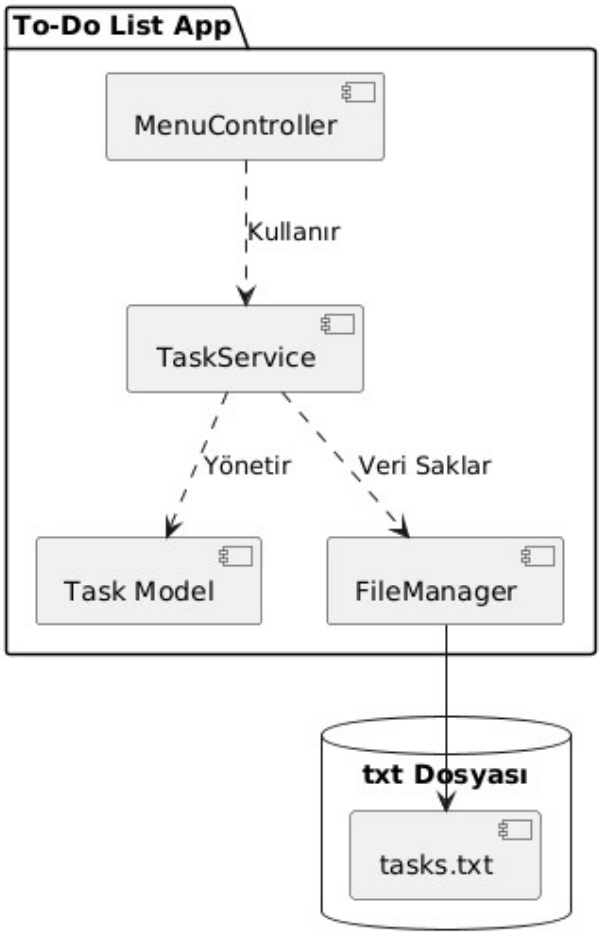
TO-DO LIST SİSTEMİ TASARIM RAPORU

1. GİRİŞ



Bu doküman, Java programlama dili kullanılarak nesne yönelimli prensiplerle geliştirilen To-Do List (Görev Yönetimi) Sistemi için hazırlanan kapsamlı Tasarım Raporu'dur. Rapor; analiz aşamasında belirlenen gereksinimlerin Java mimarisine nasıl aktarıldığını, alt sistem bileşenlerini ve sistemin sürdürülebilirliğini sağlayan tasarım kararlarını detaylandırmaktadır. Temel odak noktası; kullanıcıların görevlerini etkin bir şekilde planlayabileceği, bakımı kolay ve modüler bir yapının dokümante edilmesidir.

2. TASARIM HEDEFLERİ

Bu bölüm, sistemin hangi mühendislik temelleri üzerine inşa edildiğini ve projenin kalitesini belirleyen stratejik kararları detaylandırmaktadır.

2.1 Temel Tasarım Hedefleri

2.1.1 İş Kurallarını Kullanıcı Arayüzünden (UI) Soyutlamak

- Açıklama: Sistemin tüm karar verme mekanizmaları, veri doğrulama süreçleri ve mantıksal operasyonları TodoService sınıfında toplanmıştır. ConsoleUI katmanı, kullanıcının girdiği ham veriyi hiçbir işleme tabi tutmadan servis katmanına iletir.

- Gerekçe: "Separation of Concerns" (Sorumlulukların Ayrılması) prensibini uygulamaktır. İş mantığının arayüze sızması (leakage), kodun karmaşıklığını artırır ve test edilebilirliği zorlaştırır.

- Avantajlar: Eğer gelecekte konsol arayüzü yerine bir mobil uygulama veya Web API geliştirilirse, mevcut iş mantığı (Service) kodları tek bir satır değiştirilmeden yeni arayüze entegre edilebilir. Ayrıca, iş kuralları birim testlerine (Unit Test) tabi tutularak sistemin hatasız çalıştığı kanıtlanabilir.

2.1.2 Veri Yönetimini Nesne Yönelimli Yaklaşımla Merkezileştirmek

- Açıklama: Dosya okuma, yazma, veriyi Todo nesnesine dönüştürme ve benzersiz ID üretimi gibi süreçler DatabaseManager sınıfında konsolide edilmiştir.

