TO-DO LIST (GÖREV YÖNETİM SİSTEMİ) TASARIM VE FİNAL RAPORU

1. GİRİŞ

Bu doküman, Java programlama dili ile geliştirilen To-Do List Sistemi için hazırlanmış kapsamlı raporları içermektedir. Rapor, sistemin analiz aşamasında tanımlanan gereksinimlerinin; bakımı kolay, modüler ve nesne yönelimli bir mimari ile nasıl hayata geçirildiğini belgelemektedir.

2. TASARIM HEDEFLERİ

2.1 Temel Tasarım Hedefleri

• İş Kurallarını Kullanıcı Arayüzünden Ayırmak: Tüm görev doğrulama, tarih kontrolü ve öncelik atama gibi kritik iş mantığı TodoService sınıfında merkezileştirilmiştir. ConsoleUI (sunum katmanı) yalnızca kullanıcı girdilerini alır ve sonuçları gösterir. Bu ayrım, Separation of Concerns (SoC) prensibini sağlayarak gelecekte bir grafik arayüz (GUI) eklenmesini kolaylaştırır.

• Merkezi Veri Yönetimi: Tüm dosya okuma/yazma (TXT/JSON) ve benzersiz ID üretimi işlemleri DatabaseManager (veya FileManager) sınıfında toplanmıştır. Singleton tasarım deseni uygulanarak sistemde tek bir veri erişim noktası oluşturulmuş, böylece veri bütünlüğü garanti altına alınmıştır.

• Kod Tekrarını Önlemek (DRY): ID doğrulama, durum güncelleme ve veri formatlama gibi işlemler tek bir noktada tanımlanmıştır. Model sınıflarında (Todo) ortak yöntemler kullanılarak bakım maliyeti düşürülmüştür.

• Genişletilebilir Mimari: Katmanlı mimari sayesinde sisteme yeni özellikler (hatırlatıcılar, kategoriler) mevcut kodu bozmadan eklenebilir. Open/Closed Principle (OCP) gözetilerek tasarlanan yapıda, veritabanı desteği eklendiğinde sadece veri katmanının değiştirilmesi yeterli olacaktır.

2.2 Tasarım Kriterleri

• Kullanılabilirlik: Analiz kısıtlarına uygun olarak sade bir CLI (Komut Satırı Arayüzü) tercih edilmiştir. Kullanıcı hatalarında (geçersiz tarih, boş görev) net ve Türkçe uyarı mesajları verilerek hata toleransı artırılmıştır.

• Bakım Kolaylığı: Üç katmanlı yapı (Sunum - İş Mantığı - Veri) ile sorumluluklar net ayrılmıştır. Single Responsibility Principle (SRP) uyarınca her sınıfın tek bir sorumluluğu bulunmaktadır.

• Performans: Görevler program açılışında belleğe (ArrayList) yüklenir ve işlemler RAM üzerinden yapılarak disk erişimi minimize edilir. Bu sayede işlem hızı <1 saniye hedefinde tutulmuştur.

3. SİSTEM MİMARİSİ

Sistem, katmanlı mimari yaklaşımıyla üç ana alt sisteme ayrılmıştır:

1. Sunum Katmanı (ConsoleUI): Kullanıcı ile sistem arasındaki tüm etkileşimi yönetir. Görev ekleme, listeleme ve silme ekranlarını konsol üzerinde gösterir.

2. İş Mantığı Katmanı (TodoService): Görevlerin eklenme kurallarını, tamamlama durumlarını ve sıralama algoritmalarını yönetir. Tüm iş kuralları burada merkezi olarak tutulur.

3. Veri Erişim Katmanı (DatabaseManager & Model): Verilerin kalıcı olarak saklanmasını sağlar. todos.txt dosyası ile etkileşime girerek veri bütünlüğünü korur.

4. SINIF TASARIMI (LOW-LEVEL DESIGN)

Sistem, nesne yönelimli tasarım prensiplerine (OOP) uygun olarak 4 ana sınıftan oluşmaktadır. Bu sınıflar, katmanlı mimarinin gereksinimlerini karşılayacak ve sorumlulukların net bir şekilde ayrılmasını (Separation of Concerns) sağlayacak şekilde tasarlanmıştır.

4.1 Todo (Model) Sınıfı

Sorumluluklar: Sistemdeki her bir "Yapılacak" işini temsil eden temel veri nesnesidir. Görev bilgilerini güvenli ve tutarlı bir şekilde saklamaktan sorumludur.

