PROLAB II

Proje 1 – Ulaşım Rota Planlama Sistemi

Erdem AÇIKYÜREK

Bilgisayar Mühendisliği 3. Sınıf
öğrenci no: 220202094

I. ÖZET

Ulaşım planlama, şehir içi hareketliliği kolaylaştırmak ve kullanıcıların en uygun rotayı seçmesine yardımcı olmak için kritik bir öneme sahiptir. Ancak, farklı ulaşım türlerini (otobüs, tramvay, taksi) birleştiren ve kullanıcı ihtiyaçlarına göre optimize edilmiş bir rota planlama sistemi oluşturmak karmaşık bir süreçtir. Bu bağlamda, şehir içi ulaşım verilerini işleyen ve kullanıcıya en uygun rotayı sunan bir uygulama geliştirmek, hem zaman hem de maliyet açısından önemli bir araç olabilir.

II. GİRİŞ

Java Nedir?

Java, yüksek seviyeli, nesne yönelimli bir programlama dilidir. James Gosling tarafından Sun Microsystems'te geliştirilmeye başlanmış ve 1995 yılında ilk kez piyasaya sürülmüştür. Java, platform bağımsızlığı ve güvenilirliği ile tanınır ve birçok alanda yaygın olarak kullanılmaktadır.

Java'nın başlıca özellikleri şunlardır:

Platform Bağımsızlığı: Java, "bir kere yaz, her yerde çalıştır" (Write Once, Run Anywhere - WORA) prensibine dayanır. Java kodu, Java Virtual Machine (JVM) sayesinde farklı işletim sistemlerinde (Windows, macOS, Linux) çalışabilir.

Nesne Yönelimli Programlama: Java, nesne yönelimli programlama (OOP) prensiplerine dayanır. Bu, kodun modüler, yeniden kullanılabilir ve bakımı kolay olmasını sağlar.

Güvenilirlik ve Güvenlik: Java, otomatik çöp toplama (garbage collection) ve güçlü tip kontrolü gibi özelliklerle güvenilir bir dil olarak kabul edilir. Ayrıca, güvenlik mekanizmaları sayesinde kötü niyetli kodların çalışmasını engeller.

Geniş Kütüphane Desteği: Java, geniş bir standart kütüphane (Java API) sunar. Bu kütüphaneler, dosya işlemleri, ağ programlama, veri tabanı bağlantıları ve daha birçok alanda geliştiricilere yardımcı olur.

Çoklu İş Parçacığı (Multithreading): Java, çoklu iş parçacığı desteği ile aynı anda birden fazla görevin yürütülmesini sağlar. Bu, özellikle performans odaklı uygulamalarda faydalıdır.

Java, özellikle kurumsal uygulamalar, web geliştirme, mobil uygulama geliştirme (Android), büyük veri işleme ve bilimsel uygulamalar gibi alanlarda yaygın olarak kullanılmaktadır. Hem başlangıç düzeyindeki geliştiriciler hem de deneyimli profesyoneller için tercih edilen bir programlama dilidir.

Maven Nedir?

Maven, Java projeleri için bir yapılandırma ve bağımlılık yönetim aracıdır. Apache Software Foundation tarafından geliştirilen Maven, projelerin derlenmesi, test edilmesi, paketlenmesi ve dağıtılması süreçlerini otomatikleştirir. Maven, "Convention over Configuration" (Yapılandırma yerine kural) ve "Dependency Management" (Bağımlılık Yönetimi) prensiplerine dayanır.

Maven'in ana özellikleri şunlardır:

Bağımlılık Yönetimi: Maven, projede kullanılan kütüphaneleri (örneğin, Jackson) otomatik olarak indirir ve yönetir. Bu, geliştiricilerin bağımlılıkları manuel olarak eklemesi gerekliliğini ortadan kaldırır.

Standart Proje Yapısı: Maven, projeler için standart bir dizin yapısı sunar. Bu, projelerin tutarlı bir şekilde organize edilmesini sağlar ve ekip çalışmasını kolaylaştırır.

Otomatik Derleme ve Test: Maven, projeyi derlemek, testleri çalıştırmak ve paketlemek için komutlar sunar (örneğin, mvn clean install). Bu, geliştirme sürecini hızlandırır.

Merkezi Depo (Repository): Maven, bağımlılıkları Maven Central Repository gibi merkezi depolardan çeker. Bu, kütüphanelere kolay erişim sağlar.

