# 1. ANALİZ RAPORU (ANALYSIS REPORT)

## TO-Do-List SİSTEMİ

*Yazılım İnşaası Dersi- Iteration 1/2 Analiz Dokümanı*

# 1. GİRİŞ

Bu proje kapsamında geliştirilen **"To-Do List Sistemi"**, kullanıcıların günlük hayatlarındaki görev ve sorumluluklarını dijital ortamda kayıt altına almalarına, planlamalarına ve takip etmelerine olanak sağlayan bir yazılım çözümüdür. Günümüzdeki yoğun iş ve kişisel yaşam temposunda, zaman yönetimi ve görevlerin unutulmaması verimlilik açısından büyük önem taşımaktadır.

Sistem, Java programlama dili kullanılarak tasarlanmış olup, kullanıcıların temel ihtiyaçlarına yönelik sade ve hızlı bir arayüz sunmaktadır.

Bu rapor:

- geliştirilen sistemin gereksinimlerini
- işlevsel özelliklerini
- kullanıcıya sağladığı avantajları detaylı bir şekilde sunmak amacıyla hazırlanmıştır.

# 2. SİSTEM GEREKSİNİMLERİ

Sistemin başarılı bir şekilde çalışabilmesi ve kullanıcı ihtiyaçlarını karşılayabilmesi için belirlenen gereksinimler aşağıda iki ana başlık altında sunulmuştur:

### *3.1. Fonksiyonel Gereksinimler :*

Bu bölüm, sistemin kullanıcıya sunduğu temel yetenekleri ve gerçekleştirmesi gereken görevleri kapsamaktadır:

- **G1. Görev Oluşturma:** Kullanıcı, başlık ve açıklama içeren yeni bir görev ekleyebilmelidir.
- **G2. Görev Listeleme:** Sistem, kayıtlı tüm görevleri kullanıcıya düzenli bir liste halinde sunmalıdır.
- **G3. Görev Silme:** Kullanıcı, tamamlanan veya ihtiyaç duyulmayan görevleri listeden kalıcı olarak silebilmelidir.
- **G4. Durum Güncelleme:** Kullanıcı, bir görevi "tamamlandı" veya "yapılacak" olarak işaretleyerek durumunu değiştirebilmelidir.
- **G5. Veri Saklama:** Sistem, kullanıcı tarafından eklenen görevleri uygulama açık kaldığı sürece hafızada tutmalıdır.

### *3.2. Fonksiyonel Olmayan Gereksinimler :*

Sistemin performansı ve kalitesi ile ilgili kriterler şunlardır:

- **Hız:** Görev ekleme ve silme işlemleri 1 saniyenin altında gerçekleşmelidir.
- **Kullanılabilirlik:** Uygulama arayüzü, her seviyeden kullanıcının rehberliğe ihtiyaç duymadan kullanabileceği kadar sade olmalıdır.
- **Güvenilirlik:** Sistem, hatalı veri girişlerinde çökmeye karşı dayanıklı olmalı ve kullanıcıyı uyarmalıdır.

### *3.3. sistem kısıtları :*

Sistemin tasarımı ve geliştirilmesi aşamasında dikkate alınan temel kısıtlamalar aşağıda belirtilmiştir:

- **K1. Tek Kullanıcı Desteği:** Sistem şu anki aşamada yalnızca yerel bir kullanıcı tarafından kullanılabilir; çoklu kullanıcı girişi veya profil oluşturma desteği bulunmamaktadır.
- **K2. Veri Depolama Kısıtı:** Uygulama verileri çalışma zamanında (RAM) tutulmaktadır; uygulama kapatıldığında girilen görevler kalıcı bir veri tabanına kaydedilmemektedir.
- **K3. İnternet Gereksinimi:** Sistem çevrimdışı (offline) çalışacak şekilde tasarlanmıştır ve herhangi bir bulut senkronizasyonu içermemektedir.
- **K4. Platform Bağımlılığı:** Uygulamanın çalışabilmesi için cihazda Java Runtime Environment (JRE) yüklü olması zorunludur.
- **K5. Dosya Boyutu:** Görev açıklamaları için karakter sınırı bulunmakta olup, çok büyük metin dosyalarının sisteme aktarılması kısıtlanmıştır.