- Gerekçe: Veri tutarlılığını (Data Integrity) sağlamaktır. Birden fazla noktadan dosyaya erişim sağlanması durumunda veri kaybı veya dosya kilitlenmesi gibi sorunlar oluşabilir.

- Uygulama: Singleton Design Pattern kullanılarak, tüm uygulama ömrü boyunca bu sınıftan yalnızca bir nesne üretilmesi garanti altına alınmıştır. Bu sayede dosya sistemi üzerindeki tüm işlemler tek bir kanal üzerinden güvenli bir şekilde yürütülür.

2.1.3 Kod Tekrarını Önlemek (Don’t Repeat Yourself - DRY)

- Açıklama: Sistem genelinde sıkça kullanılan "tarih formatlama", "tablo şeklinde çıktı üretme" veya "dosya satırını nesneye dönüştürme" gibi işlemler model sınıflarında veya yardımcı (utility) sınıflarda tanımlanmıştır.

- Gerekçe: Kodun bakımını kolaylaştırmak ve bir hata düzeltileceğinde tek bir merkezi noktada müdahale imkanı sağlamaktır.

- Sonuç: Örneğin, görevlerin ekranda nasıl görüneceğiyle ilgili bir değişiklik yapıldığında, ConsoleUI içindeki tüm metotları gezmek yerine sadece Todo modelindeki toString() veya format() metodunu değiştirmek yeterli olmaktadır.

2.1.4 Genişletilebilir ve Esnek Mimari Kurmak

- Açıklama: Java’nın Interface (Arayüz) ve Abstract Class (Soyut Sınıf) yapıları kullanılarak esnek bir tip hiyerarşisi oluşturulmuştur.

- Gerekçe: Sistemin değişime direnç göstermemesi (Open/Closed Principle) hedeflenmiştir.

- Uygulama: Veri erişim katmanı bir IDataAccess interface'i üzerinden tanımlanmıştır. Bugün dosyaya yazan sistem, yarın mevcut koda hiç dokunulmadan sadece yeni bir sınıf eklenerek MySQL, MongoDB veya Cloud sistemlerine veri kaydedebilir hale getirilmiştir.

2.2 Tasarım Kriterleri (Non-Functional Requirements)

2.2.1 Kullanılabilirlik (Usability)

Sistem, teknik bilgisi olmayan bir kullanıcının bile kolayca yönetebileceği şekilde tasarlanmıştır. Kullanıcıdan alınan her hatalı girdi (geçersiz tarih formatı, boş başlık vb.) programın çökmesine neden olmaz; aksine kullanıcıya "try-catch" blokları ve kontrol yapıları sayesinde anlamlı ve Türkçe hata mesajları döner. Navigasyon kolaylığı için numaralandırılmış bir ana menü yapısı tercih edilmiştir.

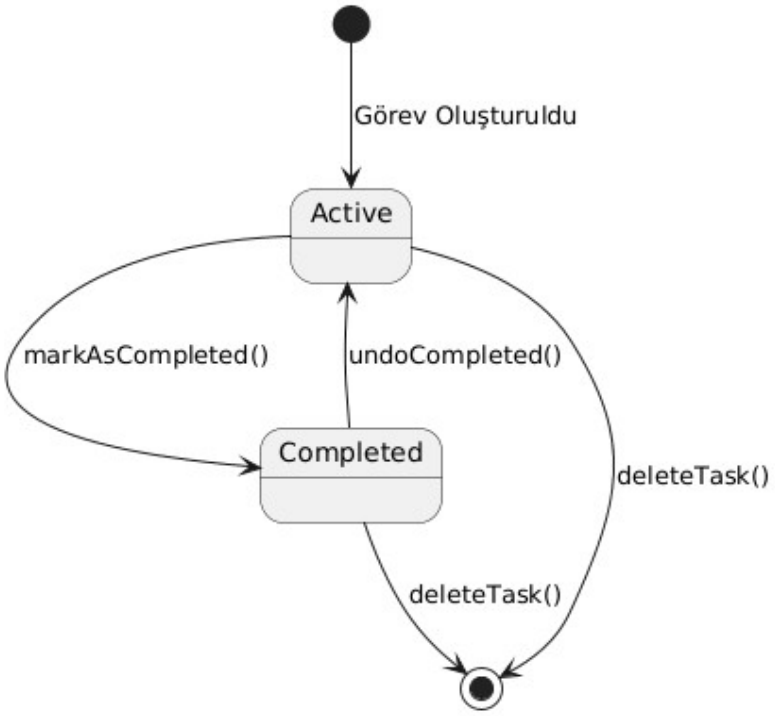
2.2.2 Bakım Kolaylığı (Maintainability)

