



Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği

Yazılım Mühendisliği Dersi Proje Ödevi

Dersin Yürütücüsü: Prof. Dr. Oya Kalıpsız

Mayıs, 2025

Konu: 2 – Motorlu Araç Filo Yönetim Sistemi

21011702 – Melih Yelman

22011615 – Halil Salih Tosun

21011069 – Ahmet Çam

1.	Proje Alan Tanımı	3
2.	Kabul ve Kısıtlar	3
3.	Proje İş-Zaman Çizelgesi.....	4
4.	Ekip Organizasyon Şeması ve Görev Dağılımları	5
5.	Proje Risk Tablosu	6
6.	Analiz.....	7
6.1.	Kullanım Senaryosu Diyagramı	7
6.2.	Kullanım Senaryoları	8
6.3.	Kavramsal Sınıf Diyagramı	19
7.	Tasarım	20
7.1.	Ayrıntılı Sınıf Diyagramı.....	20
7.2.	Sıralama (Sequence) Diyagramları	21
7.3.	Durum (State) Diyagramları.....	22
7.4.	Etkinlik (Activity) Diyagramları.....	23
8.	Birim Testi Sınamaları ve Projede Kullanılan Teknolojiler	24

1. Proje Alan Tanımı

Bu proje, filo araç kiralama ve yönetimi hizmeti sunan firmalara yönelik bir araç yönetim ve raporlama bilgi sistemidir. Sistem, kiralanan veya şirkete ait (özmal) araçların bakım, yakıt, kasko, parça değişimi gibi tüm harcamalarını takip eder; kilometre verilerini kaydeder; görev ve sürücü atamalarını yönetir. Hem firma yetkilileri hem de dış kaynak firmaları tarafından veri girişi ve raporlama yapılabilir. Ayrıca geçmiş harcama verilerine göre Hareketli Ortalama Yöntemi ile gelecek dönem harcamaları tahminlenir.

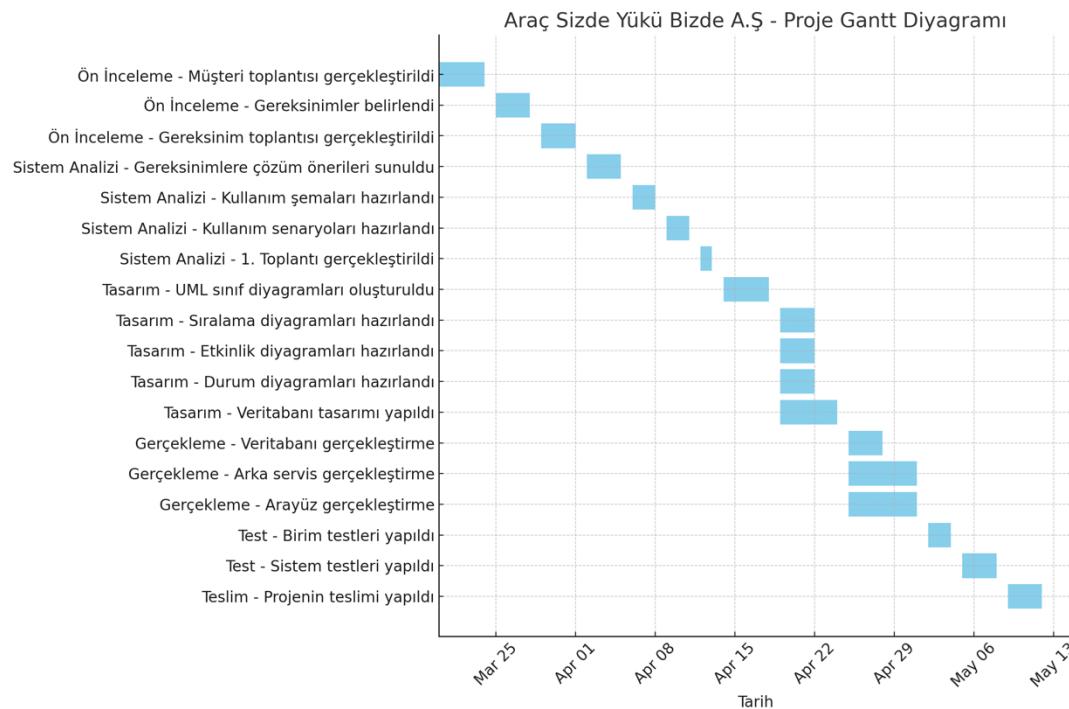
2. Kabul ve Kısıtlar

Bu kısımda projeyi gerçekleştirirken kabul ettiğimiz bilgiler yer almaktadır.

- Veri girişi dış kaynak firması tarafından yapılacaktır.
- Raporlama işlemleri hem firma yetkilileri hem de dış kaynak firma yetkilileri tarafından yapılabilecektir.
- Sistem hem kiralanmış hem de özmal (şirketin sahip olduğu) araçları destekleyecektir.
- Her ay başında araçların kilometre bilgisi sisteme girilecek ve bir önceki aya ait veri saklanacaktır.
- Havuzda bekleyen araçlara görev ataması ve ardından sürücü ataması yapılmadan araç kullanımı mümkün değildir.
- Belli araçlar belirli çalışanlara atanabilir ve bu durumda çalışanların kullandıkları kilometre bilgisi sisteme girilir.
- Aracı kullanan kişi kilometre bilgisini girdikten sonra bu bilgi dış kaynak firması tarafından doğrulanır.
- Tüm araçlar için bakım, kasko, yakıt, tamir ve genel toplam olmak üzere harcama bilgileri tutulacaktır.
- Harcamalar tablo ve grafiklerle raporlanabilecektir.
- Sistem, geçmiş harcama verilerine dayanarak Hareketli Ortalama Yöntemi ile bir sonraki dönemin tahmini harcamasını hesaplayacaktır.
- Hareketli Ortalama Yöntemi dışındaki tahminleme algoritmaları kullanılmayacaktır.
- Kullanıcı rolleri sabittir: dış kaynak firması, firma yetkilisi ve sürücü.
- Sistemde çok dilli destek bulunmamaktadır; arayüz Türkçe olacaktır.
- Araç türüne (otomobil, minibüs, kamyon vs.) göre özel bir işlem yapılmayacaktır.
- Sisteme dışsal veriler (örneğin yakıt endeksi, hava durumu vs.) dahil edilmeyecektir.
- Sistem web tabanlı olarak geliştirilecek, farklı lokasyonlardan erişim sağlanabilecektir.
- Projeye büyük ölçekli yeni özellikler eklenmeyecektir; yalnızca belirtilen işlevler geliştirilecektir.
- Veri girişi sadece dış kaynak firması tarafından yapılacaktır; firma içi kullanıcılar sadece görüntüleme ve raporlama yapabilecektir.
- Her harcama kalemi tüm araç türleri için geçerli olacak, ayrılmayacaktır.
- Raporlar geçmiş yıllara ve dönemsel harcamalara göre detaylı alınabilecektir.

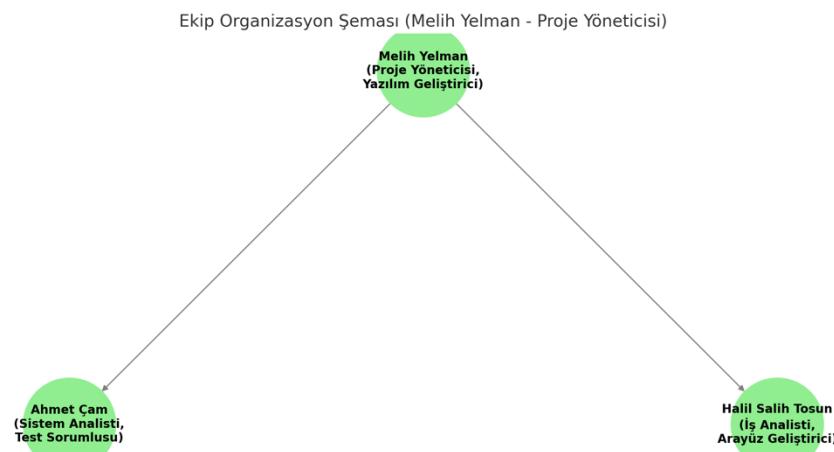
3. Proje İş-Zaman Çizelgesi

Bu kısımda projeye dair gantt diyagramı bulunmaktadır.



4. Ekip Organizasyon Şeması ve Görev Dağılımları

Şekil 4.1'de ekip şeması verilmiştir. Aşağıda proje ekibi üyeleri ve üstlendikleri roller belirtilmiştir.



Şekil 4.1 - Ekip Organizasyon Şeması

- Ahmet Çam
İş Analisti, Test Süreci Sorumlusu
- Melih Yelman
Yazılım Geliştirici, Veritabanı Tasarımı ve Uygulaması
- Halil Salih Tosun
Sistem Analisti, Arayüz Geliştirici

5. Proje Risk Tablosu

Bu kısımda projeye dair riskleri içeren risk tablosu bulunmaktadır.

