**Hazırlık ve Planlama Aşamaları**

**1. Proje Hedefinin Netleştirilmesi**

* (1.1) **Projenin amacını açıkça tanımla**:  
  Bu proje ile amacımız; belirli bir SaaS sağlayıcısından (örneğin Microsoft 365) çektiğimiz lisans, kullanım, maliyet verilerini işleyerek, Raynet platformuna entegre edilebilecek bir analitik katman yaratmak. Bu katman müşterilere hangi lisansların kullanılmadığını, nasıl maliyet optimizasyonu yapabileceklerini ve kullanım trendlerini sunan bir çözüm üretecek.
* (1.2) **Kapsam Belirle**:  
  İlk etapta tek bir SaaS sağlayıcısı ile başlayacağız (Microsoft 365). Bu, projenin karmaşıklığını azaltıp prototipin hızlı ortaya çıkmasını sağlar. İleride Salesforce, Jira Cloud veya Zoom gibi diğer SaaS platformları eklenebilir.
* (1.3) **Başarı kriterlerini belirle**:  
  Başarı kriterlerimiz örneğin şöyle olabilir:
  + Çektiğimiz verilerden en az %90 doğrulukla kullanılmayan lisansları tespit edebilmek.
  + En az bir maliyet optimizasyon önerisi sunabilmek (örneğin, 5 kullanıcının premium lisans yerine standart lisansa geçmesiyle aylık X dolar tasarruf sağlanacağı).
  + Kullanım verilerini grafiklerle gösteren, filtreleme imkânı olan bir dashboard üretmek.

**2. Teknik Gereksinimlerin ve Kaynakların Belirlenmesi**

* (2.1) **Yazılım dili ve çerçeveleri seç:**
  + Backend için: Python (veri işleme, analitik ve API entegrasyonu için uygun) ve FastAPI (hızlı REST API geliştirmek için)
  + Veri işleme için: Pandas, NumPy (analitik ve veri dönüşümleri)
  + Frontend için: React (komponent tabanlı yapı, hızlı prototipleme ve geniş eklenti ekosistemi için)
  + Veritabanı için: PostgreSQL (ölçeklenebilirlik, veri bütünlüğü ve JSON desteği açısından avantajlı)  
    Bu seçimlere katılıyor musunuz, ya da alternatif önerileriniz var mı? Veya farklı bir araç/dil kullanmak ister misiniz?
* (2.2) **Donanım/Geliştirme Ortamı:**
  + Geliştirmeyi kendi bilgisayarınızda yapacağız. Burada Docker kullanarak bir container tabanlı ortam oluşturabiliriz. Bu sayede tüm bileşenler (PostgreSQL, backend, frontend) birbiriyle uyumlu çalışır.
  + Geliştirme aşamasında örnek veriler, test senaryoları ve yerel bir database kurulumu ile çalışacağız.  
    Bu yaklaşım size uygun mu? Geliştirme ortamında eklemek istediğiniz özel bir araç veya yöntem var mı?
* (2.3) **API Bilgisi Toplama:**
  + Seçtiğimiz SaaS sağlayıcısı olarak Microsoft 365’i düşünelim. Bu durumda Microsoft Graph API dokümantasyonunu inceleyeceğiz.
  + OAuth2 kimlik doğrulaması ile Access Token alma, kullanıcı listesi, lisans bilgileri, kullanım istatistikleri gibi endpoint’leri belirleyeceğiz.
  + Gerekirse test amaçlı Microsoft Graph Playground veya dummy veri setleri kullanarak ilk çağrıları simüle edeceğiz.

