıntılı bilgi "Gerçekleştirilen Optimizasyon Çalışmaları" bölümünde verilmektedir.

3.1.1 Gerçekleştirilen çalışmalar

Genetik algoritma yardımıyla, arka uç tarafından sağlanan siparişlere ait koordinat bilgileri kullanılarak, kurye veya kuryelerin siparişleri en az yol maliyetiyle adreslere teslim ederek depoya geri döndüğü sipariş sıralamasının hesaplanma işlemleri gerçekleştirilmiştir. Bu işlem, arka uç tarafından geliştirilmiş olan flask API'ye gönderilen istek neticesinde tetiklenerek çalışmaktadır. Bu tetiklenme sonucunda genetik algoritma çalışarak istenilen çıktı değerlerini cevap olarak arka uca geri dönmektedir. Bu işlemin başarılı bir şekilde sonuçlanması için optimizasyon tarafında gerçekleşen işlemler sırayla şu şekildedir;

Not: Gerçekleştirilen çalışmaların reel bir örnek üzerinden anlaşılır bir şekilde açıklanabilmesi için geliştirilen optimizasyon API'sine Postman uygulaması kullanılarak Görükle Mahallesinde bulunan 20 farklı adresin bilgileri istek olarak gönderilmiştir. Optimizasyon tarafına ait gerçekleştirilen her bir işlem adımı bu örnek üzerinden adım adım işlenmiş ve anlatılmıştır.

1. OSM üzerinden bölge haritasına ait graf yapısının alınması

İlk olarak bölge haritasına ait graf yapısı alınmıştır. Arka uç tarafından sağlanan koordinatlar OSM üzerinden alınan reel koordinatlar olduğundan bu koordinatlar arasındaki mesafe matrisinin hesaplanması için OSMnx kütüphanesi yardımıyla Görükle Mahallesine ait cadde ve sokakların graf yapısı alnmıştır. Graf yapısına ait görsel Şekil 4.9'da verilmektedir.

Şekil 4.9: Görükle Mahallesine ait graf yapısı

2. Adres ve depoya ait uzaklık matrisinin hesaplanması

Graf yapısıyla beraber mahalle içerisinde bulunan sokak ve caddelerin uzaklığı ve graf yapısını oluşturan düğümlerin arasındaki uzaklık bilgileri de alınmıştır. Bölge haritası alındıktan sonra arka uç tarafından gelen sipariş adresleri koordinatları kullanılarak metre cinsinden uzaklık matrisi oluşturulmuştur. Uzaklık matrisi kesitine ait görsel Şekil 4.10'da verilmektedir.

	Depo	Adres1	Adres2	Adres3	Adres4	Adres5
Depo	0.000	625.375	809.634	1369.973	1233.015	949.549
Adres1	632.335	0.000	397.200	1017.050	1007.356	1189.135
Adres2	959.766	397.200	0.000	679.111	621.071	1231.822
Adres3	1451.988	1017.050	679.111	0.000	459.005	1366.374
Adres4	1341.243	1033.569	647.284	485.218	0.000	1014.684
Adres5	953.496	1189.135	1231.822	1366.374	1049.916	0.000

Şekil 4.10: 20 adrese ve depoya ait uzaklık matrisi kesiti(metre)

Yukarıda verilen görselde görüldüğü üzere Depo →Adres1 ile Adres1 → Depo arası mesafe aynı değildir. Bunun sebebi iki nokta arasındaki mesafenin OSMnx kütüphanesi içerisinde bulunan "shortest_path()" fonksiyonu kullanılarak hesaplanmasıdır. Bu fonksiyon iki düğüm arasındaki sokak ve caddelerin yol tiplerini de hesaplamaya dahil eden bir fonksiyondur. İki nokta arasındaki sokak ve caddelerin hangi yöne trafiğe açık olup olmadığı, bu yol üzerindeki hız sınırı vb. yol durumuna ait özellikler bölgeye ait graf yapısıyla beraber OSM üzerinden alınmaktadır.

Uzaklık matrisi oluşturma işlemleri tamamlandıktan sonra matris, bölüm 4.1.2'de açıklaması ayrıntılı olarak verilmiş olan genetik algoritmaya optimizasyon işleminin gerçekleştirilmesi için girdi olarak verilmektedir. Genetik algoritma, kurye sayısı ve kapasitesine bağlı olarak adresleri toplamda kuryelerin en az yol maliyetiyle gezeceği ve depoya geri döneceği şekilde kuryelere dağıtmaktadır.

Not: Genetik algoritma, "1 adrese götürülecek 1 sipariş vardır" ve "kurye kapasitesi toplam alabileceği sipariş sayısına eşittir" maddeleri esas alınarak geliştirilmiştir. Örneğin kurye kapasitesi parametre olarak 2 olarak belirlendiyse bu kuryenin bir seferde 2 farklı adrese sahip olan 2 adet sipariş taşıyabileceğini göstermektedir.

Şekil 4.8'de algoritmaya girdi olarak verilmek amacıyla hazırlanmış olan Görükle Mahallesinde bulunan 20 farklı adresin uzaklık matrisi kesiti verilmektedir. Bu matriste 0.indekte bulunan adres değeri kuryeler tarafından depo olarak kullanılan yerin adresidir. Bu adres değerleri ve müsait olan kurye bilgileri geliştirilmiş olan Flask API'ye arka uç tarafından atılan istek sorgusuyla optimizasyon tarafına alınmaktadır.

3. Genetik algoritma ve kurye parametrelerinin belirlenmesi

Arka uç tarafından atılan istekle beraber müsait olan kurye kapasitesi ve sayısı bilgisi de optimizasyon tarafına alınmaktadır. Postman uygulaması üzerinden 20 adres bilgisi ve kurye bilgilerine ait gönderilen istek sorgusuna ait kesit görsel Şekil 4.11'de verilmektedir.

```
{
    "parameters": [
        {
            "fleet_size": [5],
            "capacity": [4]
        }
    ],
    "orders": [
        {
            "ord_id": 11,
            "latitude": 40.2278,
            "longitude": 28.8496
        },
        {
            "ord_id": 22,
            "latitude": 40.2291,
            "longitude": 28.8469
        },
        {
            "ord_id": 33,
            "latitude": 40.2322,
            "longitude": 28.8431
        },
        {
            "ord_id": 44,
            "latitude": 40.2299,
            "longitude": 28.8396
        },
        {
            "ord_id": 55,
            "latitude": 40.2214,
            "longitude": 28.8404
        },
```


Şekil 4.11: Postman uygulaması üzerinden atılan istek sorgusu kesiti

Şekil 4.9'da verilen istek sorgusunda görüldüğü üzere genetik algoritma için gerekli olan kurye parametreleri ve adres bilgileri optimizasyon tarafına arka uç tarafından sağlanmaktadır. Bu istek üzerinden alınan parametre değerleri ile genetik algoritma parametre değerlerine ait görsel Şekil 4.12'de verilmektedir.

```
# Araç Parametreleri
capacity      = 4
fleet_size    = 5

# GA Parametreleri
population_size = len(address) * 2
mutation_rate   = 0.10
elite           = 1
generations     = 10
```

Şekil 4.12: Genetik algoritma ve araç parametrelerine ait görsel

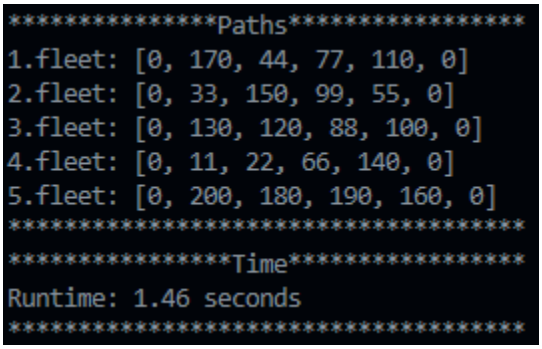
Literatürde yapılan çalışmalar incelendiğinde oluşturulacak olan populasyon büyüklüğünü ifade eden "population_size" parametre değerinin toplam adres uzunluğunun 2 katı olarak verildiği görülmektedir. Çaprazlama oranı 0.5 sabit tutularak kullanılan farklı çaprazlama değerlerine aynı gerçekleşme olasılığı tanımlanmıştır. Bu çalışmalar dikkate alınarak diğer parametre değerleri de belirlenmiştir.