Esneklik: Maven, pom.xml (Project Object Model) dosyası üzerinden özelleştirilebilir. Geliştiriciler, projenin ihtiyaçlarına göre yapılandırmalar yapabilir.

Maven, özellikle büyük ölçekli Java projelerinde bağımlılık yönetimi ve yapılandırma süreçlerini kolaylaştırmak için yaygın olarak kullanılır. Bu proje de Maven kullanılarak oluşturulmuştur.

Eclipse Nedir?

Eclipse, Java geliştiricileri için popüler bir entegre geliştirme ortamıdır (IDE - Integrated Development Environment). Açık kaynaklı bir yazılım olan Eclipse, kod yazma, hata ayıklama, test etme ve proje yönetimi gibi işlemleri kolaylaştırır.

Eclipse'in temel özellikleri şunlardır:

Kod Düzenleme ve Hata Ayıklama: Eclipse, sözdizimi vurgulama, otomatik tamamlama ve hata ayıklama araçları sunar. Bu, geliştiricilerin kod yazma sürecini hızlandırır ve hataları kolayca bulmasını sağlar.

Maven Entegrasyonu: Eclipse, Maven projelerini doğrudan destekler. pom.xml dosyası üzerinden bağımlılıkları yönetebilir ve Maven komutlarını çalıştırabilir.

Eklenti Desteği: Eclipse, geniş bir eklenti ekosistemine sahiptir. Geliştiriciler, ihtiyaçlarına göre eklentiler ekleyerek IDE'yi özelleştirebilir.

Çoklu Proje Desteği: Eclipse, aynı anda birden fazla projeyi yönetme imkanı sunar. Bu, büyük ölçekli uygulamalar geliştirirken faydalıdır.

Platform Bağımsızlığı: Eclipse, Windows, macOS ve Linux gibi farklı işletim sistemlerinde çalışabilir.

Eclipse, bu projede geliştirme ortamı olarak kullanılmış
ve Maven projesi Eclipse üzerinde oluşturulmuştur.

JSON Nedir?

JSON (JavaScript Object Notation), veri alışverişi ve depolama için kullanılan hafif bir veri formatıdır. İnsan tarafından okunabilir bir yapıya sahip olan JSON, hem makineler hem de insanlar için kolayca anlaşılabilir bir formattır.

JSON'un temel özellikleri şunlardır:

Hafif ve Okunabilir: JSON, basit bir sözdizimine sahiptir ve hem insanlar hem de makineler tarafından kolayca okunabilir.

Veri Yapısı: JSON, anahtar-değer çiftleri (key-value pairs) ve diziler (arrays) gibi veri yapılarını destekler. Bu, karmaşık veri yapılarının temsil edilmesini sağlar.

Dil Bağımsızlığı: JSON, birçok programlama dili tarafından desteklenir (Java, Python, JavaScript vb.). Bu, farklı sistemler arasında veri alışverişini kolaylaştırır.

Geniş Kullanım Alanı: JSON, API'ler, yapılandırma dosyaları ve veri depolama gibi birçok alanda kullanılır.

Kolay İşleme: JSON, Jackson gibi kütüphanelerle kolayca işlenebilir ve Java nesnelerine dönüştürülebilir.

Bu projede, şehir verileri (duraklar, mesafeler, ücretler, transfer bilgileri) JSON formatında bir dosyada saklanmış ve Jackson kütüphanesi kullanılarak okunmuştur.

Rota Planlama Nedir?

Rota planlama, bir başlangıç noktasından hedef noktasına en uygun ulaşım yolunu belirleme işlemidir. Bu işlem, farklı ulaşım türlerini (otobüs, tramvay, taksi, yürüme) birleştirerek kullanıcıya zaman, maliyet ve aktarma sayısı gibi kriterlere göre optimize edilmiş rotalar sunmayı amaçlar.

Rota planlamanın bazı yaygın kullanım alanları şunlardır:

Şehir İçi Ulaşım: Kullanıcıların şehir içinde en hızlı veya en ekonomik rotayı bulmasına yardımcı olur.

Zaman ve Maliyet Optimizasyonu: Rota planlama, kullanıcıların zaman ve maliyet açısından en uygun seçeneği seçmesini sağlar.

Aktarma Yönetimi: Farklı ulaşım türleri arasında aktarma yaparak daha verimli rotalar oluşturur.