## 3. AKTÖRLER

Sistemin işleyişinde rol alan ve sistemle etkileşime giren aktörler aşağıda tanımlanmıştır:

- **5.1. Son Kullanıcı (Kullanıcı):** Sistemin ana aktörüdür. Görevleri oluşturma‹ listeleme‹ silme ve güncelleme yetkisine sahip olan kişidir. Sistemin sunduğu tüm fonksiyonel özelliklerden doğrudan yararlanır.

- **5.2. Sistem (Yazılım Katmanı):** Kullanıcının komutlarını işleyen, verileri geçici olarak saklayan ve arayüz üzerinden geri bildirim sağlayan aktördür. Java tabanlı çalışma mantığını yürüten arka plan birimidir.

# 4. SİSTEM MODELLERİ

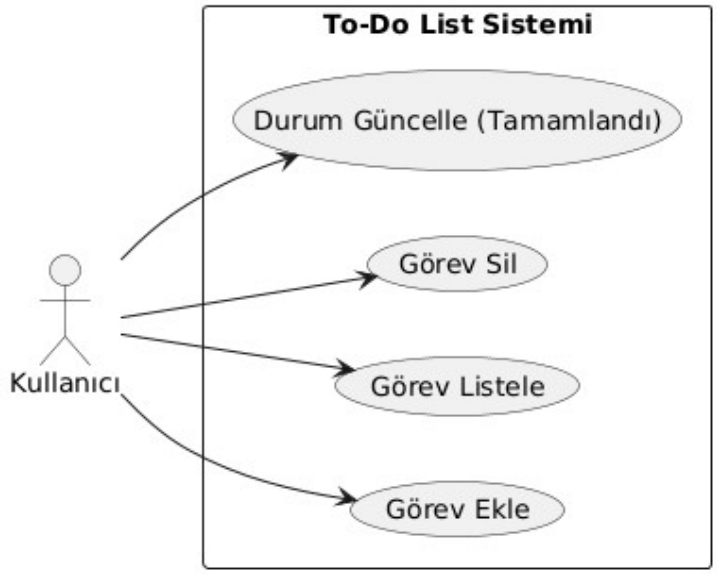
Sistemin yapısını, davranışını ve bileşenleri arasındaki ilişkileri tanımlamak amacıyla aşağıdaki modeller oluşturulmuştur:

### *4.1. Kullanım Durumu Modeli (Use Case Model)*

Bu model, sistemin sunduğu işlevlerin aktörler tarafından nasıl tetiklendiğini gösterir.

- **Kullanım Durumları:** Görev Ekle, Görev Sil, Görevleri Listele ve Durum Güncelleme süreçlerini kapsar.
- **İlişkiler:** Aktör (Kullanıcı) ile bu işlevler arasındaki doğrudan etkileşimi tanımlar.

#### *4.1.2. Kullanım Durumu Diyagramı (Use Case Diagram)*




Bu diyagram, sistemin dış dünya (aktörler) ile olan etkileşimini ve kullanıcıya sunduğu temel hizmetleri görselleştirir.

- **Aktör (Kullanıcı):** Sistemin tek kullanıcısıdır ve tüm işlemleri başlatma yetkisine sahiptir.
- **Sistem Sınırı (To-Do List Sistemi):** Uygulamanın kapsamını ve kullanıcıya açık olan fonksiyonları belirler.