4. Genetik algoritmanın çalıştırılması ve sonuçların incelenmesi

Arka uç tarafından gelen istek doğrultusunda elde edilen araç parametre ve koordinat değerlerine göre parametre değerleri belirlenmiştir. Bu işlemden sonra genetik algoritma belirtilen parametre ve girdi değerleriyle çalıştırılmıştır. Genetik algoritma başarılı bir şekilde 20 adres değerini 5 kurye arasında dağıtmaktadır. Çıktı değerlerine ait görse Şekil 4.13 ve Şekil 4.14'te verilmektedir.

```
**********Optimization**********
Generation =  1  Distance =  33280.66
Generation =  2  Distance =  30653.98
Generation =  3  Distance =  30653.98
Generation =  4  Distance =  30653.98
Generation =  5  Distance =  30025.21
Generation =  6  Distance =  27965.81
Generation =  7  Distance =  26239.58
Generation =  8  Distance =  26167.93
Generation =  9  Distance =  23411.49
Generation =  10  Distance =  22449.45
****************************************
```

Şekil 4.13: Genetik algoritma adımlarına ait toplam alınan mesafe değişimi



Şekil 4.14: Genetik algoritma çıktı değeri ve çalışma süresine ait görsel

Bu işlem sonucunda sipariş kapasitesi 4 olan 5 kurye için 20 sipariş adresi genetik algoritma aracılığıyla kuryelerin toplamda en az yol alacağı şekilde dağıtılmıştır. Genetik algoritmanın bu işlemi gerçekleştirmesi 1,46 saniye sürmüştür. Algoritmanın çalışmaya başladığı noktada hesaplanan toplam yol mesafesi 33280 metre iken genetik algoritma, işlem sonunda bu mesafeyi 22449 metreye kadar indirmiştir. Uzaklık matrisinde bulunan 0.index değerinin depoyu ifade ettiği bilinmektedir. Örnek olarak 1.kurye depodan yola çıkarak sırayla 170,44,77 ve 110 numaralı siparişleri adreslerine teslim etmiş ve tekrar depoya dönmüştür.

5. Genetik algoritma hiper parametre optimizasyonuyla çıktı değerinin iyileştirilmesi

Genetik algoritma gibi tüm meta sezgisel algoritmalarda algoritmanın kullanılmış olduğu alan ve problemle ilgili optimum sonucu veren parametre değerleri değişkenlik göstermektedir. Geliştirilmiş olan genetik algoritma, belirlenen 20 sipariş adresi ve kurye bilgileriyle Şekil 4.12'de verilen parametre değerleri kullanılarak çalıştırıldığında Şekil 4.13. ve Şekil 4.14.'te ki sonuçları vermektedir. Bu kapsamda sonuçların iyileştirilmesi için manuel arama yöntemi kullanılarak hiper parametre optimizasyonu gerçekleştirilmiştir. Populasyon büyüklüğü, mutasyon oranı, elitizim sayısı ve jenerasyon sayısı için Şekil 4.15'te verilen değerler adres ve kurye bilgileri kullanılarak denenmiştir.

```
mutation_list = [0.10,0.40,0.70]
population_size_list = [15,25,40,75]
generations_list = [10,100,200]
elitizm_list = [1,2,4]
```

Şekil 4.15: Hiper parametre optimizasyonunda kullanılan parametre değerleri

Genetik algoritmanın farklı parametre değerleri kullanılarak çalıştırılması sonucunda elde edilen çıktı değerleri karşılaştırılmak üzere kaydedilmiştir. Manuel arama işlemi sonucunda 108 farklı parametre dizilimiyle genetik algoritma çalıştırılmıştır. Her çalıştırmada kullanılan parametre değerleri, elde edilen toplam alınan yol ve algoritmanın çalışma süresi kaydedilmiştir. Elde edilen değerler alınan yol uzaklık değerine(distance) göre sıralanmıştır. Bu sıralama sonucunda elde edilen en iyi ve en kötü 10 çözüm değerine ait parametre ve sonuç değerleri Şekil 4.16. ve Şekil 4.17'de verilmektedir.

population_size	mutation_rate	elite	generation	time_second	distance
40	0.1	2	200	24.25	17265.02
40	0.4	4	200	24.53	17265.02
40	0.1	1	200	24.33	17559.55
25	0.1	1	100	7.63	17569.07
75	0.4	2	200	50.02	18034.34
75	0.1	2	200	49.30	18190.00
75	0.1	1	200	50.01	18361.55
75	0.4	4	200	49.11	18418.86
25	0.7	4	200	14.71	18433.94
75	0.7	2	200	50.60	18446.12

Şekil 4.16: En iyi 10 çözüm değerine ait görsel

population_size	mutation_rate	elite	generation	time_second	distance
75	0.1	2	10	2.37	25968.07
25	0.1	2	10	0.79	26021.94
40	0.7	2	10	1.35	26075.54
25	0.7	2	10	0.84	26720.07
25	0.4	4	10	0.77	27143.38
15	0.7	1	10	0.54	27295.11
40	0.7	1	10	1.42	27334.32
75	0.7	4	10	2.48	27612.55
25	0.4	1	10	0.82	28458.93
15	0.1	2	10	0.42	28888.01

Şekil 4.17: En kötü 10 çözüm değerine ait görsel

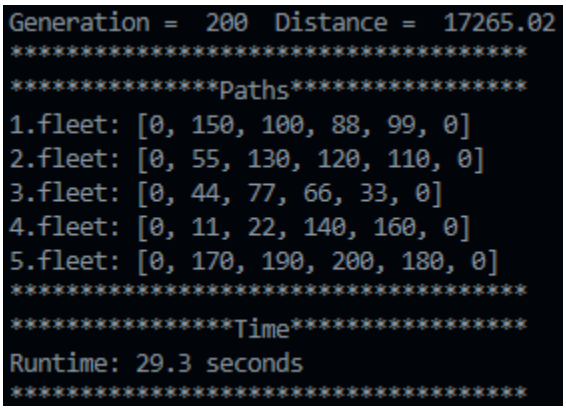
Manuel arama işlemi sonucunda 20 sipariş adresi ve kurye bilgileri için en optimum çözümü veren genetik algoritma parametreleri belirlenmiştir. Bu örneklemin hiper parametre optimizasyonu için seçilmesinin nedeni proje hedefi doğrultusunda optimizasyon tarafının karşılaması beklenen gereklilikleri içermesidir.

Bu işlem sonucunda hiper parametre değerleri manuel arama sonucunda en iyi çözümü veren parametre değerlerine göre düzenlenmiştir.

population_size	mutation_rate	elite	generation	time_second	distance
40	0.1	2	200	24.25	17265.02

Şekil 4.18: Manuel arama sonucu belirlenen en iyi çözüme ait hiper parametre değerleri

Şekil 4.18'de belirlenen parametre değerleri kullanılarak 20 sipariş adresi için genetik algoritma tekrar çalıştırılmıştır. Manuel arama işleminde bu parametreler kullanılarak elde edilen sonuçlara çok benzer sonuçlar tekrar elde edilmiştir. İşleme ait sonuç değerleri Şekil 4.19'da verilmektedir.



Şekil 4.19: Yeni hiper parametre değerlerine ait çıktı

Şekil 4.13. ve 4.14.'te verilen genetik algoritma sonuçlarında kuryelerin alacağı toplam yol mesafesi minimum 22449 metreye kadar düşürülmüştü. Manuel arama sonucunda belirlenen hiper parametre değerleri kullanılarak kuryelerin alacağı toplam yol mesafesi 17265 metreye düşürülmüştür. Bu işlem sonucunda kuryelerin alacağı toplam yol mesafesi 5184 metre azalmıştır.

Şekil 4.19'de her bir kuryenin depodan çıkarak bırakacağı siparişlerin sıralaması verilmektedir. Arka uç tarafına bu sıralamanın istenilen JSON formatında sağlanması gerekmektedir. Bu işlem doğrultusunda JSON formatı hazırlanarak geliştirilen flask API aracılığıyla sonuç değerleri arka uca iletilmektedir. 1 numaralı kuryeye ait cevap JSON içeriği Şekil 4.20'de verilmektedir.