Kullanıcı Deneyimi: Kullanıcılara, ihtiyaçlarına göre özelleştirilmiş rota seçenekleri sunarak ulaşım deneyimini iyileştirir.

Veri Analizi: Ulaşım verilerini analiz ederek, şehir planlama ve trafik yönetimi gibi alanlarda kullanılabilir.

Bu proje, şehir içi ulaşım verilerini işleyerek kullanıcıya en uygun rotayı sunan bir uygulama geliştirmeyi amaçlamıştır.

III. YÖNTEM

A. Yolculuk Planlama Sisteminin Tasarımı

Kullanıcıdan Yolcu Tipi ve İsim Bilgilerinin Alınması: Program başlatıldığında, kullanıcıdan yolcu tipi (öğrenci, yaşlı veya normal) seçmesi istenir. Kullanıcının seçimi kaydedilir ve bu seçim doğrultusunda indirim oranları belirlenir. Ardından, kullanıcıdan adı istenir, girilen isim kaydedilir ve sistemde saklanır.

Taksi Ücretlerinin Sabitlenmesi: Taksi açılış ücreti ve kilometre başına ücret sabit değerler olarak tanımlanır. Bu değerler, taksi adımlarının maliyet hesaplamalarında kullanılmak üzere sistemde tutulur.

Karşılama Mesajı ve Rota Planlama Sürecine Geçiş: Kullanıcıya bir karşılama mesajı gösterilir ve ardından rota planlama sürecine geçilir. Bu aşamada, kullanıcıdan seyahat başlangıç zamanı (örneğin, saat 14:30) istenir. Girilen zaman kaydedilir.

Başlangıç ve Hedef Noktalarının Koordinatlarının Alınması: Kullanıcıdan başlangıç noktasının enlem ve boylam koordinatları istenir. Girilen değerler bir nokta olarak kaydedilir. Aynı şekilde, hedef noktasının enlem ve boylam koordinatları alınır ve bu değerler de bir nokta olarak saklanır.

Cüzdan Bilgilerinin Toplanması: Kullanıcıdan ödeme yöntemlerine ilişkin bilgiler istenir: nakit miktarı, kredi kartı limiti ve kentkart bakiyesi. Bu bilgiler ayrı ayrı kaydedilir ve ödeme işlemlerinde kullanılmak üzere sistemde tutulur.

Şehir Verilerinin JSON Dosyasından Okunması: Şehir verileri bir JSON dosyasından okunur. Bu veriler; durakların kimliklerini, isimlerini, türlerini (otobüs, tramvay vb.), koordinatlarını, sonraki duraklara olan mesafeleri, süreleri, ücretleri ve transfer bilgilerini içerir. Bu veriler, rota planlama sürecinin temelini oluşturur.

B. En Yakın Durakların Belirlenmesi

Başlangıç ve Hedef Noktalarına En Yakın Durakların Bulunması: Başlangıç noktasına en yakın durak, tüm duraklarla başlangıç noktası arasındaki mesafelerin Haversine formülüyle hesaplanmasıyla belirlenir. En küçük mesafeye sahip durak seçilir. Aynı yöntemle, hedef noktasına en yakın durak da bulunur.

Mesafe Kontrolü ve Adım Türünün Belirlenmesi: Başlangıç noktasından en yakın durağa olan mesafe kontrol edilir. Eğer mesafe 3 kilometreden küçükse, bir yürüme adımı eklenir; yürüme süresi mesafeye göre hesaplanır ve maliyet sıfır olarak belirlenir. Mesafe 3 kilometreden büyükse, bir taksi adımı eklenir; taksi maliyeti, açılış ücreti ve kilometre başına ücret üzerinden hesaplanır, taksi süresi ise mesafeye göre belirlenir. Hedef noktasından en yakın durağa olan mesafe için de aynı kontrol yapılır ve uygun adım (yürüme veya taksi) eklenir.

C. Rota Seçeneklerinin Hesaplanması

Farklı Rota Türlerinin Oluşturulur. Beş farklı rota seçeneği hesaplanır:

- Sadece Otobüs Rotası
- Sadece Tramvay Rotası
- Otobüs ve Tramvay Aktarması Rotası
- Taksi ve Toplu Taşıma Kombinasyonu Rotası
- En Az Aktarmalı Rota

Rota Değerlendirme ve Skorlama: Her rota için toplam mesafe, toplam süre, toplam maliyet ve aktarma sayısı hesaplanır. Rotalar, maliyet, süre ve aktarma sayısına dayalı bir skor sistemine göre sıralanır. En yüksek skora sahip rota, en uygun rota olarak kullanıcıya önerilir.

