**Kocaeli Üniversitesi**

**Bilgisayar Mühendisliği Bölümü**

**Yazılım Laboratuvarı I**

**Kargo Dağıtım Sistemi**

*Oğuz Narlı Alperen Tan*

[*180201074@kocaeli.edu.tr*](mailto:180201074@kocaeli.edu.tr)[*190201054@kocaeli.edu.tr*](mailto:190201054@kocaeli.edu.tr)

**Giriş:**

Yazılım laboratuvarı 1. Proje kapsamında bizden kullanıcıların uygulama üzerinden kargo ekleyebilen, silebilen, teslim edebilen ve kargoları teslim etmek için en kısa yol oluşturan bir program geliştirmemiz beklenmektedir. Projemiz C# diliyle, bulut üzerindeki MySQL veri tabanı yönetim sistemi kullanılarak, Postman arayüzü üzerinden kontrollerini yaptığımız ve kendi yazdığımız bulut üzerindeki API ile veri tabanının haberleşmesi sağlanarak geliştirilmiştir. Ayrıca harita işlemleri için GMap paketi ve GoogleMap API'si kullandık. Kısa yol algoritması için Gezgin Satıcı Problemini tercih ettik.

**Özet:**

Uygulamaya üye olan kullanıcı (kargo görevlisi), müşterinin ismi ve adresiyle kargo oluşturabilir, var olan kargoların takibini yapabilir, silebilir ve teslim edilme durumunu değiştirebilir. Bu kargoların teslimat yolu en kısa mesafe baz alınarak haritada gösterilir. Haritaya API arayüzünden eklenen teslimat şubeleri, teslim edilen kargolar ve teslim edilmeyen kargoları farklı renk işaretçi ile gösterdik.

**Veri Tabanı:**

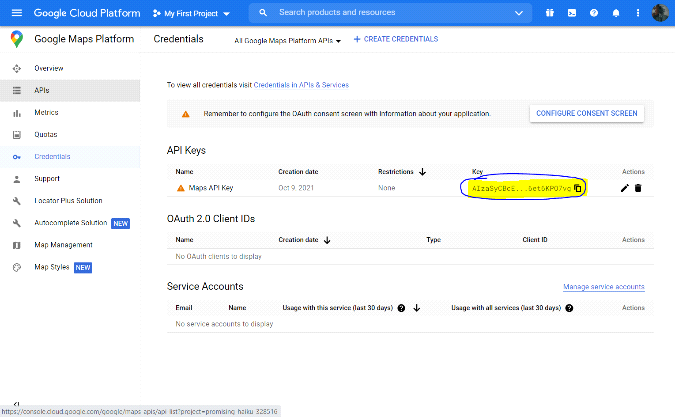
Google Cloud Platform üzerinden oluşturduğumuz MYSQL veri tabanına MYSQL workbench üzerinden uzaktan IP adresiyle bağlanarak 3 adet table (users,deliveries,deliveryPoints) oluşturduk. User tablosunda uygulamaya kayıt olan kullanıcıların kullanıcı ID’sini, kullanıcı adını, şifresini, adresini (enlem ve boylam değerleri) tuttuk. Ardından deliveries tablosunda oluşturulan kargoların kargo ID’sini, müşteri ismini, kargo lokasyonunu, lokasyonun enlem ve boylam değerlerini ve son olarak kargonun teslim edilme durumunu tuttuk. Son olarak deliveryPoints tablosunda teslimat şubelerinin ID’sini, ismini, lokasyonunu ve enlem-boylam değerlerini tuttuk.