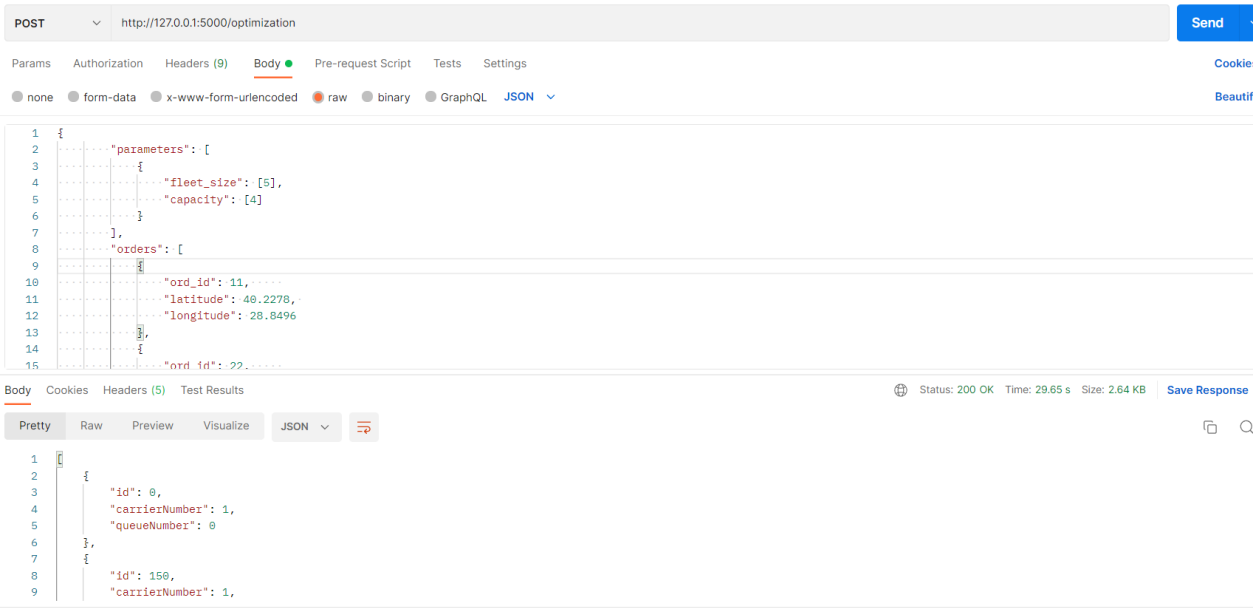

```
[
    {
        "id": 0,
        "carrierNumber": 1,
        "queueNumber": 0
    },
    {
        "id": 150,
        "carrierNumber": 1,
        "queueNumber": 1
    },
    {
        "id": 100,
        "carrierNumber": 1,
        "queueNumber": 2
    },
    {
        "id": 88,
        "carrierNumber": 1,
        "queueNumber": 3
    },
    {
        "id": 99,
        "carrierNumber": 1,
        "queueNumber": 4
    },
    {
        "id": 0,
        "carrierNumber": 1,
        "queueNumber": 5
    },
```


Şekil 4.20: 1.kurye için arka uca iletilen cevap JSON içeriği

Optimizasyon tarafında genetik algoritma aracılığıyla oluşturulan kurye sıralamaları arka uca flask API aracılığıyla tek bir JSON dosyasında gönderilmektedir. Bu JSON formatında "id" değeri, arka uç tarafından gönderilen istek ile beraber gelen sipariş adreslerini ifade eden değerdir. "carrierNumber" değeri bu "id" değerini hangi kuryenin taşıyacağını ifade etmektedir.

1.kurye için bu değer 1 olarak verilmektedir. “queueNumber” ise sipariş adresinin ilgili kurye tarafından kaçıncı sırada ziyaret edileceğini tutan değerdir. Örneğin “id” değeri 88 olan sipariş 1 numaralı kurye tarafından üçüncü sırada bırakılacak olan siparişitir.

Bu şekilde proje hedefi doğrultusunda optimizasyon tarafının sağlaması beklenen “kuryelerin alması gereken toplam yol mesafesinin azaltılması” işlemi başarılı bir şekilde gerçekleştirilmiştir. Postman uygulaması üzerinden optimizasyon tarafı ile haberleşme için geliştirilen flask API’ye atılan istek ve döndürülen cevap bilgilerinin bulunduğu ekran görüntüsü Şekil 4.21’de verilmektedir. Bu ekran görüntüsünden optimizasyon tarafının iletilen koordinatları ve kurye parametre değerlerini kullanarak beklenilen çıktı değerini başarılı bir sağlayabildiği görülmektedir.

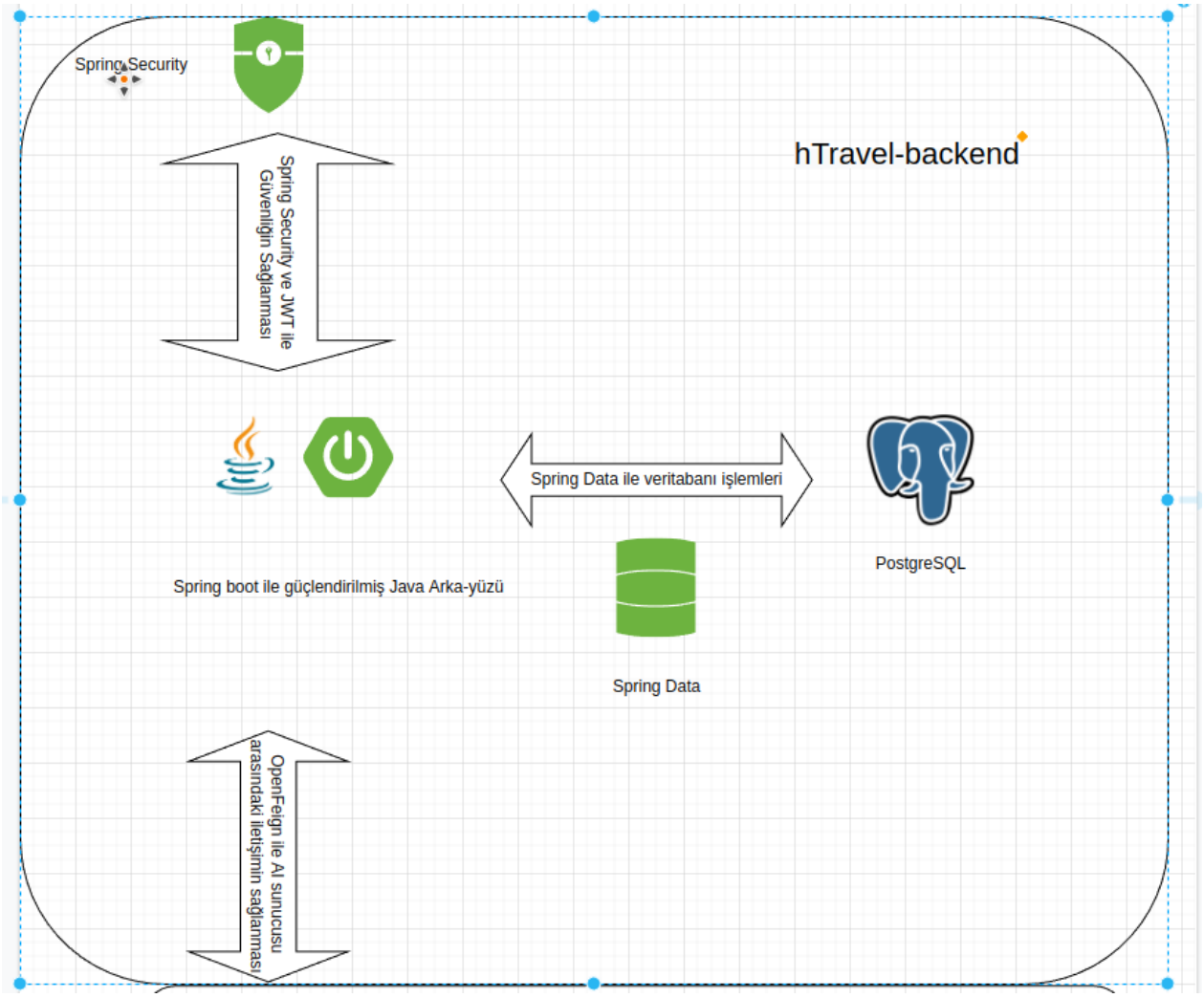


Şekil 4.21: Geliştirilen Optimizasyon API’nin çalışmasına ait görsel

4.2.Arka Uç Çalışmaları

Arka-yüz servisinin görevi, kullanıcı bilgileri, sipariş bilgileri gibi verileri depolayarak bu veriler üzerinde veri manipülasyonu gerçekleştirmektir. Arka-yüz servisi yapay zekâ servisini kullanarak kuryeler ve siparişler arasındaki optimizasyon verilerini depolamaktadır ve gerekli verileri kullanıcı ara yüzüne sunmaktadır.