***Kullanım Durumu Senaryoları :***

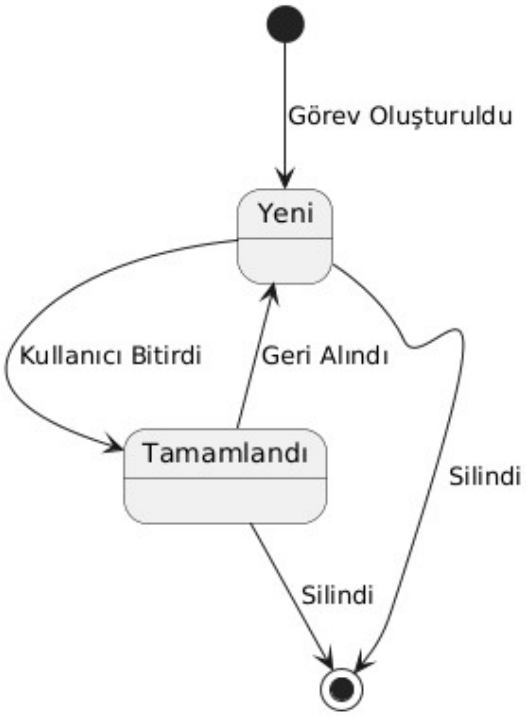
***Senaryo 1: Görev Ekleme***

- **Ön Koşul:** Uygulamanın açık olması.
- **Temel Akış:**
  - Kullanıcı "Yeni Görev" butonuna tıklar.
  - Sistem bir giriş alanı sunar.
  - Kullanıcı görev başlığını yazar ve kaydeder.
  - Sistem listeyi günceller ve yeni görevi görüntüler.
- **Son Koşul:** Görev başarıyla listeye eklenmiştir.

***Senaryo 2: Görev Silme***

- **Ön Koşul:** Listede en az bir görev bulunması.
- **Temel Akış:**
  - Kullanıcı listeden silmek istediği görevi seçer.
  - "Sil" komutunu tetikler.
  - Sistem nesneyi listeden kaldırır.
- **Son Koşul:** Görev artık listede görünmez.

## 4.2. DİNAMİK MODELLER



Dinamik model, sistemdeki nesnelerin belirli bir işlevi yerine getirmek için birbirleriyle nasıl etkileşime girdiğini ve zaman içindeki mesaj akışını tanımlar. Bu kapsamda "Görev Ekleme" süreci için etkileşim diyagramı oluşturulmuştur.

### *4.2.1 Etkileşim (Sequence) Diyagramı*

Aşağıdaki diyagram, kullanıcının arayüz üzerinden yeni bir görev ekleme talebi gönderdiğinde sistem bileşenleri arasındaki iletişimi göstermektedir:

**Etkileşim Adımları :**

**Kullanıcı:** Arayüzdeki (UI) "Ekle" butonuna basar ve görev verisini girer.

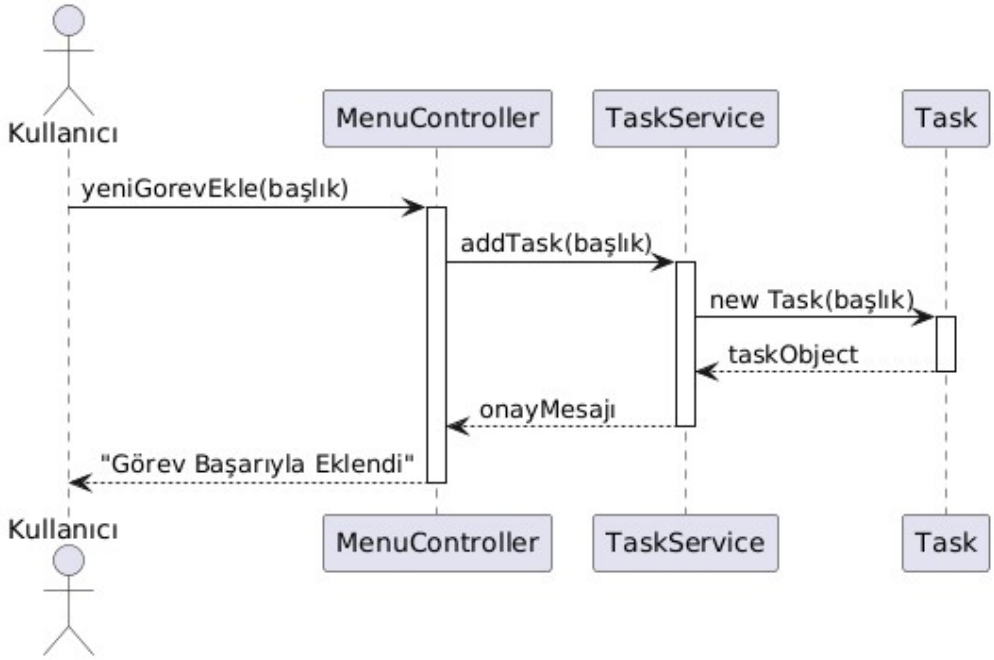
1. **MenuController:** Arayüzden gelen bu veriyi yakalar ve geçerliliğini kontrol eder.
2. **TaskList (Model):** Controller, `addTask()` metodunu çağırarak yeni görev nesnesini listeye ekler.
3. **Geri Bildirim:** İşlem tamamlandığında, sistem kullanıcıya listenin güncellendiğine dair onay mesajı döner.