Rota Detaylarının Kullanıcıya Sunulması: Tüm rota seçenekleri kullanıcıya gösterilir. Her rota için toplam maliyet, toplam süre, toplam mesafe, aktarma sayısı ve varış zamanı listelenir. Ayrıca, her rota seçeneğinin detaylı adımları (taksi, otobüs, tramvay veya yürüme) mesafe, maliyet ve tür bilgileriyle birlikte sunulur.

D. Kullanıcı Seçimi ve Ödeme İşlemleri

Rota Seçimi: Kullanıcıdan bir rota seçmesi istenir. Kullanıcı bir sayı girer; eğer geçersiz bir giriş yapılırsa, bir hata mesajı gösterilir ve tekrar giriş yapması talep edilir. Geçerli bir rota seçildiğinde, bu seçim kaydedilir.

Ödeme Yöntemi Seçimi: Kullanıcıdan ödeme yöntemi (nakit, kredi kartı veya kentkart) seçmesi istenir. Seçilen yöntem kaydedilir.

Rota Detaylarının Onayı ve Cüzdan Güncellemesi: Seçilen rotanın detayları (adımlar, başlangıç zamanı, varış zamanı, ödeme yöntemi, toplam maliyet, toplam süre ve aktarma sayısı) kullanıcıya tekrar gösterilir. Seçilen ödeme yöntemine göre cüzdan bakiyesi güncellenir; örneğin, kentkart seçildiyse toplam maliyet kentkart bakiyesinden düşülür.

Güncellenmiş Cüzdan Bilgilerinin Sunulması: Kalan nakit miktarı, kredi kartı limiti ve kentkart bakiyesi kullanıcıya gösterilir. Program bu aşamada sona erer.

IV. YENİ ÖZELLİKLER VE ULAŞIM SİSTEMİNE ENTEGRASYONU

Yeni Ulaşım Yöntemlerinin (Elektrikli Scooter) Sisteme Entegrasyonu:

Elektrikli scooter gibi yeni bir ulaşım yönteminin sisteme eklenmesi, mevcut sistemin hem veri yapılarında hem de hesaplama mantığında bir dizi değişikliği gerektirir. İlk olarak, sistemin veri modeli olan JSON dosyasında, elektrikli scooter durakları, bu duraklar arasındaki mesafeler, süreler ve ücretlendirme bilgileri tanımlanmalıdır. Örneğin, scooter duraklarının koordinatları, kiralama ücretleri ve kilometre başına ücret gibi bilgiler JSON yapısına eklenmelidir. Ardından, sistemin durakları ve ulaşım türlerini temsil eden veri yapıları güncellenmelidir. Şu anda sistem, otobüs ve tramvay duraklarını "type" alanı ile ayırt etmektedir; bu alana "scooter" türü eklenerek yeni ulaşım yöntemi desteklenebilir.

Rota hesaplama mantığında da değişiklikler yapılması gerekir. Mevcut sistem, otobüs ve tramvay için ayrı ayrı rota hesaplama metotlarına sahiptir ("hesaplaSadeceOtobus" ve "hesaplaSadeceTramvay"). Elektrikli scooter için benzer bir metot ("hesaplaSadeceScooter") eklenmelidir. Ayrıca, scooter kullanımının belirli mesafelerle sınırlı olabileceği (örneğin, 5 km'den kısa mesafelerde tercih edilebilir) göz önüne alınarak, rota planlama algoritması bu kısıtlamaları dikkate alacak şekilde güncellenmelidir. Örneğin, başlangıç noktasından en yakın scooter durağına olan mesafe 3 km'den küçükse scooter kullanımı önerilebilir, aksi halde taksi veya diğer ulaşım yöntemleri tercih edilebilir.

Maliyet hesaplama mantığı da scooter için uyarlanmalıdır. Şu anda taksi için açılış ücreti ve kilometre başına ücret üzerinden bir hesaplama yapılmaktadır; scooter için ise kiralama ücreti ve süre bazlı ücretlendirme gibi farklı bir model kullanılabilir. Bu, maliyet

hesaplama fonksiyonlarının ("taksiMaliyetHesapla" gibi) genelleştirilmesi veya yeni bir fonksiyon ("scooterMaliyetHesapla") eklenmesiyle gerçekleştirilebilir.