Risk ID	Adı	Türü/Grubu	Etkisi	Olasılık	Önemi
*	1 Gereksinim Değişiklikleri	Proje	Büyük	Yüksek	Büyük
	2 Güvenlik Açıkları	Teknik	Büyük	Orta	Büyük
	3 Performans Sorunları	Teknik	Orta	Orta	Orta
	4 Veri Kaybı veya Tutarlılığı	Teknik	Büyük	Düşük	Orta
	5 Zaman Çizelgesi Gecikmeleri	Proje	Büyük	Yüksek	Büyük

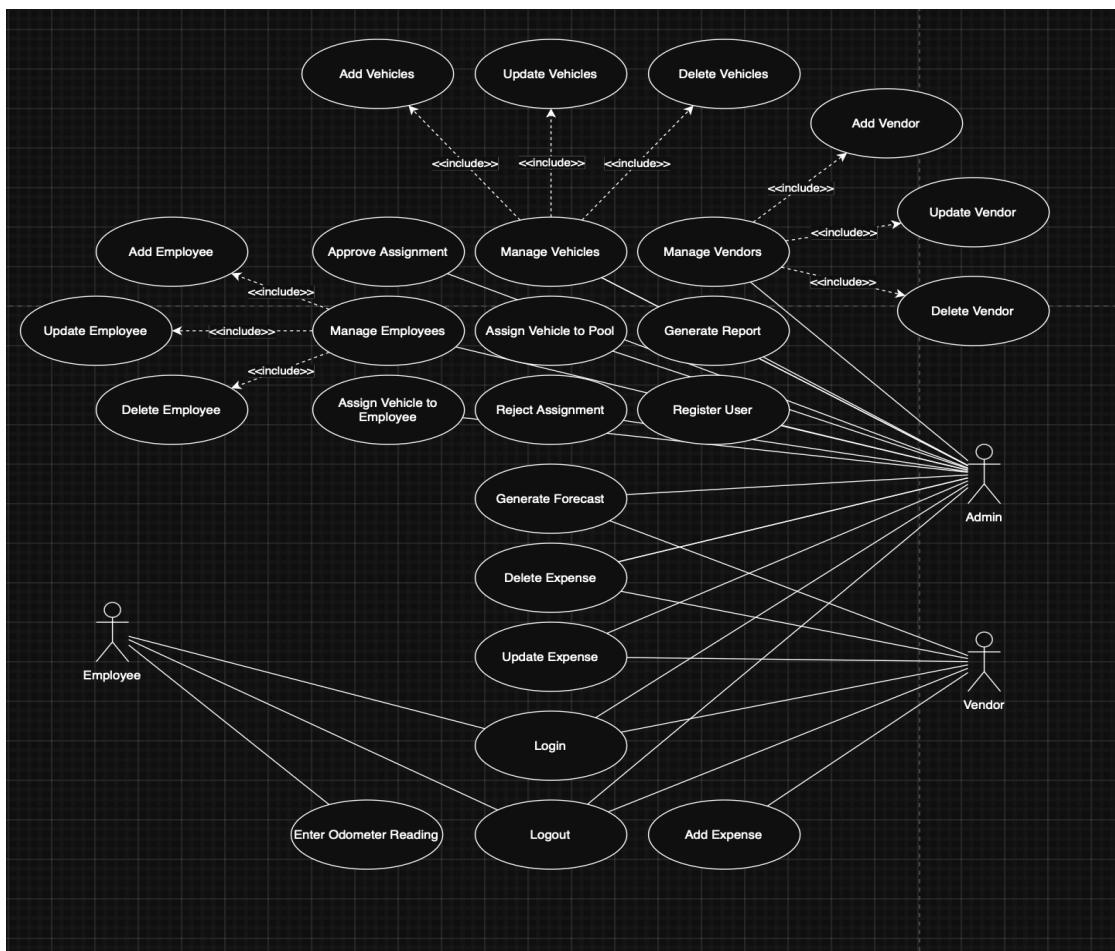
Tablo 5.1 - Proje Risk Tablosu

6. Analiz

Bu kısımda analize dair diyagramlar ve kullanım senaryoları bulunmaktadır.

6.1. Kullanım Senaryosu Diyagramı

Bu kısımda sistemin genel kullanım senaryosu bulunmaktadır.



Şekil 6.1.1 - Use Case Diyagramı

6.2. Kullanım Senaryoları

Bu kısımda kullanım senaryoları bulunmaktadır.

Use Case Adı	Update Vehicles
Amaç	Sistemde kayıtlı bir aracın bilgisini (plaka, model, marka, lease tarihleri vb.) güncellemek
Aktörler	ADMIN
Önkoşullar	<p>ADMIN sisteme giriş yapmış olmalıdır</p> <p>Güncellenecek araç veritabanında bulunmalıdır</p>
Temel Akış	<ol style="list-style-type: none">1. ADMIN, güncellenecek araca ait sayfaya veya form ekranına gider2. ADMIN, değiştirilecek alanları (örneğin model, leaseEndDate vb.) düzenler3. Sistem, girilen verileri doğrular4. Sistem, <code>VehicleService.update(id, updatedVehicle)</code> metodunu çağrıarak ilgili kaydı veritabanında günceller5. Sistem, güncellenmiş <code>VehicleDTO</code> bilgisini döndürür ve "Araç bilgileri güncellendi" mesajı gösterir
Son Koşullar	<p>Araç kaydi, veritabanında yeni değerlerle güncellenmiştir</p> <p>ADMIN, güncellenmiş bilgilere anında erişebilir</p>

Use Case Adı	Delete Vehicles
Amaç	Mevcut bir araç kaydını sistemden silmek
Aktörler	ADMIN
Önkoşullar	<p>ADMIN sisteme giriş yapmış olmalıdır</p> <p>Silinecek araç ID'si mevcut olmalıdır</p>
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, silinmek istenen aracı seçer 2. ADMIN, "Sil" komutunu onaylar 3. Sistem, <code>VehicleService.deleteById(id)</code> metodunu çağırarak araç kaydını veritabanından siler 4. Sistem, "Araç başarıyla silindi" mesajı döndürür
Son Koşullar	<p>İlgili <code>Vehicle</code> kaydı veritabanından kaldırılmıştır</p> <p>Bu araca ait ek tablolar veya ilişkilendirmeler (kodun silme stratejisine göre) güncellenir veya silinir</p>

Use Case Adı	Add Vehicles
Amaç	Yeni bir araç kaydının sisteme eklenmesini sağlar
Aktörler	ADMIN
Önkoşullar	<p>ADMIN sisteme giriş yapmış olmalıdır</p> <p>Araç ekleme işlemine erişim izni olmalıdır (kodda <code>VehicleController</code> → POST <code>/api/vehicles</code> normalde ADMIN tarafından kullanılır)</p>
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, "Yeni Araç Ekle" formuna girer 2. ADMIN, araç plaka, marka, model, mülkiyet tipi gibi zorunlu alanları doldurur 3. Sistem, girilen verileri doğrular 4. Sistem, <code>VehicleService.save()</code> metodunu çağırarak verileri veritabanına kaydeder 5. Sistem, kaydedilen aracın <code>VehicleDTO</code> bilgilerini döndürür ve "Araç başarıyla eklendi" mesajı gösterir.
Son Koşullar	<p>Yeni <code>Vehicle</code> kaydı veritabanına eklenir</p> <p>ADMIN, oluşturulan aracın ID'sine ve detaylarına erişebilir</p>

Use Case Adı	Manage Vehicles
Amaç	Araçları yönetmek için yüksek seviyeli bir Use Case. “Add Vehicles”, “Update Vehicles” ve “Delete Vehicles” alt Use Case’lerini içerir
Aktörler	ADMIN
Önkoşullar	ADMIN sisteme giriş yapmış olmalıdır
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, “Araç Yönetimi” ekranına girer 2. Sistem, tüm araçların listesini gösterir 3. ADMIN, ekleme veya güncelleme veya silme işlemlerinden birini başlatır
Son Koşullar	Araç ekleme, güncelleme veya silme işlemlerinden biri başarıyla tamamlanır

Use Case Adı	Add Employee
Amaç	Sisteme yeni bir çalışan kaydetmek
Aktörler	ADMIN
Önkoşullar	<p>ADMIN giriş yapmış olmalıdır</p> <p>Kullanıcı ekleme yetkisi bulunmalıdır</p>
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, “Çalışan Ekle” formuna girer 2. ADMIN, çalışanın adı, departmanı, kullanıcı adı, şifre bilgilerini doldurur 3. Sistem, <code>EmployeeService.save(e)</code> metodunu çağırarak veritabanına kaydeder (arka planda <code>UserRole.EMPLOYEE</code> set edilir). 4. Sistem, “Çalışan eklendi” mesajı döndürür
Son Koşullar	<p>Yeni bir Employee kaydı (aynı zamanda User) oluşturulmuştur</p> <p>ADMIN, eklenen çalışanın ID'sine ve bilgilerine erişebilir</p>

Use Case Adı	Update Employee
Amaç	Mevcut bir çalışanın bilgilerini (isim, departman, kullanıcı adı, rol vb.) güncellemek
Aktörler	ADMIN
Önkoşullar	ADMIN giriş yapmış olmalıdır Güncellenecek çalışan var olmalıdır
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, güncellemek istediği çalışanın detay ekranına gider 2. ADMIN, değiştirilecek alanları doldurur (örneğin departman, username) 3. Sistem, <code>EmployeeService.save()</code> veya <code>EmployeeService.update()</code> metoduyla kaydı günceller 4. Sistem, güncellenmiş Employee bilgilerini döndürür ve "Çalışan güncellendi" mesajı verir <p>Çalışan kaydı yeni değerleriyle veritabanında saklanmıştır</p>
Son Koşullar	ADMIN güncellenmiş bilgilere anında ulaşabilir
Use Case Adı	Delete Employee
Amaç	Sistemde kayıtlı bir çalışanı silmek
Aktörler	ADMIN
Önkoşullar	ADMIN giriş yapmış olmalıdır Silinecek çalışan kaydı var olmalıdır
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, silmek istediği çalışanın kimliğini (ID) seçer 2. Sistem, <code>EmployeeService.deleteById(id)</code> metodunu çağırır 3. Sistem, başarı mesajı gösterir
Son Koşullar	Çalışan kaydı veritabanından kaldırılır Bu çalışana bağlı varsa "EmployeeAssignment" gibi varlıklar da güncellenir veya silinir

Use Case Adı	Manage Employees
Amaç	Çalışan ekleme, güncelleme ve silme işlemlerini tek çatı altında toplar
Aktörler	ADMIN
Önkoşullar	ADMIN giriş yapmış olmalıdır
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, "Çalışan Yönetimi" menüsüne girer 2. Sistem, çalışan listesini sunar 3. ADMIN, yeni ekleme veya mevcut kayıt güncelleme/silme adımlarından birini seçer
Son Koşullar	Seçilen eylem başarıyla tamamlanır Çalışan listesi güncel hale gelir
Use Case Adı	Approve Assignment
Amaç	Bir çalışan atamasını (EmployeeAssignment) onaylamak
Aktörler	ADMIN
Önkoşullar	ADMIN giriş yapmış olmalıdır Onaylanacak atama, "beklemede" (approved=false) konumunda olmalıdır
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, onaylamak istediği atamayı seçer 2. Sistem, EmployeeAssignmentService.approve(id, km) metodunu çağırır, onay sırasında km bilgisi de girilir ve TripLog oluşturulur 3. Sistem, atamanın approved alanını true yapar, yeni TripLog kaydı ekler 4. Sistem, "Atama onaylandı" mesajı gösterir
Son Koşullar	İlgili EmployeeAssignment "onaylı" hale gelir Yeni TripLog kaydı eklenmiştir (km bilgisiyle)