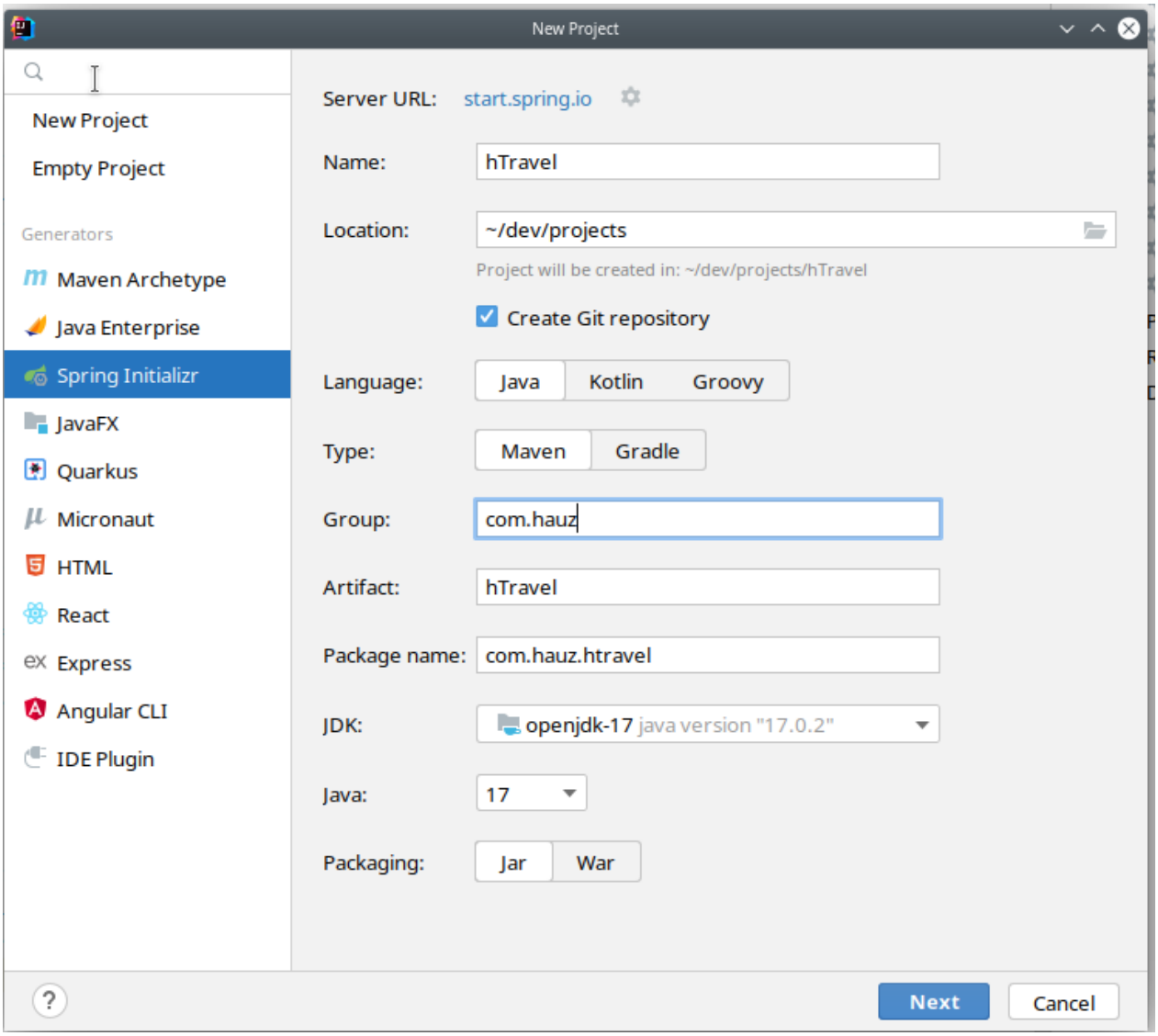
Arka-yüz servisi için şekilde gibi bir mimari düşünüldü. Uygulamayı ayağa kaldırmak adına Spring Boot, veritabanı olarak PostgreSQL, veritabanı erişimi için Spring Data, kimlik doğrulama ve yetkilendirme adına Spring Security, yapay zekâ servisi ile iletişim adına OpenFeign kütüphanesi kullanılması uygun görüldü.



Şekil 4.22: Arka uç kavramsalına ait görsel

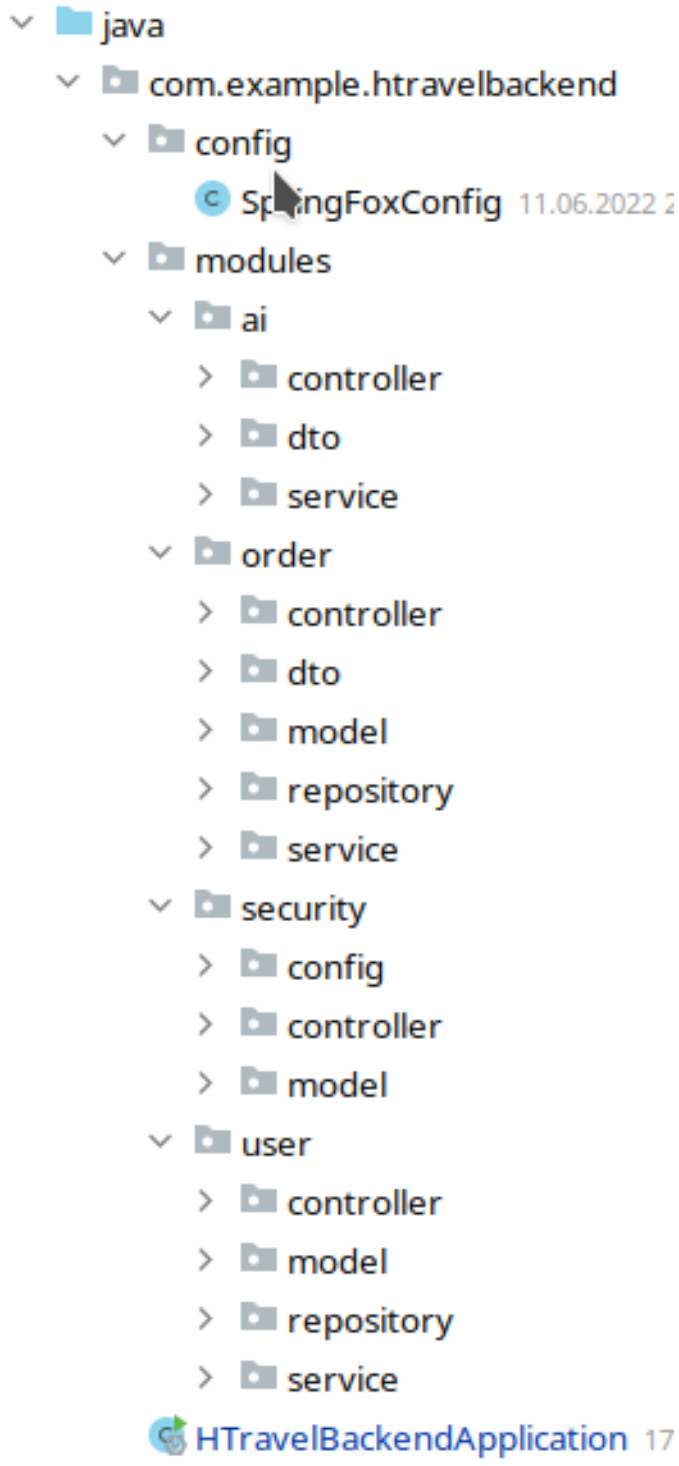
4.2.1 Spring boot implementasyonu

Spring boot başlatıcısı aracı sayesinde proje hızlı bir şekilde inşa edildi.



Şekil 4.23: IntelliJ ile Spring Initialize kullanımı

Dosya mimarisi aşağıda belirtilen şekilde kuruldu. Konfigürasyon sınıfları ayrı uygulamanın modülleri ayrı olarak saklanmaktadır. Modüller kendi içerisinde, web isteklerini yönetmek adına controller sınıflarını, veritabanı varlıklarının Java karşılıkları saklamak adına model sınıflarını, veritabanı erişim işlemlerini gerçekleştirmek adına repository sınıflarını ve business logic dediğimiz veri değişikliklerinin yapıldığı service sınıflarını içermektedir.



Şekil 4.24: Arka Yüz Paket Mimarisi

4.2.2 Örnek arka-yüz akışı

- **Kimlik Doğrulama ve Yetkilendirme Süreci**

Spring Security ile kimlik doğrulama ve yetkilendirme süreçlerini gerçekleştirmek adına OAuth 2.0 (Open Authorization) standartları kullanıldı [17]. OAuth 2.0 standartı çalışma mantığı şu şekildedir.

1. İstemci sunucuya bir kimlik doğrulama isteği gönderir. Bu istek kullanıcı adı ve şifreyi içeren bir body içerir.