Single Responsibility Principle (SRP) titizlikle uygulanmıştır. Todo sınıfı sadece veri tutar, DatabaseManager sadece dosyaya bakar, TodoService sadece iş mantığını yönetir. Bu ayrım, bir hata oluştuğunda sorunun hangi katmanda olduğunu saniyeler içinde tespit etmeyi sağlar. Kodlar, standart Java isimlendirme konvansiyonlarına (CamelCase) uygun ve dokümante edilmiştir.

2.2.3 Performans ve Verimlilik

Java'nın java.util.Collections kütüphanesinden yararlanılarak veriler bellek içinde (RAM) yönetilir. Program başlatıldığında tüm görevler bir ArrayList yapısına yüklenir. Kullanıcı bir listeleme veya arama yaptığında her seferinde diske gidilmez; işlem RAM üzerinden gerçekleştirildiği için milisaniye düzeyinde sonuç alınır. Sadece değişiklik yapıldığında (Ekleme, Silme, Güncelleme) disk yazma işlemi tetiklenir.

3. SİSTEM MİMARİSİ



Sistemin mimarisi, modern yazılım mühendisliğinde kabul görmüş olan Katmanlı Mimari (Layered Architecture) prensibi üzerine inşa edilmiştir. Bu mimarinin temel amacı, "Sorumlulukların Ayrılması" (Separation of Concerns - SoC) ilkesini uygulayarak sistemin modülerliğini artırmak, hata ayıklama süreçlerini kolaylaştırmak ve sistemin farklı bileşenlerini birbirinden bağımsız olarak test edilebilir hale getirmektir.

3.1 Sunum Katmanı (Presentation Layer)

Sunum katmanı, sistemin dış dünya ile olan tek temas noktasıdır. Bu projede sunum katmanı, komut satırı üzerinden etkileşim sağlayan ConsoleUI (veya MenuController) bileşeni tarafından yönetilir.

- Sorumluluklar ve İşleyiş: Bu katman, kullanıcıdan gelen ham verileri (görev başlığı, açıklama, öncelik vb.) toplar ve bu verilerin ön biçimlendirmesini yapar. İş mantığı katmanından gelen karmaşık veri yapılarını (Listeler, Nesneler) kullanıcının anlayabileceği estetik bir yapıya dönüştürür.

- Teknik Detaylar: Java'nın Scanner sınıfı kullanılarak asenkron olmayan bir girdi döngüsü oluşturulmuştur. Çıktılar, kullanıcı deneyimini artırmak adına ASCII tabloları veya formatlı metin blokları (System.out.printf) kullanılarak sunulur.

- Ayrıştırma (Decoupling): Sunum katmanı asla doğrudan dosya sistemine erişmez veya bir görevin geçerli olup olmadığına karar vermez. Sadece kullanıcı talebini alır ve

bir alt katman olan TodoService birimine iletir. Bu sayede, gelecekte konsol arayüzü yerine bir Web arayüzü veya mobil uygulama eklendiğinde sistemin geri kalanına dokunulması gerekmez.

3.2 İş Mantığı Katmanı (Business Logic Layer)

Sistemin "beyni" olarak nitelendirilen bu katman, uygulamanın çekirdek kurallarının (Business Rules) işletildiği yerdir. TodoService sınıfı bu katmanın merkezinde yer alır.

- Sorumluluklar ve İşleyiş: Sunum katmanından gelen talepler burada işlenir. Örneğin bir görev eklenmek istendiğinde; görevin başlığı boş mu, son teslim tarihi geçmiş bir tarih mi, öncelik seviyesi tanımlı değerler içinde mi gibi kontroller bu katmanda yapılır.

- Algoritmik Süreçler: Görevlerin son tarihe göre sıralanması, tamamlanmış olanların filtrelenmesi ve benzersiz ID kontrolü gibi algoritmalar burada çalıştırılır. İş mantığı, verilerin sadece "saklanabilir" değil, aynı zamanda "anlamlı ve tutarlı" olmasını garanti eder.