• Alanlar (Attributes):

• id: Her görev için sistem tarafından atanan benzersiz kimlik numarası (int).

• title: Görevin kısa ve açıklayıcı adı (String).

• description: Görevin detaylarını içeren açıklama metni (String).

• priority: Görevin aciliyet durumunu (Düşük, Orta, Yüksek) belirten seviye (String).

• isCompleted: Görevin tamamlanma durumunu takip eden mantıksal bayrak (boolean).

• Temel Yöntemler (Methods):

• toggleStatus(): Görevin durumunu "Tamamlandı" veya "Devam Ediyor" olarak tersine çevirir.

• toFileString(): Nesne verilerini, metin dosyasında saklanmaya uygun (örneğin virgülle ayrılmış) satır formatına dönüştürür.

• toString(): Konsol ekranında kullanıcıya estetik ve okunabilir bir görünüm sunmak için verileri biçimlendirir.

4.2 DatabaseManager (Data) Sınıfı

Sorumluluklar: Veri katmanının merkezini oluşturur. Tüm dosya okuma ve yazma işlemlerini yürüterek verilerin kalıcı (persistence) olmasını sağlar.

• Özellikler:

• Singleton Tasarım Deseni: Sistemde aynı anda yalnızca bir DatabaseManager örneğinin bulunmasını sağlar. Bu sayede dosya erişim çakışmaları ve veri bozulmaları engellenir.

• Dosya Yönetimi: todos.txt dosyasının varlığını kontrol eder, yoksa otomatik olarak oluşturur.

• Temel Yöntemler:

• loadTodos(): Dosyadaki her bir satırı okur, parçalar ve bunları Todo nesnelerine dönüştürerek bir liste halinde döner.

• saveTodos(List<Todo>): Bellekteki güncel görev listesini alır ve dosyanın üzerine en güncel haliyle yazar.

• generateNextId(): Dosyadaki mevcut en büyük ID değerini tespit ederek yeni eklenecek görev için çakışmayan bir sonraki ID'yi üretir.

4.3 TodoService (Logic) Sınıfı

Sorumluluklar: Sistemin "İş Mantığı" (Business Logic) katmanıdır. Kullanıcı taleplerini işler, doğrular ve veri katmanı ile koordinasyonu sağlar.

• Façade Benzeri Davranış: Sunum katmanının (UI) karmaşık dosya işlemlerinden ve doğrulama süreçlerinden haberdar olmasını engelleyerek sade bir yöntem seti sunar.

• Temel Yöntemler:

• addTodo(): Yeni görev verilerini alır, boş alan kontrolü gibi doğrulamaları yapar ve DatabaseManager aracılığıyla kaydeder.

• completeTodo(int id): Belirli bir ID'ye sahip görevi bulur, durumunu günceller ve değişiklikleri kalıcı hale getirir.

• deleteTodo(int id): İlgili görevi listeden çıkarır ve veri bütünlüğünü sağlamak için dosyayı günceller.

• filterTasks(): Tamamlanmış veya devam eden görevleri birbirinden ayırarak listeleme operasyonlarını yönetir.

4.4 ConsoleUI (Presentation) Sınıfı

Sorumluluklar: Kullanıcı etkileşimini (CLI) yönetir. Kullanıcıdan veri toplamak ve sistem sonuçlarını görselleştirmekle yükümlüdür.

• Temel Yöntemler:

• start(): Uygulamanın ana döngüsünü başlatır ve kullanıcıyı seçim yapması için menüye yönlendirir.

• showMenu(): Kullanılabilir tüm işlemleri (Ekle, Sil, Listele vb.) ekranda listeler.

• handleInput(): Java Scanner sınıfını kullanarak kullanıcıdan girdi alır; sayısal ve metinsel girdilerin temel kontrolünü yapar.

• displayTasks(List<Todo>): Görev listesini hizalı bir tablo formatında (ID | Başlık | Durum vb.) ekrana basarak kullanılabilirliği artırır.

5. GERÇEKLEŞTİRİM VE SONUÇ

Bu bölüm, tasarım aşamasında teorik olarak planlanan mimarinin Java dilinde nasıl somutlaştırıldığını, geliştirme sürecindeki teknik tercihleri ve karşılaşılan mühendislik problemlerine üretilen çözümleri kapsamaktadır.

5.1 Uygulama Süreci ve Geliştirme Ortamı

Projenin gerçekleştirim aşaması, modern yazılım geliştirme standartları takip edilerek tamamlanmıştır.