Use Case Adı	Assign Vehicle to Pool
Amaç	Aracı havuz görevine (PoolAssignment) atar
Aktörler	ADMIN
Önkoşullar	ADMIN giriş yapmış olmalıdır Atanacak araç kaydı bulunmalıdır
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, "Havuz Ataması" sayfasına girer 2. ADMIN, araç ID, başlangıç-bitiş tarihi ve "mission" (görev) bilgisini girer 3. Sistem, <code>VehicleService.assignToPool(vehicleId, mission, start, end)</code> metodunu çağırır 4. Kodda "PoolAssignment" nesnesi oluşturulur, "mission" set edilir, <code>vehicle</code> field'i ilgili araca bağlanır ve kaydedilir 5. Sistem, "Araç havaşa atandı" mesajı döndürür
Son Koşullar	Yeni PoolAssignment kaydı oluşur ve ilgili Vehicle nesnesinin atamalar listesine eklenir

Use Case Adı	Assign Vehicle to Employee
Amaç	Aracı bir çalışana (EmployeeAssignment) atamak
Aktörler	ADMIN
Önkoşullar	ADMIN giriş yapmış olmalıdır. Atanacak araç ve çalışanın kayıtlı olması gereklidir
Temel Akış	<ol style="list-style-type: none"> 1. ADMIN, atama ekranına girer 2. ADMIN, araç ID, çalışan ID, başlangıç-bitiş tarihi bilgilerini girer 3. Sistem, <code>VehicleService.assignToEmployee(vehicleId, employeeId, start, end)</code> metodunu çağırır 4. Kodda "EmployeeAssignment" oluşturulur, çalışan ve araç ile ilişkilendirilir, veritabanına kaydedilir 5. Sistem, "Araç çalışana atandı" mesajı döndürür
Son Koşullar	EmployeeAssignment kaydı eklenir (approved=false varsayılan) İlgili Vehicle ve Employee entity'leri "assignments" listesine yeni kaydı ekler

Use Case Adı	Reject Assignment
Amaç	Bir çalışan atamasını reddetmek (onay vermemek).
Aktörler	ADMIN
Önkoşullar	ADMIN giriş yapmış olmalıdır Reddedilecek atama kaydı onaylanmamış veya bekliyor durumda olabilir
Temel Akış	<p>1. ADMIN, "Atamayı Reddet" seçenekini kullanır</p> <p>2. Sistem, <code>EmployeeAssignmentService.reject(id)</code> metodunu çağırır</p> <p>3. Kodda atamanın <code>approved=false</code> set edilmesi sağlanır; var olan TripLog silinir</p> <p>4. Sistem, "Atama reddedildi" mesajı verir</p>
Son Koşullar	<code>EmployeeAssignment.approved</code> değeri <code>false</code> olarak güncellenir İlgili TripLog kaydı silinir

Use Case Adı	Generate Report
Amaç	Belirli bir tarih aralığı için araçların veya tek bir aracın km, masraf, atama bilgileri vb. özetlenmiş raporunu üretir
Aktörler	ADMIN
Önkoşullar	ADMIN giriş yapmış olmalıdır Rapor alınacak tarih aralıkları geçerli olmalıdır (<code>from <= to</code>)
Temel Akış	<p>1. ADMIN, "Rapor Oluştur" sayfasına girer</p> <p>2. ADMIN, tarih aralığını (<code>from, to</code>) ve isterse araç ID'sini seçer</p> <p>3. Sistem, <code>ReportService.generateAllReports(...)</code> veya <code>generateVehicleReport(...)</code> metodunu çağırır</p> <p>4. Kod, <code>VehicleService</code>, <code>ExpenseService</code>, <code>AssignmentService</code> gibi servislerden verileri toplar; masrafları tür bazında, km farkını ve atamaları raporlar</p> <p>5. Sistem, <code>Report entity</code>'sına rapor içeriğini yazar ve veritabanına kaydeder</p> <p>6. Sistem, oluşturulan raporun temel özeti (<code>ReportDTO</code>) ekranda gösterir veya indirilebilir hale getirir</p>
Son Koşullar	Rapor kaydı (<code>Report entity</code>) veritabanına eklenmiştir ADMIN rapor içeriğini inceleyebilir

Use Case Adı	Register User
Amaç	Sisteme yeni kullanıcı (User entity) kaydı yapar
Aktörler	ADMIN
Önkoşullar	Kullanıcı kaydı yapma izni veya ilgili endpoint'e erişim olması gereklidir
Temel Akış	<ol style="list-style-type: none"> 1. Aktör, "Register User" formunu açar 2. Aktör, username, password, role bilgilerinin tamamını girer 3. Sistem, <code>UserService.create(username, rawPassword, role)</code> metodunu çağırır 4. Kod, şifreyi Bcrypt ile hashleyerek <code>UserRepository</code> üzerinden kaydı saklar 5. Sistem, "Kullanıcı kaydedildi" mesajı döndürür
Son Koşullar	Yeni User veritabanına eklenir, "id" değeri oluşturulur Sistemde belirtilen role ile giriş yapabilecek yeni kullanıcı mevcuttur
Use Case Adı	Generate Forecast
Amaç	Belirli bir geçmiş döneme bakarak (ör. son 3 ay) masrafların tür bazında ortalamasını hesaplayıp tahmin değeri üretir
Aktörler	ADMIN VENDOR
Önkoşullar	Aktör giriş yapmış olmalıdır
Temel Akış	<ol style="list-style-type: none"> 1. Aktör, "Forecast" sayfasını açar ve "period" değerini (kaç aylık veri) girer 2. Sistem, <code>ForecastService.generateExpenseForecast(periodInMonths)</code> metodunu çağırır 3. Kod, <code>ExpenseRepository.findAll()</code> ile masrafları toplar ve son X ayın verilerini filtreler 4. Kod, her bir <code>ExpenseType</code> için aylık ortalama hesaplar 5. Sistem, sonuçları aktöre JSON veya arayüzde özet olarak döndürür
Son Koşullar	1. Belirtilen dönem için masraf tahminleri hesaplanır 2. ADMIN veya VENDOR, farklı gider türleri bazında ortalamaları görüntüler

Use Case Adı	Delete Expense
Amaç	Sistemdeki bir gider kaydını silmek
Aktörler	ADMIN VENDOR
Önkoşullar	Aktör giriş yapmış olmalıdır Silinecek masraf kaydı mevcut olmalıdır
Temel Akış	1. Aktör, "Gider Listesi" ekranında silmek istediği masrafı seçer 2. Sistem, <code>ExpenseService.deleteById(id)</code> metodunu çağırır 3. Sistem, "Gider başarıyla silindi" mesajı döndürür
Son Koşullar	1. İlgili <code>Expense</code> veritabanından kaldırılmıştır 2. Gider toplamlarında veya raporlamada bu masraf artık görünmez

Use Case Adı	Update Expense
Amaç	Bir masraf kaydını (miktar, tarih, açıklama, vendor, vehicle) güncellemek
Aktörler	ADMIN VENDOR
Önkoşullar	Aktör giriş yapmış olmalıdır Güncellenecek masraf kaydı mevcut olmalıdır
Temel Akış	1. Aktör, güncellemek istediği masrafın detayına girer 2. Aktör, yeni tarih, tutar, açıklama vb. alanları düzenler 3. Sistem, <code>ExpenseService.findById(id)</code> ile kaydı bulur ve gelen verilerle günceller 4. Sistem, başarı durumunu bildirir ve güncellenmiş masraf kaydını döndürür
Son Koşullar	1. Masraf kaydı yeni bilgilerle saklanır 2. Rapor ve hesaplamalar yeni değerleri yansıtır