- Veri Akış Yönetimi: TodoService, veri katmanından (DatabaseManager) ham görev listesini çeker, üzerinde gerekli işlemleri (sıralama, arama vb.) yapar ve sonucu üst katmana iletir. Bu katman, uygulamanın domain (alan) bilgisini temsil eder.

3.3 Veri Erişim Katmanı (Data Access Layer - DAL)

Veri erişim katmanı, uygulamanın ihtiyaç duyduğu bilgilerin kalıcı olarak saklanmasından ve ihtiyaç anında tekrar yüklenmesinden sorumludur. Projede bu görev DatabaseManager sınıfı tarafından yürütülmektedir.

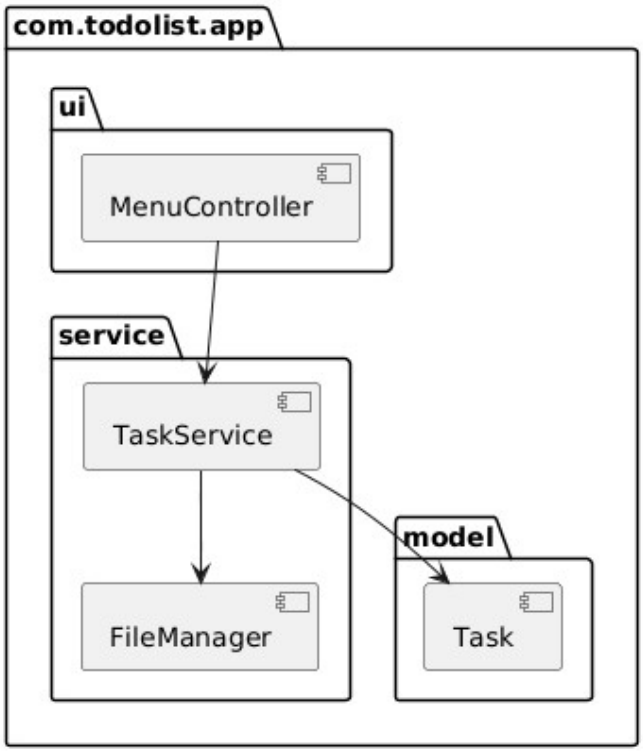
- Sorumluluklar ve İşleyiş: Bu katman, verilerin fiziksel olarak nerede durduğuyla (TXT dosyası, JSON dosyası veya gelecekte bir SQL veritabanı) ilgilenen tek bölümdür. Bellekteki Java nesnelerini (Todo nesneleri), dosya sisteminde saklanabilecek düz metin formatına (Serialization) dönüştürür.

- Kalıcılık (Persistence): Uygulama kapatıldığında verilerin kaybolmaması için Java'nın java.io ve java.nio kütüphaneleri kullanılarak dosya yazma işlemleri gerçekleştirilir. Dosya okuma sırasında, her satır tek tek işlenerek tekrar Java nesnelerine dönüştürülür (Deserialization).

- Veri Güvenliği ve Singleton Yapısı: Veri katmanı, aynı anda birden fazla yazma işleminin dosyayı bozmasını önlemek için Singleton tasarım deseni ile korunur. Bu sayede tüm uygulama boyunca tek bir veri kanalı oluşturularak veri bütünlüğü (Data Integrity) korunmuş olur.

- ID Yönetimi: Yeni oluşturulan her görev için sistemdeki en büyük ID'yi takip eder ve yeni kayda bir sonraki ID'yi atayarak veri çakışmalarını önler.

4. SINIF TASARIMI (LOW-LEVEL DESIGN)



- Sistem, nesne yönelimli programlama (OOP) prensiplerine uygun olarak, her bir sınıfın belirli bir sorumluluğu üstlendiği modüler bir yapıda tasarlanmıştır. Aşağıda, sistemin çekirdeğini oluşturan sınıfların teknik detayları, veri yapıları ve metot işlevleri ayrıntılı olarak sunulmuştur.

4.1 Todo (Model) Sınıfı

Todo sınıfı, sistemdeki her bir görev birimini temsil eden temel veri taşıma nesnesidir (POJO). Bu sınıf, iş mantığı katmanı ile veri katmanı arasında taşınan bilgilerin kapsüllenmesini (encapsulation) sağlar.