**API:**

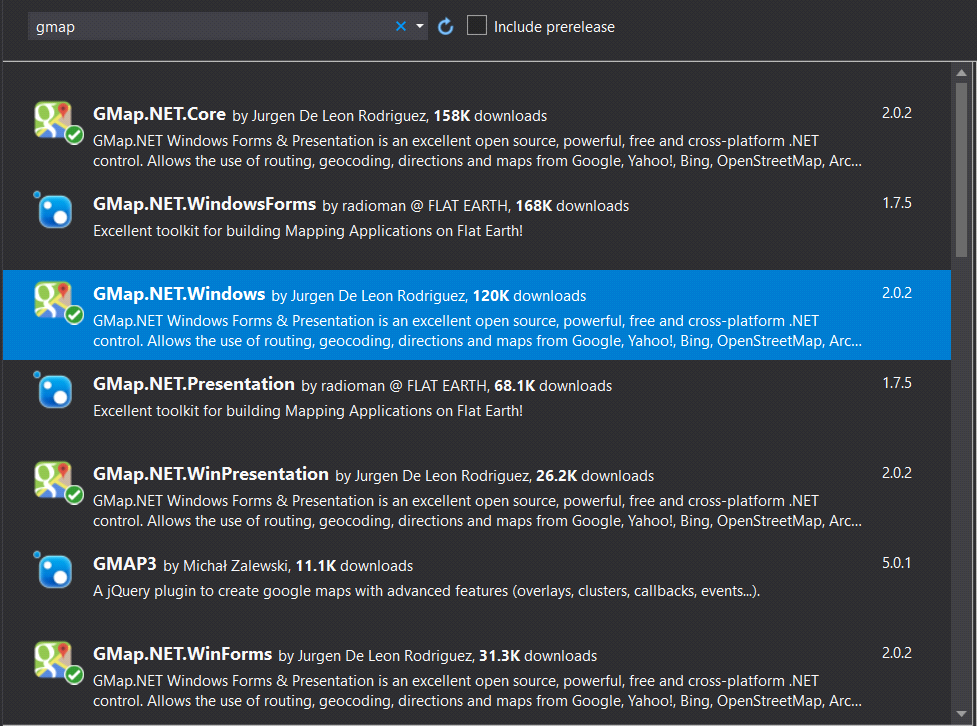
Visual Studio üzerinden ASP.net Web API projesi oluşturarak API yazmaya başladık. API kodlarını, API’nin sadece JSON metin biçimi kullanarak çalışacak şekilde düzenledik. API’de kullanacağımız veri tabanındaki her bir veri tipi için get set metodu oluşturduk. Appsettings kısmında API ile veri tabanı bağlantısını sağladık. Ardından her bir API metodu için controller classı oluşturduk. Örneğin userController.cs classında API/users/ kısmının GET ile veri tabanından veri çekilmesini, POST ile veri tabanına veri aktarılmasını, PUT ile veri tabanının düzenlenmesini ve güncellenmesini, DELETE ile veri tabanından veri silinmesi işlemlerini SQL komutlarını veri tabanına API üzerinden göndererek gerçekleştirdik. GET isteğini kullanıcının uygulamaya giriş yaparken ve şifresini değiştirirken, POST isteğini yeni kullanıcı oluştururken, PUT isteğini kullanıcı şifresini değiştirirken, DELETE isteğini kullanıcı silerken kullanacağımızı varsayarak oluşturduk. Aynı işlemleri deliveriesController ve deliveryPointsController sınıfları için de veri tabanına uyumlu şekilde uyguladık. Yazdığımız API’nin çalışabilirliğini localhost üzerinden Postman arayüzü ile kontrol ettikten sonra Visual Studio’yu kullanarak Azure bulut platformuna aktarıp yayınladık.

**Form1.cs ile Google Map bağlantısı:**

Haritalar için projemizde kullandığımız yapı GMap’tir. GMap C#’ın sağladığı harita işlemlerinin yapılabileceği bir pakettir. Bu API ile beraber istediğimiz harita hizmeti sağlayan sistemlerini (Google Map,Yandex Haritalar...) kullanmamızı izin verir. Biz çalışmamızda Google Map kullandı. Ancak Google Map kullanılması için Google Cloud hizmetinden API key alınması gerekir. Bunun için Google Cloud Map Platform’a gidip ilk olarak hesap oluşturuldu. Ardından bulut içerisinde Map için proje oluşturulduktan sonra projeye özel bir API key oluşmaktadır.



Bu API key’i uygulamada kullanmak için GMap paketini kurmamız gerekmekte. Bunun için ilk olarak VS Community 2019 idesinde project->Nutget Package Manager bölümünde GMap paketini aratıldı.



Ardından paketi projeye kuruldu. Kurulan paket winform yapısında harita için araç sağladığından toolbox’ı yenilendi. Böylece arayüze haritayı da ekleyebilir hale geldi. Bunu da project-> refresh toolbox item seçeneği ile gerçekleştirebilmekte. Şimdi ise API’yi proje ile haberleştirmemiz gerektirmekte. Bunun için Form1 kısmındaki Contructer metodu içerisinde tanımlayarak GoogleMap Api’si kullanıldı.