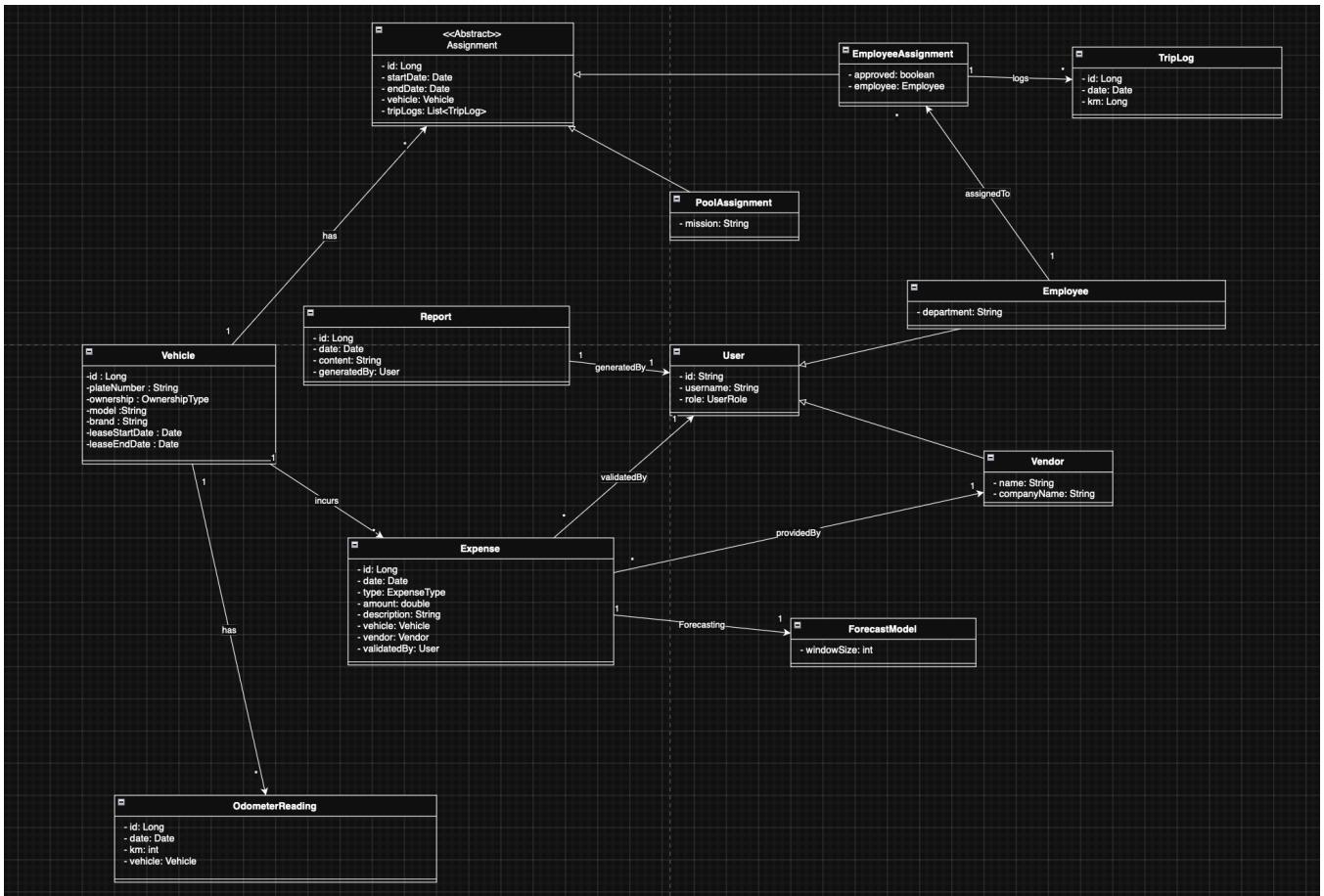
Use Case Adı	Login
Amaç	Kullanıcının (ADMIN, EMPLOYEE, VENDOR) JWT tabanlı kimlik doğrulama ile sisteme giriş yapmasını sağlar
Aktörler	ADMIN VENDOR EMPLOYEE
Önkoşullar	Sistemde kayıtlı bir kullanıcı (User) kaydı olmalıdır Kullanıcı, kullanıcı adı ve şifre bilgilerine sahip olmalıdır
Temel Akış	<p>1. Aktör, "Login" sayfasında username ve password'ü girer</p> <p>2. Sistem, AuthController.login() → AuthenticationManager.authenticate() metodunu çağırır.</p> <p>3. Şifre doğrusa sistem JWT üretecek döndürür</p> <p>4. Kullanıcı bu token'i sonraki isteklerde Authorization header'ında taşıır</p>
Son Koşullar	1. Kullanıcıya geçerli bir JWT token verilmiştir 2. Kullanıcı, token ile korunan endpoint'lere erişebilir

Use Case Adı	Logout
Amaç	Kullanıcı oturumunu sonlandırır
Aktörler	ADMIN VENDOR EMPLOYEE
Önkoşullar	Aktör giriş yapmış olmalıdır
Temel Akış	<p>1. Aktör, "Logout" komutuna basar</p> <p>2. Sistem, istemci tarafında token'i silmeye veya geçersiz kılmaya yönelik bir mekanizma çalıştırır</p>
Son Koşullar	<p>1. Kullanıcının geçerli token'i devre dışı kalır veya istemci tarafında unutulur</p> <p>2. Kullanıcı korunan kaynaklara erişemez</p>

Use Case Adı	Add Expense
Amaç	Yeni bir masraf kaydını (Expense) sisteme eklemek
Aktörler	ADMIN VENDOR
Önkoşullar	<p>Aktor giriş yapmış olmalıdır</p> <p>Masrafın ait olduğu araç ID'si ve varsa vendor ID'si geçerli olmalıdır</p>
Temel Akış	<p>1. Aktör, "Yeni Gider Ekle" formuna girer</p> <p>2. Tarih, tutar, açıklama, type (FUEL, MAINTENANCE vb.), vehicleId, vendorId gibi alanları doldurur</p> <p>3. Sistem, <code>ExpenseService.save(e, vehicleId, vendorId, validatedById)</code> metodunu çağırır</p> <p>4. Sistem, ilgili Vehicle ve Vendor kaydını bularak Expense nesnesine set eder</p> <p>5. Sistem, masraf kaydını veritabanına ekler ve "Gider eklendi" mesajı gösterir</p>
Son Koşullar	<p>1. Yeni Expense kaydı veritabanına eklenir</p> <p>2. Raporda ve tahmin hesaplarında bu masraf dikkate alınır</p>
Use Case Adı	Enter Odometer Reading
Amaç	Bir aracın kilometre bilgisini (OdometerReading) sisteme girmek
Aktörler	EMPLOYEE
Önkoşullar	<p>Kullanıcı giriş yapmış olmalıdır</p> <p>Km bilgisinin girileceği araç sisteme kayıtlı olmalıdır</p>
Temel Akış	<p>1. Aktör, "Odometer Reading Ekle" ekranına girer</p> <p>2. Tarih ve km değerini girer, ayrıca vehicle_id seçer</p> <p>3. Sistem, <code>OdometerReadingService.save(reading)</code> metoduyla kaydı ekler</p> <p>4. Sistem, "Km bilgisi kaydedildi" mesajı gösterir</p>
Son Koşullar	<p>1. Yeni OdometerReading kaydı veritabanına eklenmiştir</p> <p>2. Aracın km takibi raporlara yansır</p>

6.3. Kavramsal Sınıf Diyagramı

Bu kısımda sınımlara dair methodların yer almadığı kavramsal sınıf diyagramı bulunur



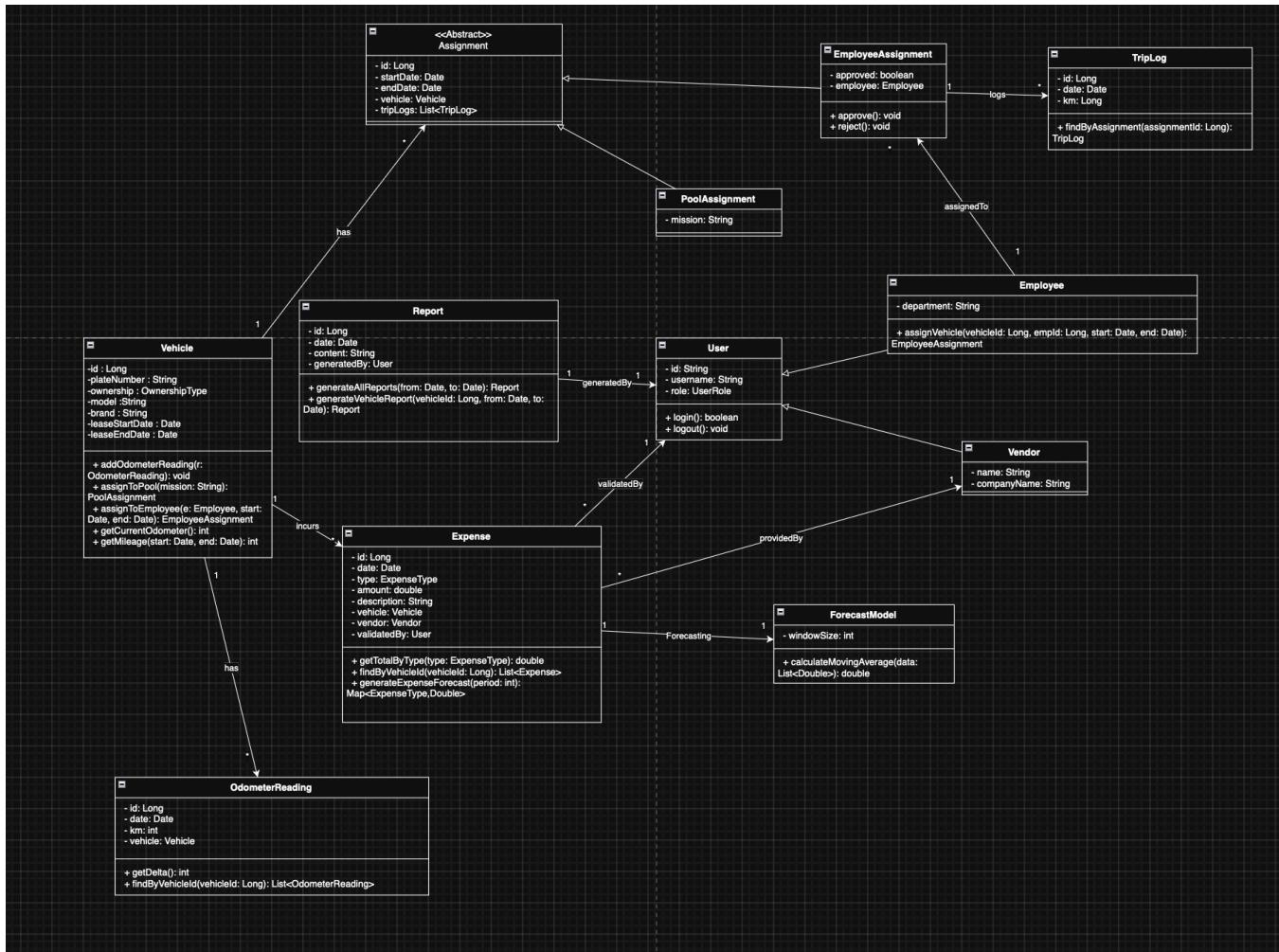
Şekil 6.4.1. - Kavramsal Sınıf Diyagramı

7. Tasarım

Bu kısımda tasarıma dair diyagramlar bulunmaktadır.

7.1. Ayrıntılı Sınıf Diyagramı

Bu kısımda ayrıntılı sınıf diyagramına yer verilmektedir.