## 4.2.2 Durum (State) Diyagramı:

Bir görevin (Task) yaşam döngüsü boyunca geçebileceği durumları tanımlar:

- **Yeni (New):** Görev ilk oluşturulduğunda bu durumdadır.
- **Tamamlandı (Completed):** Kullanıcı görevi bitirdiğinde durum güncellenir.
- **Silindi (Deleted):** Görev listeden kaldırıldığında sistemden tamamen çıkarılır.

## 4.3.1 Sequence Diyagramları :




### *4.3.1. Etkileşim (Sequence) Diyagramları*

Sistemdeki nesnelerin bir işlevselliği gerçekleştirmek için birbirleriyle hangi sırayla etkileşime girdiğini gösterir. Aşağıda en temel iki senaryo için taptığımız tasarımlar yer almaktadır:

***A. Görev Ekleme Senaryosu***

Bu diyagram, kullanıcının yeni bir görev girmesiyle başlayan süreci açıklar.

- **Adım 1:** Kullanıcı arayüz üzerinden `yeniGorevEkle()` komutunu tetikler.
- **Adım 2: MenuController**, kullanıcıdan alınan verileri doğrular ve **TaskList** sınıfına iletir.
- **Adım 3: TaskList**, yeni bir **Task** nesnesi oluşturur ve bunu listeye ekler.
- **Adım 4:** Sistem, işlemin başarılı olduğunu kullanıcıya geri bildirir.

***B. Görev Silme Senaryosu***

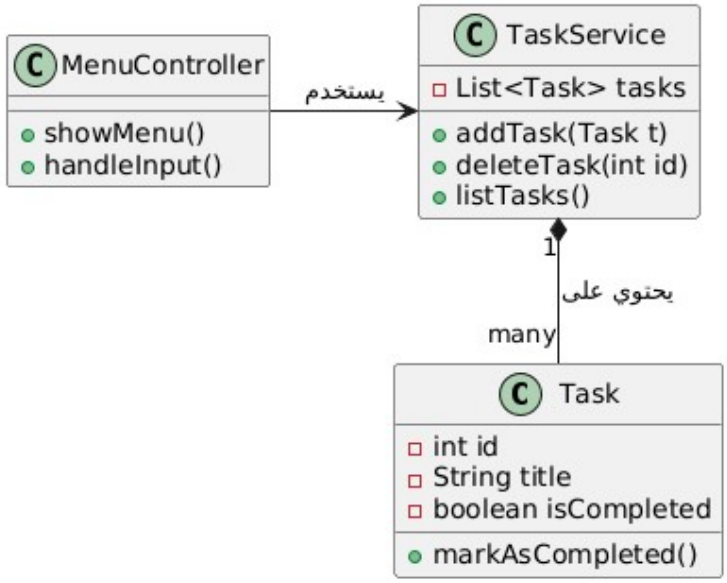
Kullanıcının mevcut bir görevi listeden kaldırma sürecini gösterir.

- **Adım 1:** Kullanıcı silmek istediği görevin ID bilgisini seçer.
- **Adım 2: MenuController**, silme talebini **TaskList**'e gönderir.
- **Adım 3: TaskList**, ilgili ID'ye sahip nesneyi listeden çıkarır.
- **Adım 4:** Arayüz güncellenerek silinen görevin artık listede olmadığı gösterilir.