Ayrıca GoogleMap Api kullanarak yapılan her işlemi Google Cloud servisi üzerinde bize chart olarak izlenmesi sağlanmaktadır. Bunun için Cloud hizmetinde projenin Metrics bölümünde incelenebilir.

**Form1.cs:**

**-Kargo oluşturma paneli:**

Bu panel içerisinde kullanıcı kargo eklemektedir. Panel içesinde yer alan araçlar müşteri konumu seçimi için Gmap API’ sinin sağladığı harita, kullanıcının elle adres girebildiği richtext box, kullanıcın haritadan mı yoksa elle giriş yapıp yapamayacağı ile ilgili seçim yaptığı checkbox, müşteri ismi eklediğimiz textbox ve ekleme işlemi gerçekleşmesi için ekle butonu yer almaktadır. Kullanıcı kargo eklendiğinde veri tabanında müşterinin adresi, enlem boylam bilgisi ve ismi tutulmaktadır. Kullanıcı haritadan seçim yaptığından API adres bilgisini bize sunabilmektedir. Eğer elle adres girmeyi tercih ederse aynı şekilde adres alanında girilen anahtar kelimelere göre koordinatları elde edilebilmektedir. Ancak giriş formatı her anahtar kelime arası “, ” olacak şekilde olmalıdır. Kullanıcı kargo eklediği esnada 2. arayüz açık ise thread işlemi ile eş zamanlı olarak kargo araç rotası güncellenebilmektedir.

**-Kargo Takip Paneli:**

Bu panel içerisinde kullanıcı bütün müşterilerini görebilmeyi, kargo iptalini ve teslim bilgisini değiştirmeyi amaçlamaktadır. Kullanıcılar datagrid aracı içerisinde görüntülenmektedir. Datagrid üzerinde satır seçerek gerçekleştirmek istediğiniz işlem ile ilgili butona basarak kargo durumunu değiştirebilirsiniz. İptal butonuna basarsanız seçtiğiniz kargo silinecektir. Eğer ikinci ekran açıksa eş zamanlı olarak kısa yol güncellemesi de yapacaktır. Ancak teslim edildi butonuna basarsa kargo durumu güncellenecek ve eğer ki 2. arayüz açıksa thread işlemi gerçekleşerek tekrar rota hesabı yapılacak. Ancak rota hesaplamaya kuryenin geldiği nokta teslim edildi olarak değiştirdiğimiz nokta olacağından hesaplamaya oradan başlayacak.

**Form1.cs ile API bağlantısı:**

Form1 tasarımında oluşturduğumuz input alanlarından text verisini çekip bir stringe aktardıktan sonra ilgili async fonksiyonların (tuşa basma, thread vs.) içinde HttpClient ve RestClient oluşturup bağlantı adresini Azure bulut servisine yerleştirdiğimiz API’nin adresi (\*\*\*\*.azurewebsites.net) şeklinde ayarladık. API’nin hangi metodu kullanacağını belirledikten sonra (ör: api/users) göndereceğimiz isteği yazıp, gelen yanıtı kontrol ederek fonksiyonların tamamlanmasını sağladık. Eğer gelen yanıt olumlu değilse kullanıcının veri girişinde hata olduğunu varsayıp bir mesaj kutusu ile kullanıcıyı bilgilendirdik. Olumlu yanıtlarda ise yine kullanıcıyı bilgilendirmek amacıyla mesaj kutusu kullandık. Arayüzdeki hemen hemen bütün butonlar için yukarıdaki şekilde API bağlantısı sağladık.

**Form2.cs:**

Bu arayüz ekranında kuryenin rotasını, müşterinin kargosunun verebileceği şubeleri ve teslim edilmiş kargoların konumunu kullanıcıya gösterdik. Ekran açıldığında arka planda yani yazılım kısmında “Gezgin Satıcı Problemi” kullanarak en kısa rota hesabı yaptık. “Gezgin Satıcı Problemi” kuryenin gidebileceği tüm yolların çıkartılıp en kısa yolun seçilmesine dayanan bir algoritmadır. Ayrıca istenilen noktaların haritada işaretlenmesi de sağlanmaktadır. Yeşil renk işaret kullanıcının ya da kuryenin ilk çıkış noktasını, mavi işaret veri tabanında elle eklenmesi istenen kendi yazdığımız API tarafından çekilmiş şubeleri, sarı işaret teslim edilmiş kargoları ve kırmızı renk teslim edilecek kargoları temsil etmektedir. Mevcut arayüz açıkken eğer müşteri durumu güncellenirse thread işlemi ile eş zamanlı olarak rota durumu güncellenebilmektedir.

