



Yıldız Teknik Üniversitesi
Elektrik-Elektronik Fakültesi
Bilgisayar Mühendisliği Bölümü

“AYSIZ AUTO”

BLM3722

Yazılım Mühendisliği

Gr: 3

Prof.Dr. Mehmet Sıddık AKTAŞ

Proje Rapor

Proje Grubu: 5

İsim: Zeynep EKİNCİ

No: 21011068

E-posta:

zeynep.ekinci1@std.yildiz.edu.tr

İsim: Sude ELİTOK

No: 21011062

E-posta:

sude.elitok@std.yildiz.edu.tr

İsim: İrem ÇELİK

No: 21011097

E-posta:

irem.celik3@std.yildiz.edu.tr

İsim: Ahmet Mahir DEMİRELLİ

No: 21011063

E-posta: mahir.demirelli@std.yildiz.edu.tr

İsim: Emirhan Yusuf TOPTAŞ

No: 21011039

E-posta: emirhan.toptas@std.yildiz.edu.tr

Proje Planı:

- Proje Alan Tanımı:

Projenin amacı, henüz çevrimiçi faaliyet göstermeyen birinci el araç satış firmasının, tüm fonksiyonlarını dijital ortamda gerçekleştirebilen kapsamlı bir sistem tasarlamaktır. Bu proje kapsamında, müşteriler istedikleri araç için test sürüşü talebinde bulunabilir, fiyat teklifi alabilir ve sipariş oluşturabilirler.

- Kabul ve Kısıtlar:

Sistemde, bayi ve depo işlemlerini yürüten birer sistem üyesinin bulunduğu kabul edilmekte olup, bu yapı isteğe bağlı olarak genişletilebilir.

Sistemin depo bölümünden sorumlu olan üye, depodaki mevcut stokları görüntüleyebilmekte, isteğe bağlı olarak stok güncellemesi ve araç eklemesi yapabilmektedir.

Sistemin bayi bölümünden sorumlu olan üye, bayideki stok durumlarını görüntüleyebilmekte, depodan bayiye araç çekebilmektedir.

Müşteri, bayide var olan araçlar için test sürüşü talebi gönderebilmektedir. Bayi yöneticisi bu talebi uygunluk durumuna göre onaylar ya da reddeder.

Müşteri bir araç için fiyat teklifi isteyebilmektedir. Bayi tarafından fiyat teklifi gönderilir ve bu teklifin geçerlilik süresi 30 gündür. Müşteri 30 gün içerisinde aracı satın alma hakkına sahiptir.

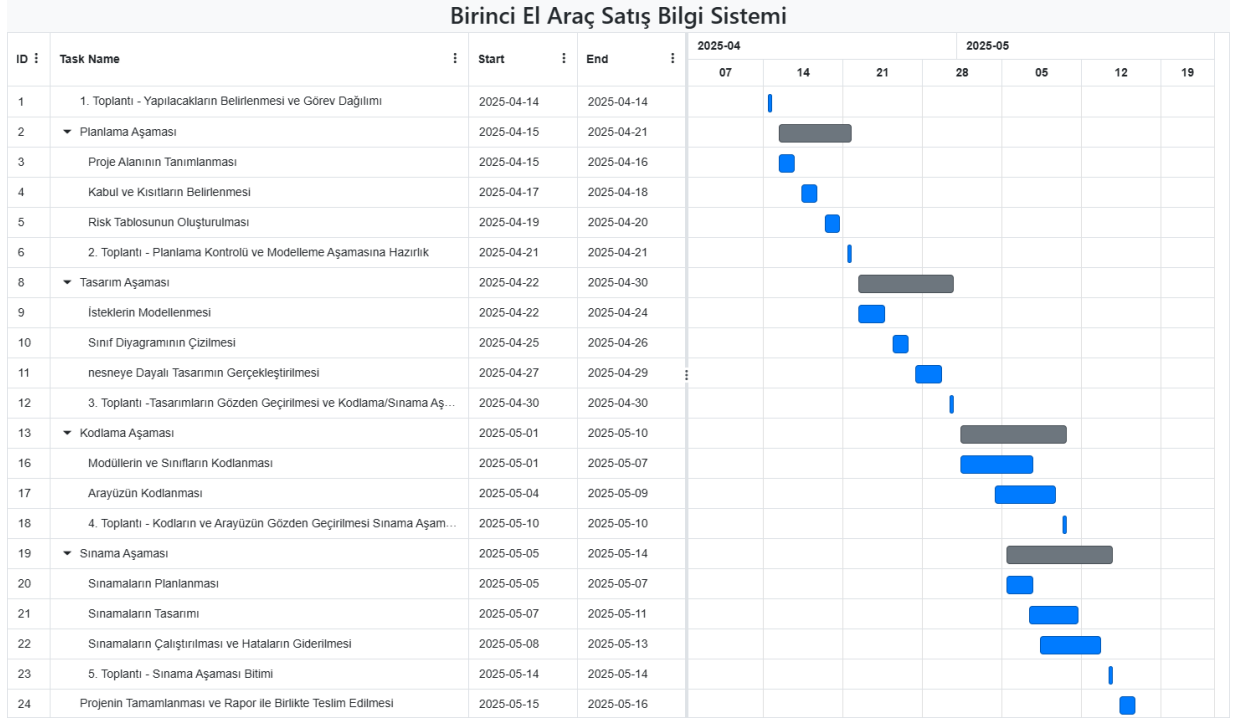
Müşterinin satın alma talebi, bayinin satış onayını vermesiyle birlikte işleme alınır ve sipariş süreci müşteri tarafından anlık olarak takip edilebilir.

Müşteri hareketleri sistemde kayıt altına alınmakta ve hedef kitlenin analiz edilmesi amacıyla kullanılmaktadır. Bu raporlar, bayi yöneticisi tarafından görüntülenebilir.

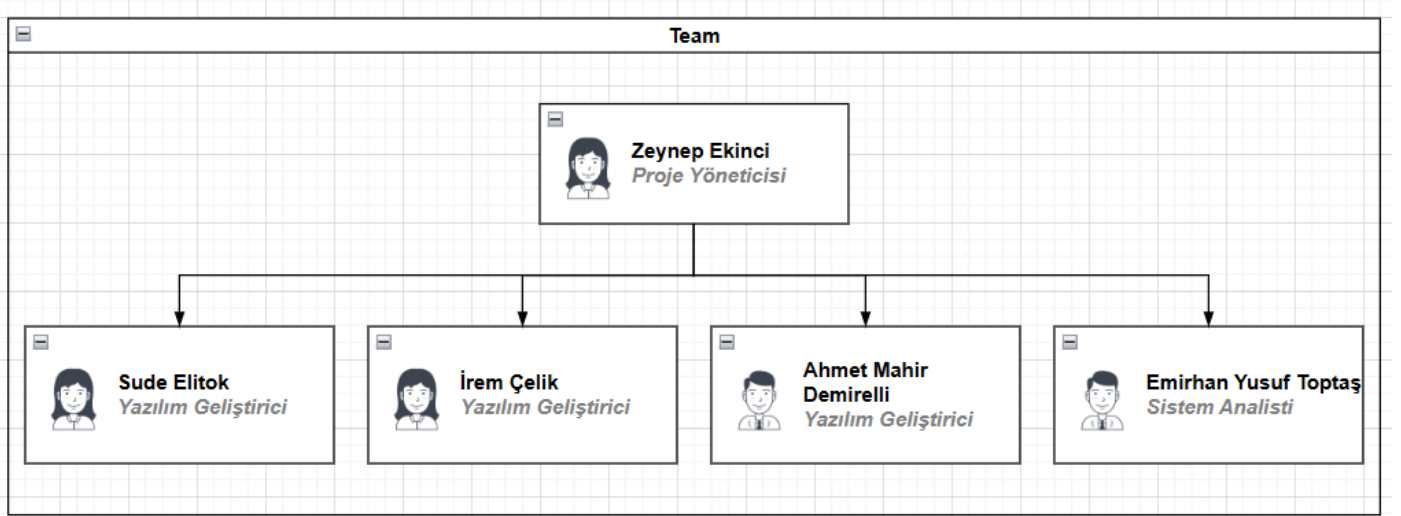
Gerçekleştirilen araç satışları, model, yıl, paket ve adet düzeylerinde tablo ve grafiklerle raporlanabilmektedir. Bu veriler kullanılarak, gelecek 3 yıl için satış tahminleri yapılabilmektedir.

- Proje İş-Zaman Çizelgesi:

Planlama, tasarım, geliştirme, test ve teslimat aşamalarını kapsayan proje, ekip içi iş dağılımı ve görev takiplerini kolaylaştırmak için Gantt diyagramı kullanılarak hazırlanmıştır.



- Ekip Organizasyon Şeması ve Görev Dağılımı



- Risk Yönetimi

Proje sürecinde oluşabilecek riskleri tespit edebilme amacıyla risk tablosu ve risk bilgi sayfaları oluşturulmuştur. Bu sayede ortaya çıkabilecek riskler erken tespit edilerek gerekli önlemler alınabilir ve proje sürecinde oluşabilecek olumsuz etkiler en aza indirgenebilir.

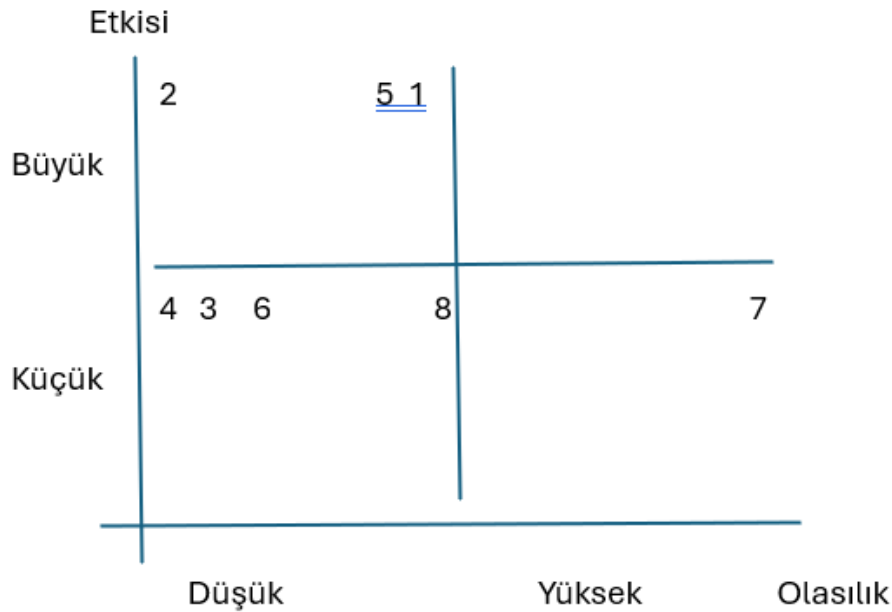
- Risk Tablosu

Bu risk tablosu, proje sürecinde karşılaşılabilecek olası risklerin belirlenmesi, değerlendirilmesi ve etkilerinin analiz edilmesi amacıyla hazırlanmıştır. Tablo, her bir riskin Risk ID, Adı, Türü, Olasılık ve Etkisi gibi temel bilgilerini içermektedir. Bu bilgiler, projenin başarısını ve sürdürülebilirliğini sağlamak için gerekli önlemlerin planlanması ve uygulanması açısından oldukça önemlidir.

Risk ID	Adı	Türü	Olasılık	Etki
1	Veritabanı performans sorunları	Teknik	Orta	Büyük
2	Güvenlik açıkları nedeniyle müşteri verilerinin sızması	Teknik	Düşük	Büyük
3	Zamanında teslim edilememe riski.	Proje	Düşük	Orta
4	Sistem arayüzünün karmaşık veya kullanıcı dostu olmaması.	Teknik	Düşük	Orta
5	Test süreçlerinin yetersiz olması.	Teknik	Orta	Büyük
6	Satış tahmininin hatalı yapılması	İş	Düşük	Orta
7	Ekip üyeleri arasında iş yükü dengesizliği veya rollerin belirsiz olması.	Proje	Yüksek	Orta
8	Kullanılan araçların yetersizliği	Teknik	Orta	Orta

- Risk Grafiği

Risk grafiği, risk tablosundaki bilgiler ile oluşturulmuştur. Hangi riskler için risk bilgi sayfası oluşturulması gerektiğinin tespit edilmesini sağlamak amacıyla yapılmıştır.



1,2,5 ve 7 numaralı riskler için risk bilgi sayfaları oluşturulması gerekmektedir.

RİSK #01: Veritabanı Performans Sorunları

Olasılık: Orta

Etki: Yüksek

Türü: Teknik

Açıklama: Veritabanı sorgularının yavaş olması, veri kaybı ve düşük sistem verimliliğine yol açabilir. Büyük veri işlemlerinde veya yüksek kullanıcı trafiğinde kritik performans sorunları yaşanabilir.

İşaretleri:

1. Sorgu sürelerinin uzaması.
2. Sunucunun sık sık donması veya çökmesi.
3. Yüksek bellek veya işlemci kullanımı.

Önlemler:

1. Sorgu optimizasyonu ve indeksleme.
2. Yük dengeleme ve önbellekleme kullanımı.
3. Sunucu kapasitesini artırmak.

RİSK #02: Güvenlik Açıkları Nedeniyle Müşteri Verilerinin Sızması

Olasılık: Düşük

Etki: Yüksek

Türü: Teknik

Açıklama: Yetersiz güvenlik önlemleri, müşteri verilerinin yetkisiz kişiler tarafından ele geçirilmesine yol açabilir. Bu durum, itibar kaybı, yasal yaptırımlar ve müşteri güveninin azalması gibi ciddi sonuçlar doğurabilir.

İşaretleri:

1. Sistem erişiminde olağan dışı giriş denemeleri.
2. Bilinmeyen veri hareketleri veya veri kaybı.
3. Güvenlik yamalarının eksik veya güncellenmemiş olması.

Önlemler:

1. Güçlü şifreleme ve iki faktörlü kimlik doğrulama kullanımı.
2. Güvenlik duvarları ve izinsiz giriş tespit sistemleri (IDS) uygulamak.
3. Düzenli güvenlik testleri ve zafiyet taramaları yapmak.

RİSK #05: Test Süreçlerinin Yetersiz Olması

Olasılık: Orta

Etki: Yüksek

Türü: Teknik

Açıklama: Yetersiz test süreçleri, sistemde hataların tespit edilememesine ve kullanıcı deneyiminin olumsuz etkilenmesine yol açabilir. Bu durum, müşteri memnuniyetsizliği ve yüksek bakım maliyetleriyle sonuçlanabilir.

İşaretleri:

1. Sık sık beklenmedik sistem hataları ve çökme sorunları.
2. Kullanıcı şikayetlerinin artması.
3. Yeni sürümlerde sık sık hata çıkması.

Önlemler:

1. Otomasyon test araçları kullanmak.
2. Test planları ve senaryolarını genişletmek.

RİSK #07: Ekip Üyeleri Arasında İş Yükü Dengesizliği veya Rollerinin Belirsiz Olması

Olasılık: Yüksek **Etki:** Orta **Türü:** Proje

Açıklama: Ekip üyeleri arasındaki iş yükü dengesizliği, motivasyon düşüklüğüne ve proje teslim süresinin gecikmesine yol açabilir. Belirsiz roller, sorumluluk karmaşasına ve verimsizliğe neden olabilir.

İşaretleri:

1. Ekip üyeleri arasında iş yüküyle ilgili sık şikayetler.
2. Proje ilerlemesinde yavaşlama veya gecikmeler.
3. Bazı ekip üyelerinin aşırı yoğun çalışması, bazılarının ise düşük katkı sağlaması.

Önlemler:

1. Görev ve sorumlulukları net bir şekilde tanımlamak.
2. Düzenli ekip toplantıları ve performans değerlendirmeleri yapmak
3. İş yükünü dengeli dağıtmak ve gerektiğinde ek kaynak sağlamak

● Yazılım Ölçümü

Yazılım ölçümü, ürün kalitesini değerlendirmek, iş yükünü tahmin etmek ve maliyetleri kontrol etmek amacıyla yapılmıştır. İyi bir ölçüm, projelerin zamanında ve bütçeye uygun tamamlanmasına yardımcı olur.