{

“username”:”guven”,

“password”:”sifre123”

}

2. Yetkilendirme sunucusu istemcinin kimliğini doğrular. Kullanıcıya erişim token’ı döndürür. Bu uygulamada erişim token’ı adına Json Web Token kullanılmıştır.

{

“token”:
“eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMTFzYSIsIm5hbWUiOiJndmVuIiwiaWF0IjoxNTE2MzMxMjAyMn0.lNSyNvGx7Zop8sii8m6aHLpBbmtNGbFwsuHoK_Y_Sm4”

}

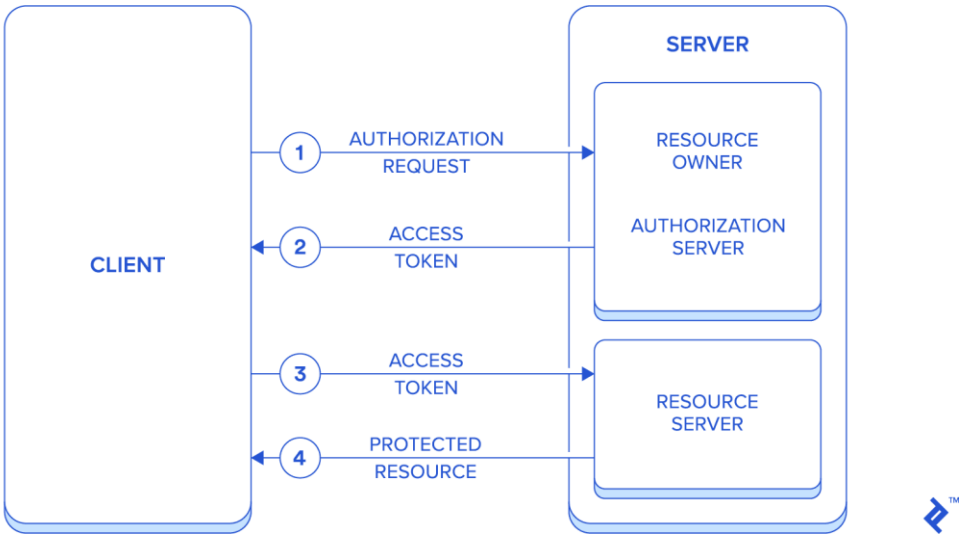
3. Kullancı önceki adımda aldığı token ile birlikte kaynaklara erişim sağlayabilir. Bunun için gönderilen isteğin başlık kısmına “Authorization” anahtarı ve “Bearer + {alınan token}” formatındaki değer eklenerek istekleri gönderilir.

{

“Authorization”:”Bearer {alınan token}”

}

4. Sunucu token sayesinde isteği gönderen kaynağın kim olduğunu ve neler yapmaya izninin olduğunu bilir. İstemcinin erişmek istediği kaynaya erişmeye yetkisi varsa erişimi sağlar.



Şekil 4.25: OAuth2 Çalışma Mimarisi

- **Kimlik Doğrulaması Yapılmış Bir İstemci İçin Örnek Akış**

Bir sipariş teslim edildikten sonra siparişin durumunun tamamlandı olarak güncellenmesini istiyoruz. Bunun için api/v1/order/complete-order/{orderId} uç noktasına sipariş ID'si ile birlikte bir post isteğinin atılması uygun görüldü. Bunun implementasyonu için öncelikle OrderController sınıfı ilgili uç nokta adresi için post isteklerini yakalamalı. Bunun için @PostMapping anotasyonu kullanıldı. OrderId değerini almak adına @PathVariable anotasyonu kullanıldı. Geridönüş tipi ise veritabanında güncelleme işlemi yapıldığı için kaç satırın güncellendiği bilgisi olmalıdır. Örneğin bu işlem için maksimum 1 satır güncellenebilir ve 1 sayısı döndürülür. Eğer hiçbir satır güncellenmezse 0 değeri döndürülür.

```
@PostMapping("/complete-order/{orderId}")
public ResponseEntity<Integer> completeOrder(@PathVariable Long orderId) {
return ResponseEntity.ok(orderService.completeOrder(orderId));
}
```

Kod 4.1: /complete-order/{orderId} uç noktasının implementasyonu

OrderController sınıfının completeOrder metodu OrderService sınıfının completeOrder metodunu çağırır. Bu metod ise OrderCommandRepository interface'inin updateOrderStatusByIdEquals motodunu çağırır. Herhangi bir mantıksal işlem gerekseydi bu sınıf içerisinde gerçekleştirilirdi. Örneğin alınan değeri bir arttır şeklinde. Fakat bu örnek adına başka bir işlem gerekmediği için doğrudan Repository interface'inin ilgili metodu çağırıldı.

```
@Override
public int completeOrder(Long orderId) {
return orderCommandRepository.updateOrderStatusByIdEquals(orderId);
}
```

Kod 4.2: Siparişi tamamlama işlemini gerçekleştiren metodu

Repository interface'leri (repository'ler için interface kullanılır) veritabanı işlemlerini gerçekleştirmemizi sağlar. OrderCommandRepository sınıfı bir Order tablosunda bir Command işlemi gerçekleştireceği için bu şekilde adlandırılmıştır. Query yani sadece sorgu işlemlerinin gerçekleşeceği interfaceler tablo_ismiQueryRepository şeklinde isimlendirildi.

Veritabanındaki değişiklikler bir transaction içinde yapılmalıdır. Aksi taktirde işlemler sırasında bir hata meydana gelirse transaction rollback edilebilmeli yani geri sarılabilmelidir. Tabiki hatalı işlemlerde veri değişikliği yapmak istemeyiz. Bunu sağlamak adına @Transactional anotasyonunu kullanıyoruz.

@Modifying ise yine veritabanı güncellemelerinde kullanılır. @Query anotasyonu ise çalıştırmak istediğimiz sorgudur. Metod isimlerini uygun şekilde vererek sorgu yazmaya gerek kalmadan da dilediğimiz işlemi gerçekleştirebilirdik fakat özellikle seçme sorguları başta olmak üzere her zaman @query anotasyonu ile özel sorgu yazmak best practice dediğimiz bir işi yapmanın en doğru yoludur. Çünkü özel sorgu yazmadığımızda metod ismi ile üretilen sorgu ihtiyacımız olmayan farklı verileri getirebilir ihtiyacımız olmayan işlemleri de gereksiz yere yerine getirebilir.

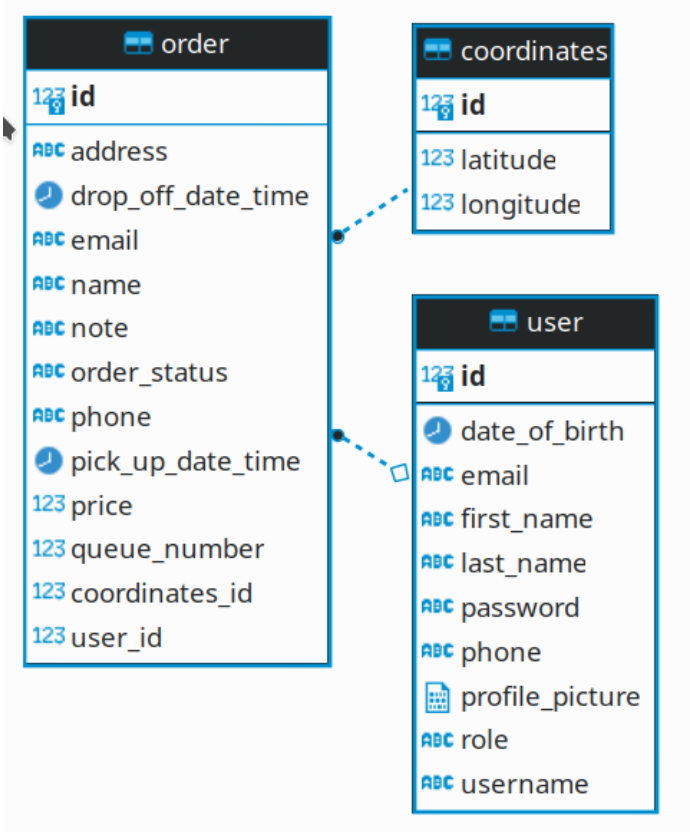
```
public interface OrderCommandRepository extends CrudRepository<Order, Long> {
@Transactional
@Modifying
@Query("update Order o set o.orderStatus =
com.example.htravelbackend.modules.order.model.OrderStatus.DELIVERED where o.id = :id")
int updateOrderStatusByIdEquals(@NonNull Long id);
}
```

Kod 4.3: Siparişi tamamlamak için veri tabanı güncelleme kodu

Repository sınıfı ilgili işlemi gerçekştirir sonuç değeri önce service sınıfına oradan controller sınıfına döndürür. Controller sınıfı da isteğin geldiği adrese sonucu döndürür.