**Kısa yol belirlemede kullanılan algoritma (Gezgin Satıcı):**

Seyyar satıcı problemi [yöneylem araştırması](https://tr.wikipedia.org/wiki/Y%C3%B6neylem_ara%C5%9Ft%C4%B1rmas%C4%B1) ve [teorik bilgisayar bilimi](https://tr.wikipedia.org/wiki/Teorik_bilgisayar_bilimi) alanlarında incelenen bir "kombinatorik optimizasyon" problemidir.

Seyyar satıcı problemi şu şekilde tanımlanabilir:

* Bir seyyar satıcı var;
* Bu satıcı, mallarını n şehirde satmak istiyor;
* Öte yandan, mantıklı bir şekilde, bu satıcı bu şehirleri mümkün olan en kısa şekilde ve her bir şehre maksimum bir kere uğrayarak turlamak istiyor.

Problemin amacı, satıcıya bu en kısa yolu sunabilmektir. Algoritmanın çözümlemesi ise;

* Başlangıç için seçebileceği n değişik şehir vardır
* İlk şehirde, satıcının n-1 değişik şehir arasında seçim hakkı vardır
* İkinci şehirde, satıcının n-2 değişik şehir arasında seçim hakkı vardır
* vs.

Dolayısıyla, sonuç olarak satıcının n! kadar değişik tur arasından seçim hakkı olacaktır. Amaç bu seçimler arasında en kısa yolu belirlemektir.

Bu yapıyı kurmak için yazılımda ilk olarak gerçekleştirilen şey, aracın gideceği her nokta için düğüm sınıfı tanımlanmasıdır. Düğümün içerdiği özellikler ise id’si, koordinatları, komşulukları, teslim edilip edilmeme durumu ve kullanıcı olup olmadığıdır. Komşuluk için ise aralarını uzaklıklarını belirten bir değişken ve komşu düğümün kendisinin tanımlı olduğu bir sınıf tanımlandı. Sonrasında gerçekleştirilen işlem her düğüm için komşu düğümlerini ve onlara olan uzaklığı tanımladı. Devamında ise Gezgin Satıcı Problemi için gerekli algoritmalar oluşturuldu. Bunun için ilk olarak tüm seçimleri türetebilmek için Heap Algoritmasını kullanıldı. Algoritma her seçimi düğümlerin içerisinde yer alan id’lere göre türetmektedir. Her türetildiğinde ise kısa yol hesabı yapılmaktadır. Eğer yol türetildiğinde daha önce seçilmiş yolda daha kısaysa yeni en kısa yol olarak güncellenir. En sonunda ise seçilmiş en kısa rotaya göre harita üzerinde rota çizdirilir.

**Sonuç:**

Bu projede C# programlama diliyle temel kargo işlemleri yapabilen bir uygulama oluşturmayı, gezgin satıcı algoritmasıyla kısa yol hesaplaması yapmayı, bulut üzerinde veri tabanı ve API oluşturmayı kodlayarak öğrendik.

**Yalancı kod:**

API:

JSON metin biçimi kullanacak şekilde ayarla

Veri tipleri için get/set metodları oluştur

Veri tabanı bağlantısı sağla

Controller classları oluştur

Her bir tablo için ayrı oluşturulan controller classları içinde get/post/put/delete istekleri için veri tabanına uygun SQL komutları yardımıyla fonksiyon oluştur

Yazdığın API’yi Azure bulut platformuna yayınla

Form1.cs:

Google map API bağlantılarını yap

//Buton tıklama fonksiyonu

User input verisini çek, stringe aktar

Yeni http veya restclient oluştur

API adresini yaz

API metodunu yaz

Verileri çek veya gönder

Gelen verileri işleyip kullanıcıya yanıt ver

//Yukarıdaki fonksiyonu bütün buton fonksiyonları için uygula

Gezgin satıcı algoritması:

List<Node> path;

double minDistance;

func getMin(List<Node> a, int n):

double dis = 0;