Otonom Taksi ve Benzeri Yeni Ulaşım Araçlarının Entegrasyonu:

Otonom taksi gibi yenilikçi ulaşım araçlarının sisteme eklenmesi, elektrikli scooter'a benzer bir yaklaşım gerektirir, ancak bazı ek gereksinimler de ortaya çıkar. Otonom taksiler, mevcut taksi sistemine benzer bir şekilde çalışabilir, ancak farklı ücretlendirme modelleri (örneğin, sabit bir ücret veya enerji tüketimine dayalı bir ücret) ve süre hesaplama yöntemleri (otonom araçların trafik kurallarına daha sıkı bağlı olması nedeniyle farklı bir hız varsayımı) gerekebilir. JSON veri yapısında, otonom taksiler için bir "type" değeri (örneğin, "otonom_taksi") tanımlanmalı ve bu araçların kullanılabilir olduğu bölgeler, ücretlendirme bilgileri ve hız varsayımları eklenmelidir.

Rota hesaplama mantığında, otonom taksi için mevcut taksi hesaplama mantığı ("hesaplaSadeceTaksi") temel alınabilir, ancak otonom taksilerin belirli bölgelerde veya mesafelerde (örneğin, şehir merkezinde) daha uygun olabileceği bir filtreleme mekanizması eklenebilir. Ayrıca, otonom taksilerin kullanımına ilişkin ek kısıtlamalar (örneğin, yalnızca belirli saatlerde kullanılabilir olması) hesaba katılmalıdır. Bu, rota planlama algoritmasının genelleştirilmesi ve yeni bir ulaşım türü için ek koşullar eklenmesiyle sağlanabilir.

Değişiklik Gerektiren Fonksiyonlar:

Yeni ulaşım yöntemlerinin eklenmesi, sistemdeki birkaç temel fonksiyonu doğrudan etkiler. İlk olarak, "RotaPlanlayici" sınıfındaki rota hesaplama metotları ("hesaplaSadeceOtobus", "hesaplaSadeceTramvay", "hesaplaOtobusTramvayAktarma", "hesaplaSadeceTaksi" ve "enAzAktarmaliRotaHesapla") güncellenmelidir. Bu metotlar, yeni ulaşım türlerini desteklemek için ya genelleştirilmeli ya da her yeni ulaşım türü için ayrı bir metot eklenmelidir. Örneğin, elektrikli scooter için "hesaplaSadeceScooter" ve otonom taksi için "hesaplaSadeceOtonomTaksi" gibi metotlar yazılabilir.

Ayrıca, "MesafeHesaplayici" sınıfındaki mesafe ve süre hesaplama mantığı, yeni ulaşım yöntemlerinin hız ve erişim özelliklerine göre uyarlanmalıdır. Örneğin, elektrikli scooter'ların ortalama hızı taksiden daha düşük olabilir, bu nedenle süre hesaplama formülü buna göre ayarlanmalıdır. "RotaSonuc" sınıfındaki adım ekleme fonksiyonları ("ekleYurume", "ekleTaksi") de yeni ulaşım türleri için genelleştirilmeli veya yeni fonksiyonlar ("ekleScooter", "ekleOtonomTaksi") eklenmelidir. Son olarak, maliyet hesaplama fonksiyonları ("taksiMaliyetHesapla") her yeni ulaşım türü için uyarlanmalıdır.

Açık/Kapalı Prensibi Doğrultusunda Yeni Ulaşım Araçlarının Entegrasyonu:

Yeni bir ulaşım aracı entegrasyonu için aşağıdaki adımlar takip edilebilir:

Yeni bir ulaşım aracı eklendiğinde, mevcut kod tabanına bu araca özel metotlar ya da fonksiyonlar doğrudan eklenebilir. Örneğin, bir "elektrikli scooter" eklemek istiyorsanız, diğer araçlar için yapılana benzer şekilde "scooterRotaHesapla", "scooterMaliyetHesapla" gibi metotlar "RotaPlanlayici" sınıfına ya da ilgili modüle dahil edilebilir. Bu metotlar, scooter'a özgü hesaplama mantığını (mesafe, hız, maliyet vb.) içerecek şekilde yazılır ve sistemin mevcut işleyişine entegre edilir.