Bu projede, dolaylı bir ölçüm yöntemi olan Fonksiyon Noktası yöntemi kullanılarak yazılımın verimlilik, kalite, gider ve belgeleme düzeyleri hesaplanmıştır. Hesaplamalar, fonksiyon noktası bileşenleri ile yazılım parametrelerine ait tablolar esas alınarak gerçekleştirilmiştir.

a) Fonksiyon noktası bileşenleri

i	Nicelik	Sayısı (S)	Ağırlık Faktörü (AF) - Basit	Ağırlık Faktörü (AF) - Orta	Ağırlık Faktörü (AF) - Karmaşık	Fpi (S*AF)
1	Kullanıcının yazılıma giriş sayısı (user input)	10	3	4	6	40
2	Kullanıcının aldığı çıktı sayısı (output)	1	4	5	7	7
3	Kullanıcının sorgulama sayısı (query)	5	3	4	6	15
4	Kütük sayısı (record)	7	7	10	15	70
5	Dış arabirim sayısı	0	5	7	10	0
Toplam=						132

b) Yazılım Parametreleri

i	Parametre	Fi (0-5 arası)
1	Güvenli yedekleme ve geri yükleme gerekli mi?	5
2	İletişim altyapısı gerekli mi?	5
3	Dağıtılmış işleme fonksiyonları var mı?	0
4	Performans kritik mi?	3
5	Sistem yükü fazla mı?	3
6	Çevrimiçi veri girişi var mı?	5
7	Çok ekranlı hareket girişleri var mı?	3
8	Ana dosyalar çevrimiçi güncelleniyor mu?	5
9	Giriş, Çıkış ve Sorgular karmaşık mı?	3
10	İçsel işlemler karmaşık mı?	2
11	Yeniden kullanılabilirlik var mı?	5
12	Yükleme tasarıma dahil mi?	2
13	Farklı şirketlerde de çalışması söz konusu mu?	3
14	Uygulama kullanıcı tarafından kolayca değiştirilebilir mi?	0
Toplam =		44

Sistemdeki her bir fonksiyon tipi için sayılar ve ağırlık faktörleri çarpılarak $\sum FPi = 132$ değeri elde edilmiştir. Ardından, yazılımın 14 genel özelliği değerlendirilerek toplam karmaşıklık düzeyi $\sum Fi = 44$ olarak belirlenmiştir.

Standart formül olan $FP = \sum FPi \times (0.65 + 0.01 \times \sum Fi)$ kullanılarak, fonksiyon noktası toplamı hesaplanmıştır:

$$FP = 132 \times (0.65 + 0.01 \times 44) = 132 \times 1.09 = 144.$$

Elde edilen bu değer, yazılımın büyüklüğünü ifade etmekte olup; verimlilik, kalite, gider ve belgeleme gibi diğer yazılım ölçütlerinin hesaplanmasında kullanılmıştır.

- Verimlilik: Verimlilik, bir kişinin belirli bir zaman diliminde üretebildiği fonksiyon noktası miktarını gösterir.

Formül: Verimlilik = FP / (Kişi × Ay)

Hesaplama: Verimlilik = 144 / (5 × 1) = 28.8

- Kalite: Kalite, yazılımın ne kadar hatayla geliştirildiğini ifade eder. Daha düşük değer daha yüksek kalite anlamına gelir.

Formül: Kalite = Hata Sayısı / FP

Hesaplama: Kalite = 10 / 144 = 0.069

- Gider: Gider, bir fonksiyon noktasının maliyetini ifade eder. Toplam giderin FP değerine bölünmesiyle elde edilir.

Formül: Gider = Toplam Gider / FP

Hesaplama: Gider = 100000 / 144 = 694

- Belgeleme: Belgeleme, her bir fonksiyon noktası başına kaç sayfa belge üretildiğini gösterir.

Formül: Belgeleme = Belge Sayfası / FP

Hesaplama: Belgeleme = 50 / 144 = 0.347

İsteklerin Modellenmesi

- Kullanım Senaryosu Diyagramı



- Kullanım Senaryosu Metinleri

a) Kullanım Senaryosu: Test Sürüşü

Kullanım Senaryosu	Test Sürüşü
Birincil Aktör	Müşteri
İlgililer ve İlgili Alanları	<p>Müşteri: En kısa zamanda test sürüş talebinin onaylanmasını ve aracın hazırlanmasını ister.</p> <p>Bayi: Test sürüşü yapılmak istenen aracın stokta olmasını, test sürüşü sırasında aracın herhangi bir zarara uğramamasını ister.</p>
Ön Koşullar	<p>Müşteri sistemde kayıtlıdır.</p> <p>Müşterinin ehliyeti doğrulanır.</p>
Son Koşullar	<p>Test sürüşü gerçekleştirilen araç bayi tarafından teslim alınır.</p> <p>Aracın hasar durumu kontrol edilir.</p> <p>Test sürüşü bilgileri sisteme kaydedilir.</p>
Ana Senaryo	<ol style="list-style-type: none">1. Müşteri sisteme giriş yapar.2. Müşteri test yapmak istediği aracı seçerek sürüş talebini gönderir.3. Bayi test aracının stoğunun uygunluğunu kontrol eder.4. Bayi tarih uygunluğunu kontrol eder.5. Bayi, sürüş talebine onay verir.6. Test sürüşü gerçekleştirilir.7. Sürüş sonunda bayi, aracı teslim alır, olası hasar kontrolü yapar ve sisteme giriş yapar.8. Bayi, test sürüşü bilgilerini sisteme işler ve veri analizi için saklar.
Alternatif Senaryo	<p>1a. Müşteri kayıtlı değil</p> <ol style="list-style-type: none">1. Müşteri sisteme kaydolar. <p>3a. Araç stokta yok</p> <ol style="list-style-type: none">1. Araç depodan bayiye çekilir. <p>4a. Tarih uygun değil</p> <ol style="list-style-type: none">1. Bayi talebi reddeder.

b) Kullanım Senaryosu: Fiyat Teklifi Alma

Kullanım Senaryosu	Fiyat Teklifi Alma
Birincil Aktör	Müşteri
İlgililer ve İlgili Alanları	<p>Müşteri: İstediği aracın için fiyat teklifi almak ister. Teklifin bütçesine uygun olmasını ister.</p> <p>Bayi: Müşteriyi memnun edecek rekabetçi bir fiyat sunmak ister.</p>
Ön Koşullar	<p>Müşteri, sistemde kayıtlıdır.</p> <p>Müşteri, fiyatını öğrenmek istediği arabayı seçmiştir.</p>
Son Koşullar	Aracın güncel fiyatı müşteriye bildirilmiştir.
Ana Senaryo	<ol style="list-style-type: none">1. Müşteri, sisteme giriş yapar.2. Müşteri, fiyat teklifi almak istediği aracı seçer.3. Müşteri, talebi gönderir.4. Bayi, araç için uygun bir fiyat belirler.5. Bayi, fiyatı müşteriye bildirir.
Alternatif Senaryo	<p>1a. Müşteri kayıtlı değil</p> <ol style="list-style-type: none">1. Müşteri sisteme kaydolur. <p>3a. Müşteri daha önce talep göndermiş ancak bayi henüz teklif yapmamış</p> <ol style="list-style-type: none">1. Sistem, kullanıcıya "Bu araç için daha önce talep gönderildi, teklif bekleniyor." diye uyarı verir. <p>3b. Bayi son 30 gün içinde o araç için fiyat teklif yapmış</p> <ol style="list-style-type: none">1. Sistem, kullanıcıya "Bu araç için geçerli fiyat teklifi bulunmaktadır." diye uyarı verir.

c) Kullanım Senaryosu: Araç Satın Alımı

Kullanım Senaryosu	Araç Satın Alımı
Birincil Aktör	Müşteri
İlgililer ve İlgili Alanları	Müşteri: İstediği aracın satın almak ister. Bayi: Satın alma sürecinin eksiksiz bir şekilde ve zamanında tamamlanmasını, ödeme ve araç teslim işlemlerinin sorunsuz gerçekleşmesini ister.
Ön Koşullar	Müşteri, sistemde kayıtlıdır. Müşteri, fiyat teklifi almıştır.
Son Koşullar	Satın alma işlemi tamamlanmış, ödeme alınmıştır. Araç müşteriye teslim edilmiştir. Satış bilgileri sisteme kaydedilmiştir.
Ana Senaryo	<ol style="list-style-type: none">1. Müşteri, sisteme giriş yapar.2. Müşteri, daha önce aldığı fiyat teklifleri arasından almak istediği arabayı seçer.3. Bayi, aracın stoğunu kontrol eder.4. Bayi, satın alma işlemi için gerekli belgeleri düzenler.5. Müşteri ödeme işlemini tamamlar.6. Bayi, aracı müşteriye teslim eder ve araç stoğunu günceller.7. Bayi, satış işlemi verilerini sisteme kaydeder ve veri analizi için saklar.
Alternatif Senaryo	<ol style="list-style-type: none">1a. Müşteri kayıtlı değil<ol style="list-style-type: none">1. Müşteri sisteme kaydolar.2a. Fiyat teklifinin verilmesinin üstünden 30 gün geçmiş<ol style="list-style-type: none">1. Sistem müşterinin yeni fiyat teklifi alması için uyarı verir.2. Müşteri yeni fiyat teklifi için talep gönderir.3a. Araç stokta yok<ol style="list-style-type: none">1. Araç depodan bayiye çekilir.5a. Ödeme gerçekleştirilemez<ol style="list-style-type: none">1. Farklı ödeme seçenekleri müşteriye sunulur.2. Müşteri kendisine uygun yöntemi seçer ve ödemeyi gerçekleştirir.

d) Kullanım Senaryosu: Depoya Yeni Araç Eklenmesi

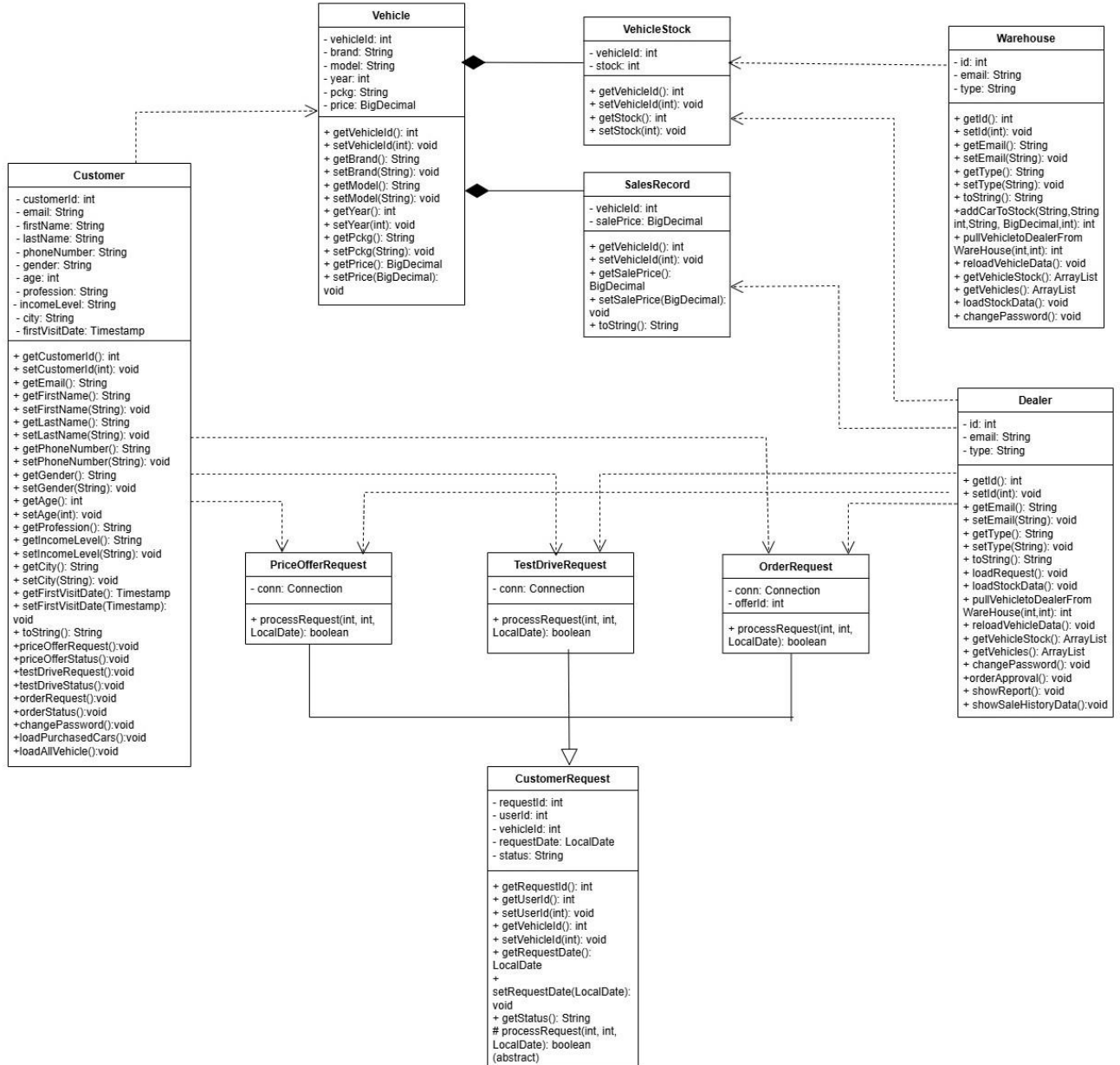
Kullanım Senaryosu	Depoya Yeni Araç Eklenmesi
Birincil Aktör	Depo Sorumlusu
İlgililer ve İlgili Alanları	Depo Sorumlusu: Yeni gelen aracın birinci el olmasını, depoya eklenmesini ve stoğunun tutulmasını ister. Bayi: Araç kaydının doğru olmasını ve aracın stokta olmasını ister.
Ön Koşullar	Depo sorumlusu sisteme giriş yapmıştır. Yeni birinci el araç teslim alınmıştır.
Son Koşullar	Yeni araç depoya girmiştir. Aracın bilgileri sisteme işlenmiştir.
Ana Senaryo	<ol style="list-style-type: none">1. Depo sorumlusu, sisteme giriş yapar.2. Depo sorumlusu, yeni araç ekleme sayfasına girer..3. Aracın markası modeli yılı paketi, fiyatı ve adeti sisteme girilir.4. Araç sisteme kaydedilir.
Alternatif Senaryo	4a. Araç deponun sistemine daha önce kaydedilmiş. <ol style="list-style-type: none">1. Sistem aracın önceden depoya eklendiğine dair uyarı verir.2. Depo sorumlusu yeni araç olarak eklemek yerine o aracın stoğunu artırır.

• Sınıf Düzeyi İzlenebilirlik Tablosu

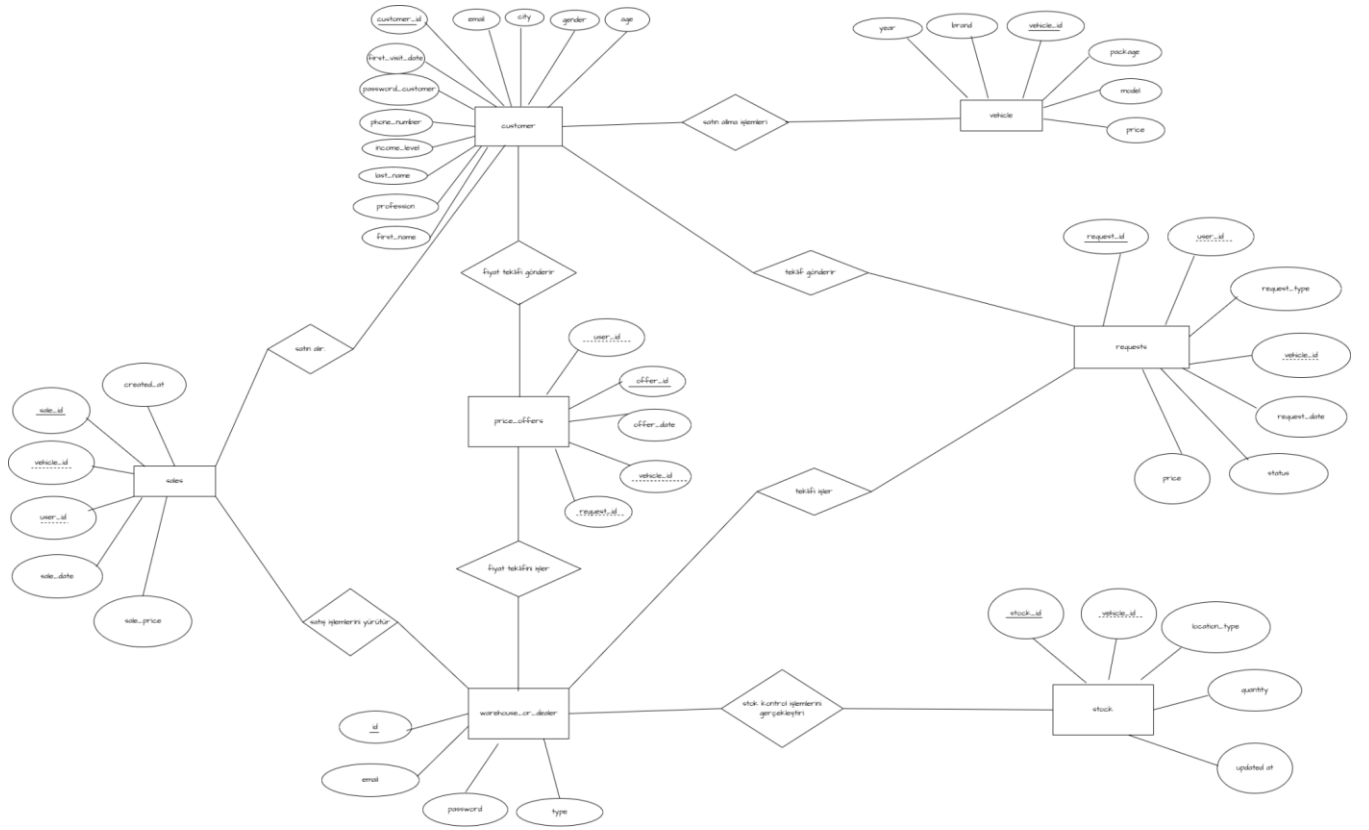
Gereksinim ID	Gereksinim Tanımı	Sınıf Adı	Test Vaka ID
REQ-001	Müşteri Girişi	CustomerLoginPage	TV-001
REQ-002	Bayi Girişi	DealerLoginPage	TV-002
REQ-003	Depo Girişi	WarehouseLoginPage	TV-003
REQ-004	Stoktaki Araçları Sorgulama	ShowCarStockPage	TV-004
REQ-005	Test Sürüşü Talep Etme	TestDriveRequestPage	TV-005
REQ-006	Fiyat Teklifi İsteme	PriceOfferRequestPage	TV-006
REQ-007	Sipariş Verme	OrderRequestPage	TV-007
REQ-008	Test Sürüşü Onaylama	DealerRequestPage	TV-008
REQ-009	Fiyat Teklifi Yapma	DealerRequestPage	TV-009
REQ-010	Siparişi Onaylama	DealerOrderApprovalPage	TV-010
REQ-011	Stoktan Araç Çekme	PullCarFromStockPage	TV-011
REQ-012	Araç Ekleme	AddCarToStockPage	TV-012
REQ-001	Araç Stoğu Güncelleme	UpdateStockPage	TV-013