List<Node> proccess = new List<Node>();

proccess.Add(fs);

for (int i = 0; i < n; i++):

proccess.Add(a[i]);

for (int i = 0; i < n; i++):

for (int j=0;j< proccess[i].neghs.count;j++):

if (proccess[i].neghs[j].neg.id == proccess[i + 1].id):

dis += j.len;

if (minDs == 0 || dis < minDs):

path = proccess;

minDs = dis;

func travellingSalesman(List<Node> a, int nodeSize, int nodeSize2):

if(nodeSize==1):

getMin(a,n)

for (int i = 0; i < size; i++):

travellingSalesman(a, size - 1, n);

if (size % 2 == 1):

Node temp = a[0];

a[0] = a[size - 1];

a[size - 1] = temp;

else:

Node temp = a[i];

a[i] = a[size - 1];

a[size - 1] = temp;

Form2.cs:

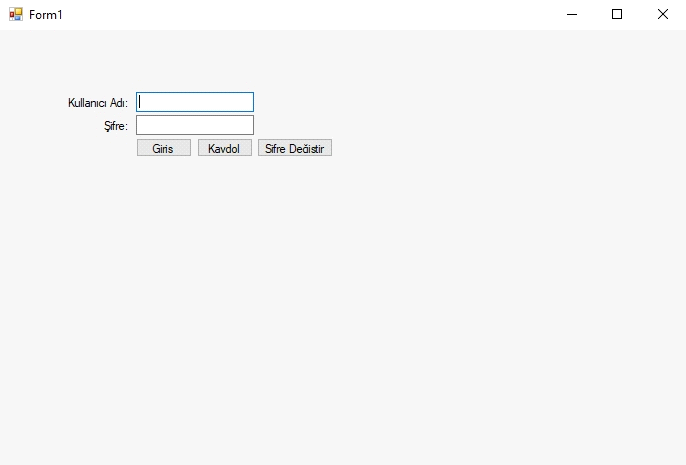
func cargoRoute():

setNeighbor();

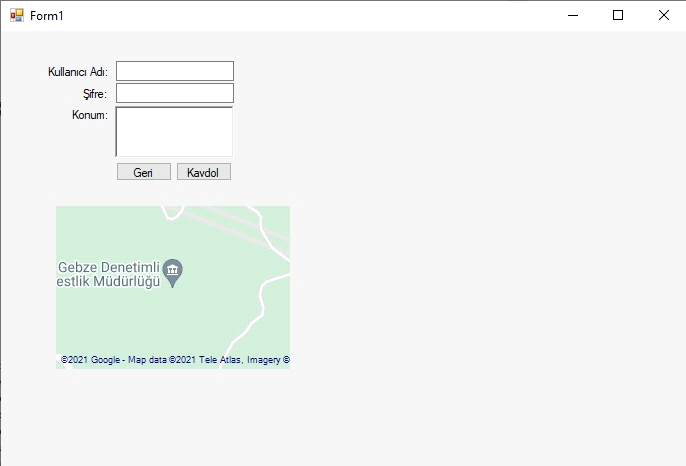
travellingSalesman(allNode,allNode.count,allNode.count);

getRoute();