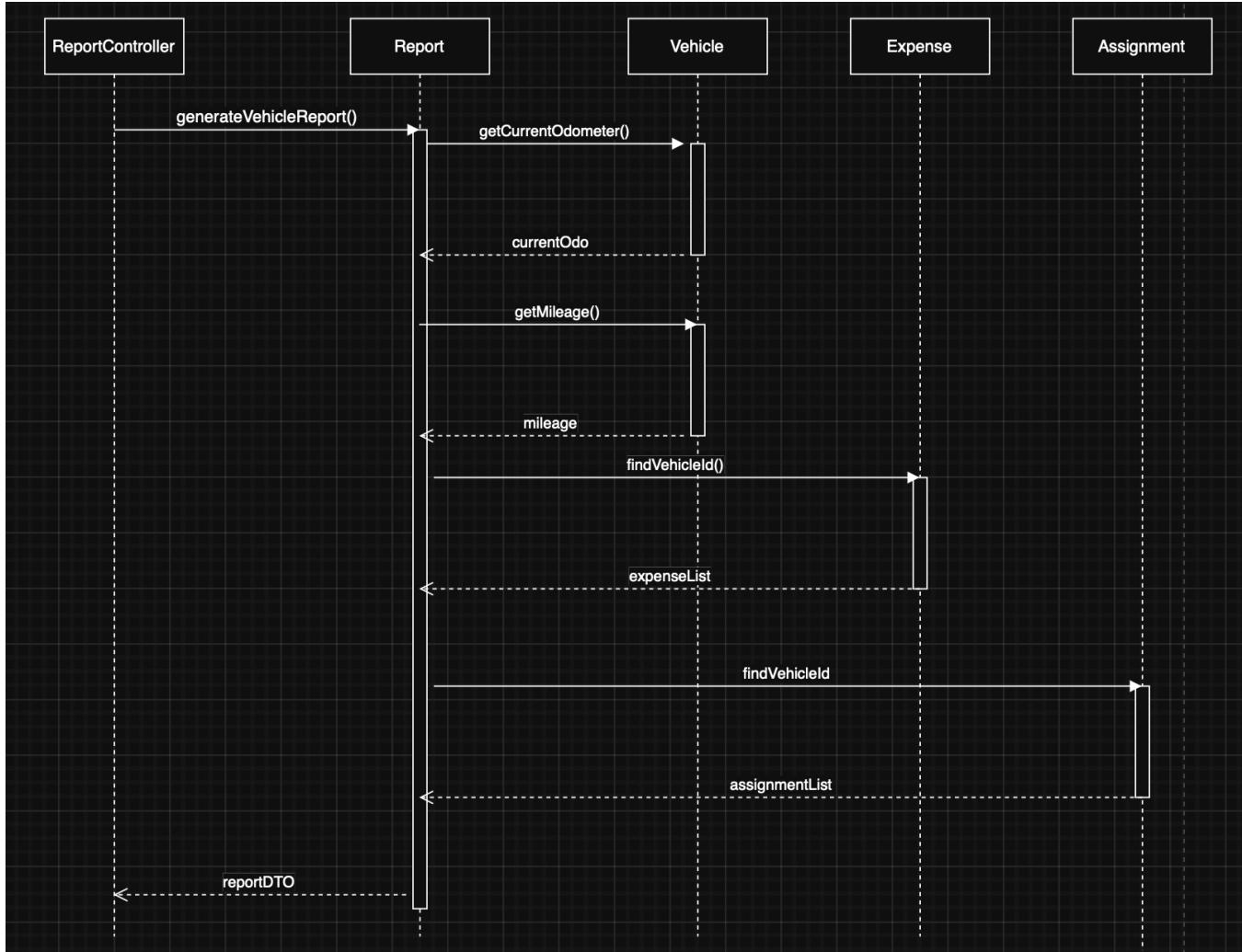


Şekil 7.1.1. - Detaylandırılmış UML Sınıf Diyagramı

7.2. Sıralama (Sequence) Diyagramları

Bu kısımda ardışıl diyagramlar bulunmaktadır.

a. Araç Raporu Oluşturma

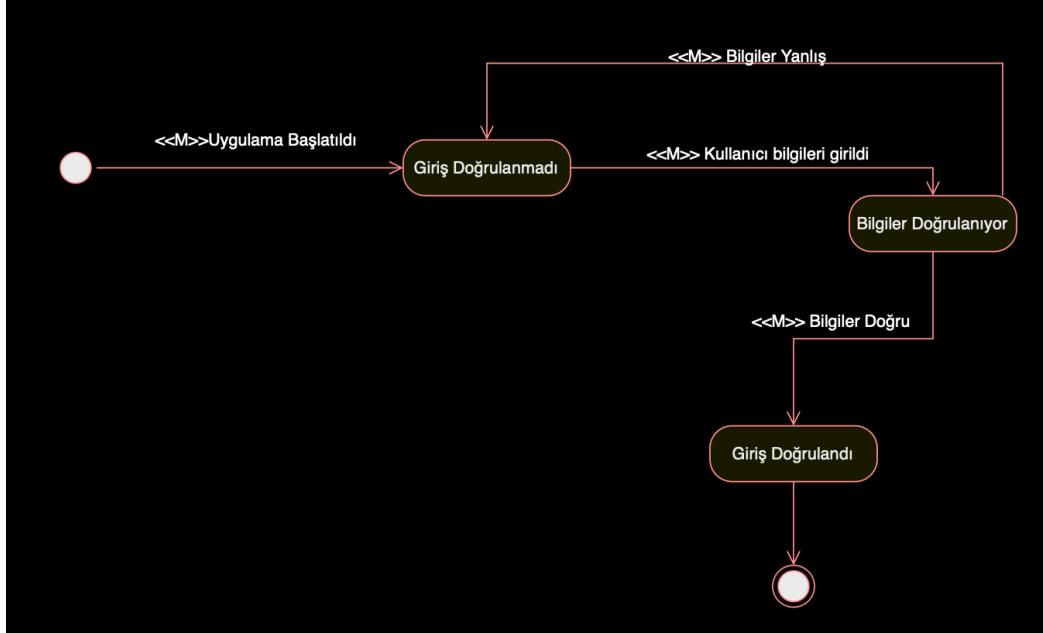


Şekil 7.2.1. – Araç Raporu Oluşturma Ardışıl Diyagram

7.3. Durum (State) Diyagramları

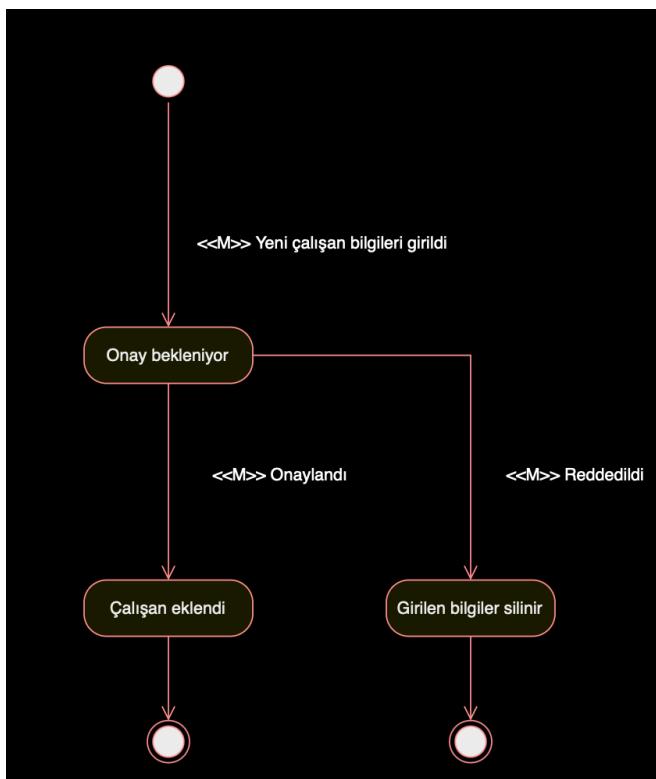
Bu kısımda durum diyagamları bulunmaktadır

a. Login



Şekil 7.3.1. – Login Durum Diyagramı

b. Çalışan Ekleme

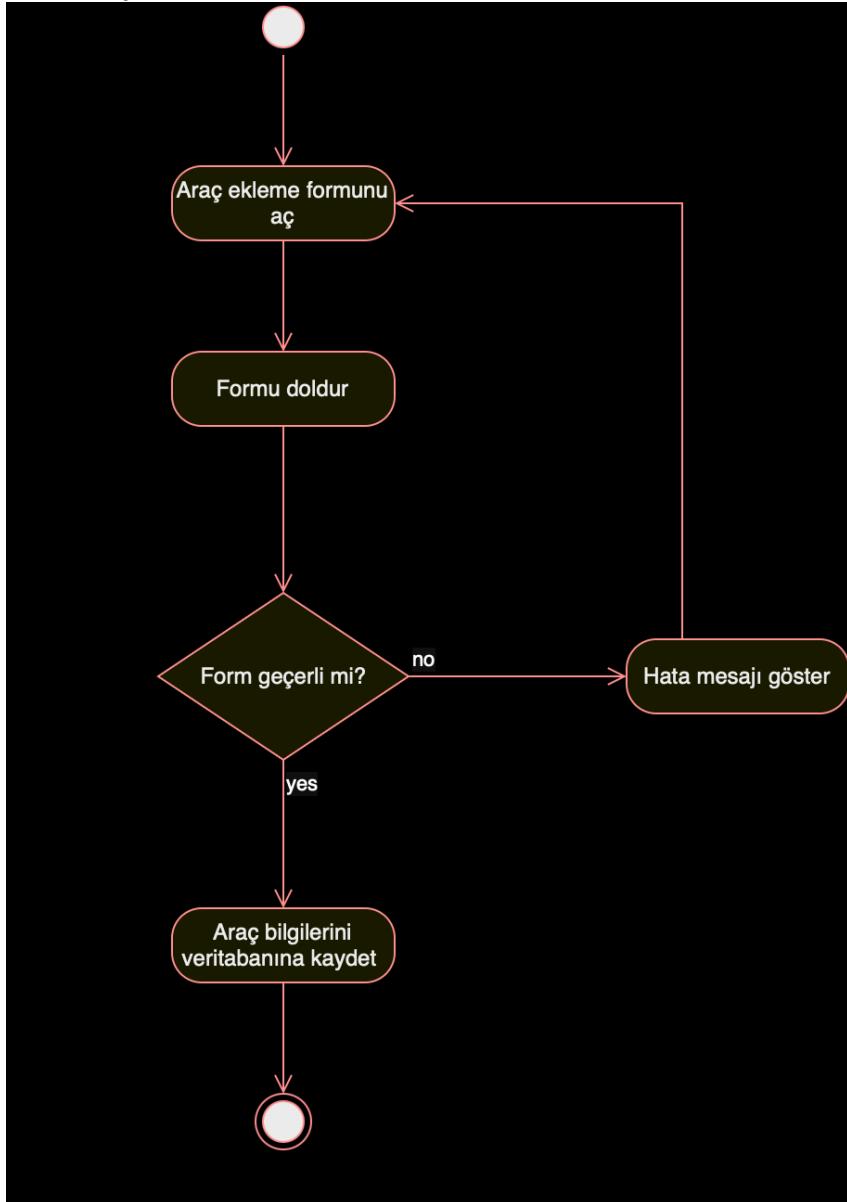


Şekil 7.3.2. - Proje Bitirme Durum Diyagramı

7.4. Etkinlik (Activity) Diyagramları

Bu kısımda etkinlik diyagramları bulunmaktadır.