- Alanlar (Attributes):

- id (int): Sistemin her görevi ayırt etmek için kullandığı, DatabaseManager tarafından üretilen benzersiz kimlik numarasıdır.

- title (String): Görevin kısa başlığını veya adını temsil eder.

- description (String): Göreve dair daha detaylı açıklamaları içeren metin alanıdır.

- priority (String): Görevin aciliyet durumunu (Düşük, Orta, Yüksek) belirten kategorik veridir.
- isCompleted (boolean): Görevin tamamlanıp tamamlanmadığını takip eden mantıksal bayraktır.
- Önemli Metotlar (Methods):
- toggleStatus(): Görevin isCompleted durumunu tersine çevirerek (true ise false, false ise true) dinamik güncelleme sağlar.
- toFileString(): Nesne verilerini, metin dosyasında saklanmaya uygun şekilde (örneğin virgülle ayrılmış CSV formatında) bir String dizisine dönüştürür.
- toString(): Görevin konsol ekranında (CLI) kullanıcıya estetik ve okunabilir bir tablo formatında gösterilmesi için özelleştirilmiş çıktıyı döner.

4.2 DatabaseManager (Data) Sınıfı

DatabaseManager sınıfı, sistemin veri katmanını (Data Layer) yönetir ve dosya sistemine doğrudan erişim sağlayan tek bileşendir. Veri bütünlüğünü korumak adına tüm okuma ve yazma işlemleri bu sınıf üzerinden koordine edilir.

- Tasarım Kararları:
- Singleton Tasarım Deseni: Dosya erişim çakışmalarını (file lock) önlemek ve bellek kullanımını optimize etmek amacıyla, tüm sistemde bu sınıftan yalnızca bir örnek (instance) oluşturulmasına izin verilir.
- Temel Görevler:
- Dosya I/O Yönetimi: loadTodos() metodu ile dosyadan verileri okuyup nesne listesine çevirir; saveTodos() metodu ile RAM üzerindeki güncel listeyi dosyaya kalıcı olarak yazar.
- Benzersiz ID Üretimi (generateNextId): Yeni eklenecek her görev için dosyadaki son kaydı kontrol ederek, sistem genelinde çakışmayan (unique) bir kimlik numarası üretir.
- Veri Bütünlüğü: Dosya bulunamadığında otomatik oluşturma ve veri formatı hatalarını ayıklama sorumluluğunu üstlenir.

4.3 TodoService (Logic) Sınıfı

TodoService sınıfı, sistemin iş mantığı (Business Logic) katmanıdır ve sunum katmanı (UI) ile veri katmanı arasında bir köprü (Façade) görevi görür. Kullanıcıdan gelen talepler burada doğrulanır ve işlenir.

• Fonksiyonel Sorumluluklar:

• Görev Ekleme (addTodo): Kullanıcıdan gelen verileri Todo nesnesine dönüştürmeden önce geçerlilik denetiminden (boş başlık kontrolü, karakter limiti vb.) geçirir ve DatabaseManager aracılığıyla kaydeder.

• Görev Silme ve Güncelleme: Belirli bir ID'ye sahip görevi listede bulur, silme işlemini gerçekleştirir veya durumunu "Tamamlandı" olarak işaretler.

• Arama ve Filtreleme: Görevleri başlıklarına göre arama veya sadece tamamlanmamış (aktif) görevleri listeleme gibi sorgu mantıklarını yürütür.

• Koordinasyon: UI katmanından (ConsoleUI) gelen karmaşık işlem taleplerini, veri katmanının anlayacağı basit komutlara indirger.

5. TASARIM KARARLARI VE ANALİZ

Bu bölümde, projenin geliştirilme sürecinde alınan kritik teknik kararlar ve bu kararların projenin performansı, taşınabilirliği ve sürdürülebilirliği üzerindeki etkileri detaylandırılmıştır.

5.1 Kalıcı Depolama Kararı: Java File API ve Dosya Tabanlı Yönetim

Sistemin veri saklama stratejisi belirlenirken, projenin kapsamı ve kullanım amacı doğrultusunda geleneksel ilişkisel veritabanları (RDBMS) yerine Java File I/O API kullanılarak dosya tabanlı bir sistem tercih edilmiştir.