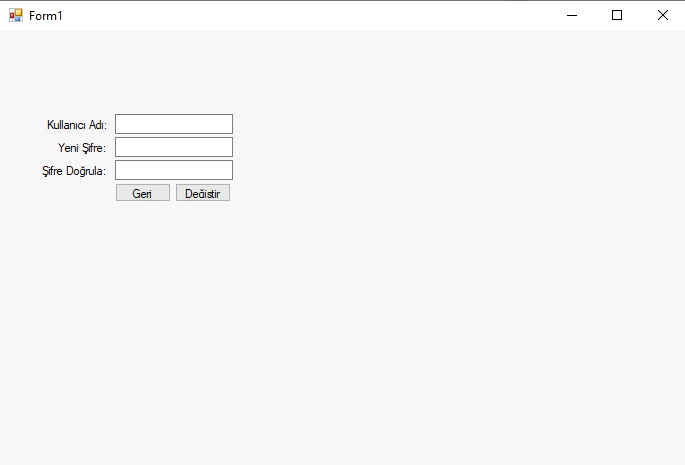
**Deneysel Sonuçlar:**



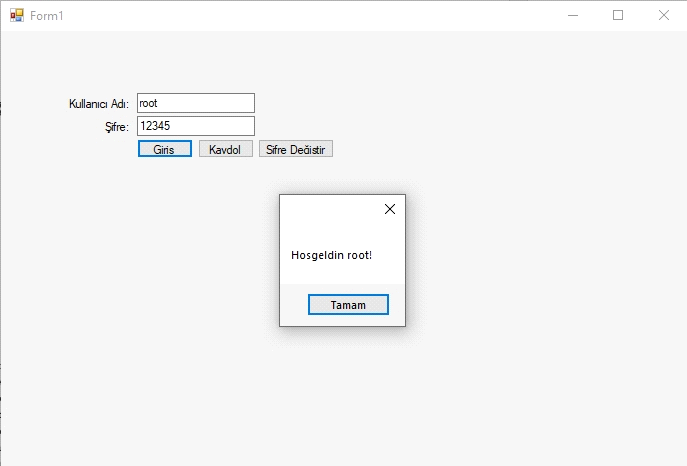
*Kullanıcı giriş ekranı*



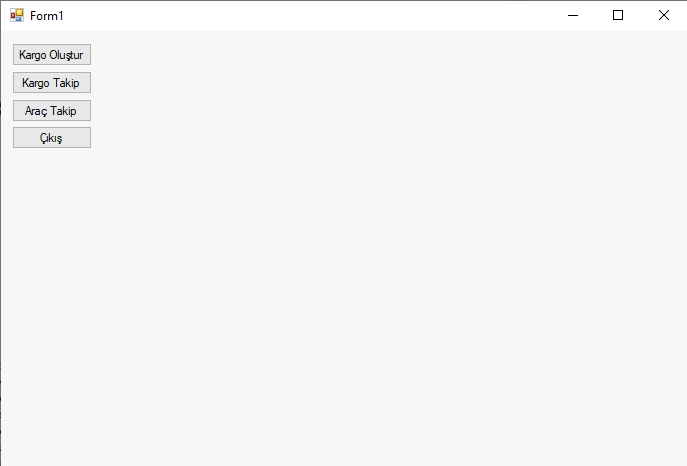
*Kullanıcı kayıt olma ekranı*



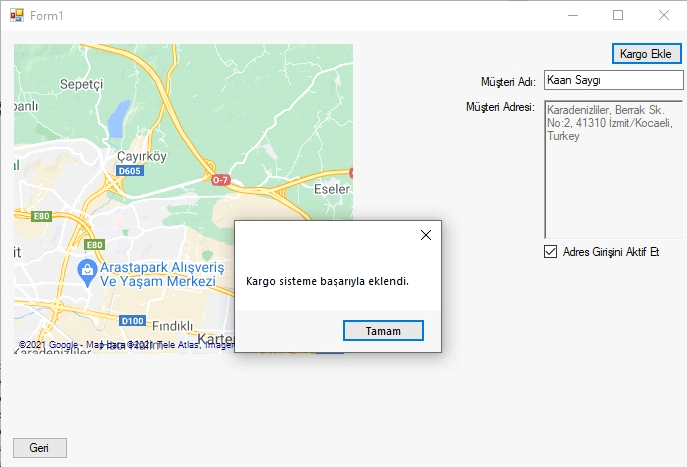
*Kullanıcı şifre değiştirme ekranı*



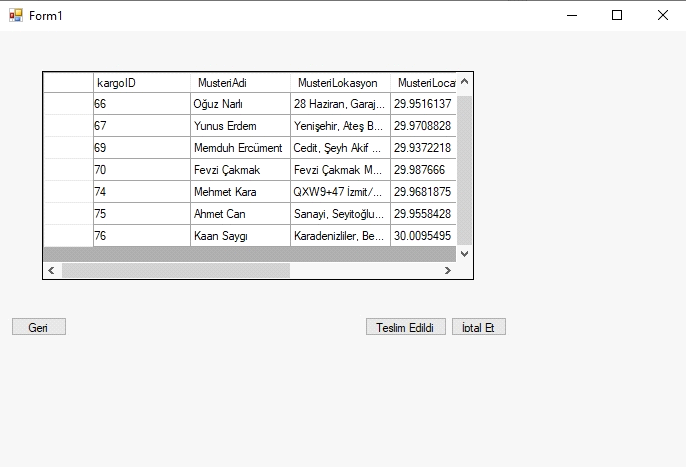
*Başarılı giriş uyarısı*



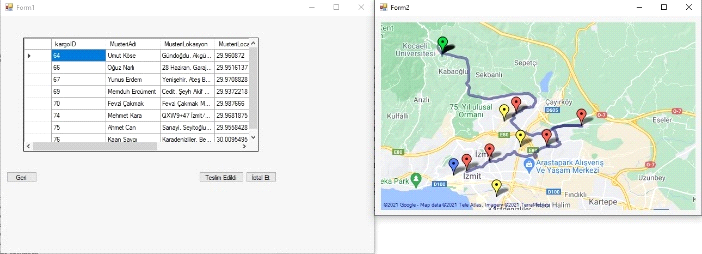
*Uygulama ana ekran arayüzü*



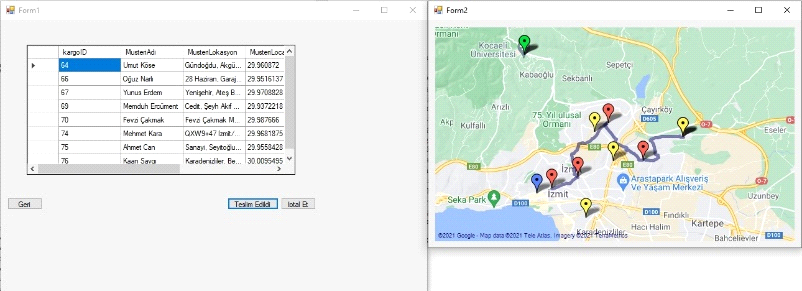
*Kargo oluşturma ekranı*



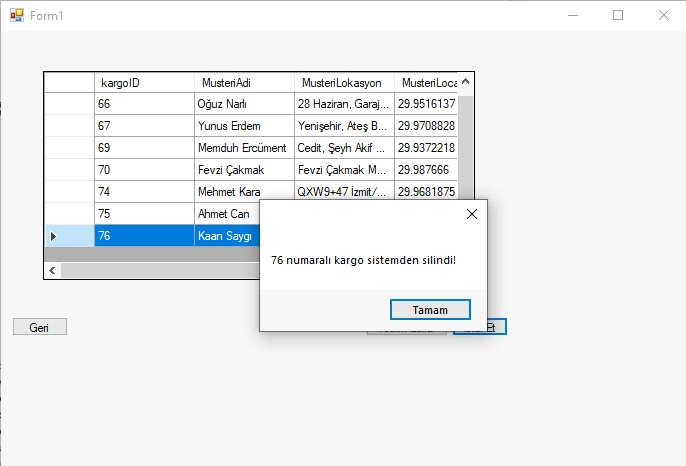
*Kargo takip ekranı*



*Kargo aracı takip ekranı*



*Kargo teslim edildikten sonra haritadaki değişiklik*



*Kargo silme uyarısı*

**Kaynakça:**

[*https://stackoverflow.com/questions*](https://stackoverflow.com/questions)

[*https://www.youtube.com/watch?v=Br8H-TuSSO8*](https://www.youtube.com/watch?v=Br8H-TuSSO8)

[*https://www.youtube.com/watch?v=dScLxUQUK8I*](https://www.youtube.com/watch?v=dScLxUQUK8I)