Nesneye Dayalı Modelleme Ve Tasarım

a) Sınıf Diyagramı:

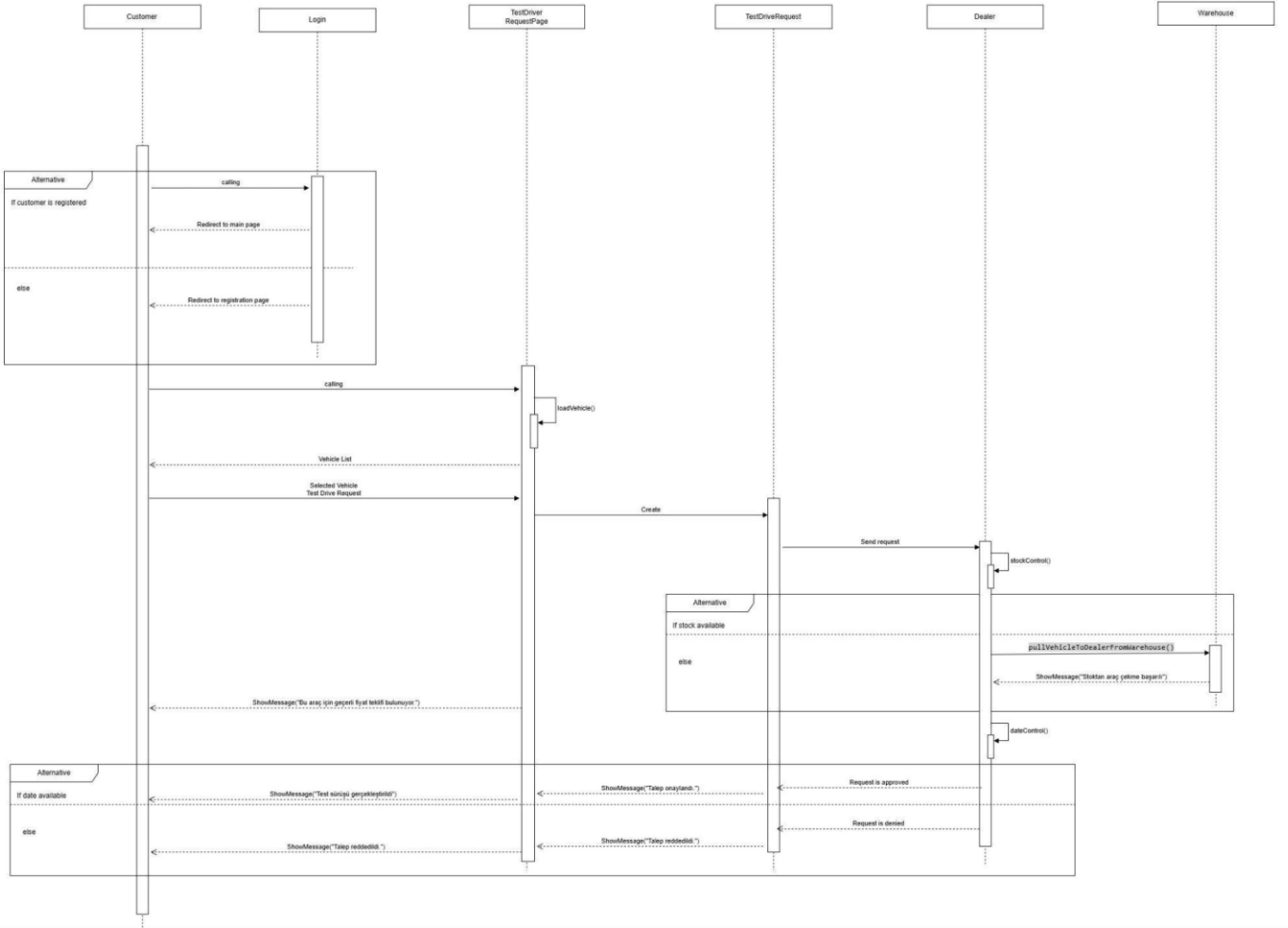


b) ER Diyagramı:

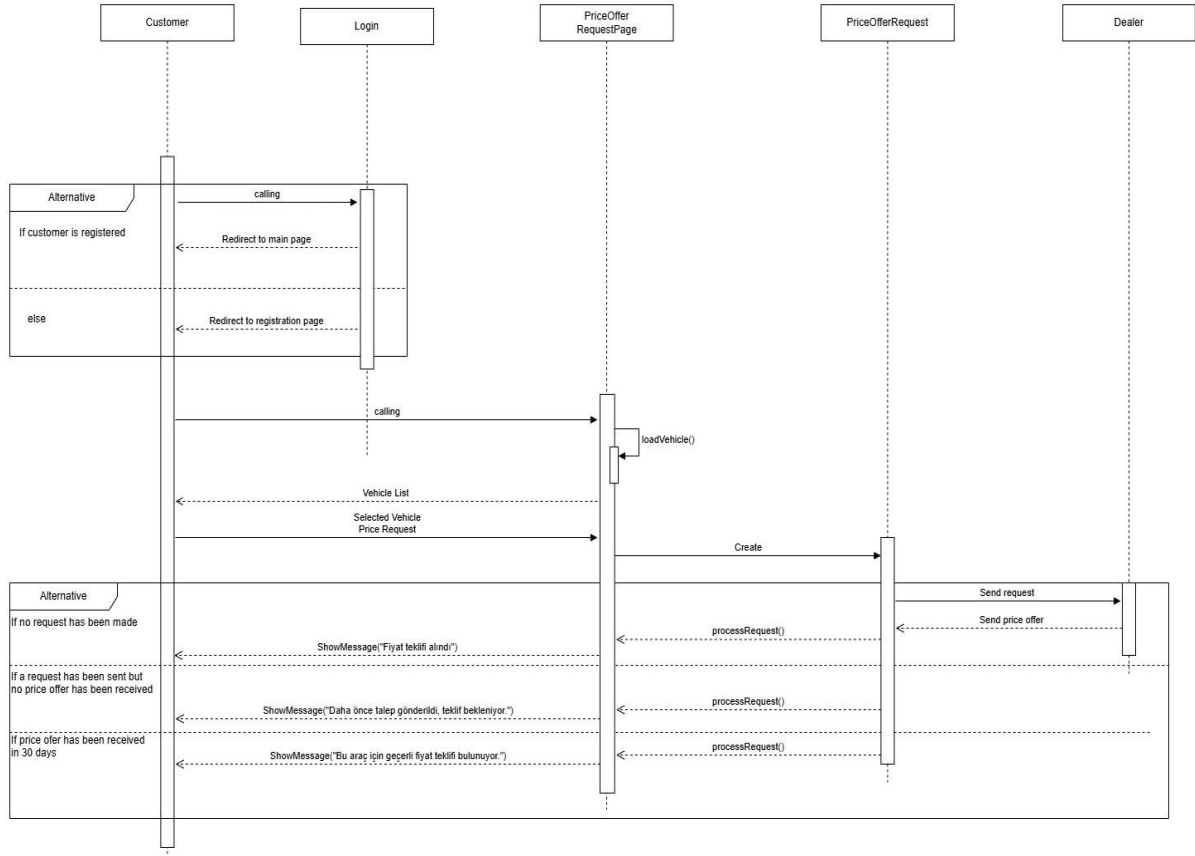


- Tasarım Sıralama (Sequence) Diyagramı:

- Test sürüşü senaryosu:

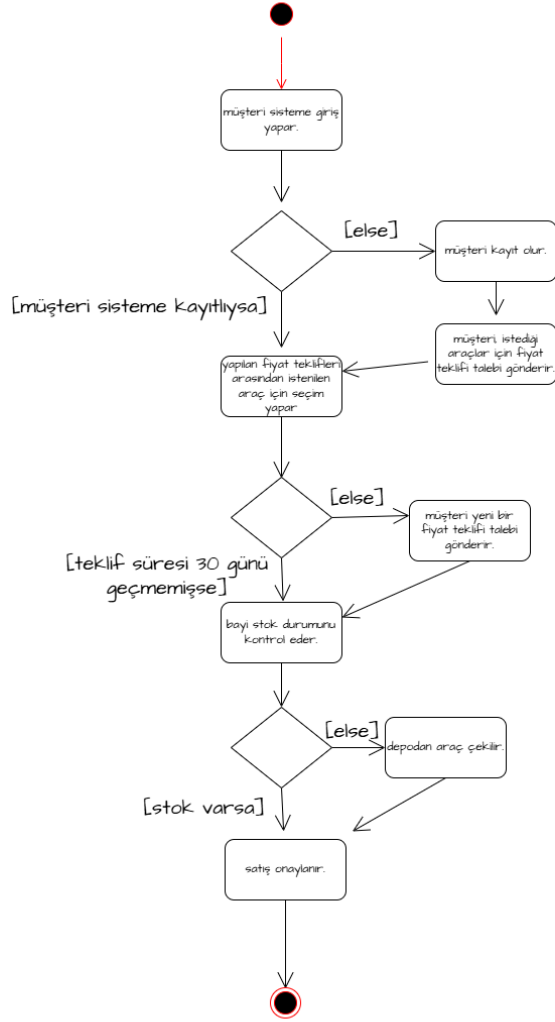


- Fiyat teklifi alma senaryosu:

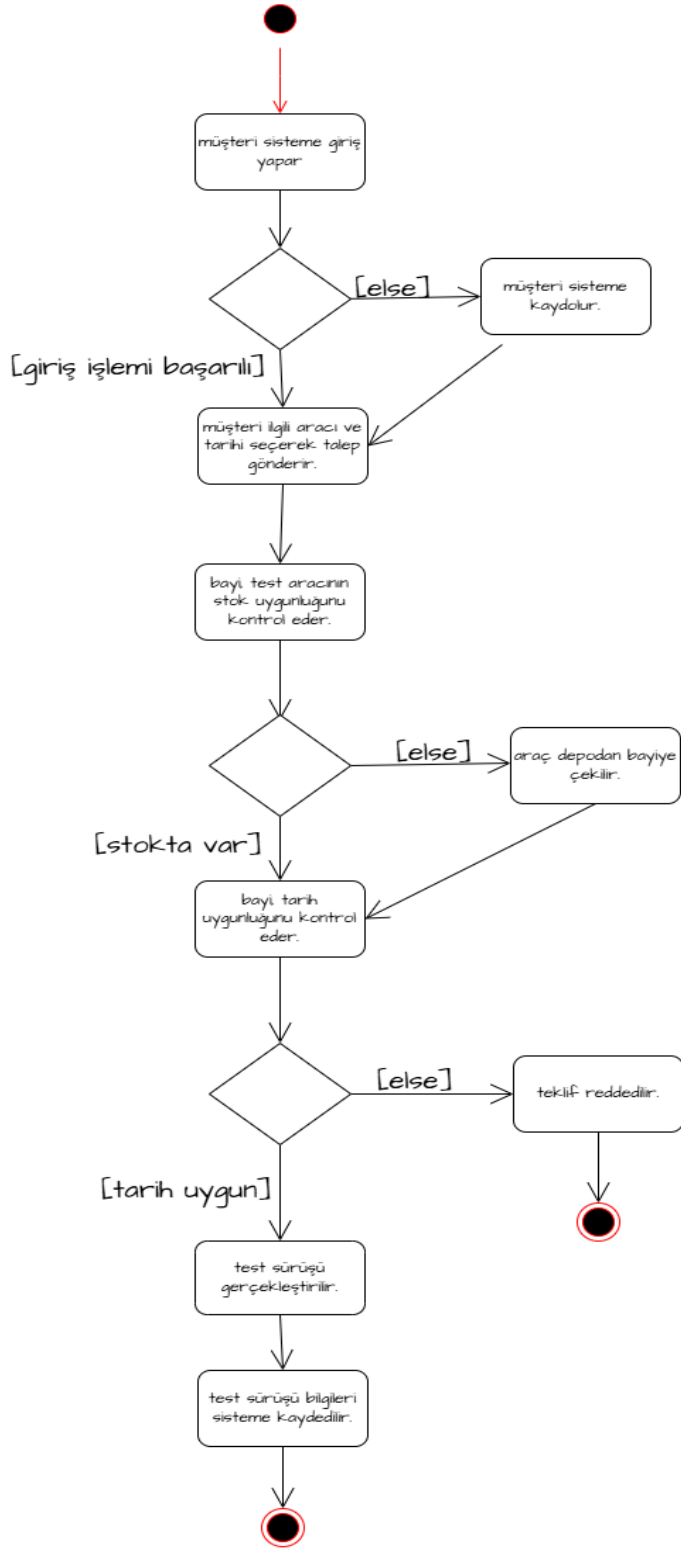


- Etkinlik (Activity) Diyagramı:

- Satın alma senaryosu:

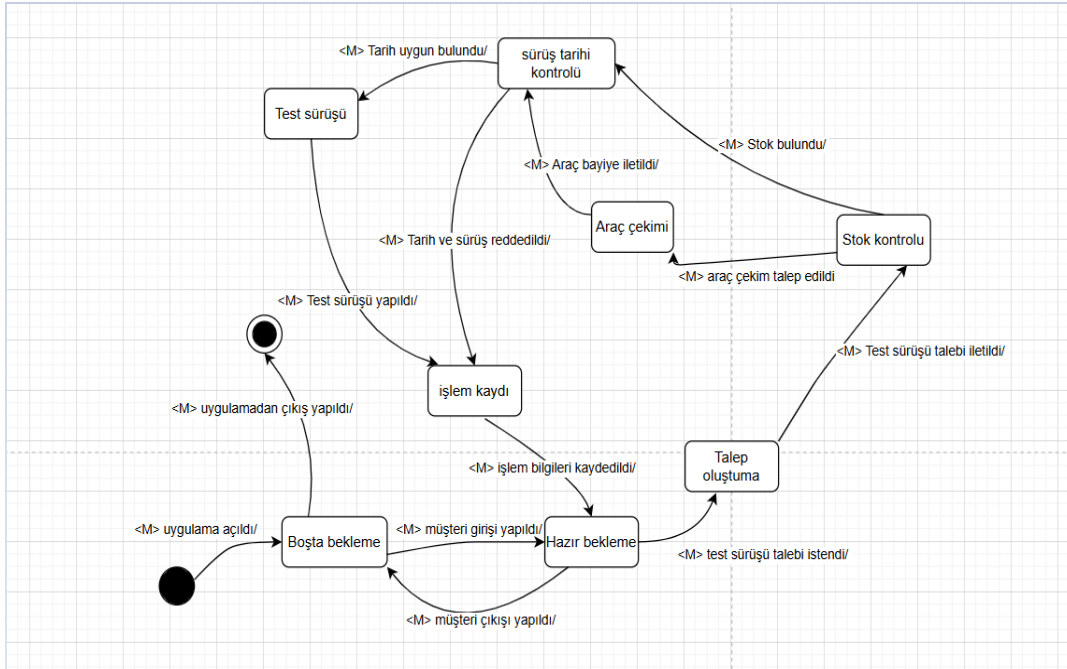


- Test sürüşü talebi senaryosu:

