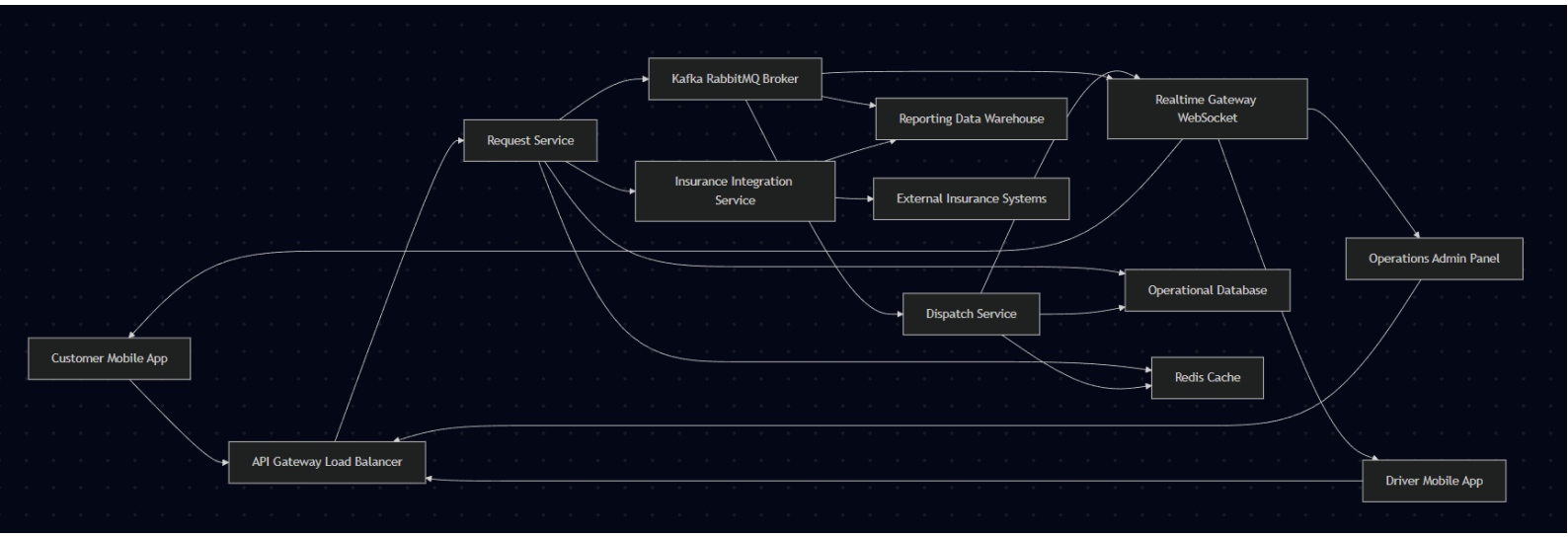

Teknolojiler ve Katmanlar

- **Client Katmanı:**
  - *Customer Mobile App, Driver Mobile App, Operations Admin Panel*
  - Hepsi backend ile **HTTP/REST + JSON** ve gerçek zamanlı için **WebSocket** üzerinden konuşuyor.
- **Edge Katmanı:**
  - **Nginx / Traefik + API Gateway**
  - Auth, rate limiting, routing gibi işler burada. Arkadaki servisler stateless, gerektiğinde yatayda çoğaltılabiliyor.
- **Uygulama Katmanı (Python / Django):**
  - Çekirdek servisler: **Request Service, Dispatch Service, Insurance Integration Service, Realtime Gateway backend'i**
  - Hepsi **Django + Django REST Framework** ile yazılmış bağımsız servisler (veya aynı proje içinde modüler app'ler).
  - Aralarındaki senkron iletişim REST; arka plan işler için Celery kullanıyorum.
- **Mesajlaşma ve Kuyruk:**
  - **Celery + RabbitMQ** (veya Redis broker)
  - Yükü kuyruk üzerinde toplamak, talep/atama gibi işlemleri asenkron yönetmek için.
- **Gerçek Zamanlı Katman:**

    - **Django Channels + Redis**
    - Driver, müşteri ve admin paneline WebSocket üzerinden canlı durum ve konum bilgisi push ediyorum.
- **Veri Katmanı:**
    - **PostgreSQL**: Operasyonel veriler (requests, assignments, drivers, users...).
    - **Redis**: Sık okunan ve düşük latency gereken veriler için cache.
    - **Reporting Data Warehouse**: Kafka/Celery event'lerinden beslenen ayrı bir raporlama/veri ambarı (örneğin ayrı Postgres/ClickHouse).