a. Araç ekleme



Şekil 7.4.1. – Araç Ekleme Etkinlik Diyagramı

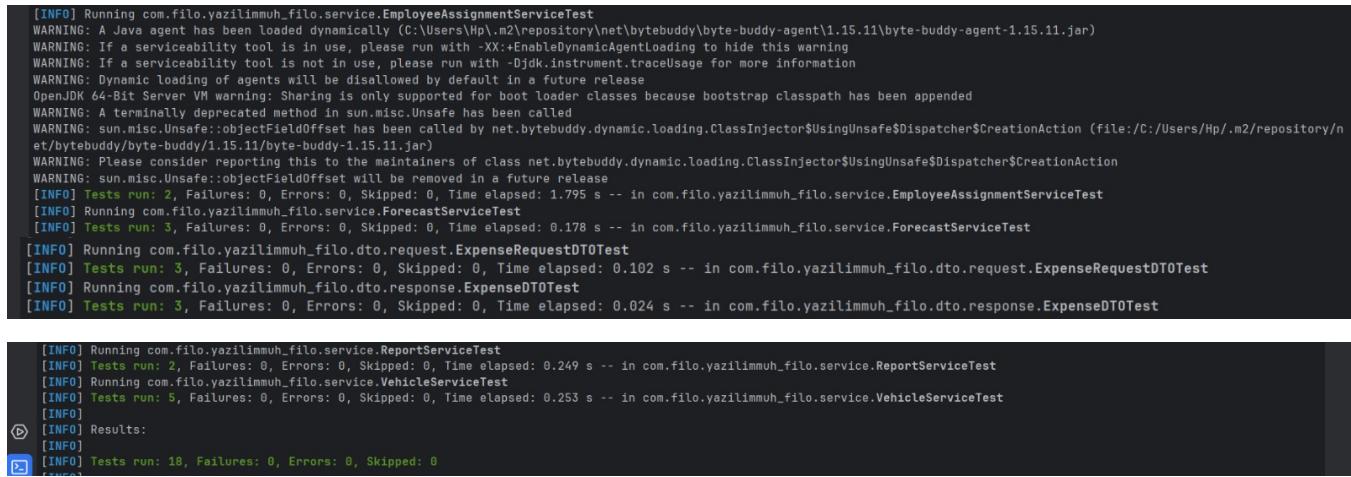
8. Birim Testi Sınamaları ve Projede Kullanılan Teknolojiler

Bu bölümde birim testlerinin kodları bulunmaktadır. Testlerinin başarılı olduğuna dair ekran çıktıları Şekil 8.1'de verilmiştir. Ayrıca projede kullanılan teknolojiler ve projenin doğru derlenebilmesi için gerekli adımlardan bahsedilmiştir.

Proje geliştirme sürecinde hem back-end hem de front-end tarafında modern ve güncel teknolojiler tercih edilmiştir. Arka uç (back-end) geliştirme için Java 21 sürümü ve Spring Boot 3 framework'ü kullanılmıştır. Veritabanı olarak yerel geliştirme ortamlarında hızlı ve hafif bir çözüm sunan **H2 file-based database** tercih edilmiş, .data/filo-db.mv.db dosyası üzerinden bağlantı sağlanmıştır.

Front-end geliştirme sürecinde ise React tabanlı bir framework olan **Next.js 15** kullanılmış ve proje TypeScript ile yazılmıştır. Kullanıcı arayüz bileşenlerinin geliştirilmesinde ise modern ve özelleştirilebilir bir bileşen kütüphanesi olan **Material UI (MUI)** tercih edilmiştir.

Projemizi Windowsta derlemek için Backend klasörünün içerisindeki '.\mvnw.cmd clean install -U' komutunu kullanabilirsiniz. Test için ise '.\mvnw.cmd test' komutu ile tüm testleri çalıştırabilirsiniz. Mac için ise 'mvn clean install -U' ile derleyebilir, 'mvn test' komutu ile testleri çalıştırabilirsiniz.



```
[INFO] Running com.filo.yazilimmuh_filo.service.EmployeeAssignmentServiceTest
WARNING: A Java agent has been loaded dynamically (C:\Users\Hp\.m2\repository\net\bytebuddy\byte-buddy-agent\1.15.11\byte-buddy-agent-1.15.11.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by net.bytebuddy.dynamic.loading.ClassInjector$UsingUnsafe$Dispatcher$CreationAction (file:/C:/Users/Hp/.m2/repository/net/bytebuddy/byte-buddy/1.15.11/byte-buddy-1.15.11.jar)
WARNING: Please consider reporting this to the maintainers of class net.bytebuddy.dynamic.loading.ClassInjector$UsingUnsafe$Dispatcher$CreationAction
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.795 s -- in com.filo.yazilimmuh_filo.service.EmployeeAssignmentServiceTest
[INFO] Running com.filo.yazilimmuh_filo.service.ForecastServiceTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.178 s -- in com.filo.yazilimmuh_filo.service.ForecastServiceTest
[INFO] Running com.filo.yazilimmuh_filo.dto.request.ExpenseRequestDTOTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.102 s -- in com.filo.yazilimmuh_filo.dto.request.ExpenseRequestDTOTest
[INFO] Running com.filo.yazilimmuh_filo.dto.response.ExpenseDTOTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.024 s -- in com.filo.yazilimmuh_filo.dto.response.ExpenseDTOTest

[INFO] Running com.filo.yazilimmuh_filo.service.ReportServiceTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.249 s -- in com.filo.yazilimmuh_filo.service.ReportServiceTest
[INFO] Running com.filo.yazilimmuh_filo.service.VehicleServiceTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.253 s -- in com.filo.yazilimmuh_filo.service.VehicleServiceTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 18, Failures: 0, Errors: 0, Skipped: 0
```

Şekil 8.1 - Test Sonuçlarının Ekranı

a. Halil Salih Tosun

- Test1:

```
class ExpenseDTOTest {  
  
    private Expense expense;  
    private Vehicle vehicle;  
    private Vendor vendor;  
    private User validator;  
    private LocalDate date;  
  
    @BeforeEach  
    void setUp() {  
        date = LocalDate.of(2025, 5, 11);  
  
        vehicle = new Vehicle();  
        vehicle.setId(11L);  
        vehicle.setPlateNumber("34ABC34");  
        vehicle.setBrand("Toyota");  
        vehicle.setModel("Corolla");  
  
        vendor = new Vendor();  
        vendor.setId(22L);  
        vendor.setUsername("vendorUser");  
        vendor.setRole(UserRole.VENDOR);  
        vendor.setName("Acme Supplies");  
        vendor.setCompanyName("Acme Inc.");  
  
        validator = new User();  
        validator.setId(33L);
```

```

validator.setUsername("adminUser");
validator.setRole(UserRole.ADMIN);

expense = new Expense();
expense.setId(123L);
expense.setDate(date);
expense.setType(ExpenseType.MAINTENANCE);
expense.setAmount(250.75);
expense.setDescription("Fren balatası değişimi");
expense.setVehicle(vehicle);
expense.setVendor(vendor);
expense.setValidatedBy(serializer);
}

@Test
void testConstructorMapsAllFieldsCorrectly() {
    ExpenseDTO dto = new ExpenseDTO(expense);

    assertEquals(123L, dto.getId(), "ID doğru set edilmeli");
    assertEquals(date, dto.getDate(), "Tarih doğru set edilmeli");
    assertEquals("MAINTENANCE", dto.getType(), "Enum tipi String olarak set edilmeli");
    assertEquals(250.75, dto.getAmount(), "Tutar doğru set edilmeli");
    assertEquals("Fren balatası değişimi", dto.getDescription(), "Açıklama doğru set edilmeli");

    assertNotNull(dto.getVehicle(), "Vehicle DTO null olmamalı");
    VehicleRequestDTO vDto = dto.getVehicle();
    assertEquals(11L, vDto.getId(), "Vehicle ID doğru map edilmeli");
    assertEquals("34ABC34", vDto.getPlateNumber(), "PlateNumber doğru map edilmeli");
}

```

```

assertNotNull(dto.getVendor(), "Vendor DTO null olmamalı");

VendorRequestDTO vendDto = dto.getVendor();

assertEquals(22L, vendDto.getId(), "Vendor ID doğru map edilmeli");

assertEquals("vendorUser", vendDto.getUsername(), "Vendor username doğru map edilmeli");

assertEquals("Acme Inc.", vendDto.getCompanyName(), "CompanyName doğru map edilmeli");



assertNotNull(dto.getValidatedBy(), "ValidatedBy DTO null olmamalı");

UserResponse uResp = dto.getValidatedBy();

assertEquals(33L, uResp.getId(), "Validator ID doğru map edilmeli");

assertEquals("adminUser", uResp.getUsername(), "Validator username doğru map edilmeli");

assertEquals(UserRole.ADMIN, uResp.getRole(), "Validator role doğru map edilmeli");

}

@Test
void testSettersAndGettersWork() {
    ExpenseDTO dto = new ExpenseDTO();

    dto.setId(7L);
    dto.setDate(date.minusDays(5));
    dto.setType("FUEL");
    dto.setAmount(99.9);
    dto.setDescription("Yakıt alımı");
}

VehicleRequestDTO vDto = new VehicleRequestDTO(vehicle);

VendorRequestDTO vendDto = new VendorRequestDTO(vendor);

