Sunucu Mimarisi Tasarımı

Proje için **mikro servis mimarisi (Microservices Architecture)** tercih edilmelidir. Bu yapı, ölçeklenebilirliği ve esnekliği artırarak farklı hizmetlerin bağımsız olarak yönetilmesini sağlar.

Mimari Bileşenler:

1. **API Gateway:** Kullanıcı isteklerini uygun mikro servislere yönlendiren bir katmandır. (Örn: Nginx, API Gateway)
2. **Authentication & Authorization Service:** Kullanıcı kimlik doğrulama ve yetkilendirme işlemlerini yönetir. (Örn: OAuth2, JWT)
3. **User Service:** Kullanıcı bilgilerini yönetir.
4. **Content Service:** Kullanıcıların gönderdiği içerikleri (metin, resim, video) saklar.
5. **Interaction Service:** Beğeni, yorum ve paylaşımlar gibi etkileşimleri yönetir.
6. **Notification Service:** Bildirimleri yönetir (e-posta, push notifications vb.).
7. **Database Cluster:** Verilerin güvenli ve ölçeklenebilir şekilde saklanmasını sağlar.
8. **Cache Layer:** Veritabanına yüklenmeden sık kullanılan verileri saklar.
9. **Load Balancer:** Trafiği dengeler.
10. **CDN (Content Delivery Network):** Büyük medya dosyalarının hızlı dağıtımını sağlar.

Veritabanı Yönetimi

Seçilen Veritabanı:

- **PostgreSQL + Redis + Elasticsearch**
  - PostgreSQL: Kullanıcı bilgileri, içerik verileri ve ilişkisel işlemler için.
  - Redis: Gerçek zamanlı önbellekleme için.
  - Elasticsearch: Hızlı arama ve indeksleme işlemleri için.

Veritabanı Kümesi (Cluster) Yapısı:

- **Master-Slave Replikasyonu:** PostgreSQL üzerinde, okuma/yazma işlemleri ana (master) sunucuda, okuma işlemleri ise yedek (replica) sunucular tarafından gerçekleştirilir.
- **Sharding:** Büyük veri kümelerini farklı sunuculara bölerek ölçeklenebilirliği artırır.
- **Backup & Disaster Recovery:** Günlük otomatik yedekleme ve felaket kurtarma planları uygulanır.

Ölçeklenebilirlik Stratejileri

Yatay Ölçekleme (Horizontal Scaling)

- Sunucu yükü arttığında, yeni mikro servis kopyaları çalıştırılarak yük dengelenir.

Otomatik Yük Dengeleme (Auto Scaling & Load Balancing)

- Sunucu yükünü otomatik olarak yönetmek için **Docker Compose** veya **basit bir load balancer** kullanılabilir.
- **Nginx veya Apache HTTP Server** gibi araçlarla gelen istekler dengelenerek sunucuya aşırı yük binmesi engellenir.

Mesaj Kuyruğu (Message Queue) Kullanımı

- **Redis** veya **Microsoft Azure** gibi daha kolay yönetilebilen araçlar kullanılabilir.
- Kuyruklama sayesinde, yoğun isteklerde işlemler daha düzenli bir şekilde sıralanır ve sistem daha verimli çalışır.

Önbellekleme ve Hızlandırma

- **Redis / Memcached:** Sık kullanılan verileri önbelleğe almak için.
- **CDN (Cloudflare, AWS CloudFront):** Statik içeriklerin hızlı teslimi için.
- **Indexing & Query Optimization:** Veritabanında indeksleme ve optimize edilmiş sorgular kullanılır.

Güvenlik Stratejileri

- **JWT & OAuth2 ile Kimlik Doğrulama:** Kullanıcı oturum yönetimi ve yetkilendirme.
- **Rate Limiting ve DDOS Koruması:** API isteklerini sınırlamak için. (Örn: Cloudflare, AWS WAF)
- **Veri Şifreleme:** Kullanıcı verileri SHA-256 ile şifrelenir.
- **SQL Injection & XSS Koruması:** ORM kullanımı ve giriş doğrulama mekanizmaları.
- **İki Aşamalı Kimlik Doğrulama (2FA):** Kullanıcı hesaplarını koruma amaçlı.

Performans İzleme ve Analiz

- **Log Yönetimi:**
  - **ELK Stack (Elasticsearch, Logstash, Kibana):** Log analizi için.
  - **Grafana + Prometheus:** Gerçek zamanlı sunucu ve uygulama izleme.
- **APM (Application Performance Monitoring):**
  - **New Relic, Datadog veya Jaeger:** Mikro servislerin performansını ölçmek için.
- **Health Checks ve Alerts:**
  - Kubernetes Liveness & Readiness Probes
  - Prometheus Alarms