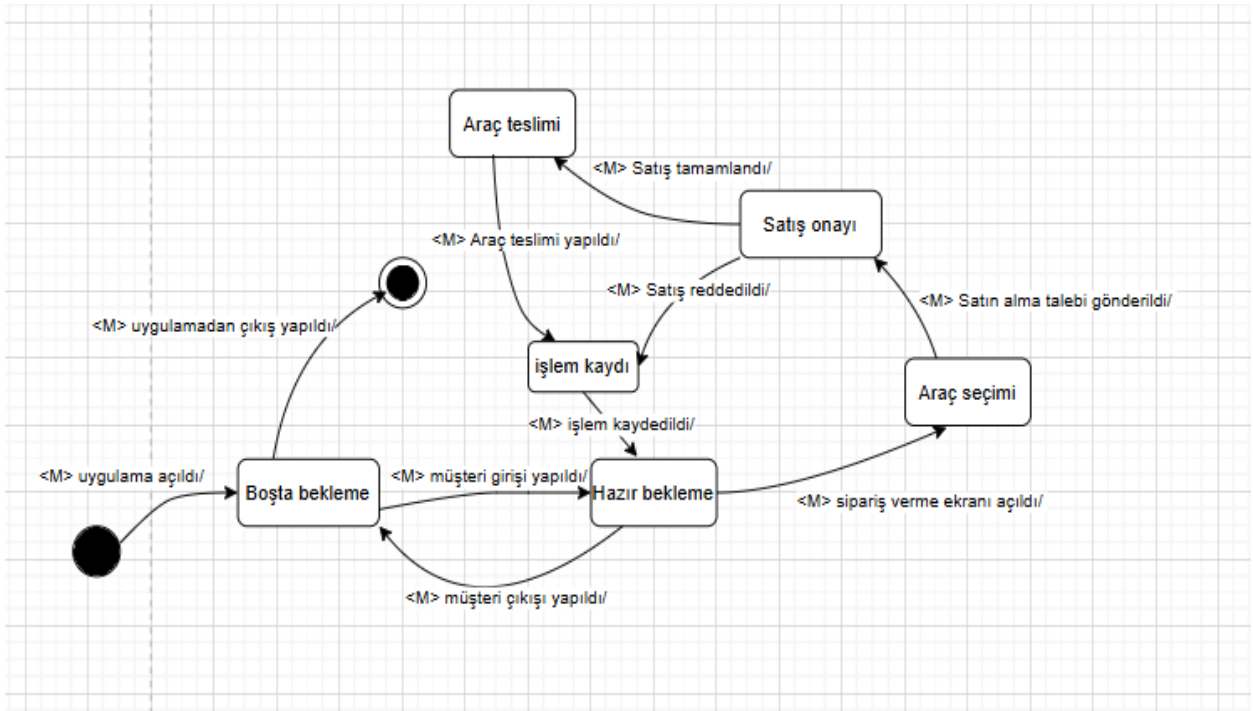


- Durum (State) Diyagramı:

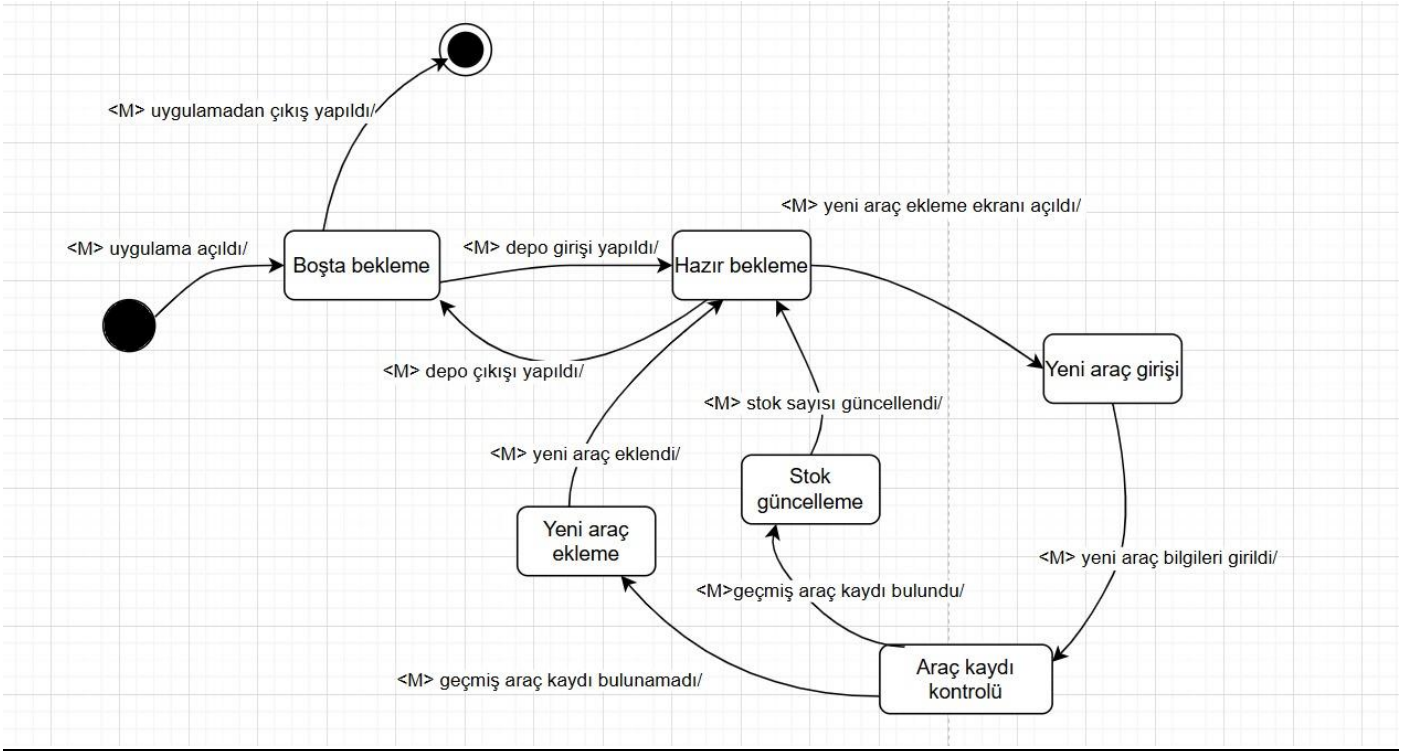
- Test sürüşü talebi senaryosu:



- Satın alma senaryosu:



- Depoya yeni araç ekleme senaryosu:



Birim Testi Sınamaları

- Satın alınan araçları görüntüleme testi- Zeynep Ekinci:
 - Kaynak Kodu:

```
package yazilim.tests;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import yazilim.CustomerCarPage;
import yazilim.classes.Customer;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.time.LocalDate;
import java.util.Date;
import static org.junit.jupiter.api.Assertions.*;

public class CustomerCarPageTest {
    private Connection conn;
    private CustomerCarPage customerCarPage;
    private final Customer customer = new Customer(100, "testuser@example.com", "Test",
"User", "555555555", "male", 20, "tester", "high", "Istanbul", new
Timestamp(System.currentTimeMillis()));

    @BeforeEach
    void setUp() throws SQLException {
        // Veritabanı bağlantısını açma
        String url = "jdbc:postgresql://localhost:5432/YazilimMuhProje";
```

```

String username = "postgres";
String password = "12345";
conn = DriverManager.getConnection(url, username, password);
// Test kullanıcı verisi ekleme
try (PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO customer (customer_id, email, password_customer,
first_name, last_name) VALUES (?, ?, ?, ?, ?) ON CONFLICT DO NOTHING")) {
    stmt.setInt(1, customer.getCustomerId());
    stmt.setString(2, customer.getEmail());
    stmt.setString(3, "testpassword");
    stmt.setString(4, customer.getFirstName());
    stmt.setString(5, customer.getLastName());
    stmt.executeUpdate();
}
// Test araç verisi ekleme (100 ID)
try (PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO vehicle (vehicle_id, brand, model, year) VALUES (?, ?, ?,
?) ON CONFLICT DO NOTHING")) {
    stmt.setInt(1, 100); // Vehicle ID 100 olarak ayarlandı
    stmt.setString(2, "Toyota");
    stmt.setString(3, "Corolla");
    stmt.setInt(4, 2020);
    stmt.executeUpdate();
}
// Test satış verisi ekleme (Vehicle ID 100 kullanarak)
try (PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO sales (sale_id, user_id, vehicle_id, sale_date, sale_price)
VALUES (?, ?, ?, ?, ?) ON CONFLICT DO NOTHING")) {
    stmt.setInt(1, 100);
    stmt.setInt(2, customer.getCustomerId());
    stmt.setInt(3, 100);
    stmt.setDate(4, java.sql.Date.valueOf("2023-05-01"));
    stmt.setDouble(5, 300000);
    stmt.executeUpdate();
}

try (PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO vehicle (vehicle_id, brand, model, year) VALUES (?, ?, ?,
?) ON CONFLICT DO NOTHING")) {
    stmt.setInt(1, 101);
    stmt.setString(2, "Honda");
    stmt.setString(3, "Civic");
    stmt.setInt(4, 2022);
    stmt.executeUpdate();
}
try (PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO sales (sale_id, user_id, vehicle_id, sale_date, sale_price)
VALUES (?, ?, ?, ?, ?) ON CONFLICT DO NOTHING")) {
    stmt.setInt(1, 101);
    stmt.setInt(2, customer.getCustomerId());
    stmt.setInt(3, 101);
    stmt.setDate(4, java.sql.Date.valueOf("2023-06-01"));
    stmt.setDouble(5, 500000);
    stmt.executeUpdate();
}

customerCarPage = new CustomerCarPage(customer, conn);
}

```

```

//test verilerini silme
@AfterEach
void tearDown() throws SQLException {
    // stock tablosundan silme
    try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM stock WHERE
vehicle_id IN (?, ?)")) {
        stmt.setInt(1, 100);
        stmt.setInt(2, 101);
        stmt.executeUpdate();
    }
    // sales tablosundan silme
    try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM sales WHERE
user_id = ?")) {
        stmt.setInt(1, customer.getCustomerId());
        stmt.executeUpdate();
    }
    // vehicle tablosundan silme
    try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM vehicle WHERE
vehicle_id IN (?, ?)")) {
        stmt.setInt(1, 100);
        stmt.setInt(2, 101);
        stmt.executeUpdate();
    }
    // customer tablosundan silme
    try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM customer WHERE
customer_id = ?")) {
        stmt.setInt(1, customer.getCustomerId());
        stmt.executeUpdate();
    }
    if (conn != null && !conn.isClosed()) {
        conn.close();
    }
}

@Test
void testLoadPurchasedCars() {
    System.out.println("\n=== Testing Purchased Cars ===");
    JTable table = customerCarPage.table;
    DefaultTableModel model = (DefaultTableModel) table.getModel();
    int rowCount = model.getRowCount();
    System.out.println("Row count: " + rowCount);
    // Tablodaki tüm verileri yazdırma
    for (int i = 0; i < rowCount; i++) {
        for (int j = 0; j < model.getColumnCount(); j++) {
            System.out.print("Cell (" + i + ", " + j + "): " + model.getValueAt(i, j)
+ "\t");
        }
        System.out.println();
    }
    assertEquals(2, rowCount, "Table should contain two purchased cars.");
    // İlk satırın doğru olduğunu kontrol etme
    String vehicleName1 = (String) model.getValueAt(0, 0);
    String saleDate1 = (String) model.getValueAt(0, 1);
    String salePrice1 = (String) model.getValueAt(0, 2);
    System.out.println("Expected: Honda Civic (2022)");
    System.out.println("Actual: " + vehicleName1);
    System.out.println("Expected price: 500000,00 TL");
    System.out.println("Actual price: " + salePrice1);
    assertEquals("Honda Civic (2022)", vehicleName1);
    assertEquals("2023-06-01", saleDate1);
}

```

```

        assertEquals("500000,00 TL", salePrice1);
        // İkinci satırın doğru olduğunu kontrol etme
        String vehicleName2 = (String) model.getValueAt(1, 0);
        String saleDate2 = (String) model.getValueAt(1, 1);
        String salePrice2 = (String) model.getValueAt(1, 2);
        System.out.println("Expected: Toyota Corolla (2020)");
        System.out.println("Actual: " + vehicleName2);
        System.out.println("Expected price: 300000,00 TL");
        System.out.println("Actual price: " + salePrice2);
        assertEquals("Toyota Corolla (2020)", vehicleName2);
        assertEquals("2023-05-01", saleDate2);
        assertEquals("300000,00 TL", salePrice2);
    }

    @Test
    void testEmptyPurchasedCars() throws SQLException {
        System.out.println("\n=== Testing Empty Purchased Cars ===");
        System.out.println("Clearing all sales for user ID: " +
customer.getCustomerId());
        // Tüm satışları temizle
        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM sales WHERE
user_id = ?")) {
            stmt.setInt(1, customer.getCustomerId());
            int rowsDeleted = stmt.executeUpdate();
            System.out.println("Deleted " + rowsDeleted + " sales records for user ID: "
+ customer.getCustomerId());
        }
        // Test edilen sınıfı yeniden başlatma
        customerCarPage = new CustomerCarPage(customer, conn);

        JTable table = customerCarPage.table;
        DefaultTableModel model = (DefaultTableModel) table.getModel();
        int rowCount = model.getRowCount();
        System.out.println("Row count after clearing sales: " + rowCount);
        // Tablodaki tüm verileri yazdırma
        for (int i = 0; i < rowCount; i++) {
            for (int j = 0; j < model.getColumnCount(); j++) {
                System.out.print("Cell (" + i + ", " + j + "): " + model.getValueAt(i, j)
+ "\t");
            }
            System.out.println();
        }
        // Boş tablo mesajını kontrol etme
        assertEquals(1, rowCount, "Table should show a message if no cars have been
purchased.");
        String message = (String) model.getValueAt(0, 0);
        System.out.println("Expected message: Henüz araç satın alınmamış.");
        System.out.println("Actual message: " + message);
        assertEquals("Henüz araç satın alınmamış.", message);
    }
}

```


- Ekran Çıktısı:

```
Runs: 2/2 Errors: 0 Failures: 0
> CustomerCarPageTest [Runner: JUnit 5] (0,536)

=== Testing Empty Purchased Cars ===
Clearing all sales for user ID: 100
Deleted 2 sales records for user ID: 100
Row count after clearing sales: 1
Cell (0,0): Henüz araç satın alınmamış. Cell (0,1): Cell (0,2):
Expected message: Henüz araç satın alınmamış.
Actual message: Henüz araç satın alınmamış.

=== Testing Purchased Cars ===
Row count: 2
Cell (0,0): Honda Civic (2022) Cell (0,1): 2023-06-01 Cell (0,2): 500000,00 TL
Cell (1,0): Toyota Corolla (2020) Cell (1,1): 2023-05-01 Cell (1,2): 300000,00 TL
Expected: Honda Civic (2022)
Actual: Honda Civic (2022)
Expected price: 500000,00 TL
Actual price: 500000,00 TL
Expected: Toyota Corolla (2020)
Actual: Toyota Corolla (2020)
Expected price: 300000,00 TL
Actual price: 300000,00 TL
```

- Sipariş Verme Testi - Zeynep Ekinci:
- Kaynak Kodu:

```
package yazilim.tests;

import yazilim.requests.OrderRequest;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDate;
import static org.junit.jupiter.api.Assertions.*;

class OrderRequestTest {
    private Connection conn;
    private OrderRequest orderRequest;
    private final int userId = 100;
    private final int vehicleId = 100;
    private final int offerId = 100;
    private final LocalDate requestDate = LocalDate.now();
    @BeforeEach
    void setUp() throws SQLException {
        // Veritabanı bağlantısı ayarları
        String url = "jdbc:postgresql://localhost:5432/YazilimMuhProje";
        String username = "postgres";
        String password = "12345";
        conn = DriverManager.getConnection(url, username, password);
        // Test verilerini ekleme
    }
}
```

```

        String userInsert = "INSERT INTO customer (customer_id, email,
password_customer, first_name, last_name) VALUES (?, ?, ?, ?, ?) ON CONFLICT DO
NOTHING";
        try (PreparedStatement stmt = conn.prepareStatement(userInsert)) {
            stmt.setInt(1, userId);
            stmt.setString(2, "testuser@example.com");
            stmt.setString(3, "testpassword");
            stmt.setString(4, "Test");
            stmt.setString(5, "User");
            stmt.executeUpdate();
        }
        String vehicleInsert = "INSERT INTO vehicle (vehicle_id, brand, model) VALUES
(?, ?, ?) ON CONFLICT DO NOTHING";
        try (PreparedStatement stmt = conn.prepareStatement(vehicleInsert)) {
            stmt.setInt(1, vehicleId);
            stmt.setString(2, "TestBrand");
            stmt.setString(3, "TestModel");
            stmt.executeUpdate();
        }
        // Test request kaydı ekleme
        String requestInsert = "INSERT INTO requests (user_id, vehicle_id, request_type,
request_date, status) VALUES (?, ?, ?, ?, ?) RETURNING request_id";
        int generatedRequestId = -1;
        try (PreparedStatement stmt = conn.prepareStatement(requestInsert)) {
            stmt.setInt(1, userId);
            stmt.setInt(2, vehicleId);
            stmt.setString(3, "order");
            stmt.setDate(4, java.sql.Date.valueOf(requestDate.minusDays(15)));
            stmt.setString(5, "pending");
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                generatedRequestId = rs.getInt("request_id");
            }
        }
        // price_offers kaydı ekleme
        String offerInsert = "INSERT INTO price_offers (offer_id, request_id, user_id,
vehicle_id, offer_date, offered_price) VALUES (?, ?, ?, ?, ?, ?) ON CONFLICT DO
NOTHING";
        try (PreparedStatement stmt = conn.prepareStatement(offerInsert)) {
            stmt.setInt(1, offerId);
            stmt.setInt(2, generatedRequestId);
            stmt.setInt(3, userId);
            stmt.setInt(4, vehicleId);
            stmt.setDate(5, java.sql.Date.valueOf(requestDate.minusDays(10)));
            stmt.setBigDecimal(6, new java.math.BigDecimal("50000"));
            stmt.executeUpdate();
        }

        orderRequest = new OrderRequest(userId, vehicleId, requestDate, conn, offerId);
    }
    //test kayıtları silinir
    @AfterEach
    void tearDown() throws SQLException {
        // price_offers tablosundan silme
        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM price_offers
WHERE user_id = ?")) {
            stmt.setInt(1, userId);
            stmt.executeUpdate();
        }
    }