### 4.3.2. Dinamik Modelin Değerlendirilmesi

Bu diyagramlar, Java tarafında yazılacak olan metodların (örneğin: `addTask()`, `deleteTask()`) hangi sınıflar altında bulunması gerektiğini ve çalışma mantığını netleştirir. Özellikle **MenuController** sınıfı, kullanıcı ile iş mantığı arasındaki ana koordinatör görevini üstlenmektedir.

### 4.3.3. Aktivite Diyagramı (Activity Diagram)




Aktivite diyagramı, sistemdeki iş akışını ve karar mekanizmalarını görselleştirir. Aşağıdaki diyagram, bir kullanıcının uygulamayı başlatmasından bir görevi yönetmesine kadar geçen süreci kapsamaktadır:

**İş Akışı Adımları (خطوات تدفق العمل):**

1. **Başlangıç:** Uygulama çalıştırılır ve ana menü görüntülenir.
2. **Seçim Yapma:** Kullanıcı yapmak istediği işlemi (Ekle, Sil, Listele veya Çıkış) seçer.
3. **Karar Mekanizması:**
    a. Eğer **"Ekle"** seçilirse; sistem veri girişi bekler, veriyi kaydeder ve menüye döner.
    b. Eğer **"Sil"** seçilirse; sistem silinecek görevi sorar, listeyi günceller ve menüye döner.
    c. Eğer **"Listele"** seçilirse; mevcut görevler ekrana basılır.
4. **Bitiş:** Kullanıcı "Çıkış" seçeneğini seçtiğinde süreç sonlanır.

## 4.4. SINIF MODELLERİ (Class Models)

### *4.4.1. Task Sınıfı (Görevi Temsil Eden Sınıf)*

Bu sınıf, her bir yapılacak işin özelliklerini tutar:

- **Öznitelikler (Attributes):** `id` (int), `title` (String), `description` (String), `isCompleted` (boolean).
- **Metotlar:** `getTitle()`, `setTitle()`, `markAsCompleted()`.

### *4.4.2. MenuController (Kullanıcı Arayüzü Denetleyicisi)*

Kullanıcıdan gelen girdileri alan ve sistemi yöneten sınıftır:

- **Görevi:** Konsol ekranındaki menü seçeneklerini yönetir ve kullanıcının seçimine göre ilgili iş kurallarını çağırır.

## 4.4. SINIF MODELLERİ

Bu bölümde, sistemin statik yapısını oluşturan sınıflar‹ bu sınıfların öznitelikleri ve metotları detaylandırılmıştır.

### *4.4.1. Task Sınıfı (Eşya/Item yerine)*

Bu sınıf, sistemdeki her bir "yapılacak işi" temsil eden temel veri modelidir.

- **Öznitelikler:** * `id`: Görevin benzersiz kimlik numarası.
    - `title`: Görev başlığı.
    - `description`: Görev açıklaması.
    - `isCompleted`: Görevin tamamlanma durumu (boolean).
- **Metotlar:** `getTitle()`, `setTitle()`, `markAsCompleted()`.

### *4.4.2. User Sınıfı (Customer yerine)*

Sistemi kullanan bireyi temsil eden sınıftır.

- **Öznitelikler:** `userName`, `userId`.
- **Metotlar:** `getUserInfo()`.

#### *4.4.3. TaskAssignment Sınıfı (Rental yerine)*

Bir görevin bir kullanıcıya atanmasını veya listenin yönetimini temsil eder.

- **Öznitelikler:** `assignmentId`, `creationDate`, `priorityLevel`.

#### *4.4.4. FileManager (Dosya Yönetimi)*

Verilerin kalıcı olarak saklanması ve dosyadan okunması işlemlerini yürüten yardımcı sınıftır.

- **Görevi:** Görev listesini bir metin dosyasına (.txt) kaydeder ve uygulama açıldığında tekrar yükler.

#### *4.4.5. TaskService (RentalService yerine)*

Sistemin "İş Kurallarını" (Business Logic) barındıran ana sınıftır.

- **Görevi:** Görev ekleme mantığı, silme doğrulama işlemleri ve listenin filtrelenmesi gibi fonksiyonel süreçleri yönetir.