```

```

UserResponse uResp = new UserResponsevalidator);

dto.setVehicle(vDto);
dto.setVendor(vendDto);
dto.setValidatedBy(uResp);

assertEquals(7L, dto.getId());
assertEquals(date.minusDays(5), dto.getDate());
assertEquals("FUEL", dto.getType());
assertEquals(99.9, dto.getAmount());
assertEquals("Yakıt alımı", dto.getDescription());
assertSame(vDto, dto.getVehicle(), "Vehicle DTO referansı korunmalı");
assertSame(vendDto, dto.getVendor(), "Vendor DTO referansı korunmalı");
assertSame(uResp, dto.getValidatedBy(), "UserResponse referansı korunmalı");
}

```

@Test

```

void testNullsHandledGracefullyInConstructor() {
    Expense empty = new Expense();
    ExpenseDTO dto = new ExpenseDTO(empty);

    assertNull(dto.getId(), "ID null kalmalı");
    assertNull(dto.getDate(), "Date null kalmalı");
    assertNull(dto.getType(), "Type null kalmalı");
    assertNull(dto.getAmount(), "Amount null kalmalı");
    assertNull(dto.getDescription(), "Description null kalmalı");
    assertNull(dto.getVehicle(), "Vehicle DTO null kalmalı");
    assertNull(dto.getVendor(), "Vendor DTO null kalmalı");
    assertNull(dto.getValidatedBy(), "ValidatedBy DTO null kalmalı");
}

```

```
 }  
 }
```

- Test2:

```
class ExpenseRequestDTOTest {  
  
    private Expense expense;  
    private Vehicle vehicle;  
    private Vendor vendor;  
    private User validator;  
    private LocalDate date;  
  
    @BeforeEach  
    void setUp() {  
  
        date = LocalDate.of(2025, 5, 11);  
  
        vehicle = new Vehicle();  
        vehicle.setId(42L);  
  
        vendor = new Vendor();
```

```

        vendor.setId(17L);

        validator = new User();
        validator.setId(99L);

        expense = new Expense();
        expense.setId(123L);
        expense.setDate(date);
        expense.setType(ExpenseType.MAINTENANCE);
        expense.setAmount(250.75);
        expense.setDescription("Fren balatası değişimi");

        expense.setVehicle(vehicle);
        expense.setVendor(vendor);
        expense.setValidatedByvalidator);
    }

    @Test
    void testConstructorMapsAllFieldsCorrectly() {
        ExpenseRequestDTO dto = new ExpenseRequestDTO(expense);

        assertEquals(123L, dto.getId(), "ID doğru kopyalanmalı");
        assertEquals(date, dto.getDate(), "Tarih doğru kopyalanmalı");
        assertEquals("MAINTENANCE", dto.getType(), "Tür enum'dan String'e dönüştürülmeli");
        assertEquals(250.75, dto.getAmount(), "Tutar doğru kopyalanmalı");
        assertEquals("Fren balatası değişimi", dto.getDescription(), "Açıklama doğru kopyalanmalı");
        assertEquals(42L, dto.getVehicleId(), "Vehicle ID doğru kopyalanmalı");
        assertEquals(17L, dto.getVendorId(), "Vendor ID doğru kopyalanmalı");
        assertEquals(99L, dto.getValidatedById(), "ValidatedBy ID doğru kopyalanmalı");
    }
}

```

```
@Test  
void testSettersAndGettersWork() {  
    ExpenseRequestDTO dto = new ExpenseRequestDTO();  
  
    dto.setId(7L);  
    dto.setDate(date.minusDays(5));  
    dto.setType("FUEL");  
    dto.setAmount(99.9);  
    dto.setDescription("Yakit alımı");  
    dto.setVehicleId(5L);  
    dto.setVendorId(6L);  
    dto.setValidatedById(8L);  
  
    assertEquals(7L, dto.getId());  
    assertEquals(date.minusDays(5), dto.getDate());  
    assertEquals("FUEL", dto.getType());  
    assertEquals(99.9, dto.getAmount());  
    assertEquals("Yakit alımı", dto.getDescription());  
    assertEquals(5L, dto.getVehicleId());  
    assertEquals(6L, dto.getVendorId());  
    assertEquals(8L, dto.getValidatedById());  
}
```

```
@Test  
void testNullsHandledGracefullyInConstructor() {  
  
    Expense e2 = new Expense();  
    ExpenseRequestDTO dto = new ExpenseRequestDTO(e2);
```

```

        assertNull(dto.getId());
        assertNull(dto.getDate());
        assertNull(dto.getType());
        assertNull(dto.getAmount());
        assertNull(dto.getDescription());
        assertNull(dto.getVehicleId());
        assertNull(dto.getVendorId());
        assertNull(dto.getValidatedById());
    }
}

```

b. Melih Yelman

- Test1:

```

public class ForecastServiceTest {

    @Mock
    private ExpenseRepository expenseRepository;

    @InjectMocks
    private ForecastService forecastService;

    private List<Expense> testExpenses;
    private Vehicle testVehicle;
    private LocalDate now;

    @BeforeEach
    public void setup() {
        MockitoAnnotations.openMocks(this);

        now = LocalDate.now();
        testExpenses = new ArrayList<>();
        testVehicle = new Vehicle();
        testVehicle.setId(1L);
        testVehicle.setBrand("Toyota");
        testVehicle.setModel("Corolla");

        createExpense(ExpenseType.FUEL, 100.0, now.minusMonths(2));
        createExpense(ExpenseType.FUEL, 120.0, now.minusMonths(1));
    }
}

```

```

        createExpense(ExpenseType.FUEL, 110.0, now.minusMonths(0));

        createExpense(ExpenseType.MAINTENANCE, 200.0, now.minusMonths(2));
        createExpense(ExpenseType.MAINTENANCE, 0.0, now.minusMonths(1));
        createExpense(ExpenseType.MAINTENANCE, 400.0, now.minusMonths(0));

        createExpense(ExpenseType.INSURANCE, 600.0, now.minusMonths(0));

        when(expenseRepository.findAll()).thenReturn(testExpenses);
    }

private void createExpense(ExpenseType type, double amount, LocalDate date) {
    Expense expense = new Expense();
    expense.setType(type);
    expense.setAmount(amount);
    expense.setDate(date);
    expense.setVehicle(testVehicle);
    testExpenses.add(expense);
}

@Test
public void testGenerateExpenseForecast_ThreeMonthPeriod() {
    Map<ExpenseType, Double> forecast = forecastService.generateExpenseForecast(3);

    assertTrue(forecast.containsKey(ExpenseType.FUEL), "Forecast should contain FUEL expenses");
    assertTrue(forecast.containsKey(ExpenseType.MAINTENANCE), "Forecast should contain MAINTENANCE expenses");
    assertTrue(forecast.containsKey(ExpenseType.INSURANCE), "Forecast should contain INSURANCE expenses");

    verify(expenseRepository, times(1)).findAll();

    final double DELTA = 0;
    assertEquals(110, forecast.get(ExpenseType.FUEL), DELTA, "FUEL expense forecast should match expected value");
    assertEquals(200, forecast.get(ExpenseType.MAINTENANCE), DELTA, "MAINTENANCE expense forecast should match expected value");
    assertEquals(200, forecast.get(ExpenseType.INSURANCE), DELTA, "INSURANCE expense forecast should match expected value");
}

@Test
public void testGenerateExpenseForecast_OneMonthPeriod() {
    Map<ExpenseType, Double> forecast = forecastService.generateExpenseForecast(1);

    assertTrue(forecast.containsKey(ExpenseType.FUEL), "Forecast should contain FUEL expenses");
    assertTrue(forecast.containsKey(ExpenseType.MAINTENANCE), "Forecast should contain MAINTENANCE expenses");
}

```

```

        assertTrue(forecast.containsKey(ExpenseType.INSURANCE), "Forecast should contain
INSURANCE expenses");

        verify(expenseRepository, times(1)).findAll();

        final double DELTA = 0;
        assertEquals(110, forecast.get(ExpenseType.FUEL), DELTA, "FUEL expense forecast should
match expected value");
        assertEquals(400, forecast.get(ExpenseType.MAINTENANCE), DELTA, "MAINTENANCE
expense forecast should match expected value");
        assertEquals(600, forecast.get(ExpenseType.INSURANCE), DELTA, "INSURANCE expense
forecast should match expected value");
    }

    @Test
    public void testGenerateExpenseForecast_NoExpenses() {
        when(expenseRepository.findAll()).thenReturn(new ArrayList<>());

        Map<ExpenseType, Double> forecast = forecastService.generateExpenseForecast(3);

        assertTrue(forecast.isEmpty(), "Forecast should be empty when there are no expenses");
    }
}

```

- **Test2:**

```

public class ReportServiceTest {

    @Mock
    private VehicleService vehicleService;

    @Mock
    private ExpenseService expenseService;

    @Mock
    private AssignmentService assignmentService;

    @Mock
    private ReportRepository reportRepository;

    @InjectMocks
    private ReportService reportService;

    private Vehicle testVehicle;
    private User testUser;
    private LocalDate fromDate;
    private LocalDate toDate;
    private List<Expense> expenses;
    private List<Assignment> assignments;
}