**3. Mimari ve Veri Modeli Tasarımı**

* (3.1) **Yüksek seviye mimari diyagramı oluştur**:  
  Bu diyagram, SaaS API → Data Ingestion Layer → Veritabanı → Analitik Katman → Web Dashboard arasındaki veri akışını gösteren basit bir şema olacak.
  + **3.1.1: Mimari Diyagram Oluşturma**  
    Çizmek istediğimiz diyagram, en azından aşağıdaki bileşenleri içermeli:
    - **SaaS API (Microsoft 365 / Microsoft Graph API)**: Dış veri kaynağı
    - **Data Ingestion Service (Backend Ingestion Katmanı)**: Python tabanlı bir servis, OAuth2 ile kimlik doğrulaması yapar, API’dan veri çeker.
    - **Veritabanı (PostgreSQL)**: raw\_data ve normalized\_data tabloları ile ek tablo(lar)
    - **Analitik Katman (Python Scriptler / Backend API):** Normalized veriler üzerinde sorgular yapar, trend analizi, grup bazlı istatistikler çıkarır.
    - **REST API (FastAPI)**: Frontend’in istekte bulunacağı endpoint’leri sunar. Bu endpoint’ler raporlama, filtreleme, maliyet analizleri gibi verileri döndürür.
    - **Frontend (React)**: Kullanıcının erişeceği dashboard. Filtreler, grafikler ve tablolar burada görselleştirilecek.
  + **3.1.2: Veri Akışı Detayları**
* Periodik bir görev (örn. her gece saat 01:00) Data Ingestion Service, Microsoft Graph API’ye bağlanır, Access Token alır, kullanıcı ve lisans bilgilerini JSON olarak çeker.
* Bu JSON verileri Veritabanı’ndaki raw\_data tablosuna eklenir.
* Ardından bir normalizasyon script’i devreye girer (Cron job ile tetiklenebilir veya ingestion’dan hemen sonra çalışır). Bu script raw\_data’dan veriyi okuyup lisans tiplerini standartlaştırır, kullanıcı kimliklerini birleştirir, süre bilgilerini zaman serisi formatında normalized\_data tablosuna yazar.
* Analitik fonksiyonlar (Python script veya API endpoint’i), normalized\_data tablosunda sorgular yapar. Bu sorgulardan elde edilen sonuçlar (kullanılmayan lisans sayısı, aylık maliyet ortalaması, departman bazlı kullanım gibi) API üzerinden frontend’e sunulur.
* Frontend bu verileri grafiğe, tabloya dönüştürerek kullanıcıya gösterir.
* (3.2) **Veritabanı şeması tasarla**:
  + raw\_data tablosu: API’dan gelen ham veriyi saklayacak. Kolonlar: id, fetched\_at (tarih/zaman damgası), raw\_json (JSONB kolonu), source\_platform (örn. "M365").
  + normalized\_data tablosu: Temizlenmiş ve anlamlı hale getirilmiş veri. Kolonlar: id, user\_id, license\_type, last\_activity\_date, monthly\_cost, active\_user\_flag, fetched\_period (verinin hangi dönemi kapsadığı), vb.
* **3.2.1: raw\_data Tablosu**
  + Amaç: API’dan çekilen ham veriyi saklamak.
  + Kolonlar:
    - id (SERIAL, primary key)
    - fetched\_at (TIMESTAMP, ne zaman çekildi)
    - raw\_json (JSONB, tüm ham veri)
  + Burada her API çağrısı sonrası gelen veri tek bir kayıt olarak saklanabilir. Sonraki aşamada script bu JSONB içinden gerekli alanları okuyacak.
* **3.2.2: normalized\_data Tablosu**
  + Amaç: Ham veriyi işlenmiş, sorgulanabilir formata dönüştürmek.
  + Önerilen kolonlar:
  + id (SERIAL, primary key)
  + user\_id (TEXT): Kullanıcıyı tanımlayan eşsiz kimlik. Microsoft Graph API ile gelen kullanıcı id’si.
  + department (TEXT): Kullanıcının departmanı (API’dan veya ek bir haritalama tablosundan alınabilir)
  + license\_type (TEXT): Standart, Premium, Enterprise vb.
  + last\_activity\_date (DATE): Kullanıcının son aktif olduğu tarih
  + cost\_per\_license (NUMERIC): Bu lisans tipinin aylık maliyeti
  + record\_date (DATE): Bu kaydın geçerli olduğu tarih (zaman serisi analizi için)
  + Bu tabloya her normalizasyon çalışmasında insert yapılır. Eski veriler zaman serisi incelemeleri için saklanır.
* **3.2.3: Ek Tablolar (Opsiyonel)**
  + license\_info tablosu: Her lisans tipinin adını, üreticisini, varsayılan maliyetini, destek bitiş tarihlerini tutabilir.
  + department\_info tablosu: Departman adlarını, açıklamalarını tutabilir. Böylece normalized\_data’da sadece department\_id tutulur.
* **3.2.4: İndeks ve Performans Stratejisi**
  + normalized\_data üzerinde user\_id, record\_date, license\_type kolonlarına indeks koyarak sorgu hızını artır.
  + Özellikle zaman serisi analizleri için record\_date’e indeks eklemek trend analizinde performans sağlar.
  + Sık sorgulanan kombinasyonlar (ör. department + license\_type) için bileşik indeks düşünülebilir.