#### *4.4.6. MenuController (Kullanıcı Arayüzü Denetleyicisi)*

Kullanıcı ile sistem arasındaki etkileşimi yöneten sınıftır.

- **Görevi:** Konsol ekranındaki menü seçeneklerini (1-Ekle, 2-Sil vb.) görüntüler, kullanıcı girişlerini alır ve bu girişleri ilgili Service sınıflarına yönlendirir.

## 4.5. NESNE MODELİ

Nesne modeli, sistemin belirli bir anındaki durumunu ve sınıflardan türetilen nesnelerin birbirleriyle olan somut ilişkilerini görselleştirir. Sınıf diyagramının aksine, burada statik yapılar değil, çalışma zamanındaki (runtime) gerçek veriler temsil edilir.

#### *4.5.1. Nesne Diyagramı (Object Diagram)*

Aşağıdaki diyagram, bir kullanıcının sistemde iki farklı görev oluşturduğu bir anı temsil etmektedir:

- **u1:User** `* userName = "sara123"`
- **t1:Task**
  - `id = 1`
  - `title = "Java Ödevi"`

    - `isCompleted = false`
- **t2:Task**
    - `id = 2`
    - `title = "Market Alışverişi"`
    - `isCompleted = true`
- **tl:TaskList**
    - `items = [t1, t2]`

#### *4.5.2. Nesneler Arası İlişkiler*

- **User - TaskList İlişkisi:** Kullanıcı nesnesi (u1), kendi oluşturduğu görevleri içeren listeye (tl) doğrudan erişim sağlar.
- **TaskList - Task İlişkisi:** TaskList nesnesi, içerisinde birden fazla Task nesnesini (t1, t2) barındırarak bunları yönetir.

## 5. KULLANICI ARAYÜZÜ TASARIMI

Bu bölümde, To-Do List uygulamasının kullanıcıyla etkileşime girdiği ekran tasarımları ve menü yapıları sunulmaktadır. Uygulama, kullanım kolaylığı sağlamak amacıyla sade ve anlaşılır bir yapıda tasarlanmıştır.

### *5.1. Ana Menü Tasarımı*

Uygulama başlatıldığında kullanıcıyı karşılayan ana ekran, tüm temel fonksiyonlara erişim imkanı sunar.

- **Menü Seçenekleri:** Kullanıcı klavye üzerinden 1-4 arasında bir rakam seçerek işlem yapar.
- **Görünüm:** Sade bir liste formatında görevler başlıklarıyla birlikte sergilenir.

### *5.2. Görev Ekleme Ekranı*

Kullanıcı "Görev Ekle" seçeneğini seçtiğinde sistem yeni bir giriş alanı açar.

- **Girdi Alanları:** Görev Başlığı ve Görev Açıklaması.
- **Onay:** Veriler girildikten sonra sistem "Görev başarıyla eklendi" mesajı verir.

### *5.3. Görev Listeleme ve Durum Güncelleme*

Tüm görevlerin durumlarıyla (Tamamlandı / Yapılacak) birlikte listelendiği ekrandır.

- **İşaretleme:** Kullanıcı, listedeki bir görevin yanındaki kutucuğu seçerek durumu güncelleyebilir.

## 6.4. AYRINTILI SINIF MODELLERİ VE YAZILIM MİMARİSİ

Sistemin nesne yönelimli yapısı (OOP), modülerlik ve sürdürülebilirlik prensiplerine uygun olarak tasarlanmıştır. Aşağıda her bir sınıfın sorumlulukları, öznitelikleri ve metotları detaylandırılmıştır:

### *6.4.1. Task Sınıfı (Temel Veri Modeli)*

Bu sınıf, sistemin kalbi olan "görev" nesnesini temsil eder. Sınıf içerisinde kapsülleme (encapsulation) prensibi uygulanmıştır.