[*https://www.youtube.com/watch?v=31dtFLQykLw*](https://www.youtube.com/watch?v=31dtFLQykLw)

[*https://www.youtube.com/watch?v=PwH5sc-Q\_Xk*](https://www.youtube.com/watch?v=PwH5sc-Q_Xk)

[*https://www.youtube.com/watch?v=ZiJkEAfvFUc*](https://www.youtube.com/watch?v=ZiJkEAfvFUc)

[*http://edestek2.kocaeli.edu.tr/*](http://edestek2.kocaeli.edu.tr/)

[*https://youtube.com/playlist?list=PLlD7n\_T-mUjVuqIhWVfaNhnpqCZmNcA9e*](https://youtube.com/playlist?list=PLlD7n_T-mUjVuqIhWVfaNhnpqCZmNcA9e)

[*https://www.geeksforgeeks.org/heaps-algorithm-for-generating-permutations/*](https://www.geeksforgeeks.org/heaps-algorithm-for-generating-permutations/)

[*https://www.youtube.com/watch?v=hvDx7q6vcWM*](https://www.youtube.com/watch?v=hvDx7q6vcWM)

[*https://www.youtube.com/watch?v=XaXsJJh-Q5Y*](https://www.youtube.com/watch?v=XaXsJJh-Q5Y)

[*https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/*](https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/)