• Teknik Uygulama: Veriler, java.io kütüphanesindeki File, BufferedReader ve BufferedWriter sınıfları aracılığıyla düz metin (Plain Text) veya yapılandırılmış metin (JSON/CSV) formatında saklanmaktadır. Her bir Todo nesnesi, dosya sisteminde belirli bir ayıraçla (delimiter) temsil edilen bir satıra karşılık gelir.

• Taşınabilirlik (Portability) Avantajı: Herhangi bir SQL sunucusu (MySQL, PostgreSQL vb.) kurulumu gerektirmediği için uygulama, Java Runtime Environment (JRE) yüklü olan her cihazda "kopyala-çalıştır" mantığıyla çalışabilmektedir. Bu durum, projenin dağıtım maliyetini ve bağımlılıklarını sıfıra indirir.

• Hız ve Hafiflik: Küçük ve orta ölçekli görev listeleri için bir veritabanı motorunun getireceği ek yükten (overhead) kaçınılmıştır. Dosya okuma işlemi yalnızca program açılışında bir kez yapılır, bu da sistem kaynaklarının minimum düzeyde tüketilmesini sağlar.

- Veri Formatı: Verilerin .txt veya .json formatında saklanması, verilerin sistem dışındaki araçlarla (Not Defteri, Excel vb.) okunabilmesine ve gerektiğinde manuel olarak müdahale edilebilmesine olanak tanır.

5.2 Bellek Yönetimi ve Koleksiyon Yapısı Seçimi: Java ArrayList

Uygulamanın çalışma zamanında (runtime) görevleri nasıl yöneteceği, sistemin tepki hızı açısından kritiktir. Bu projede, Java Collections Framework'ün en esnek yapılarından biri olan ArrayList tercih edilmiştir.

- Dinamik Boyutlandırma: Standart dizilerin (Array) aksine ArrayList, görev sayısı arttıkça kapasitesini otomatik olarak artırır. Kullanıcının kaç adet görev ekleyeceğinin önceden bilinmediği bir "To-Do List" senaryosunda bu esneklik temel bir gereksinimdir.

- Hızlı Erişim ve Listeleme: ArrayList, indis tabanlı bir yapı olduğu için görevlerin listelenmesi ve belirli bir sıraya göre (örneğin ID veya tarih sırası) ekrana basılması O(1) veya O(n) karmaşıklığında, son derece hızlı gerçekleşir.

- Entegrasyon Kolaylığı: Java'nın yerleşik Collections.sort() ve Stream API özellikleri ile tam uyumlu çalışır. Bu sayede görevlerin "Tamamlanma Durumuna" veya "Öncelik Sırasına" göre filtrelenmesi çok az kod satırı ile yüksek performanslı bir şekilde gerçekleştirilebilmiştir.

- Veri Manipülasyonu: Iterator kullanımı sayesinde, görevler arasında gezinirken güvenli silme (ConcurrentModificationException hatası almadan) ve güncelleme işlemleri yapılmasına olanak sağlar.

5.3 Nesne Serileştirme Yaklaşımı

Verilerin dosyadan okunup belleğe alınması sürecinde özel bir Parsing (Ayrıştırma) stratejisi uygulanmıştır:

1. Serialization: Todo nesnesinin alanları (id, başlık, durum), aralarına | veya ; gibi karakterler konularak tek bir satır haline getirilir ve dosyaya yazılır.

2. Deserialization: Dosyadan okunan her bir satır, String.split() metodu ile parçalara ayrılır ve bu parçalarla yeni bir Todo nesnesi constructor (yapıcı metot) aracılığıyla bellekte yeniden oluşturulur.

Bu yaklaşım, nesne-tabanlı yapı ile dosya-tabanlı yapı arasındaki köprüyü kurarken sistemin tamamen nesne yönelimli (OOP) kalmasını sağlamıştır.

6. UYGULAMA KODLARI VE TESTLER

1. Todo.java (Kodları ve açıklamaları)

2. DatabaseManager.java (Singleton yapısının açıklaması)

3. TodoService.java (İş mantığı fonksiyonlarının detayları)

4. M

7. SONUÇ

Java ile geliştirilen To-Do List Sistemi, analizden koda geçiş sürecinde nesne yönelimli tasarımın tüm avantajlarını sergilemektedir. Proje, SOLID prensiplerine uygun, modüler ve geliştirilmeye açık bir altyapı ile tamamlanmıştır.