```

```

        // requests tablosundan silme
        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM requests WHERE
user_id = ?")) {
            stmt.setInt(1, userId);
            stmt.executeUpdate();
        }
        // vehicles tablosundan silme
        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM vehicle WHERE
vehicle_id = ?")) {
            stmt.setInt(1, vehicleId);
            stmt.executeUpdate();
        }
        // customers tablosundan silme
        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM customer WHERE
customer_id = ?")) {
            stmt.setInt(1, userId);
            stmt.executeUpdate();
        }
        if (conn != null && !conn.isClosed()) {
            conn.close();
        }
    }

    @Test
    void testProcessRequestValidOffer() throws SQLException {
        System.out.println("\n=== Testing Valid Offer Processing ===");
        System.out.println("User ID: " + userId);
        System.out.println("Vehicle ID: " + vehicleId);
        System.out.println("Request Date: " + requestDate);
        System.out.println("Testing processRequest()...");
        boolean result = orderRequest.processRequest(userId, vehicleId, requestDate);
        if (result) {
            System.out.println("✓ processRequest() returned true - Offer processed successfully.");
        } else {
            System.out.println("X processRequest() returned false - Offer processing failed.");
        }
        assertTrue(result, "Request should be processed successfully for a valid
offer.");
        // Veritabanında kaydın olup olmadığını kontrol etme
        try (PreparedStatement stmt = conn.prepareStatement("SELECT * FROM requests
WHERE user_id = ? AND vehicle_id = ? AND request_type = 'order'")) {
            stmt.setInt(1, userId);
            stmt.setInt(2, vehicleId);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                System.out.println("✓ Request successfully found in the database:");
                System.out.println("Request ID: " + rs.getInt("request_id"));
                System.out.println("Request Type: " + rs.getString("request_type"));
                System.out.println("Request Date: " + rs.getDate("request_date"));
                System.out.println("Status: " + rs.getString("status"));
            } else {
                System.out.println("X No matching request found in the database.");
            }
            assertTrue(rs.next(), "The order request should be recorded in the
database.");
        }
    }
}

```

```

@Test
void testProcessRequestExpiredOffer() throws SQLException {
    System.out.println("\n=== Testing Expired Offer Processing ===");
    System.out.println("User ID: " + userId);
    System.out.println("Vehicle ID: " + vehicleId);
    System.out.println("Current Request Date: " + requestDate);
    LocalDate expiredOfferDate = requestDate.minusDays(40);
    System.out.println("Setting offer date to: " + expiredOfferDate);
    try (PreparedStatement stmt = conn.prepareStatement("UPDATE price_offers SET
offer_date = ? WHERE offer_id = ?")) {
        stmt.setDate(1, java.sql.Date.valueOf(expiredOfferDate));
        stmt.setInt(2, offerId);
        int rowsUpdated = stmt.executeUpdate();
        if (rowsUpdated > 0) {
            System.out.println("✓ Offer date updated successfully in the database.");
        } else {
            System.out.println("X Offer date update failed. No matching offer found.");
        }
    }
    // Teklifin süresi dolmuş mu kontrolü
    boolean result = orderRequest.processRequest(userId, vehicleId, requestDate);
    if (!result) {
        System.out.println("✓ processRequest() returned false - Offer is correctly detected as
expired.");
    } else {
        System.out.println("X processRequest() returned true - Expired offer should not be
processed.");
    }

    assertFalse(result, "Request should not be processed for an expired offer.");
}
}

```

Ekran Çıktısı

The screenshot shows an IDE window with a test runner on the left and a console on the right. The test runner shows 'OrderRequestTest [Runner: JUnit 5] (0,326 s)' with 'Runs: 2/2', 'Errors: 0', and 'Failures: 0'. The console output shows the following:

```

=== Testing Valid Offer Processing ===
User ID: 100
Vehicle ID: 100
Request Date: 2025-05-14
Testing processRequest()...
✓ processRequest() returned true - Offer processed successfully.
✓ Request successfully found in the database:
Request ID: 82
Request Type: order
Request Date: 2025-04-29
Status: pending

=== Testing Expired Offer Processing ===
User ID: 100
Vehicle ID: 100
Current Request Date: 2025-05-14
Setting offer date to: 2025-04-04
✓ Offer date updated successfully in the database.
Teklif süresi dolmuş.
✓ processRequest() returned false - Offer is correctly detected as expired.

```

- Müşteri Kayıt Testi - Sude Elitok:
 - Kaynak Kodu:

```
package yazilim.tests;

import static org.junit.jupiter.api.Assertions.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import yazilim.CustomerRegisterPage;
import yazilim.classes.Customer;

class TestCustomerRegisterPage {
    private static Connection conn;
    private static CustomerRegisterPage registerPage1;
    private static CustomerRegisterPage registerPage2;
    private static int newId;

    @BeforeAll
    static void setUpBeforeClass() throws Exception {
        conn =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje",
"postgres", "12345");
        //CustomerRegisterPage nesneleri
        registerPage1 = new CustomerRegisterPage(conn);
        registerPage2 = new CustomerRegisterPage(conn);
    }
    @AfterEach
    void tearDownAfterClass() throws Exception {
        if (conn != null && !conn.isClosed() && newId > 0) {
            PreparedStatement deleteCustomer = conn.prepareStatement("DELETE FROM
customer WHERE customer_id = ?");
            deleteCustomer.setInt(1, newId);
            deleteCustomer.executeUpdate();
        }
    }

    @Test
    void SuccessfulRegistration() throws SQLException {
        // İlk kullanıcı için geçerli verilerle kayıt işlemi
        registerPage1.getNameField().setText("Ali");
        registerPage1.getSurnameField().setText("Veli");
        registerPage1.getPhoneField().setText("5012345678");
        registerPage1.getEmailField().setText("ali.veli@example.com");
        registerPage1.getPasswordField().setText("securepass");
        registerPage1.getCityField().setText("Ankara");
        registerPage1.getProfessionField().setText("Mühendis");
        registerPage1.getGenderComboBox().setSelectedIndex(0);
        registerPage1.getAgeSpinner().setValue(28);
        registerPage1.getIncomeLevelComboBox().setSelectedIndex(1);
        Customer customer = registerPage1.registerCustomerIfValid();
        newId = customer.getCustomerId();
    }
}
```

```

        assertNotNull(customer);
        assertEquals("Ali", customer.getFirstName());
        assertEquals("ali.veli@example.com", customer.getEmail());
        assertEquals("Veli", customer.getLastName());
        assertEquals("5012345678", customer.getPhoneNumber());
        assertEquals("ali.veli@example.com", customer.getEmail());
        assertEquals("Ankara", customer.getCity());
        assertEquals("Mühendis", customer.getProfession());
        assertEquals("male", customer.getGender());
        assertEquals(28, customer.getAge());
        assertEquals("medium", customer.getIncomeLevel());
    }

    @Test
    void testMissingName() throws SQLException {
        // name boş
        registerPage1.getNameField().setText("");
        registerPage1.getSurnameField().setText("Veli");
        registerPage1.getPhoneField().setText("5012345678");
        registerPage1.getEmailField().setText("ali.veli@example.com");
        registerPage1.getPasswordField().setText("securepass");
        registerPage1.getCityField().setText("Ankara");
        registerPage1.getProfessionField().setText("Mühendis");
        registerPage1.getGenderComboBox().setSelectedIndex(0);
        registerPage1.getAgeSpinner().setValue(28);
        registerPage1.getIncomeLevelComboBox().setSelectedIndex(1);
        Customer customer = registerPage1.registerCustomerIfValid();
        assertNull(customer);
    }

    @Test
    void testInvalidPhoneNumber() throws SQLException {
        // telefon numarası 10 rakamdan oluşmuyor
        registerPage1.getNameField().setText("Ali");
        registerPage1.getSurnameField().setText("Veli");
        registerPage1.getPhoneField().setText("12345");
        registerPage1.getEmailField().setText("ali.veli@example.com");
        registerPage1.getPasswordField().setText("securepass");
        registerPage1.getCityField().setText("Ankara");
        registerPage1.getProfessionField().setText("Mühendis");
        registerPage1.getGenderComboBox().setSelectedIndex(0);
        registerPage1.getAgeSpinner().setValue(28);
        registerPage1.getIncomeLevelComboBox().setSelectedIndex(1);
        Customer customer = registerPage1.registerCustomerIfValid();
        assertNull(customer);
    }

    @Test
    void testEmailAlreadyRegistered() throws SQLException {
        // daha önce aynı email kayıt edilmiş
        registerPage1.getNameField().setText("Ali");
        registerPage1.getSurnameField().setText("Veli");
        registerPage1.getPhoneField().setText("5012345678");
        registerPage1.getEmailField().setText("ali.veli@example.com");
        registerPage1.getPasswordField().setText("securepass");
        registerPage1.getCityField().setText("Ankara");
        registerPage1.getProfessionField().setText("Mühendis");
        registerPage1.getGenderComboBox().setSelectedIndex(0);
        registerPage1.getAgeSpinner().setValue(28);
        registerPage1.getIncomeLevelComboBox().setSelectedIndex(1);
    }

```

```


        Customer customer1 = registerPage1.registerCustomerIfValid();
        newId = customer1.getCustomerId();
        registerPage2.getNameField().setText("Ali Veli");
        registerPage2.getSurnameField().setText("Toprak");
        registerPage2.getPhoneField().setText("5076543210");
        registerPage2.getEmailField().setText("ali.veli@example.com");
        registerPage2.getPasswordField().setText("newpass");
        registerPage2.getCityField().setText("Istanbul");
        registerPage2.getProfessionField().setText("Doktor");
        registerPage2.getGenderComboBox().setSelectedIndex(0);
        registerPage2.getAgeSpinner().setValue(30);
        registerPage2.getIncomeLevelComboBox().setSelectedIndex(2);
        Customer customer2 = registerPage2.registerCustomerIfValid();
        assertNull(customer2);
    }
}





```

- Ekran Çıktısı:

Finished after 6.371 seconds

Runs: 4/4 ✖ Errors: 0 ✖ Failures: 0

▼  TestCustomerRegisterPage [Runner: JUnit 5] (5.388 s)

-  SuccessfulRegistration() (1.721 s)
-  testEmailAlreadyRegistered() (1.607 s)
-  testInvalidPhoneNumber() (0.929 s)
-  testMissingName() (1.129 s)

- Deneme Sürüşü İsteği Gönderme Testi - Sude Elitok:
 - Kaynak Kodu:

```

package yazilim.tests;

import static org.junit.jupiter.api.Assertions.*;
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Timestamp;
import java.time.LocalDate;
import org.junit.jupiter.api.*;
import yazilim.classes.Customer;
import yazilim.classes.Vehicle;
import yazilim.requests.TestDriveRequest;

class TestTestDriveRequest {
    private static Connection conn;
    private static final int TEST_USER_ID = 9000000;
    private static final int TEST_VEHICLE_ID = 9000000;
    private static Customer testCustomer;
    private static Vehicle testVehicle;
}

```

```

@BeforeAll
static void setUpBeforeClass() throws Exception {
    conn =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje",
"postgres", "12345");
    testCustomer = new Customer(TEST_USER_ID, "testuser@example.com", "Test",
"User", "555555555", "male", 20, "tester", "high", "Istanbul", new
Timestamp(System.currentTimeMillis()));
    testVehicle = new Vehicle(TEST_VEHICLE_ID, "MarkaX", "ModelY", 2024, "Basic",
new BigDecimal("25000.00"));
    PreparedStatement insertCustomer = conn.prepareStatement(
        "INSERT INTO customer (customer_id, email, password_customer, first_name,
last_name, phone_number, gender, age, profession, income_level, city, first_visit_date)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?) ON CONFLICT (customer_id) DO NOTHING");
    insertCustomer.setInt(1, testCustomer.getCustomerId());
    insertCustomer.setString(2, testCustomer.getEmail());
    insertCustomer.setString(3, "test_password");
    insertCustomer.setString(4, testCustomer.getFirstName());
    insertCustomer.setString(5, testCustomer.getLastName());
    insertCustomer.setString(6, testCustomer.getPhoneNumber());
    insertCustomer.setString(7, testCustomer.getGender());
    insertCustomer.setInt(8, testCustomer.getAge());
    insertCustomer.setString(9, testCustomer.getProfession());
    insertCustomer.setString(10, testCustomer.getIncomeLevel());
    insertCustomer.setString(11, testCustomer.getCity());
    insertCustomer.setTimestamp(12, testCustomer.getFirstVisitDate());
    insertCustomer.executeUpdate();
    PreparedStatement insertVehicle = conn.prepareStatement(
        "INSERT INTO vehicle (vehicle_id, brand, model, year, package, price)
VALUES (?, ?, ?, ?, ?, ?) ON CONFLICT (vehicle_id) DO NOTHING");
    insertVehicle.setInt(1, testVehicle.getVehicleId());
    insertVehicle.setString(2, testVehicle.getBrand());
    insertVehicle.setString(3, testVehicle.getModel());
    insertVehicle.setInt(4, testVehicle.getYear());
    insertVehicle.setString(5, testVehicle.getPckg());
    insertVehicle.setBigDecimal(6, testVehicle.getPrice());
    insertVehicle.executeUpdate();
    PreparedStatement deleteStmt = conn.prepareStatement(
        "DELETE FROM requests WHERE user_id = ? AND vehicle_id = ? AND
request_type = ?");
    deleteStmt.setInt(1, TEST_USER_ID);
    deleteStmt.setInt(2, TEST_VEHICLE_ID);
    deleteStmt.setString(3, "test_drive");
    deleteStmt.executeUpdate();
}

@Test
void testSuccessRequest() throws Exception {
    LocalDate today = LocalDate.now();
    TestDriveRequest request = new TestDriveRequest(testCustomer.getCustomerId(),
testVehicle.getVehicleId(), today, conn);
    boolean result = request.processRequest(testCustomer.getCustomerId(),
testVehicle.getVehicleId(), today);
    assertTrue(result, "Geçerli kullanıcı ve araç id'si ile istek başarılı olmalı");
}

@Test
void testInvalidUserIdRequest() throws Exception {
    LocalDate today = LocalDate.now();
    int invalidUserId = -1; // Geçersiz kullanıcı id'si

```



```

        TestDriveRequest request = new TestDriveRequest(invalidUserId,
testVehicle.getVehicleId(), today, conn);
        boolean result = request.processRequest(invalidUserId,
testVehicle.getVehicleId(), today);
        assertFalse(result, "Geçersiz kullanıcı id'si ile istek başarısız olmalı");
    }
    @Test
    void testInvalidVehicleIdRequest() throws Exception {
        LocalDate today = LocalDate.now();
        int invalidVehicleId = -123; // Geçersiz araç id'si
        TestDriveRequest request = new TestDriveRequest(testCustomer.getCustomerId(),
invalidVehicleId, today, conn);
        boolean result = request.processRequest(testCustomer.getCustomerId(),
invalidVehicleId, today);
        assertFalse(result, "Geçersiz araç id'si ile istek başarısız olmalı");
    }
    @Test
    void testSuccessRequestDatabaseCorrectness() throws Exception {
        LocalDate today = LocalDate.now();
        TestDriveRequest request = new TestDriveRequest(testCustomer.getCustomerId(),
testVehicle.getVehicleId(), today, conn);
        boolean result = request.processRequest(testCustomer.getCustomerId(),
testVehicle.getVehicleId(), today);
        assertTrue(result, "İstek başarılı olmalı");
        // Gönderilen istek ile veritabanındaki kayıt aynı olmalı
        PreparedStatement stmt = conn.prepareStatement(
            "SELECT user_id, vehicle_id, request_type, request_date, status FROM
requests " +
            "WHERE user_id = ? AND vehicle_id = ? AND request_type = ?");
        stmt.setInt(1, testCustomer.getCustomerId());
        stmt.setInt(2, testVehicle.getVehicleId());
        stmt.setString(3, "test_drive");
        ResultSet rs = stmt.executeQuery();
        assertTrue(rs.next(), "Veritabanında kayıt bulunmalı");
        assertEquals(testCustomer.getCustomerId(), rs.getInt("user_id"), "Kullanıcı
id'si doğru olmalı");
        assertEquals(testVehicle.getVehicleId(), rs.getInt("vehicle_id"), "Araç id'si
doğru olmalı");
        assertEquals("test_drive", rs.getString("request_type"), "İstek tipi doğru
olmalı");
        assertEquals(today, rs.getDate("request_date").toLocalDate(), "Tarih doğru
olmalı");
        assertEquals("pending", rs.getString("status"), "Durum doğru olmalı");
    }
    @AfterAll
    static void tearDownAfterClass() throws Exception {
        if (conn != null && !conn.isClosed()) {
            PreparedStatement deleteRequests = conn.prepareStatement(
                "DELETE FROM requests WHERE user_id = ? OR vehicle_id = ?");
            deleteRequests.setInt(1, TEST_USER_ID);
            deleteRequests.setInt(2, TEST_VEHICLE_ID);
            deleteRequests.executeUpdate();
            PreparedStatement deleteVehicle = conn.prepareStatement(
                "DELETE FROM vehicle WHERE vehicle_id = ?");
            deleteVehicle.setInt(1, TEST_VEHICLE_ID);
            deleteVehicle.executeUpdate();
            PreparedStatement deleteCustomer = conn.prepareStatement(
                "DELETE FROM customer WHERE customer_id = ?");
            deleteCustomer.setInt(1, TEST_USER_ID);