- **Kullanılan veritabanı mimarisi**

Order tablosu sipariş bilgilerini tutmaktadır. Siparişin kordinatları coordinates ayrı bir tabloda bulunup order tablosu ile birebir ilişkiye sahiptir. User tablosu ise kullanıcı bilgilerini taşır.



Şekil 4.26: Veri tabanı Mimarisi

4.3. Mobil Uygulama Çalışmaları

Bu bölüm geliştirilmiş olan iOS mobil uygulamasının gereksinimlerini, insan-bilgisayar arayüzlerini, tasarımlarını ve işleme mantıklarımı ele alınacaktır.

4.3.1 Gereksinimler

Gereksinimler bölümü, kullanıcı gereksinimleri ve sistem gereksinimleri olmak üzere iki ana bölüme ayrılmıştır.

4.3.1.1 Kullanıcı gereksinimleri

- **Kayıt Olma**

Uygulamada, kullanıcılar izlemesi gereken yolları ve konumları görebilmek için sisteme kaydolabilmelidir.

- **Giriş Yapma**

Sisteme daha önce kaydolmuş Hauz kullanıcıları, çıkış yaptıklarında sisteme giriş yaparak herhangi bir iOS cihazından verilerine tekrar ulaşabilmelidir.

- **Tüm Rotayı Görüntüleme**

Kullanıcı giriş yaptığında üzerine atanmış bir iş var ise bu işin rotasını görüntüleyebilmeli ve dilediğinde gerekli yönlendirmeleri alabilmelidir.

- **Yönlendirmeleri Alma**

Kullanıcı Rotayı görüntüleyip yönlendirmeleri almak için butona tıkladığında gidilmesi gereken ilk adres için gerekli yönlendirmeleri alabilmelidir.

- **Yönlendirme Adımları Arası Gezinme**

Uygulama kullanıcının konumunu takip ederek sonraki adımı gösterebilmeli, aynı zamanda GPS sinyalinin kaybolma durumuna karşılık manuel olarak da yönlendirme adımları arasında ileri ve geri hareket edebilmelidir.

- **Kullanıcı Bilgileri**

Kullanıcı istediğinde sisteme kayıtlı olan isim soyisim, telefon ve mail adreslerini görüntüleyebilmelidir.

4.3.1.2 Sistem gereksinimleri

Bu bölüm, sistem gereksinimleri açısından kullanıcı gereksinimleri hakkında ayrıntılı bilgiler içermektedir. Aynı zamanda sistem gereksinimleri doğrultusunda yapılan model çalışmalar da ilgili başlığın altında yer alacaktır.

- **Kayıt Olma**
  - Kayıt sayfasında isim, soy isim, e-posta, şifre ve telefon numarası olmak üzere 5 farklı alan olacaktır.
  - Uygulama, bu alanların geçerli olup olmadığını kontrol edecektir (Örneğin e-posta adresi kurallara uygun yazılmış mı?). Aynı zamanda kayıt işleminin gerçekleşebilmesi için tüm alanlar dolu olmalıdır. Aksi takdirde, kullanıcı bir uyarı mesajı görecektir.
  - Kullanıcının yazdığı tüm bilgiler "register" uç noktası üzerinden kontrol edilecektir.
  - Kayıt işlemi başarılı olursa, kullanıcı kayıt bilgileri ile giriş yapabilmek için giriş sayfasına yönlendirilecektir.

Bu gereksinimler için yapılan model çalışması Şekil 4.27'deki gibidir.

Şekil 4.27: Kayıt Ol ekranı model çalışması

- **Giriş Yapma**

  - Giriş sayfasında kullanıcı adı ve şifre alanları olmak üzere iki alan olacaktır.
  - Uygulama, alanların boş olup olmadığını kontrol edecektir. Giriş işleminin gerçekleşebilmesi için tüm alanlar dolu olmalıdır. Aksi takdirde, kullanıcı bir uyarı mesajı görecektir.
  - Kullanıcının yazdığı tüm bilgiler "auth" uç noktası üzerinden kontrol edilecektir.
  - Giriş işlemi başarılı olursa, kullanıcı doğrudan ana sayfalara yönlendirilecek ve arka-yüz üzerinden üretilen token iOS uygulamasına kaydedilecektir.

  Bu gereksinimler için yapılan model çalışması Şekil 4.28'deki gibidir.

Şekil 4.28: Giriş ekranı model çalışması

- **Tüm Rotayı Görüntüleme**
  - Rotanın görüntülenebilmesi için ana sayfa üzerindeki harita kullanılacaktır.
  - Kullanıcın merkez ofisi ve teslimat bırakacağı noktalar "getOrders" uç noktası ile elde edilecek ve uygulama içi sabitlerde tutulacaktır.
  - Elde edilen merkez ofisi ve teslimat bırakacağı noktalar birer iğne yardımı ile gösterilecektir.
  - Bu iğneler arası yol tarifi de yine aynı uç noktadan gelen bilgilere göre çizilecektir.

Bu gereksinimler için yapılan model çalışması Şekil 4.29'daki gibidir.

Şekil 4.29: Harita ekranı model çalışması

- **Yönlendirmeleri Alma**
  - Kullanıcının teslimat yapacağı konumlar belirlendikten sonra takip edeceği yoldaki yönlendirmeler Apple Haritaların sunmuş olduğu "MapKit" ile ekranda kullanıcıya sunulacaktır.

- **Yönlendirme Adımları Arası Gezinme**
  - Bu özellik ileri ve geri butonları olmak üzere 2 bileşen ile elde edilecektir.
  - Kullanıcının yönlendirme adımları arasında gezinebilmesi için "MapKit" ile elde edilen yönlendirmeler bir listede tutulacak ve kullanıcının bu adımlar arasında ekrandaki ileri ve geri butonlarını kullanarak geçiş yapabilmesi sağlanacaktır.

Yönlendirmeleri Alma ve Yönlendirme Adımları Arası Gezinme gereksinimleri için yapılan model çalışması Şekil 4.30'daki gibidir.

Şekil 4.30: Yönlendirme ekranı model çalışması

- **Kullanıcı Bilgileri**
  - Profil sayfası profil fotoğrafı, isim – soy isim, telefon numarası, e-posta adresi olmak üzere 4 bileşenden oluşmaktadır.
  - Kullanıcı bilgileri "user" uç noktası ile elde edilecek ve bu sayfada kullanıcıya sunulacaktır.

Bu gereksinimler için yapılan model çalışması Şekil 4.31'deki gibidir.

Şekil 4.31: Profil ekranı model çalışması

4.3.2 İmplementasyon

Bu bölümde, ekran görüntüleri, her bir ekran hakkında detaylı bilgiler, her sayfa için ayrı ayrı kullanılan akış dahil olmak üzere Hauz iOS mobil uygulaması için neler yapıldığına değinilecektir.

- **Giriş Sayfası**

Şekil 4.32: Giriş ekranı

Kullanıcıya sunulan ilk sayfa giriş ekranıdır. Bu ekran içerisinde kullanıcıdan giriş bilgileri olan kullanıcı adı ve parola bilgileri istenmektedir. Kullanıcının bu ekran üzerinden yönlenebileceği 2 farklı sayfa bulunmaktadır:

1. Ana Sayfa
2. Kayıt Ol Sayfası

Giriş bilgileri doğru olduğu takdirde kullanıcı ana sayfaya yönlendirilmektedir. Kullanıcı bu iki alandan herhangi birini veya ikisini boş bırakırsa da bu alanların boş bırakılamayacağına dair uyarı bir mesajı ile karşılaşmaktadır (Şekil 4.34).

Şekil 4.33: Giriş ekranı boş alanlar uyarı mesajı

Hem bu sayfada kullanılan hem de uygulamanın tamamında kullanılan uyarı ve bilgilendirme mesajlarında işletim sisteminin kendi uyarı görünümleri değil kendi tasarladığımız görünümler kullanılmıştır.

Kullanıcı bilgileri başarılı bir şekilde alındıktan sonra API tarafından sağlanan "auth" uç noktasına bu kullanıcı bilgileri ile POST türünde istek atılmaktadır. İstek, Alamofire kütüphanesi aracılığı ile API'ye ulaşmaktadır. API'den gelen cevabı karşılayabilmek için bir model oluşturulmuştur (Örnek model Kod 4.4).

```
struct AuthResponseModel: Codable {
   var token: String?
}
```

Kod 4.4: "Auth" uç noktası cevabını karşılamak için oluşturulmuş model

API'den dönen cevap başarılı ise kullanıcının token'i KeyChain üzerinde saklanarak kullanıcı ana sayfaya yönlendirilmektedir.