Koda yapılabilecek birkaç değişiklik ile yeni araç eklenmiş olur.. Yeni aracı desteklemek için gerekli parametreler (örneğin, hız, ücret tarifesi) tanımlanır ve ilgili hesaplama mantığı, diğer araçlarla aynı düzende implemente edilir. Böylece, sistemin mevcut işleyişi bozulmadan, yeni ulaşım yöntemi çalışır hale getirilebilir. Bu yöntem, mevcut kod tabanına hızlı bir şekilde uyum sağlar ve ekleme süreci diğer araçlarla tutarlı bir şekilde ilerler.

65 Yaş ve Üzeri Bireyler İçin Ücretsiz Seyahat Sınırlandırması:

65 yaş ve üzeri bireyler için ücretsiz seyahat hakkının 20 seyahat ile sınırlandırılması, sisteme sonradan eklenebilecek bir özelliktir. Bu değişiklik, öncelikle kullanıcı verilerinin saklandığı bir mekanizma gerektirir. Şu anda sistem, kullanıcıların cüzdan bilgilerini (nakit, kredi kartı, Kentkart bakiyesi) tutuyor. Bu nedenle, kullanıcı profillerine yaş bilgisi ve ücretsiz seyahat hakkı kullanım sayısını takip eden bir sayaç eklenmelidir.

Bu özellik için, bir "Kullanici" sınıfı oluşturulabilir ve bu sınıf içinde yaş, ücretsiz seyahat hakkı limiti (20) ve kullanılan ücretsiz seyahat sayısı gibi alanlar tanımlanabilir. Her rota hesaplandığında, kullanıcının yaşı kontrol edilmeli ve eğer 65 yaş ve üzerindeyse, ücretsiz seyahat hakkı limiti aşılmamışsa ücret alınmamalıdır. Bu kontrol, "RotaPlanlayici" sınıfındaki maliyet hesaplama mantığında yapılabilir. Örneğin, "toplamMaliyet" hesaplanmadan önce kullanıcının ücretsiz seyahat hakkı durumu kontrol edilerek maliyet sıfırlanabilir.

Bu sınırlama, "RotaSonuc" sınıfındaki maliyet hesaplama ve ödeme işlemlerini de etkileyecektir. Ödeme yöntemi seçildiğinde, eğer kullanıcı ücretsiz seyahat hakkını kullanıyorsa, Kentkart bakiyesinden düşüş yapılmamalı ve ücretsiz seyahat sayacı bir artırılmalıdır. Ayrıca, bu bilgilerin kalıcı olarak saklanması için bir veritabanı veya dosya sistemi entegrasyonu gerekebilir, böylece kullanıcı her giriş yaptığında ücretsiz seyahat sayacı sıfırlanmaz.

Sınıf ve Fonksiyonların Etkilenmesi:

Bu sınırlama, öncelikle oluşturulacak "Kullanici" sınıfını ve "RotaPlanlayici" sınıfını etkileyecektir. "RotaPlanlayici" sınıfındaki "rotaHesapla" ve "maliyetHesapla" gibi metotlar, kullanıcının yaşını ve ücretsiz seyahat hakkını kontrol edecek şekilde güncellenmelidir. Ayrıca, "RotaSonuc" sınıfındaki ödeme işlemleri ("odemeYap" gibi bir metot varsa) ücretsiz seyahat hakkını dikkate alacak şekilde düzenlenmelidir. Eğer sistemde bir veritabanı entegrasyonu yoksa, "Kullanici" sınıfı içinde ücretsiz seyahat sayısını saklamak ve güncellemek için ek metotlar yazılmalıdır.

SONUÇ

Bu proje, kullanıcıların bir başlangıç noktasından hedef noktasına en uygun ulaşım rotasını bulmasını sağlayan bir uygulama geliştirmeyi amaçlamıştır. Eclipse üzerinde bir Maven projesi olarak oluşturulan bu uygulama, Jackson kütüphanesi kullanılarak JSON formatındaki şehir verilerini okumuş ve işlemiştir. Proje, ödevde gerektiği şekilde paketler, sınıflar ve arayüzler oluşturularak yapılandırılmıştır. Farklı ulaşım türlerini (otobüs, tramvay, taksi, yürüme) birleştiren rotalar hesaplanmış ve kullanıcıya zaman, maliyet ve aktarma sayısı gibi kriterlere göre optimize edilmiş seçenekler sunulmuştur. Gelecekte yapılacak iyileştirmelerle, uygulamanın daha da geliştirilmesi ve daha fazla özellik eklenmesi mümkündür.