```



```


        deleteCustomer.executeUpdate();
        conn.close();
    }
}
}





```

- Ekran Çıktısı:

Finished after 1.03 seconds

Runs: 4/4  Errors: 0  Failures: 0

✓  TestTestDriveRequest [Runner: JUnit 5] (0.074 s)

- ✓  testSuccessRequestDatabaseCorrectness() (0.026 s)
- ✓  testSuccessRequest() (0.005 s)
- ✓  testInvalidUserIdRequest() (0.033 s)
- ✓  testInvalidVehicleIdRequest() (0.005 s)

- Depo Kullanıcısı Şifre Değişimi Testi - İrem Çelik:
 - Kaynak Kodu:

```

package yazilim.tests;

import org.junit.jupiter.api.*;
import java.sql.*;
import static org.junit.jupiter.api.Assertions.*;

public class testWarehouseChangePassword {
    private static Connection conn;
    private static final int TEST_ID = 10000;
    @BeforeAll
    static void setupDatabase() throws Exception {
        conn =
        DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje",
        "postgres", "12345");
        // Eski varsa sil
        PreparedStatement deleteStmt = conn.prepareStatement("DELETE FROM warehouse
        WHERE id = ?");
        deleteStmt.setInt(1, TEST_ID);
        deleteStmt.executeUpdate();
        // Yeni test kullanıcısı ekle
        PreparedStatement insertStmt = conn.prepareStatement(
            "INSERT INTO warehouse (id, email, password) VALUES (?, ?, ?)");
        insertStmt.setInt(1, TEST_ID);
        insertStmt.setString(2, "test10000@example.com");
        insertStmt.setString(3, "eski123");
        insertStmt.executeUpdate();
    }
    @Test
    void testPasswordChange() throws Exception {
        String oldPassword = "eski123";
        String newPassword = "yeni123";
        boolean success = changePassword(TEST_ID, oldPassword, newPassword);
    }
}

```

```

    assertTrue(success, "Şifre başarıyla değiştirilmelidir.");
    System.out.println("Yeni şifre başarıyla oluşturuldu.");
    // Güncel şifreyi veritabanından kontrol et
    String updatedPassword = getPasswordFromDb(TEST_ID);
    assertEquals(newPassword, updatedPassword, "Yeni şifre veritabanına kaydedilmiş olmalı.");
    System.out.println("Yeni şifre başarıyla doğrulandı.");
}

private boolean changePassword(int id, String oldPass, String newPass) throws SQLException {
    PreparedStatement ps = conn.prepareStatement(
        "UPDATE warehouse SET password = ? WHERE id = ? AND password = ?");
    ps.setString(1, newPass);
    ps.setInt(2, id);
    ps.setString(3, oldPass);
    return ps.executeUpdate() > 0;
}

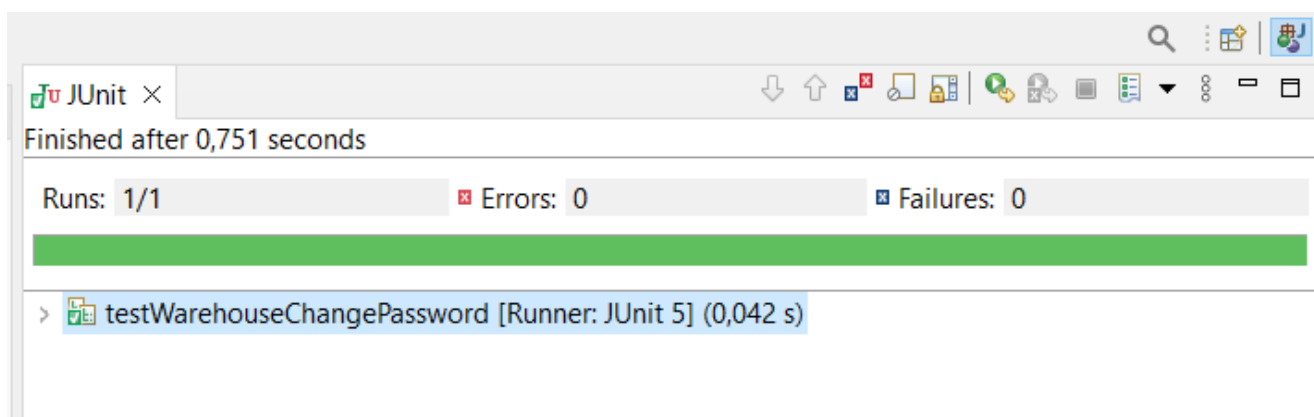
private String getPasswordFromDb(int id) throws SQLException {
    PreparedStatement ps = conn.prepareStatement("SELECT password FROM warehouse WHERE id = ?");
    ps.setInt(1, id);
    ResultSet rs = ps.executeQuery();
    if (rs.next()) {
        return rs.getString("password");
    }
    return null;
}

@AfterAll
static void tearDown() throws Exception {
    // Test kullanıcı sil
    PreparedStatement deleteUser = conn.prepareStatement(
        "DELETE FROM warehouse WHERE id = ?");
    deleteUser.setInt(1, TEST_ID);
    deleteUser.executeUpdate();

    if (conn != null) conn.close();
}
}

```

○ Ekran Çıktısı:



- Müşteri Kullanıcısı Fiyat Teklifi Testi - İrem Çelik:

- Kaynak Kodu:

```
package yazilim.tests;

import org.junit.jupiter.api.*;
import java.sql.*;
import java.time.LocalDate;
import yazilim.requests.PriceOfferRequest;
import static org.junit.jupiter.api.Assertions.*;

public class testPriceOfferRequests {
    private static Connection conn;
    private static final int TEST_USER_ID = 100000;
    private static final int TEST_VEHICLE_ID = 999;
    @BeforeAll
    static void setup() throws Exception {
        conn =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje",
"postgres", "12345");
        // Test aracını ekle
        PreparedStatement vehicleStmt = conn.prepareStatement(
            "INSERT INTO vehicle (vehicle_id, brand, model, year) VALUES (?, ?, ?, ?) ON
CONFLICT (vehicle_id) DO NOTHING");
        vehicleStmt.setInt(1, TEST_VEHICLE_ID);
        vehicleStmt.setString(2, "TestBrand");
        vehicleStmt.setString(3, "TestModel");
        vehicleStmt.setInt(4, 2024);
        vehicleStmt.executeUpdate();

        // Test müşterisi ekle
        PreparedStatement insertCustomer = conn.prepareStatement(
            "INSERT INTO customer (customer_id, email, password_customer, first_name ,
last_name) VALUES (?, ?, ?,?,?) ON CONFLICT (customer_id) DO NOTHING");
        insertCustomer.setInt(1, TEST_USER_ID);
        insertCustomer.setString(2, "testuser@example.com");
        insertCustomer.setString(3, "testpass");
        insertCustomer.setString(4, "testname");
        insertCustomer.setString(5, "testlname");
        insertCustomer.executeUpdate();
        // Aynı kullanıcı+araç+tarih+tip varsa sil
        PreparedStatement deleteReq = conn.prepareStatement(
            "DELETE FROM requests WHERE user_id = ? AND vehicle_id = ? AND request_type
= ? AND request_date = ?");
        deleteReq.setInt(1, TEST_USER_ID);
        deleteReq.setInt(2, TEST_VEHICLE_ID);
        deleteReq.setString(3, "price_offer");
        deleteReq.setDate(4, java.sql.Date.valueOf(LocalDate.now()));
        deleteReq.executeUpdate();
    }
    @Test
    void testPriceOfferRequestInsertion() throws Exception {
        PriceOfferRequest request = new PriceOfferRequest(TEST_USER_ID, TEST_VEHICLE_ID,
LocalDate.now(), conn);
        boolean result = request.processRequest(TEST_USER_ID, TEST_VEHICLE_ID,
LocalDate.now());
        assertTrue(result, "Teklif isteği başarılı olmalı");
        // Doğrula: kayıt eklendi mi?
```

```

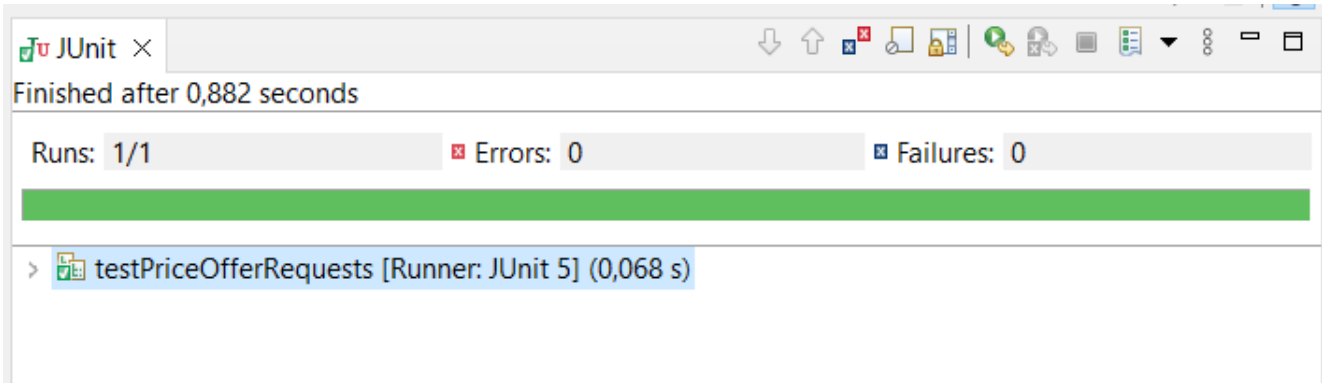
        PreparedStatement check = conn.prepareStatement(
            "SELECT COUNT(*) FROM requests WHERE user_id = ? AND vehicle_id = ? AND
request_type = ?");
        check.setInt(1, TEST_USER_ID);
        check.setInt(2, TEST_VEHICLE_ID);
        check.setString(3, "price_offer");
        ResultSet rs = check.executeQuery();
        rs.next();
        int count = rs.getInt(1);
        assertEquals(1, count, "Teklif kaydı veritabanına eklenmiş olmalı");

        System.out.println("Kayıt veritabanına eklenmiş mi?: " + result);
        PreparedStatement debug = conn.prepareStatement(
            "SELECT * FROM requests WHERE user_id = ? AND vehicle_id = ? AND
request_type = ?");
        debug.setInt(1, TEST_USER_ID);
        debug.setInt(2, TEST_VEHICLE_ID);
        debug.setString(3, "price_offer");
        ResultSet debugRs = debug.executeQuery();
        while (debugRs.next()) {
            System.out.println("BULUNDU: Request ID=" + debugRs.getInt("request_id")
                + ", date=" + debugRs.getDate("request_date")
                + ", status=" + debugRs.getString("status"));
        }
    }
    @AfterAll
    static void cleanup() throws Exception {
        // Test request sil
        PreparedStatement delete = conn.prepareStatement(
            "DELETE FROM requests WHERE user_id = ? AND vehicle_id = ? AND request_type
= ?");
        delete.setInt(1, TEST_USER_ID);
        delete.setInt(2, TEST_VEHICLE_ID);
        delete.setString(3, "price_offer");
        delete.executeUpdate();

        // Test aracını sil
        PreparedStatement deleteVehicle = conn.prepareStatement(
            "DELETE FROM vehicle WHERE vehicle_id = ?");
        deleteVehicle.setInt(1, TEST_VEHICLE_ID);
        deleteVehicle.executeUpdate();
        // Test kullanıcısını sil
        PreparedStatement deleteCustomer = conn.prepareStatement(
            "DELETE FROM customer WHERE customer_id = ?");
        deleteCustomer.setInt(1, TEST_USER_ID);
        deleteCustomer.executeUpdate();
        conn.close();
    }
}

```

- Ekran Çıktısı:



- Stoğa Araç Ekleme İşlemi Birim Testi - Ahmet Mahir Demirelli:
 - Kaynak Kodu:

```
package yazilim.tests;

import org.junit.jupiter.api.*;
import yazilim.AddCarToStockPage;
import yazilim.classes.Vehicle;
import yazilim.classes.Warehouse;
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import static org.junit.jupiter.api.Assertions.*;

public class testAddCarToStock {
    private static Connection conn;
    private AddCarToStockPage page;
    private static Warehouse warehouse = new Warehouse(2, "warehouse@example.com");
    private final static Vehicle vehicle = new Vehicle(126, "Lamborghini", "Huracan",
2021, "Full", BigDecimal.valueOf(15000000.00));
    @BeforeAll
    public static void setUp() throws SQLException {
        System.out.println("\n===== Testing Add Vehicle To Stock =====");
        conn =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje",
"postgres", "12345");
        try (PreparedStatement stmt = conn.prepareStatement(
            "INSERT INTO warehouse (id, email, password) VALUES (?, ?, ?) ON CONFLICT DO
NOTHING")) {
            stmt.setInt(1, warehouse.getId());
            stmt.setString(2, warehouse.getEmail());
            stmt.setString(3, "testpassword");
            stmt.executeUpdate();
        }
    }
    @BeforeEach
    public void init() {
        page = new AddCarToStockPage(warehouse, conn);
    }

    // Yeni araç ekleme
```

```

@Test
public void testAddVehicle_1() {
    System.out.println("\n=== testAddVehicle_1 Started ===");
    int result = page.addCarToStock(vehicle.getBrand(), vehicle.getModel(),
vehicle.getYear(), vehicle.getPkg(), vehicle.getPrice(), 5);
    assertTrue(result == 1, "Volkswagen Polo 2021 Full paket ekleme başarılı
olmalı.");
    System.out.println("Volkswagen Polo 2021 Full paket eklemesi beklendiği gibi
başarılı oldu.");
    int expectedStock = 5;
    int actualStock =
getWarehouseStockForVehicle(getVehicleIdFromFeatures(vehicle.getBrand(),
vehicle.getModel(), vehicle.getYear(), vehicle.getPkg()));
    assertEquals(expectedStock, actualStock, "Depodaki stok 5 olmalı.");
    System.out.println("Volkswagen Polo 2021 Full için depodaki stok=5
doğrulandı.");
}
// Zaten olan bir aracı eklemeye çalışma
@Test
public void testAddVehicle_2() {
    System.out.println("\n=== testAddVehicle_2 Started ===");
    int result = page.addCarToStock(vehicle.getBrand(), vehicle.getModel(),
vehicle.getYear(), vehicle.getPkg(), vehicle.getPrice(), 5);
    assertTrue(result == 0, "Volkswagen Polo 2021 Full paket ekleme başarısız
olmalı.");
    System.out.println("Volkswagen Polo 2021 Full paket eklemesi zaten kayıtlı araç
olduğu için beklendiği gibi başarısız oldu.");
}
@AfterAll
public static void closeConnection() throws Exception {
    vehicle.setVehicleId(getVehicleIdFromFeatures(vehicle.getBrand(),
vehicle.getModel(), vehicle.getYear(), vehicle.getPkg()));
    try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM stock WHERE
vehicle_id = ?")) {
        stmt.setInt(1, vehicle.getVehicleId());
        stmt.executeUpdate();
    }

    try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM vehicle WHERE
vehicle_id = ?")) {
        stmt.setInt(1, vehicle.getVehicleId());
        stmt.executeUpdate();
    }

    try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM warehouse WHERE
id = ?")) {
        stmt.setInt(1, warehouse.getId());
        stmt.executeUpdate();
    }

    if (conn != null && !conn.isClosed()) {
        conn.close();
    }
}

private static int getVehicleIdFromFeatures(String brand, String model, int year,
String pkg) {
    int vehicle_id = 0;
    String query = "SELECT vehicle_id FROM vehicle WHERE brand = ? AND model = ? AND
year = ? AND package = ?;";

```

```

        PreparedStatement stmt;
        try {
            stmt = conn.prepareStatement(query);
            stmt.setString(1, brand);
            stmt.setString(2, model);
            stmt.setInt(3, year);
            stmt.setString(4, pkg);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                vehicle_id = rs.getInt("vehicle_id");
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return vehicle_id;
    }

    private int getWarehouseStockForVehicle(int vehicleId) {
        int stock = -1;
        try (PreparedStatement stmt = conn.prepareStatement(
            "SELECT quantity FROM stock WHERE vehicle_id = ? AND location_type = 'warehouse'")) {
            stmt.setInt(1, vehicleId);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                stock = rs.getInt("quantity");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            fail("Veritabanı hatası: " + e.getMessage());
        }
        return stock;
    }
}

```

○ Ekran Çıktısı:

```

1 package yazilim.tests;
2
3 import org.junit.jupiter.api.*;
4
5
6
7
8
9
10
11
12
13
14
15
16 public class testAddCarToStock {
17     private static Connection conn;
18     private AddCarToStockPage page;
19     private static WarehouseOrDealer warehouse = new WarehouseOrDealer(2, "warehouse@example.com", "WAREHOUSE");
20     private final static Vehicle vehicle = new Vehicle(126, "Lamborghini", "Huracan", 2021, "Full", BigDecimal.valueOf(150000));
21
22     @BeforeAll
23     public static void setUp() throws SQLException {
24         System.out.println("\n==== Testing Add Vehicle To Stock =====");
25         conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje", "postgres", "12345");
26
27         ===== Testing Add Vehicle To Stock =====
28
29         === testAddVehicle 1 Started ===
30         Volkswagen Polo 2021 Full paket eklemesi beklendiği gibi başarılı oldu.
31         Volkswagen Polo 2021 Full için depodaki stok=5 doğrulandı.
32
33         === testAddVehicle 2 Started ===
34         Volkswagen Polo 2021 Full paket eklemesi zaten kayıtlı araç olduğu için beklendiği gibi başarısız oldu.