- **Kayıt Ol Sayfası**

Şekil 4.34: Kayıt Ol ekranı

Kullanıcı sisteme kaydolmak istediğinde Şekil 4.8'deki ekran ile karşılaşmaktadır. Bu ekran kullanıcı bilgilerinin elde edilebilmesi için 6 adet giriş içeren bölümden oluşmaktadır. Bu alanların hiçbirinin boş bırakılamamasının yanı sıra e-posta şifre ve telefon bilgisi girişleri, yapılan girişlerin kurallara uygun olup olmadığının kontrolü için birer düzenli ifade (regex) ile kontrol işlemine tabii tutulmuştur (Örnek Kod 4.5). Kullanıcıya bu uyarı mesajları, metin girişi yaptığı alanların altında gösterilmekte ve hata bulunan giriş alanları kırmızı renkte gösterilmektedir (Şekil 4.35).

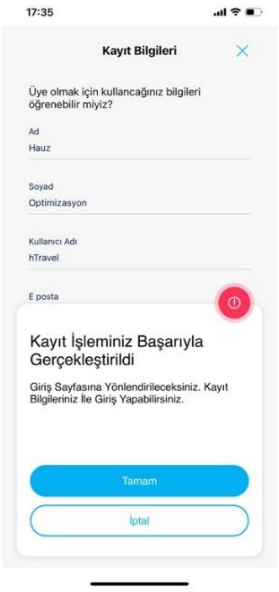
```
static func validateEmail(with email: String) -> Bool {
   let emailPattern = "[A-Z0-9a-z._%+-]+@([A-Za-z0-9.-]{2,64})+\\.[A-Za-z]{2,64}"
   let predicate = NSPredicate(format:"SELF MATCHES %@", emailPattern)

   return predicate.evaluate(with: email)
  }
```

Kod 4.5: E-posta kontrolü için oluşturulan düzenli ifade (regex) kodu

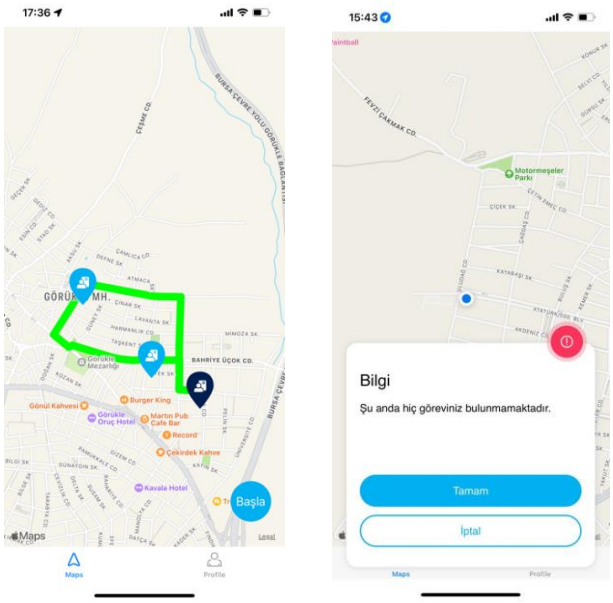
Şekil 4.35: Kayıt Ol ekranı uyarı mesajları

Kayıt için kullanıcı bilgileri alındıktan sonra API üzerinden verilerin kaydedilebilmesi için "register" uç noktasına Alamofire kütüphanesi aracılığıyla gönderilmektedir. Eğer API'den dönen sonuç başarılı ise kullanıcıya bir uyarı mesajı gösterilmekte (Şekil 4.36) ve kullanıcı, kayıt bilgileri ile giriş yapabilmek üzere giriş sayfasına yönlendirilmektedir.



Şekil 4.36: Başarılı kayıt mesajı

- **Ana Sayfa**



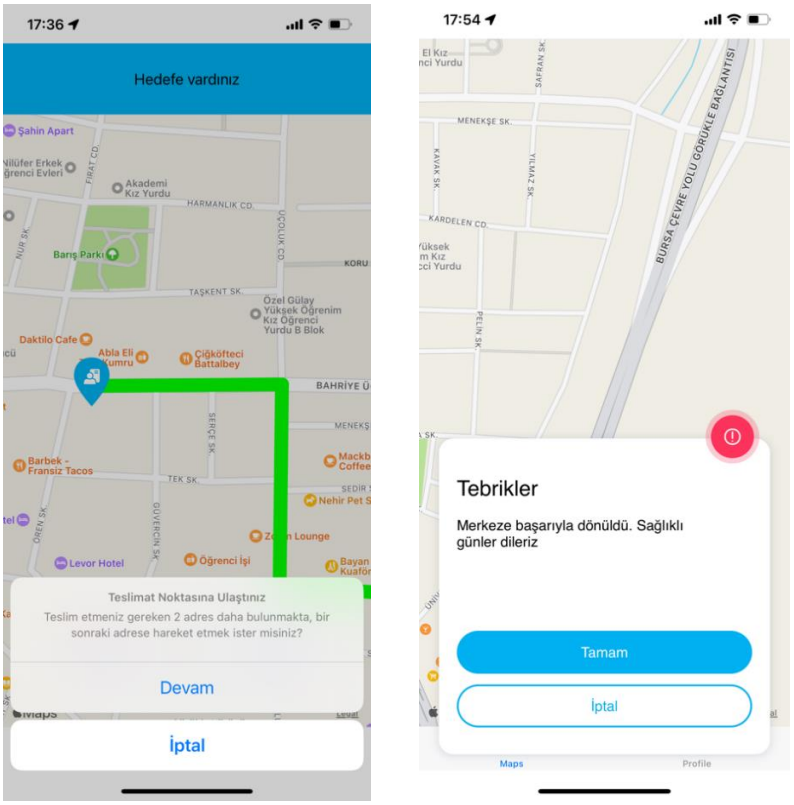
Şekil 4.37 – 4.38: Ana Sayfa ekranları

Giriş sayfası sonrası ana sayfaya yönlendirilen kullanıcıları olası 2 ekran beklemektedir. Kullanıcı bu ekrana yönlendirildiğinde ekran henüz kullanıcıya gösterilmeden önce "getOrders" uç noktasına API ile bir istek atılarak kullanıcının anlık olarak herhangi bir işe sahip olup olmadığı kontrol edilmektedir. Eğer kullanıcının üzerine atanmış bir iş varsa yani kullanıcının gitmesi rotalar var ise Şekil 4.37'de görüldüğü üzere gidilecek olan yol bilgisi kullanıcıya gösterilmektedir. Eğer kullanıcıya atanmış herhangi bir iş yok ise Şekil 4.38'deki gibi o anda kullanıcıya atanmış bir işin olmadığına dair bilgilendirme mesajı gösterilmektedir.

Kullanıcı tüm rotasını inceledikten sonra istediği zaman ekranın sağ altında bulunan "Başla" butonu ile yönlendirmeleri almaya başlayabilmektedir (Şekil 4.39). Yönlendirme başladığında ekrandaki "Başla" butonu kaybolmakta ve kullanıcının yönlendirmeleri görebilmesi için ekranın üst kısmında yönlendirme alanı gösterilmektedir. Kullanıcının konumu da takip edilmekte ve yönlendirme komutunu yerine getirdiğinde (örneğin 0.3km sonra dönülmesi gereken virajdan dönüldüğünde) otomatik olarak bir sonraki adıma geçilmektedir. Aynı zamanda uygulama kullanıcının GPS sinyalinin kesilme ihtimalini de göz önünde bulundurarak yönlendirmelerin ekranın alt bölümünde bulunan ileri ve geri okları yardımı ile gerçekleştirebilmesini sağlamaktadır. Bu özellik aynı zamanda uygulamanın test aşamasında da kullanılmıştır.

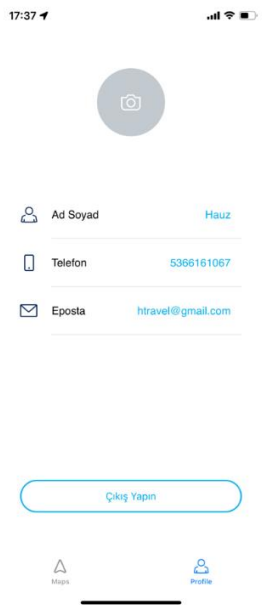
Şekil 4.39: Yönlendirme adımlarının başlaması

Yönlendirmelerin ardından kullanıcı varış noktasına ulaştığında kullanıcıya hedefe ulaşıldığına dair bir uyarı çıkmakta ve kalan hedef sayısı bildirilmektedir (Şekil 4.40). Kullanıcı teslimatını yapıp devam etmek istediğinde “complete-order” uç noktasına ilgili siparişin id’si gönderilerek siparişin veritabanındaki durumunun “Teslim Edildi” olarak kaydedilmesi sağlanmaktadır. Kullanıcı tüm siparişlerini bitirdiğinde merkeze yönlendirilmekte ve merkeze vardığında da bir bilgilendirme mesajı görmektedir (şekil 4.41).