KAYNAKÇA

[1] "JSON Veri Formatı," [Çevrimiçi]. Erişim: https://www.json.org/, [Erişim: 3 Nisan 2025].

[2] "Dijkstra Algoritması," [Çevrimiçi]. Erişim: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm, [Erişim: 3 Nisan 2025].

[3] "Haversine Formülü," [Çevrimiçi]. Erişim: https://en.wikipedia.org/wiki/Haversine_formula, [Erişim: 3 Nisan 2025].

Yalancı Kod

```
    // 1. Veri ve başlangıç ayarları
    transport_verisi = JSON_dosyadan_oku("transport.json")  // Şehir
verisi (duraklar, taksi ücretleri)
    yolcu = yolcu_tipini_seç()  // Öğrenci, Yaşlı, Normal
    rota_planlayici = yeni RotaPlanlayici(transport_verisi, yolcu)

    // 2. Kullanıcıdan giriş bilgileri al
    baslangic_zamani = kullanıcıdan_zaman_al("HH:mm formatında
başlangıç zamanı girin")
    baslangic_enlem = kullanıcıdan_sayı_al("Başlangıç enlemi girin")
    baslangic_boylam = kullanıcıdan_sayı_al("Başlangıç boylamı girin")
    hedef_enlem = kullanıcıdan_sayı_al("Hedef enlemi girin")
    hedef_boylam = kullanıcıdan_sayı_al("Hedef boylamı girin")

    // 3. Cüzdan bilgilerini al
    cuzdan = yeni Cüzdan(
        nakit = kullanıcıdan_sayı_al("Nakit miktarını girin"),
        kredi_kartı = kullanıcıdan_sayı_al("Kredi kartı limitini girin"),
        kentkart = kullanıcıdan_sayı_al("Kentkart bakiyesini girin")
    )

    // 4. En yakın durakları bul
    baslangic_durak_sonuc =
rota_planlayici.en_yakin_duragi_bul(baslangic_enlem,
baslangic_boylam)
    hedef_durak_sonuc =
rota_planlayici.en_yakin_duragi_bul(hedef_enlem, hedef_boylam)

    baslangic_durak = baslangic_durak_sonuc.durak
    baslangic_mesafe = baslangic_durak_sonuc.mesafe
    hedef_durak = hedef_durak_sonuc.durak
    hedef_mesafe = hedef_durak_sonuc.mesafe

    // 5. Rota seçeneklerini hesapla
    rota_secenekleri = boş_liste()

    // 5.1 Sadece Otobüs
    sadece_otobus = hesapla_sadece_otobus(baslangic_durak,
hedef_durak, baslangic_mesafe, hedef_mesafe)
    EĞER sadece_otobus != null İSE
        rota_secenekleri.ekle(sadece_otobus)

    // 5.2 Sadece Tramvay
    sadece_tramvay = hesapla_sadece_tramvay(baslangic_durak,
hedef_durak, baslangic_mesafe, hedef_mesafe)
    EĞER sadece_tramvay != null İSE
        rota_secenekleri.ekle(sadece_tramvay)

    // 5.3 Otobüs + Tramvay Aktarma
    otobus_tramvay = hesapla_otobus_tramvay_aktarma(baslangic_durak,
hedef_durak, baslangic_mesafe, hedef_mesafe)
    EĞER otobus_tramvay != null İSE
        rota_secenekleri.ekle(otobus_tramvay)

    // 5.4 Taksi + Toplu Taşıma Kombinasyonu
    taksi_kombinasyon = hesapla_taksi_kombinasyon(baslangic_durak,
hedef_durak, baslangic_mesafe, hedef_mesafe)
    EĞER taksi_kombinasyon != null İSE
        rota_secenekleri.ekle(taksi_kombinasyon)

    // 5.5 Sadece Taksi
    sadece_taksi = hesapla_sadece_taksi(baslangic_enlem,
baslangic_boylam, hedef_enlem, hedef_boylam)
    rota_secenekleri.ekle(sadece_taksi)

    // 5.6 En Az Aktarmalı