```


- Depodan Bayiye Araç Çekme İşlemi Birim Testi - Ahmet Mahir Demirelli:
 - Kaynak Kodu:

```
package yazilim.tests;

import org.junit.jupiter.api.*;
import yazilim.PullCarFromStockPage;
import yazilim.classes.Vehicle;
import yazilim.classes.Dealer;
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import static org.junit.jupiter.api.Assertions.*;

public class testPullVehicleToDealerFromWarehouse {
    private static Connection conn;
    private PullCarFromStockPage page;
    private final static Dealer dealer = new Dealer(340, "dealer@example.com");
    private final static Vehicle vehicle_1 = new Vehicle(124, "Volkswagen", "Tiguan",
2021, "Full", BigDecimal.valueOf(1500000.00));
    private final static Vehicle vehicle_2 = new Vehicle(125, "Ford", "Focus", 2021,
"Full", BigDecimal.valueOf(1500000.00));
    private static int stockId = 100;

    @BeforeAll
    public static void setUp() throws SQLException {
        System.out.println("\n=== Testing Pull Vehicle To Dealer From Warehouse ===");
        conn =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje",
"postgres", "12345");
        try (PreparedStatement stmt = conn.prepareStatement(
            "INSERT INTO dealer (id, email, password) VALUES (?, ?, ?) ON CONFLICT DO
NOTHING")) {
            stmt.setInt(1, dealer.getId());
            stmt.setString(2, dealer.getEmail());
            stmt.setString(3, "testpassword");
            stmt.executeUpdate();
        }
        // Araç 1 ekle
        try (PreparedStatement stmt = conn.prepareStatement(
            "INSERT INTO vehicle (vehicle_id, brand, model, year, package, price)
VALUES (?, ?, ?, ?, ? ,?) ON CONFLICT DO NOTHING")) {
            stmt.setInt(1, vehicle_1.getVehicleId());
            stmt.setString(2, vehicle_1.getBrand());
            stmt.setString(3, vehicle_1.getModel());
            stmt.setInt(4, vehicle_1.getYear());
            stmt.setString(5, vehicle_1.getPckg());
            stmt.setBigDecimal(6, vehicle_1.getPrice());
            stmt.executeUpdate();
        }
        // Araç 1 stoğa ekle
        try (PreparedStatement stmt = conn.prepareStatement(
```

```

        "INSERT INTO stock (stock_id, vehicle_id, location_type, quantity,
updated_at) VALUES (?, ?, ?, ?, ?) ON CONFLICT DO NOTHING")) {
            stmt.setInt(1, stockId);
            stmt.setInt(2, vehicle_1.getVehicleId());
            stmt.setString(3, "warehouse");
            stmt.setInt(4, 5);
            stmt.setTimestamp(5, new Timestamp(System.currentTimeMillis()));
            stmt.executeUpdate();
        }

// Araç 2 ekle
try (PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO vehicle (vehicle_id, brand, model, year, package, price)
VALUES (?, ?, ?, ?, ?, ?) ON CONFLICT DO NOTHING")) {
    stmt.setInt(1, vehicle_2.getVehicleId());
    stmt.setString(2, vehicle_2.getBrand());
    stmt.setString(3, vehicle_2.getModel());
    stmt.setInt(4, vehicle_2.getYear());
    stmt.setString(5, vehicle_2.getPckg());
    stmt.setBigDecimal(6, vehicle_2.getPrice());
    stmt.executeUpdate();
}

// Araç 2 stoğa ekle
try (PreparedStatement stmt = conn.prepareStatement(
    "INSERT INTO stock (stock_id, vehicle_id, location_type, quantity,
updated_at) VALUES (?, ?, ?, ?, ?) ON CONFLICT DO NOTHING")) {
    stmt.setInt(1, stockId + 1);
    stmt.setInt(2, vehicle_2.getVehicleId());
    stmt.setString(3, "dealer");
    stmt.setInt(4, 2);
    stmt.setTimestamp(5, new Timestamp(System.currentTimeMillis()));
    stmt.executeUpdate();
}
}

@BeforeEach
public void init() {
    page = new PullCarFromStockPage(dealer, conn);
}

// Depodan araç çekiyor
@Test
public void testPullVehicle_1() {
    System.out.println("\n=== testPullVehicle_1 Started ===");
    int result = page.pullVehicleToDealerFromWarehouse(vehicle_1.getVehicleId(), 2);
    assertTrue(result == 1, "Araç çekme işlemi başarılı olmalı. (Araç ID: " +
vehicle_1.getVehicleId() + ")");
    System.out.println(vehicle_1.getVehicleId() + " Id'li aracı bayiye çekme işlemi
beklendiği gibi başarılı oldu.");
    int expectedStock = 3;
    int actualStock = getWarehouseStockForVehicle(vehicle_1.getVehicleId());
    assertEquals(expectedStock, actualStock, "Depodaki stok 3 olmalı.");
    System.out.println(vehicle_1.getVehicleId() + " Id'li araç için stock=3 kaldığı
doğrulandı.");
}

// Zaten bayide olan aracı çekmeye çalışıyor
@Test
public void testPullVehicle_2() {

```

```

        System.out.println("\n=== testPullVehicle_2 Started ===");
        int result = page.pullVehicleToDealerFromWarehouse(vehicle_2.getVehicleId(), 1);
        assertFalse(result == 0, "Araç çekme işlemi başarısız olmalı. (Araç ID: " +
vehicle_2.getVehicleId() + ")");
        System.out.println(vehicle_2.getVehicleId() + " Id'li araç zaten bayide olduğu
için çekme işlemi beklendiği gibi başarısız oldu.");
    }

    // Stoktan daha fazla talep ediyor
    @Test
    public void testPullVehicle_3() {
        System.out.println("\n=== testPullVehicle_3 Started ===");
        int result = page.pullVehicleToDealerFromWarehouse(vehicle_1.getVehicleId(), 5);
        assertTrue(result == -1, "Araç çekme işlemi stok yetersizliği sebebi ile
beklendiği gibi başarısız oldu.");
        System.out.println(vehicle_1.getVehicleId() + " Id'li aracı bayiye çekme işlemi
stok yetersizliği sebebi ile beklendiği gibi başarısız oldu.");
    }

    // Depoda araç kalmayacak şekilde çekiyor
    @Test
    public void testPullVehicle_4() {
        System.out.println("\n=== testPullVehicle_4 Started ===");
        int result = page.pullVehicleToDealerFromWarehouse(vehicle_1.getVehicleId(), 3);
        assertTrue(result == 1, "Araç çekme işlemi başarılı olmalı (Araç ID: " +
vehicle_1.getVehicleId() + ")");
        System.out.println(vehicle_1.getVehicleId() + " Id'li aracı bayiye çekme işlemi
beklendiği gibi başarılı oldu.");
        int remainingRows = getWarehouseStockRowCount(vehicle_1.getVehicleId());
        assertEquals(0, remainingRows, "Depoda bu araçtan kayıt kalmamalı (Araç ID: " +
vehicle_1.getVehicleId() + ")");
        System.out.println(vehicle_1.getVehicleId() + " Id'li araçtan depoda kalmadığı
doğrulandı.");
    }

    @AfterAll
    public static void tearDown() throws SQLException {
        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM stock WHERE
vehicle_id = ?")) {
            stmt.setInt(1, vehicle_1.getVehicleId());
            stmt.executeUpdate();
        }

        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM stock WHERE
vehicle_id = ?")) {
            stmt.setInt(1, vehicle_2.getVehicleId());
            stmt.executeUpdate();
        }

        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM vehicle WHERE
vehicle_id = ?")) {
            stmt.setInt(1, vehicle_1.getVehicleId());
            stmt.executeUpdate();
        }

        try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM vehicle WHERE
vehicle_id = ?")) {
            stmt.setInt(1, vehicle_2.getVehicleId());

```

```

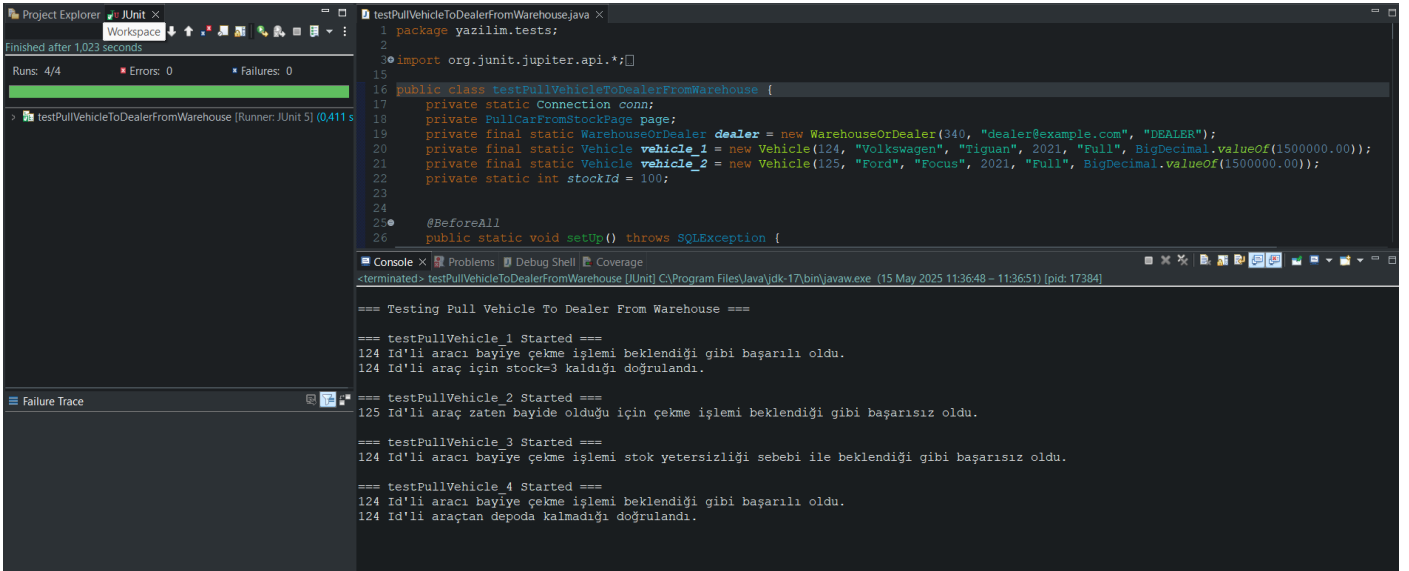
        stmt.executeUpdate();
    }
    try (PreparedStatement stmt = conn.prepareStatement("DELETE FROM warehouse WHERE
id = ?")) {
        stmt.setInt(1, dealer.getId());
        stmt.executeUpdate();
    }
    if (conn != null && !conn.isClosed()) {
        conn.close();
    }
}

private int getWarehouseStockForVehicle(int vehicleId) {
    int stock = -1;
    try (PreparedStatement stmt = conn.prepareStatement(
        "SELECT quantity FROM stock WHERE vehicle_id = ? AND location_type =
'warehouse'")) {
        stmt.setInt(1, vehicleId);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            stock = rs.getInt("quantity");
        }
    } catch (SQLException e) {
        e.printStackTrace();
        fail("Veritabanı hatası: " + e.getMessage());
    }
    return stock;
}

private int getWarehouseStockRowCount(int vehicleId) {
    int count = -1;
    try (PreparedStatement stmt = conn.prepareStatement(
        "SELECT COUNT(*) FROM stock WHERE vehicle_id = ? AND location_type =
'warehouse'")) {
        stmt.setInt(1, vehicleId);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            count = rs.getInt(1);
        }
    } catch (SQLException e) {
        e.printStackTrace();
        fail("Veritabanı hatası: " + e.getMessage());
    }
    return count;
}
}

```

- Ekran Çıktısı:



The screenshot shows an IDE with a project explorer on the left, a code editor in the center, and a console window at the bottom. The code editor displays a Java test class named `testPullVehicleToDealerFromWarehouse` with imports for `org.junit.jupiter.api` and `java.sql`. The class contains static variables for a database connection, a dealer, and two vehicles. It also has a `setUp` method that prints a message. The console window shows the output of the test, including the message "Testing Pull Vehicle To Dealer From Warehouse" and the results of four test cases.

```
package yazilim.tests;

import org.junit.jupiter.api.*;
import java.sql.*;

public class testPullVehicleToDealerFromWarehouse {
    private static Connection conn;
    private static final int testUserId = 999;
    private static final int testVehicleId = 888;
    @BeforeAll
    public static void setUp() throws SQLException {
        System.out.println("\n=== Dummy veriler ekleniyor (DealerOrderApproval Test)");
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje", "postgres", "12345");

        PreparedStatement insertCustomer = conn.prepareStatement("""
            INSERT INTO customer (customer_id, email, password_customer, first_name, last_name, phone_number, gender, age, profession, income_level, city, first_visit_date)
            VALUES (999, 'testuser@example.com', 'testpass', 'Test', 'User', '1234567890', 'male', 30, 'Engineer', 'medium', 'TestCity', CURRENT_DATE)
            ON CONFLICT DO NOTHING
            """);
        insertCustomer.executeUpdate();

        PreparedStatement insertVehicle = conn.prepareStatement("""
            INSERT INTO vehicle (vehicle_id, brand, model, year, package, price)
            VALUES (888, 'Toyota', 'Corolla', 2023, 'Premium', 480000)
            ON CONFLICT DO NOTHING
            """);
        insertVehicle.executeUpdate();

        PreparedStatement insertRequest = conn.prepareStatement("""
```