Şekil 4.40 – 4.41: Hedefe ve Merkeze varıldığına dair uyarı ekranları

- **Profil Sayfası**



Şekil 4.42: Profil Sayfası

Profil ekranında kullanıcı isim, telefon, e-posta ve profil fotoğrafı bilgilerini görüntüleyebilmektedir. TabBar üzerinden Profil sekmesi seçildiğinde "user" uç notasından kullanıcı bilgileri alınmakta ve bu sayfa üzerinde ilgili alanlara doldurulmaktadır. Bu uç noktadan dönen cevabı karşılamak için oluşturulmuş model Kod 4.6'daki gibidir. Kullanıcı fotoğraf seçtiğinde bu fotoğraf "update" uç noktası üzerinden API aracılığı ile veritabanına kaydedilmektedir. Aynı zamanda bu ekran üzerinden çıkış yapılabilmekte ve oturumu sonlandırabilmektedir.

```
struct UserResponseModel: Codable {
    var firstName: String?
    var lastName: String?
    var email: String?
    var phone: String?
}
extension UserResponseModel {
    enum CodingKeys: String, CodingKey {
        case firstName
        case lastName
        case email
        case phone
    }
     }
```

Kod 4.6: "User" uç noktasının cevabı için oluşturulan model

6. SONUÇLAR

Gerçekleştirilen çalışmalar neticesinde orta ve küçük işletmelerin yaşamış olduğu sipariş dağıtım sorununa çözüm sunabilecek mobilite tabanlı bir yol optimizasyonu uygulaması başarılı bir şekilde geliştirilmiştir. Uygulamanın kullanımıyla gün içerisinde onlarca siparişi zamanında teslim etmesi gereken işletmelerin bu işlemi en az yol masrafıyla gerçekleştirerek zaman ve maddi olarak kazanç sağlaması beklenmektedir. Siparişlerin adreslere götürülmesinde görev alan her bir kuryenin uğrayacağı sipariş adresleri toplamda en az yol maliyeti oluşacak şekilde uygulama tarafından belirlenmektedir. Her bir kuryenin sipariş götüreceği adresler ve bu adreslerin rotasının uygulama tarafından kuryelere sağlanmasıyla kurye elemanlarının sipariş götürme rotasının belirlemesinden açığa çıkan zaman ve yol maliyetinin de önüne geçilmiştir.

Uygulama, homojen kurye yapısı yani eşit sipariş taşıma kapasitesine sahip kuryeleri bulunan her işletmede kullanıma uygun olarak geliştirilmiştir. Uygulamanın gelecekte heterojen kurye yapısı

ve birden çok deposu bulunan işletmelere cevap verebilecek hale getirilme potansiyeli bulunmaktadır. Aynı zamanda uygulamanın nihai hedefi blok zincir tabanlı MaaS (Mobility as a Servis – Hizmet olarak Mobilite) yapısı sunan bir Super-App olmaktır. MaaS - Hizmet Olarak Taşıma (TaaS) olarak da adlandırılabilmektedir - tüketici merkezli bir insan taşımacılığı modelidir. Araç ve bisiklet paylaşımı, taksiler ve araç kiralama gibi ulaşım yöntemlerinin herhangi bir kombinasyonunu içerebilen ve seyahat planlamasından ödemelere kadar tüketici için her şeyi sağlayan isteğe bağlı, gerçek zamanlı bir platformdur. Bir MaaS uygulaması kullandığınızda, ihtiyacınız olduğu sürece bir scooter, bisiklet, araba veya tren yolculuğuna erişebilirsiniz. Kısaca şehir içi ve şehir dışı ulaşım sağlamanın daha kolay ve uygun fiyatlı halidir. Aynı zamanda böyle bir sistem içerisinde ekolojik ve sosyal fayda için çeşitli ulaşım yöntemlerini içeren bir kavram olan akıllı mobilite kavramını da içermektedir. Akıllı şehirler, trafik sıkışıklığını azaltmak ve verimliliği artırmak için akıllı mobiliteyi çözümün bir parçası olarak içermektedir. Daha akıllı ve daha temiz ulaşım, araç emisyonlarını düşürmeye ve yol güvenliğini artırmaya yardımcı olmaktadır. Ayrıca kapsamlı bir Hizmet Olarak Mobilite servisi, şirketler, ulaşım otoriteleri ve hatta ülkeler arasında çeşitli ulaşım araçlarının sorunsuz bir şekilde entegre olmasına bağlıdır. Herkes için tek bilet ve merkezi bir faturalandırma söz konusu olduğundan, bu durum yalnızca yolcuya fatura kesen tarafça seyahat zincirinin tüm bölümlerine erişilebiliyorsa işe yarayabilir. Bu tarz birbirine bağlı geniş çaplı verilerin birbirine bağlanması ve erişilebilmesi sorununu geçtiğimiz 10 yıl içerisinde oldukça popüler hale gelen blok zinciri ve DLT (Dağıtık Defter Teknolojisi) çözüme kavuşturabilmektedir. Blok zinciri ve MaaS yapısı birleştirilerek araç kullanma/kiralama hizmetleri, elektrikli araç şarj işlemleri, trafik kontrolü, ehliyet/ruhsat gibi belge kontrolleri ve lojistik süreç takibi gibi hizmetlerin de sağlanabilmesi mümkün olacaktır. Böylece uygulama sadece paket dağıtım servisi sunan bir optimizasyon uygulaması olmaktan çıkarak kullanıcıların ulaşım ile ilgili gerek toplu taşıma için yol optimizasyonu gerek özel araç şarj etme, kiralama, takip etme işlemleri gerekse lojistik araç takibi işlemlerimi tek bir noktadan yürütebileceği ve aynı zamanda kurye gibi dağıtım sistemi kullanan işletmelerin optimizasyon için yararlanabileceği bir Super-App haline gelebilecektir.

7. KAYNAKLAR

[1] https://www.verikaynagi.com/genel/turkiyede-online-alisveris-verileri/

[2] https://www2.deloitte.com/content/dam/Deloitte/nl/Documents/consumer-business/deloitte-nl-cb-ths-rise-of-mobility-as-a-service.pdf

[3] https://tr.wikipedia.org/wiki/Genetik_algoritma

[4] Beasley, D., Bull, D.R., ve Martin, R.R., 1993a. An Overview of Genetic Algorithms: Part 1, Fundamentals. University Computing, Vol.15(2), 58–69, UK.

[5] Beasley, D., Bull, D.R., ve Martin, R.R., 1993b. An Overview of Genetic Algorithms: Part 2, Research Topics .University Computing, Vol. 15(4), 170–181, UK.

[6] Bingül, Z., Sekmen, A.S. ve Zein, S., 1999. An Application of Multi-Dimensional Optimization Problems Using Genetic Algorithms. Proceedings of the IASTED International Conference Intelligent Systems and Control, Santa Barbara, CA, USA

[7] Hybrid Genetic Algorithms with Great Deluge For Course Timetabling - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Flowchart-of-a-simple-GAs-process_fig1_266863507 [accessed 10 Dec, 2021]

[8] https://tr.wikipedia.org/wiki/OpenStreetMap

[9] www.oracle.com

[10] https://www.aternity.com/blogs/what-is-java-used-for/#:~:text=Java%20can%20be%20used%20to,for%20any%20of%20these%20platforms.

[11] spring.io/why-spring

[12] https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

[13] https://tr.m.wikipedia.org/wiki/Dosya:Postgresql_elephant.svg

[14] https://projectlombok.org/

[15] https://swagger.io/

[16] https://www.toptal.com/spring/spring-boot-oauth2-jwt-rest-protection

[17] https://koenig-media.raywenderlich.com/uploads/2019/01/01-MVC-Diagram-480x241.png

[18] https://tr.wikipedia.org/wiki/Ara%C3%A7_rotalama_problemi

[19] Minimizing the Carbon Footprint for the Time-Dependent Heterogeneous-Fleet Vehicle Routing Problem with Alternative Paths - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Illustration-of-the-traveling-salesman-problem-TSP-and-vehicle-route-problem-VRP_fig1_277673931 [accessed 22 Jun, 2022]

[20] https://tr.wikipedia.org/wiki/Öklid_uzaklığı