```

```

@BeforeEach
public void setup() {
    MockitoAnnotations.openMocks(this);

    fromDate = LocalDate.of(2025, 1, 1);
    toDate = LocalDate.of(2025, 3, 31);

    testVehicle = new Vehicle();
    testVehicle.setId(1L);
    testVehicle.setBrand("Toyota");
    testVehicle.setModel("Corolla");
    testVehicle.setPlateNumber("34ABC123");

    testUser = new User();
    testUser.setId(1L);
    testUser.setUsername("testuser");
    testUser.setPasswordHash("password");
    testUser.setRole(UserRole.ADMIN);

    expenses = new ArrayList<>();

    Expense fuel1 = new Expense();
    fuel1.setType(ExpenseType.FUEL);
    fuel1.setAmount(100.0);
    fuel1.setDate(LocalDate.of(2025, 1, 15));
    fuel1.setVehicle(testVehicle);

    Expense fuel2 = new Expense();
    fuel2.setType(ExpenseType.FUEL);
    fuel2.setAmount(120.0);
    fuel2.setDate(LocalDate.of(2025, 2, 15));
    fuel2.setVehicle(testVehicle);

    Expense maintenance = new Expense();
    maintenance.setType(ExpenseType.MAINTENANCE);
    maintenance.setAmount(200.0);
    maintenance.setDate(LocalDate.of(2025, 3, 15));
    maintenance.setVehicle(testVehicle);

    Expense outsidePeriod = new Expense();
    outsidePeriod.setType(ExpenseType.FUEL);
    outsidePeriod.setAmount(150.0);
    outsidePeriod.setDate(LocalDate.of(2025, 4, 15));
    outsidePeriod.setVehicle(testVehicle);

    expenses.add(fuel1);
    expenses.add(fuel2);
    expenses.add(maintenance);
    expenses.add(outsidePeriod);
}

```

```

assignments = new ArrayList<>();

EmployeeAssignment assignment1 = new EmployeeAssignment();
assignment1.setStartDate(LocalDate.of(2025, 1, 1));
assignment1.setEndDate(LocalDate.of(2025, 2, 28));
assignment1.setVehicle(testVehicle);

PoolAssignment assignment2 = new PoolAssignment();
assignment2.setStartDate(LocalDate.of(2025, 3, 1));
assignment2.setEndDate(LocalDate.of(2025, 3, 31));
assignment2.setVehicle(testVehicle);
assignment2.setMission("Test Mission");

EmployeeAssignment assignment3 = new EmployeeAssignment();
assignment3.setStartDate(LocalDate.of(2025, 4, 1));
assignment3.setEndDate(LocalDate.of(2025, 4, 30));
assignment3.setVehicle(testVehicle);

assignments.add(assignment1);
assignments.add(assignment2);
assignments.add(assignment3);

when(vehicleService.findById(1L)).thenReturn(Optional.of(testVehicle));
when(vehicleService.getCurrentOdometer(1L)).thenReturn(10000);
when(vehicleService.getMileage(eq(1L), any(LocalDate.class),
any(LocalDate.class))).thenReturn(500);
when(expenseService.findByVehicleId(1L)).thenReturn(expenses);
when(assignmentService.findByVehicleId(1L)).thenReturn(assignments);
when(reportRepository.save(any(Report.class))).thenAnswer(invocation ->
invocation.getArgument(0));
}

@Test
public void testGenerateVehicleReport_ValidData() {
    ReportDTO reportDTO = reportService.generateVehicleReport(1L, fromDate, toDate,
testUser);

    assertNotNull(reportDTO, "Report should not be null");
    assertEquals(10000, reportDTO.currentOdometer(), "Current odometer should match");
    assertEquals(500, reportDTO.mileageInPeriod(), "Mileage should match");
    assertEquals(2, reportDTO.assignmentCount(), "Number of assignments in period should
match");

    assertTrue(reportDTO.expensesByType().containsKey("FUEL"), "Report should include FUEL
expenses");
    assertTrue(reportDTO.expensesByType().containsKey("MAINTENANCE"), "Report should
include MAINTENANCE expenses");
    assertEquals(220.0, reportDTO.expensesByType().get("FUEL"), 0.01, "FUEL expenses should
sum correctly");
}

```

```

        assertEquals(200.0, reportDTO.expensesByType().get("MAINTENANCE"), 0.01,
    "MAINTENANCE expenses should sum correctly");
        assertEquals(420.0, reportDTO.totalExpense(), 0.01, "Total expenses should sum correctly");

        verify(reportRepository, times(1)).save(any(Report.class));
    }

    @Test
    public void testGenerateAllReports() {
        when(vehicleService.findAll()).thenReturn(Arrays.asList(testVehicle));

        List<ReportDTO> reports = reportService.generateAllReports(fromDate, toDate, testUser);

        assertNotNull(reports, "Reports list should not be null");
        assertEquals(1, reports.size(), "Should generate one report per vehicle");

        verify(vehicleService, times(1)).findAll();
    }
}

```

c. Ahmet Çam

- Test1:

```

public class VehicleServiceTest {

    @Mock
    private VehicleRepository vehicleRepository;

    @Mock
    private OdometerReadingService readingService;

    @Mock
    private EmployeeRepository employeeRepo;

    @InjectMocks
    private VehicleService vehicleService;

    private Vehicle testVehicle;

    @BeforeEach
    public void setup() {
        MockitoAnnotations.openMocks(this);
        testVehicle = new Vehicle();
        testVehicle.setId(1L);
        testVehicle.setBrand("Toyota");
        testVehicle.setModel("Corolla");
        testVehicle.setPlateNumber("34ABC123");
    }
}

```

```

@Test
public void testFindById() {
    when(vehicleRepository.findById(1L)).thenReturn(Optional.of(testVehicle));
    Optional<Vehicle> result = vehicleService.findById(1L);
    assertTrue(result.isPresent());
    assertEquals("Toyota", result.get().getBrand());
}

@Test
public void testUpdate() {
    Vehicle updated = new Vehicle();
    updated.setBrand("Honda");
    updated.setModel("Civic");
    updated.setPlateNumber("06XYZ789");
    updated.setOwnership(OwnershipType.LEASED);
    updated.setLeaseStartDate(LocalDate.now().minusYears(1));
    updated.setLeaseEndDate(LocalDate.now().plusYears(1));

    when(vehicleRepository.findById(1L)).thenReturn(Optional.of(testVehicle));
    when(vehicleRepository.save(any(Vehicle.class))).thenAnswer(invocation ->
        invocation.getArgument(0));

    Vehicle result = vehicleService.update(1L, updated);
    assertEquals("Honda", result.getBrand());
    assertEquals("Civic", result.getModel());
    assertEquals("06XYZ789", result.getPlateNumber());
}

@Test
public void testGetCurrentOdometer() {
    OdometerReading reading1 = new OdometerReading();
    reading1.setKm(5000);
    reading1.setDate(LocalDate.now().minusDays(10));

    OdometerReading reading2 = new OdometerReading();
    reading2.setKm(5500);
    reading2.setDate(LocalDate.now().minusDays(1));

    when(readingService.findByVehicleId(1L)).thenReturn(Arrays.asList(reading1, reading2));
    int currentKm = vehicleService.getCurrentOdometer(1L);
    assertEquals(5500, currentKm);
}

@Test
public void testAssignToPool() {
    testVehicle.getAssignments().clear();
    when(vehicleRepository.findById(1L)).thenReturn(Optional.of(testVehicle));
    when(vehicleRepository.save(any(Vehicle.class))).thenAnswer(invocation ->
        invocation.getArgument(0));
}

```

```

String mission = "Delivery Mission";
LocalDate start = LocalDate.now();
LocalDate end = LocalDate.now().plusDays(5);
PoolAssignment poolAssignment = vehicleService.assignToPool(1L, mission, start, end);

assertNotNull(poolAssignment);
assertEquals(mission, poolAssignment.getMission());
assertEquals(start, poolAssignment.getStartDate());
assertEquals(end, poolAssignment.getEndDate());
assertEquals(testVehicle, poolAssignment.getVehicle());
}

@Test
public void testAssignToEmployee() {
    Employee testEmployee = new Employee();
    testEmployee.setId(1L);
    testEmployee.setName("John Doe");
    when(vehicleRepository.findById(1L)).thenReturn(Optional.of(testVehicle));
    when(employeeRepo.findById(1L)).thenReturn(Optional.of(testEmployee));
    when(vehicleRepository.save(any(Vehicle.class))).thenAnswer(invocation ->
invocation.getArgument(0));

    LocalDate start = LocalDate.now();
    LocalDate end = LocalDate.now().plusDays(3);
    EmployeeAssignment assignment = vehicleService.assignToEmployee(1L, 1L, start, end);

    assertNotNull(assignment);
    assertEquals(testVehicle, assignment.getVehicle());
    assertEquals(testEmployee, assignment.getEmployee());
    assertEquals(start, assignment.getStartDate());
    assertEquals(end, assignment.getEndDate());
}
}

```

- **Test2:**

```

public class EmployeeAssignmentServiceTest {

    @Mock
    private EmployeeAssignmentRepository repo;

    @Mock
    private TripLogService tripLogService;

    @InjectMocks
    private EmployeeAssignmentService employeeAssignmentService;

    private EmployeeAssignment testAssignment;

    @BeforeEach

```

```

public void setup() {
    MockitoAnnotations.openMocks(this);
    testAssignment = new EmployeeAssignment();
    testAssignment.setId(1L);
}

@Test
public void testReject() {
    TripLog tripLog = new TripLog();
    tripLog.setId(10L);
    when(repo.findById(1L)).thenReturn(Optional.of(testAssignment));
    when(tripLogService.findByAssignment(1L)).thenReturn(tripLog);
    when(repo.save(any(EmployeeAssignment.class))).thenAnswer(invocation ->
invocation.getArgument(0));

    EmployeeAssignment result = employeeAssignmentService.reject(1L);
    verify(tripLogService, times(1)).deleteById(10L);
    assertNotNull(result);
}

@Test
public void testApprove() {
    when(repo.findById(1L)).thenReturn(Optional.of(testAssignment));
    when(repo.save(any(EmployeeAssignment.class))).thenAnswer(invocation ->
invocation.getArgument(0));

    int km = 100;
    EmployeeAssignment result = employeeAssignmentService.approve(1L, (long) km);
    verify(tripLogService, times(1)).create(any(TripLog.class));
    assertNotNull(result);
}
}

```