- Satış Onayı Testi - Emirhan Yusuf Toptaş:
 - Kaynak Kodu:

```
package yazilim.tests;

import org.junit.jupiter.api.*;
import java.sql.*;

public class testDealerOrderApproval {
    private static Connection conn;
    private static final int testUserId = 999;
    private static final int testVehicleId = 888;
    @BeforeAll
    public static void setUp() throws SQLException {
        System.out.println("\n=== Dummy veriler ekleniyor (DealerOrderApproval Test)");
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje", "postgres", "12345");

        PreparedStatement insertCustomer = conn.prepareStatement("""
            INSERT INTO customer (customer_id, email, password_customer, first_name, last_name, phone_number, gender, age, profession, income_level, city, first_visit_date)
            VALUES (999, 'testuser@example.com', 'testpass', 'Test', 'User', '1234567890', 'male', 30, 'Engineer', 'medium', 'TestCity', CURRENT_DATE)
            ON CONFLICT DO NOTHING
            """);
        insertCustomer.executeUpdate();

        PreparedStatement insertVehicle = conn.prepareStatement("""
            INSERT INTO vehicle (vehicle_id, brand, model, year, package, price)
            VALUES (888, 'Toyota', 'Corolla', 2023, 'Premium', 480000)
            ON CONFLICT DO NOTHING
            """);
        insertVehicle.executeUpdate();

        PreparedStatement insertRequest = conn.prepareStatement("""
```

```

        INSERT INTO requests (request_id, user_id, vehicle_id, request_type,
request_date, status)
        VALUES (8000, 999, 888, 'order', CURRENT_DATE, 'pending')
        ON CONFLICT DO NOTHING
        """);
insertRequest.executeUpdate();

PreparedStatement insertPriceOffer = conn.prepareStatement("""
        INSERT INTO price_offers (offer_id, request_id, user_id, vehicle_id,
offer_date, offered_price)
        VALUES (9000, 8000, 999, 888, CURRENT_DATE, 480000)
        ON CONFLICT DO NOTHING
        """);
insertPriceOffer.executeUpdate();

PreparedStatement insertStock = conn.prepareStatement("""
        INSERT INTO stock (vehicle_id, location_type, quantity, updated_at)
        VALUES (888, 'dealer', 1, CURRENT_TIMESTAMP)
        ON CONFLICT DO NOTHING
        """);
insertStock.executeUpdate();
System.out.println("Dummy veriler başarıyla eklendi.");
}

@Test
public void testOrderApprovalFlow() throws SQLException {
    System.out.println("\n=== Satış Onaylama Testi Başladı ===");

    PreparedStatement updateStockStmt = conn.prepareStatement("""
        UPDATE stock SET quantity = quantity - 1
        WHERE vehicle_id = ? AND location_type = 'dealer' AND quantity > 0
        """);
    updateStockStmt.setInt(1, testVehicleId);
    int affected = updateStockStmt.executeUpdate();
    assertEquals(1, affected, "Stoktan 1 araç düşülmeliydi");
    System.out.println("Stoktan 1 araç başarıyla düşüldü.");

    PreparedStatement insertSale = conn.prepareStatement("""
        INSERT INTO sales (user_id, vehicle_id, sale_date, sale_price)
        VALUES (?, ?, CURRENT_DATE, ?)
        """);
    insertSale.setInt(1, testUserId);
    insertSale.setInt(2, testVehicleId);
    insertSale.setDouble(3, 480000);
    int rows = insertSale.executeUpdate();
    assertEquals(1, rows, "Satış kaydı başarılı olmalı");
    System.out.println("Satış kaydı 'sales' tablosuna başarıyla eklendi.");

    PreparedStatement deleteOffer = conn.prepareStatement("""
        DELETE FROM price_offers WHERE user_id = ? AND vehicle_id = ?
        """);
    deleteOffer.setInt(1, testUserId);
    deleteOffer.setInt(2, testVehicleId);
    int deleted = deleteOffer.executeUpdate();
    assertEquals(1, deleted, "Fiyat teklifi silinmiş olmalı");
    System.out.println("Fiyat teklifi başarıyla silindi.");
    System.out.println("=== Satış Onaylama Testi Başarıyla Tamamlandı ===\n");
}

@AfterAll
public static void cleanup() throws SQLException {

```

```
        System.out.println("\n=== Cleanup başlatıldı ===");

        try (PreparedStatement ps = conn.prepareStatement("DELETE FROM price_offers WHERE request_id = ?")) {
            ps.setInt(1, 8000);
            ps.executeUpdate();
        }

        try (PreparedStatement ps = conn.prepareStatement("DELETE FROM sales WHERE user_id = ? AND vehicle_id = ?")) {
            ps.setInt(1, 999);
            ps.setInt(2, 888);
            ps.executeUpdate();
        }

        try (PreparedStatement ps = conn.prepareStatement("DELETE FROM requests WHERE user_id = ?")) {
            ps.setInt(1, 999);
            ps.executeUpdate();
        }

        try (PreparedStatement ps = conn.prepareStatement("DELETE FROM stock WHERE vehicle_id = ?")) {
            ps.setInt(1, 888);
            ps.executeUpdate();
        }

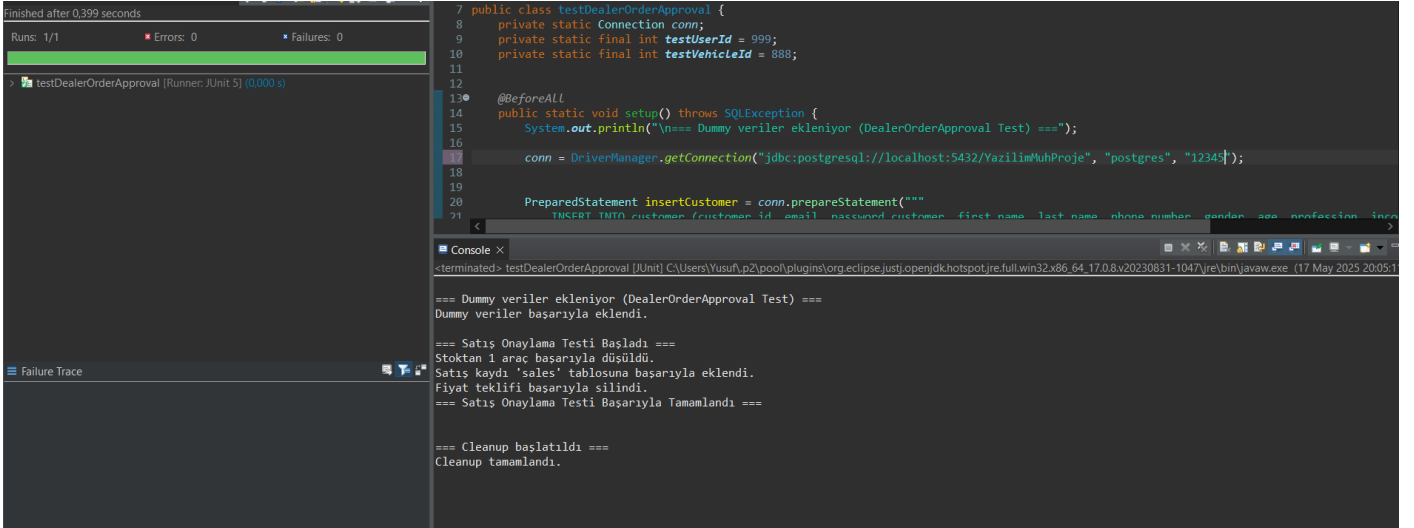
        try (PreparedStatement ps = conn.prepareStatement("DELETE FROM vehicle WHERE vehicle_id = ?")) {
            ps.setInt(1, 888);
            ps.executeUpdate();
        }

        try (PreparedStatement ps = conn.prepareStatement("DELETE FROM customer WHERE customer_id = ?")) {
            ps.setInt(1, 999);
            ps.executeUpdate();
        }

        if (conn != null && !conn.isClosed()) {
            conn.close();
        }

        System.out.println("Cleanup tamamlandı.");
    }
}
```

- Ekran Çıktısı:



The screenshot shows an IDE with a Java test class `testDealerOrderApproval` and its console output. The code includes static variables for `testUserId` and `testVehicleId`, and a `setup` method that prints a message and connects to a PostgreSQL database. The console output shows the test execution results, including the message "Dummy veriler ekleniyor (DealerOrderApproval Test) ===" and "Dummy veriler başarıyla eklendi."

```
7 public class testDealerOrderApproval {
8     private static Connection conn;
9     private static final int testUserId = 999;
10    private static final int testVehicleId = 888;
11
12
13    @BeforeAll
14    public static void setup() throws SQLException {
15        System.out.println("\n=== Dummy veriler ekleniyor (DealerOrderApproval Test) ===");
16
17        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje", "postgres", "12345");
18
19
20        PreparedStatement insertCustomer = conn.prepareStatement("
21            INSERT INTO customer (customer_id, email, password, customer_first_name, last_name, phone_number, gender, age, profession, income_level, city, first_visit_date)
22            VALUES (999, 'testuser@example.com', 'testpass', 'Test', 'User', '1234567890', 'male', 30, 'Engineer', 'medium', 'TestCity', CURRENT_DATE)
23            ON CONFLICT DO NOTHING
24        ");
25        insertCustomer.executeUpdate();
26
27        PreparedStatement insertVehicle = conn.prepareStatement("
28            INSERT INTO vehicle (vehicle_id, brand, model, year, package, price)
29            VALUES (888, 'Honda', 'Civic', 2023, 'Sport', 450000)
30            ON CONFLICT DO NOTHING
31        ");
32        insertVehicle.executeUpdate();
33
34        PreparedStatement insertRequest = conn.prepareStatement("
35            INSERT INTO requests (request_id, user_id, vehicle_id, request_type, request_date, status)
36            VALUES (?, ?, ?, 'price_offer', CURRENT_DATE, 'pending')
37            ON CONFLICT DO NOTHING
38        ");
39        insertRequest.setInt(1, testRequestId);
40        insertRequest.executeUpdate();
41    }
42}
```

```
<terminated> testDealerOrderApproval [JUnit] C:\Users\Yusuf\AppData\Local\Temp\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64-17.0.8.v20230831-1047\jre\bin\javaw.exe (17 May 2025 20:05:10)
=== Dummy veriler ekleniyor (DealerOrderApproval Test) ===
Dummy veriler başarıyla eklendi.
=== Satış Onaylama Testi Başladı ===
Stoktan 1 araç başarıyla düüldü.
Satış kaydı 'sales' tablosuna başarıyla eklendi.
Fiyat teklifi başarıyla silindi.
=== Satış Onaylama Testi Başarıyla Tamamlandı ===
=== Cleanup başlatıldı ===
Cleanup tamamlandı.
```

- Bayi Teklif Onaylama Testi - Emirhan Yusuf Toptaş:
 - Kaynak Kodu:

```
package yazilim.tests;

import org.junit.jupiter.api.*;
import java.sql.*;
import static org.junit.jupiter.api.Assertions.*;

public class testDealerRequestApproval {
    private static Connection conn;
    private static final int testUserId = 999;
    private static final int testVehicleId = 888;
    private static final int testRequestId = 8001;
    @BeforeAll
    public static void setup() throws SQLException {
        System.out.println("\n=== Dummy veriler ekleniyor (DealerRequestPage Test) ===");
        conn =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/YazilimMuhProje",
"postgres", "12345");
        conn.prepareStatement("
            INSERT INTO customer (customer_id, email, password_customer, first_name,
last_name, phone_number, gender, age, profession, income_level, city, first_visit_date)
VALUES (999, 'testuser@example.com', 'testpass', 'Test', 'User',
'1234567890', 'male', 30, 'Engineer', 'medium', 'TestCity', CURRENT_DATE)
ON CONFLICT DO NOTHING
        ").executeUpdate();
        conn.prepareStatement("
            INSERT INTO vehicle (vehicle_id, brand, model, year, package, price)
VALUES (888, 'Honda', 'Civic', 2023, 'Sport', 450000)
ON CONFLICT DO NOTHING
        ").executeUpdate();
        PreparedStatement insertRequest = conn.prepareStatement("
            INSERT INTO requests (request_id, user_id, vehicle_id, request_type,
request_date, status)
VALUES (?, ?, ?, 'price_offer', CURRENT_DATE, 'pending')
ON CONFLICT DO NOTHING
        ");
        insertRequest.setInt(1, testRequestId);
    }
}
```



```

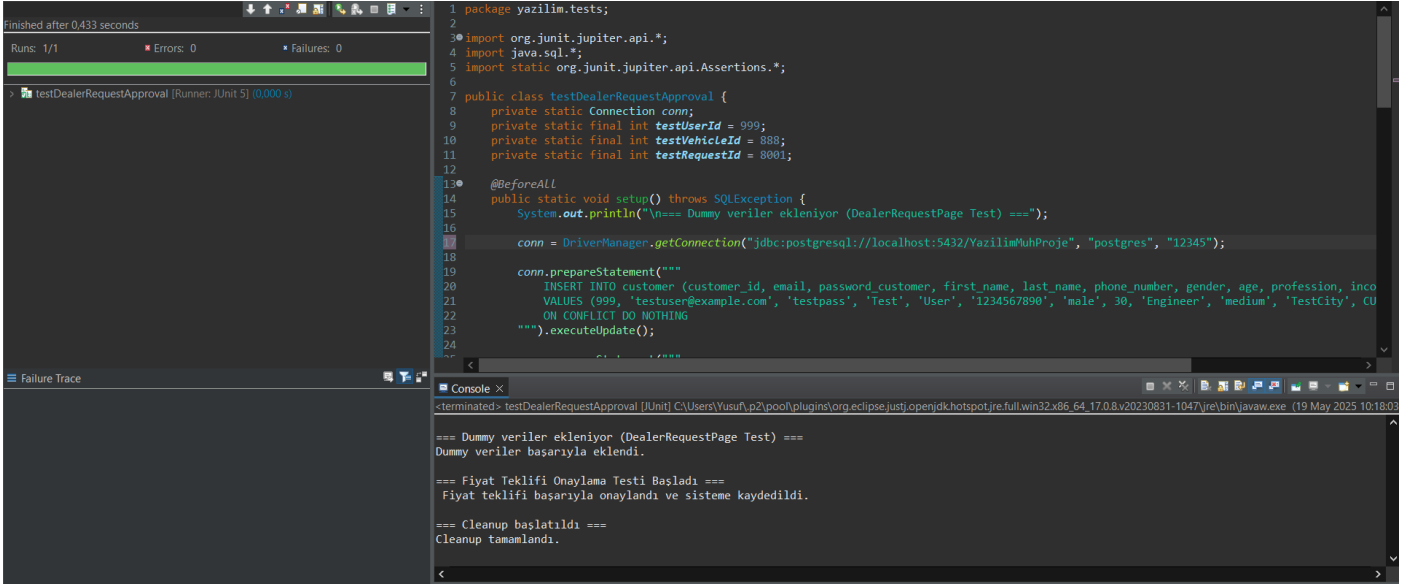
        insertRequest.setInt(2, testUserId);
        insertRequest.setInt(3, testVehicleId);
        insertRequest.executeUpdate();
        System.out.println("Dummy veriler başarıyla eklendi.");
    }
    @Test
    public void testApprovePriceOffer() throws SQLException {
        System.out.println("\n=== Fiyat Teklifi Onaylama Testi Başladı ===");

        PreparedStatement updateReq = conn.prepareStatement(
            "UPDATE requests SET status = 'accepted' WHERE request_id = ?"
        );
        updateReq.setInt(1, testRequestId);
        int updateCount = updateReq.executeUpdate();
        assertEquals(1, updateCount, "Request durumu güncellenmiş olmalı");

        PreparedStatement insertOffer = conn.prepareStatement("""
            INSERT INTO price_offers (offer_id, request_id, user_id, vehicle_id,
offer_date, offered_price)
            VALUES (9001, ?, ?, ?, CURRENT_DATE, ?)
            """);
        insertOffer.setInt(1, testRequestId);
        insertOffer.setInt(2, testUserId);
        insertOffer.setInt(3, testVehicleId);
        insertOffer.setDouble(4, 450000);
        int insertCount = insertOffer.executeUpdate();
        assertEquals(1, insertCount, "Fiyat teklifi başarıyla eklenmiş olmalı");
        System.out.println(" Fiyat teklifi başarıyla onaylandı ve sisteme kaydedildi.");
    }
    @AfterAll
    public static void cleanup() throws SQLException {
        System.out.println("\n=== Cleanup başlatıldı ===");
        conn.prepareStatement("DELETE FROM price_offers WHERE request_id =
8001").executeUpdate();
        conn.prepareStatement("DELETE FROM requests WHERE request_id =
8001").executeUpdate();
        conn.prepareStatement("DELETE FROM vehicle WHERE vehicle_id =
888").executeUpdate();
        conn.prepareStatement("DELETE FROM customer WHERE customer_id =
999").executeUpdate();
        if (conn != null && !conn.isClosed()) {
            conn.close();
        }
        System.out.println("Cleanup tamamlandı.");
    }
}

```

Ekran Çıktısı:



The screenshot displays an IDE with a Java test class and its execution results. The top-left pane shows the test runner status: "Finished after 0.433 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 0". The top-right pane shows the source code of the test class, which includes imports for JUnit and JDBC, and a test class named `testDealerRequestApproval` with a `setUp` method and a `test` method. The bottom-left pane shows the "Failure Trace" which is empty. The bottom-right pane shows the console output, which includes the test runner status, the test class name, and the test results.

```
1 package yazilim.tests;
2
3 import org.junit.jupiter.api.*;
4 import java.sql.*;
5 import static org.junit.jupiter.api.Assertions.*;
6
7 public class testDealerRequestApproval {
8     private static Connection conn;
9     private static final int testUserId = 999;
10    private static final int testVehicleId = 888;
11    private static final int testRequestId = 8001;
12
13    @BeforeAll
14    public static void setUp() throws SQLException {
15        System.out.println("\n=== Dummy veriler ekleniyor (DealerRequestPage Test) ===");
16
17        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/Vazilim%uhProje", "postgres", "12345");
18
19        conn.prepareStatement("""
20            INSERT INTO customer (customer_id, email, password, customer, first_name, last_name, phone_number, gender, age, profession, income, city, country)
21            VALUES (999, 'testuser@example.com', 'testpass', 'Test', 'User', '1234567890', 'male', 30, 'Engineer', 'medium', 'TestCity', 'CU
22            ON CONFLICT DO NOTHING
23            """).executeUpdate();
24
25    }
```

Failure Trace

Console

```
<terminated> testDealerRequestApproval [JUnit] C:\Users\Yusuflp2\pools\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.8.v20230831-1047\jre\bin\javaw.exe (19 May 2025 10:18:03)

=== Dummy veriler ekleniyor (DealerRequestPage Test) ===
Dummy veriler başarıyla eklendi.

=== Fiyat Teklifi Onaylama Testi Başladı ===
Fiyat teklifi başarıyla onaylandı ve sisteme kaydedildi.

=== Cleanup başlatıldı ===
Cleanup tamamlandı.
```