- **Öznitelikler:**
  - `private int taskId`: Her göreve atanan benzersiz bir ID.
  - `private String title`: Görevin kısa başlığı.
  - `private String description`: Görevin detaylı açıklaması.
  - `private LocalDate deadline`: Görevin tamamlanması gereken son tarih.
  - `private boolean status`: Görevin tamamlanıp tamamlanmadığını belirten bayrak.
- **Metotlar:**
  - `getters/setters`: Verilere güvenli erişim sağlar.
  - `toString()`: Görev bilgilerini okunabilir bir formatta döndürür.

### *6.4.2. User Sınıfı (Sistem Aktörü)*

Sistemi kullanan bireyin profil bilgilerini ve ayarlarını yönetir.

- **Öznitelikler:**
  - `private String userId`: Kullanıcı kimliği.
  - `private String fullName`: Kullanıcı adı ve soyadı.
  - `private List<Task> userTasks`: Kullanıcıya ait tüm görevlerin listesi.
- **Metotlar:**
  - `calculateProductivity()`: Tamamlanan görev oranına göre verimlilik analizi yapar.

### *6.4.3. TaskAssignment Sınıfı (İş Atama ve Yönetim)*

Görevlerin oluşturulma zamanı ve öncelik sıralamasını yöneten ara sınıftır.

- **Öznitelikler:**
  - `private Priority priority`: Düşük, Orta, Yüksek şeklinde öncelik seviyeleri.
  - `private LocalDateTime createdAt`: Görevin oluşturulma zaman ***damgası.***

### *6.4.4. FileManager (Kalıcı Veri Yönetimi)*

Java I/O kütüphanelerini kullanarak verilerin kaybolmasını önler.

- **Fonksiyonlar:**
  - `saveToFile(List<Task> tasks)`: Mevcut listeyi JSON veya TXT formatında diske yazar.
  - `loadFromFile()`: Uygulama açıldığında eski verileri hafızaya yükler.

### *6.4.5. TaskService (İş Mantığı Katmanı)*

Tüm algoritma ve kontrol mekanizmalarının bulunduğu sınıftır.

- **Metotlar:**
  - `addTask(Task t)`: Yeni görev eklemeden önce başlığın boş olup olmadığını kontrol eder.
  - `findTaskById(int id)`: Liste içerisinde ID ile arama yapar.
  - `sortTasksByDate()`: Görevleri tarihe göre sıralar.

### *6.4.6. MenuController (Arayüz ve Akış Denetleyicisi)*

Kullanıcıdan gelen komutları anlar ve ilgili sınıfları tetikler.

- **Görevi:** `Scanner` sınıfı ile kullanıcı girdilerini okur ve `switch-case` yapısı ile menü yönlendirmelerini gerçekleştirir.

## 7. KULLANICI ARAYÜZÜ TASARIMI VE EKRAN ANALİZLERİ

Bu bölümde, kullanıcı deneyimini (UX) optimize etmek amacıyla hazırlanan ekran tasarımları detaylandırılmıştır.

### *7.1. Terminal/Konsol Giriş Ekranı*

Uygulama ilk açıldığında kullanıcıyı şık bir ASCII art ve ana menü karşılar.

**Tasarım Detayı:** Menü numaralandırılmış bir yapıdadır. Yanlış girişlerde (örneğin harf girilmesi) sistem hata mesajı fırlatır ve tekrar giriş bekler.

### *7.2. Görev Detay ve Filtreleme Ekranı*

Kullanıcı tüm görevleri listelediğinde sadece başlıkları değil, aynı zamanda durumlarını da renkli veya sembolik olarak görür.

- **Filtreler:** "Sadece Tamamlananları Göster" veya "Bugün Yapılacakları Göster" seçenekleri arayüzde sunulur.

## 8. SONUÇ VE DEĞERLENDİRME

Bu analiz raporunda, Java tabanlı bir To-Do List uygulamasının gereksinimleri, sistem modelleri ve tasarım süreçleri detaylandırılmıştır. Projenin temel amacı olan "sadelik ve verimlilik" prensibi, hem fonksiyonel gereksinimlerde hem de arayüz tasarımında korunmuştur. Geliştirilen bu sistem, kullanıcıların günlük planlamalarını dijitalleştirerek zaman yönetimi becerilerine katkı sağlamayı hedeflemektedir.