**3.3 Analitik Katman Tasarımı**

* **Adım 3.3.1: Temel Sorgular**
  + Kullanılmayan lisansları bul:  
    SELECT user\_id, license\_type FROM normalized\_data WHERE last\_activity\_date < CURRENT\_DATE - INTERVAL '30 days';
  + Departman bazlı maliyet ortalaması:  
    SELECT department, license\_type, AVG(cost\_per\_license) AS avg\_cost FROM normalized\_data GROUP BY department, license\_type;
  + Zaman serisi trendi:  
    SELECT record\_date, COUNT(\*) AS active\_users FROM normalized\_data WHERE last\_activity\_date >= record\_date - INTERVAL '30 days' GROUP BY record\_date;
* **Adım 3.3.2: Maliyet Optimizasyon Önerileri**
  + Kullanılmayan lisanslar tespit edildiğinde, premium lisansa sahip ancak aktif olmayan kullanıcılara standart lisansa geçilmesi önerilebilir. Bunun için license\_info tablosundan premium ve standart lisans fiyat farkı hesaplanır. Örn.  
    SELECT (premium\_cost - standard\_cost) \* inactive\_user\_count AS potential\_saving;
* **Adım 3.3.3: Ek Analitik Araçlar**
  + İleride makine öğrenimi eklenecekse, normalized\_data’dan veriyi bir pandas DataFrame’e çekip cluster analizi, zaman serisi tahmini (Prophet, ARIMA) gibi ileri modeller uygulanabilir. Bu aşamada henüz prototip olduğu için basit sorgular yeterli.

**3.4 API Endpoint Tasarımı (FastAPI)**

* **Adım 3.4.1: Endpoint Örnekleri**
  + /api/unused-licenses: Kullanılmayan lisansların listesini JSON olarak döner.
  + /api/cost-analysis?department=xyz: Belirli bir departman için ortalama maliyet ve potansiyel tasarruf bilgilerini döner.
  + /api/usage-trends?start=2023-01-01&end=2023-02-01: Verilen tarih aralığı için günlük/haftalık aktif kullanıcı sayılarını döner.
  + /api/recommendations: Otomatik oluşturulmuş lisans değişikliği önerilerini JSON olarak döner.
* **Adım 3.4.2: Kimlik Doğrulama ve Yetkilendirme**
  + Başlangıçta basit tutarak herhangi bir kimlik doğrulama olmadan sunabiliriz. Ancak Raynet ile entegre edileceğini varsayarak bir API key kontrolü veya OAuth2 yetkilendirmesi ileride eklenebilir.

**3.5 Frontend Veri Yapıları ve Görselleştirme**

* **Adım 3.5.1: Frontend’de Veri Modeli**
* Frontend, /api/unused-licenses gibi endpointlerden JSON alır. Örneğin:
* { "unused\_licenses": [ {"user\_id": "u123", "license\_type": "Premium", "last\_activity": "2023-01-10", "potential\_saving": 15.00}, {"user\_id": "u456", "license\_type": "Enterprise", "last\_activity": "2023-01-05", "potential\_saving": 25.00} ] }
* Bu veriler tabloda gösterilebilir. Filtre butonlarıyla departmana göre filtre uygulanabilir.
* **Adım 3.5.2: Grafikler**
* Zaman serisi trend grafiği: record\_date ve active\_users değerlerini line chart ile göster.
* Maliyet dağılımı: license\_type bazında ortalama maliyetleri bar chart ile sun.
* Öneri listesi: Bir tabloda kullanıcıları, potansiyel tasarrufu gösteren sütun ile listeleyin.
* **3.6 Zamanlama Mekanizmaları**
* **Adım 3.6.1: Veri Çekme Sıklığı**
* Data Ingestion her gece 01:00’da çalışır (cron job)
* Normalizasyon script’i ingestion’dan 15 dakika sonra otomatik tetiklenir.
* Analitik endpointler istek bazlı çalışır (gerçek zamanlı). Raporun önbelleğe alınması istenirse Redis gibi bir cache eklenebilir.
* **Adım 3.6.2: Bildirim ve Loglama**
* Her ingestion sırasında log tutarak kaç kayıt çekildiğini yaz.
* Hata durumlarında e-posta uyarısı veya basit bir Slack webhook bildirimi kurgulanabilir.
* **3.7 Genişletilebilirlik ve Esneklik**
* **Adım 3.7.1: Ek SaaS Platformları Ekleme**  
  İleride yeni bir platform eklemek istediğinizde yapmanız gereken tek şey yeni bir ingestion script’i yazıp veriyi yine raw\_data tablosuna eklemek. Normalizasyon script’i platform bazlı filtre yaparak veri işleyebilir.
* **Adım 3.7.2: Yeni Metrikler veya Analitik Fonksiyonlar**  
  İleride lisansın sadece maliyet değil güvenlik zafiyetleri ile de korele edilmesi istenirse ek tablolar (zafiyet\_info) eklenebilir. Aynı mimari yapısı korunur, ek sorgularla veri zenginleştirilir.
* (3.3) **Analitik katmanda kullanacağın veri dönüşümleri**:  
  Örneğin:
  + Kullanım trendi için zaman bazlı agregasyon (user\_id ve tarihe göre group by).
  + Lisans tipine göre maliyet ortalamaları, inaktif kullanıcıların oranı.
  + İleride basit bir machine learning modeli (k-means) için gerekli alanların (ortalama kullanım süresi, son login sayısı vb.) hesaplanması.