1) Talep Karşılama ve Kuyruklama

Yoğunlukta gelen binlerce isteği nasıl karşılarsınız?

- Tüm istekler önce **API Gateway + Nginx**'e geliyor; burada **rate limiting** ve temel güvenlik filtreleri var.
- **Request Service (Django + DRF)**, talebi valide edip minimum veriyi PostgreSQL'e yazıyor ve işi **Celery task'i** olarak RabbitMQ'ya bırakıyor.
- **Dispatch Service** Celery worker'ları, kuyruktan gelen işleri paralel tüketiyor; yoğunluk arttığında worker sayısını artırarak yatayda ölçekleyebiliyorum.
- Böylece bayram yoğunluğunda bile yük önce kuyrukta tamponlanıyor, servisler ve DB direkt boğulmuyor.

2) Entegrasyon ve Dayanıklılık (Sigorta Servisleri)

Sigorta servisleri yavaş / cevap vermezken sistemin geri kalanı nasıl ayakta kalır?

- Tüm sigorta entegrasyonunu ayrı bir **Insurance Integration Service (Django)** altında topluyorum.
- Bu servis dış HTTP çağrılarını yapan tek nokta; burada **timeout, retry ve circuit breaker** mantığını (örn. pybreaker, custom middleware) uyguluyorum.
- Sigorta doğrulamasını mümkün olduğunca asenkron çalıştırıyorum: Request Service talebi kaydediyor, "sigorta kontrolü"nü Celery task'i olarak kuyruğa atıyor; Integration Service arkada sigorta API'sine gidip sonucu daha sonra DB'ye ve event olarak sisteme yazıyor.

- Böylece sigorta sistemi göçtüğünde bile mobil uygulama ve admin panel en azından talebi alıp “sigorta onayı bekleniyor” gibi bir state gösterebiliyor; platform komple down olmuyor.

3) Gerçek Zamanlı İletişim

Driver’ın konumu/durumu değişince müşteri ve operasyona en verimli nasıl iletirsiniz?

- **Driver Mobile App**, belirli aralıklarla konum ve job durumunu **Django Channels** üzerinden **Realtime Gateway**’e gönderiyor.
- Realtime Gateway, Customer Mobile App ve Operations Admin Panel ile açık WebSocket bağlantılarını tutuyor; driver’dan gelen güncellemeleri ilgili kullanıcılara anında push ediyor.
- Önemli state değişimlerinde (assigned, on route, arrived, completed…) hem WebSocket ile UI’ları güncelliyorum hem de gerekirse Celery üzerinden bildirim/log olayları tetikleyebiliyorum.
- Böylece “driver nerede, ne aşamada” bilgisi, polling yapmadan düşük gecikmeyle ekrana düşüyor.

4) Veri Saklama ve Raporlama

Operasyonel veriler ile raporlama/analitiği nasıl ayırırsınız?

- Günlük operasyon için **PostgreSQL** üzerinde normalleştirilmiş bir şema kullanıyorum; tüm Django servisleri transaction’ları buraya yazıyor.
- Sık kullanılan ve okuma ağırlıklı veriler için **Redis cache** devrede; özellikle Request ve Dispatch tarafında DB üzerindeki yükü ciddi düşürüyor.
- Operasyonel event’leri (request created, status changed, driver assigned vb.) Celery/Kafka üzerinde toplayıp ayrı bir **Reporting Data Warehouse**’a akıtıyorum.
- Admin paneldeki rapor ekranları doğrudan DW’ye bağlanıyor; ağır “kaç talep, hangi il, hangi saat, hangi sigorta” sorguları üretim DB’sini etkilemeden burada çalışıyor.