• Teknoloji Yığını: Uygulama, Java 17 (LTS) sürümünün sağladığı modern dil özellikleri (Records, Enhanced Switch vb.) kullanılarak geliştirilmiştir. IDE olarak IntelliJ IDEA ve Eclipse'in sunduğu statik kod analizi araçlarından yararlanılmıştır.

• Sürüm Kontrolü: Kodun geliştirilme sürecinde Git kullanılmış ve proje GitHub üzerinde depolanmıştır. Bu sayede geliştirme aşamaları versiyonlanmış, olası hatalarda geri dönüş imkanı sağlanmış ve kod bütünlüğü korunmuştur.

• Fonksiyonel Kapsam: Analiz aşamasında belirlenen tüm kullanım senaryoları (Use Cases) eksiksiz şekilde kodlanmıştır:

• Görev Ekleme: Dinamik ID ataması ve veri doğrulama ile yeni görev kaydı.

• Listeleme: Bellek üzerindeki ArrayList'in farklı parametrelere (tamamlanma durumu, öncelik) göre filtrelenerek kullanıcıya sunulması.

• Silme: ID tabanlı referans kontrolü ile verinin güvenli şekilde kaldırılması.

• Durum Güncelleme: Görevin yaşam döngüsünü (Aktif -> Tamamlandı) yöneten mantıksal anahtarlama.

5.2 Karşılaşılan Teknik Zorluklar ve Mühendislik Çözümleri

Gerçekleştirim sürecinde dosya tabanlı sistemlerin ve kullanıcı etkileşiminin doğurduğu bazı zorluklar "Savunmacı Programlama" (Defensive Programming) teknikleriyle aşılmıştır.

5.2.1 Veri Tutarlılığı ve Atomik Yazma İşlemleri

• Sorun: Dosya tabanlı veri yönetiminde, programın beklenmedik şekilde kapanması (elektrik kesintisi, hata vb.) durumunda verilerin bozulma veya eksik kalma riski bulunmaktadır.

• Çözüm: Sistem, her veri değişikliğinde (Ekle/Sil/Güncelle) "Auto-Commit" benzeri bir mantıkla çalışır. DatabaseManager sınıfı, bellekteki listeyi dosyaya yazarken dosyayı tamamen kapatmadan önce verinin disk üzerine fiziksel olarak işlendiğinden emin olan flush() ve close() rutinlerini kullanır. Ayrıca, dosya okuma sırasında karşılaşılan bozuk satırlar sistem tarafından ayıklanarak çalışma zamanı hataları engellenmiştir.

5.2.2 Giriş Doğrulama ve İstisna Yönetimi (Exception Handling)

• Sorun: Konsol tabanlı uygulamalarda kullanıcının beklenen veri tipi dışında (örneğin sayı beklenen menü seçiminde metin girmesi) giriş yapması, Java sanal makinesinin InputMismatchException fırlatarak çökmesine neden olmaktadır.

• Çözüm: Tüm kullanıcı girdileri try-catch blokları ile sarmalanmıştır. Hatalı girişlerde program sonlanmak yerine, kullanıcıya hatanın nedenini açıklayan (örn: "Lütfen 1-5

arasında bir rakam giriniz!") uyarılar vererek giriş ekranına yönlendirme yapılmıştır. Bu yaklaşım, sistemin Robustness (Sağlamlık) kriterini artırmıştır.

5.3 Genel Değerlendirme ve Sonuç

To-Do List projesi, basit bir görev yönetim aracından öte, bir yazılımın analizden ürüne dönüşme sürecini tüm aşamalarıyla temsil etmektedir.

• Mimari Başarı: Katmanlı mimari (Layered Architecture) sayesinde, kullanıcı arayüzü ile veri saklama mantığı tamamen birbirinden soyutlanmıştır. Bu durum projenin test edilebilirliğini ve modülerliğini en üst seviyeye çıkarmıştır.

• Prensiplere Uyum: Projede SOLID prensipleri, özellikle de "Single Responsibility" (Her sınıfın tek görevi olması) ve "Dependency Inversion" (Üst katmanların alt katmanlara soyutlamalar üzerinden erişmesi) ilkeleri titizlikle uygulanmıştır.

• Kişisel ve Teknik Kazanımlar: Bu süreç, yazılım inşaasında analiz ve tasarım aşamalarının ne kadar kritik olduğunu göstermiştir. Tasarımı iyi yapılmış bir sistemin kodlanma aşamasında karmaşıklığın minimuma indiği ve hata payının azaldığı somut olarak gözlemlenmiştir.