    en_az_aktarmali = hesapla_en_az_aktarmali(baslangic_durak,
hedef_durak, baslangic_mesafe, hedef_mesafe)
    EĞER en_az_aktarmali != null İSE
        rota_secenekleri.ekle(en_az_aktarmali)

    // 6. Skorları hesapla ve en iyi rotayı bul
    HER rota İÇİN rota_secenekleri İÇİNDE
        rota.skor_hesapla()  // Maliyet, süre ve aktarma sayısına göre skor
    en_iyi_rota = rota_secenekleri.içinden_minimum_skorlu_rota_seç()

    // 7. Tüm rotaları göster
    YAZDIR("Tüm Rota Seçenekleri:")
    HER rota İÇİN rota_secenekleri İÇİNDE
        YAZDIR(rota.tipi, rota.maliyet, rota.süre, rota.mesafe,
rota.aktarma_sayısı, rota.varış_zamanı)
        YAZDIR(rota.adımlar)

    // 8. Kullanıcıdan rota seçimi al
    secilen_rota = kullanıcıdan_rota_seç(rota_secenekleri)

    // 9. Ödeme yöntemini seç ve kontrol et
    odeme_yontemi = kullanıcıdan_odeme_yontemi_seç("Nakit, Kredi
Kartı, Kentkart")
    EĞER odeme_yontemi_geçerli_değil(cuzdan, secilen_rota.maliyet)
İSE
        YAZDIR("Yeterli bakiye yok, başka yöntem seçin")
        odeme_yontemi = kullanıcıdan_odeme_yontemi_seç("Nakit, Kredi
Kartı, Kentkart")
        EĞER odeme_yontemi_geçerli_değil(cuzdan, secilen_rota.maliyet)
İSE
            YAZDIR("Seyahat planlanamadı")
            BİTİR

    // 10. Ödemeyi yap ve sonucu göster
    cuzdan = odeme_yap(cuzdan, secilen_rota.maliyet, odeme_yontemi)
    YAZDIR("Seçilen Rota:", secilen_rota.adımlar)
    YAZDIR("Başlangıç Zamanı:", baslangic_zamani)
    YAZDIR("Varış Zamanı:", secilen_rota.varış_zamanı)
    YAZDIR("Toplam Maliyet:", secilen_rota.maliyet)
    YAZDIR("Kalan Bakiye:", cuzdan.nakit, cuzdan.kredi_kartı,
cuzdan.kentkart)

BİTİR

// Rota Hesaplama Fonksiyonu (Örnek: Sadece Otobüs)
FONKSİYON hesapla_sadece_otobus(baslangic_durak, hedef_durak,
baslangic_mesafe, hedef_mesafe)
    EĞER basin baslangic_durak ve hedef_durak otobüs değilse İSE
        DÖN null
    sonuc = yeni RotaSonuc("Sadece Otobüs")
    EĞER baslangic_mesafe <= 3 İSE
        sonuc.ekle_yürüme(baslangic_mesafe, "Başlangıçtan durağa
yürüme")
    DEĞİLSE
        sonuc.ekle_taksi(baslangic_mesafe,
taksi_maliyet(baslangic_mesafe), "Başlangıçtan durağa taksi")
    rota = rota_planlayici.rota_hesapla(baslangic_durak.id, hedef_durak.id)
    HER durak ÇİFTİ İÇİN rota.duraklar İÇİNDE
        mesafe = haversine(durak1.lat, durak1.lon, durak2.lat, durak2.lon)
        maliyet = yolcu.ucret_hesapla(durak1.nextStops.ucret)
        sonuc.adım_ekle(mesafe, maliyet, "Otobüs: durak1 -> durak2")
    EĞER hedef_mesafe <= 3 İSE
        sonuc.ekle_yürüme(hedef_mesafe, "Duraktan hedefe yürüme")
    DEĞİLSE
        sonuc.ekle_taksi(hedef_mesafe, taksi_maliyet(hedef_mesafe),
"Duraktan hedefe taksi")
    sonuc.toplam_süre = rota.toplam_süre
    DÖN sonuc
BİTİR

// Taksi Maliyet Hesaplama
FONKSİYON taksi_maliyet(mesafe)
    DÖN taksi.açılış_ücreti + (mesafe * taksi.km_başına_ücret)
BİTİR
```