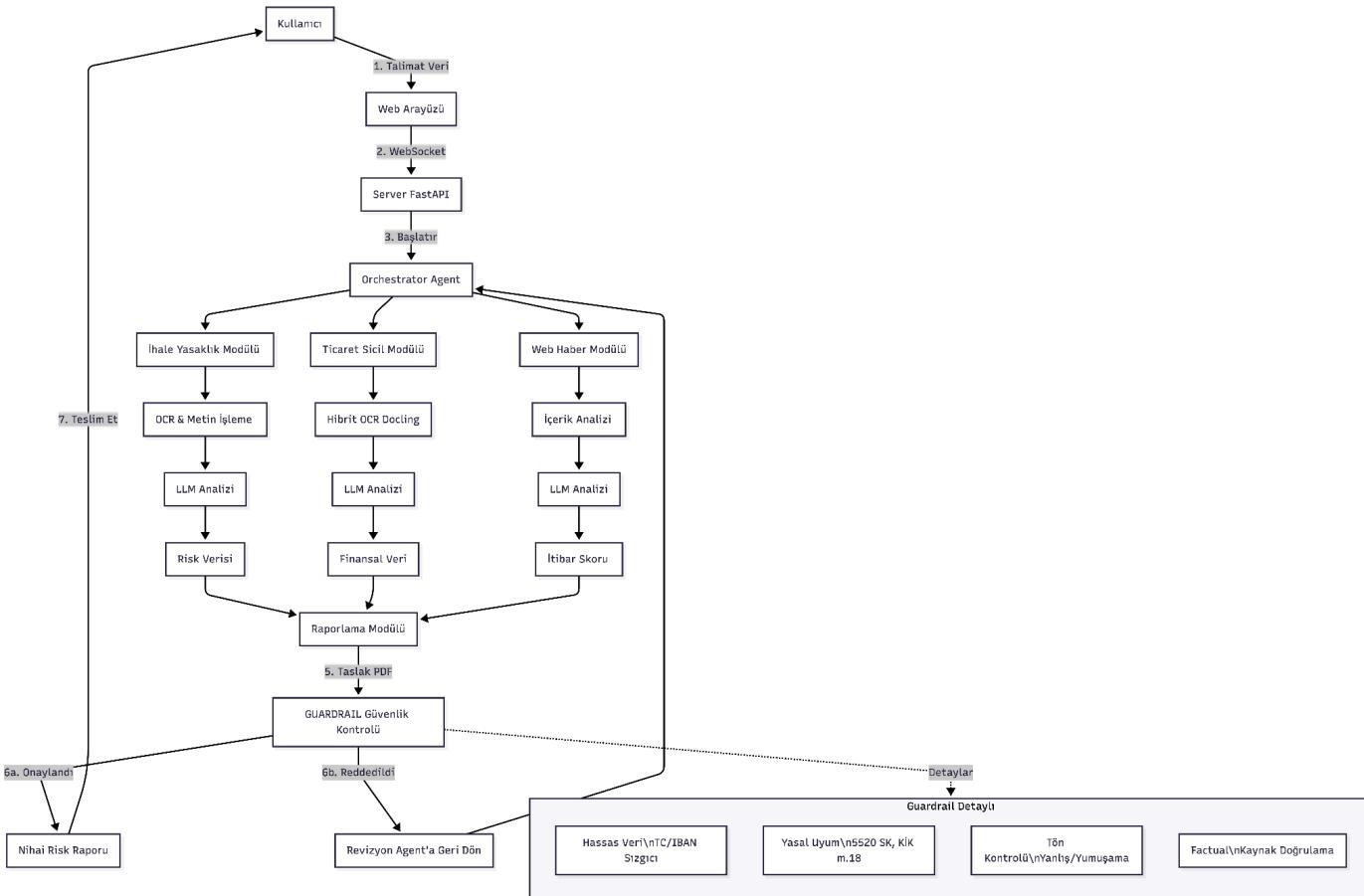


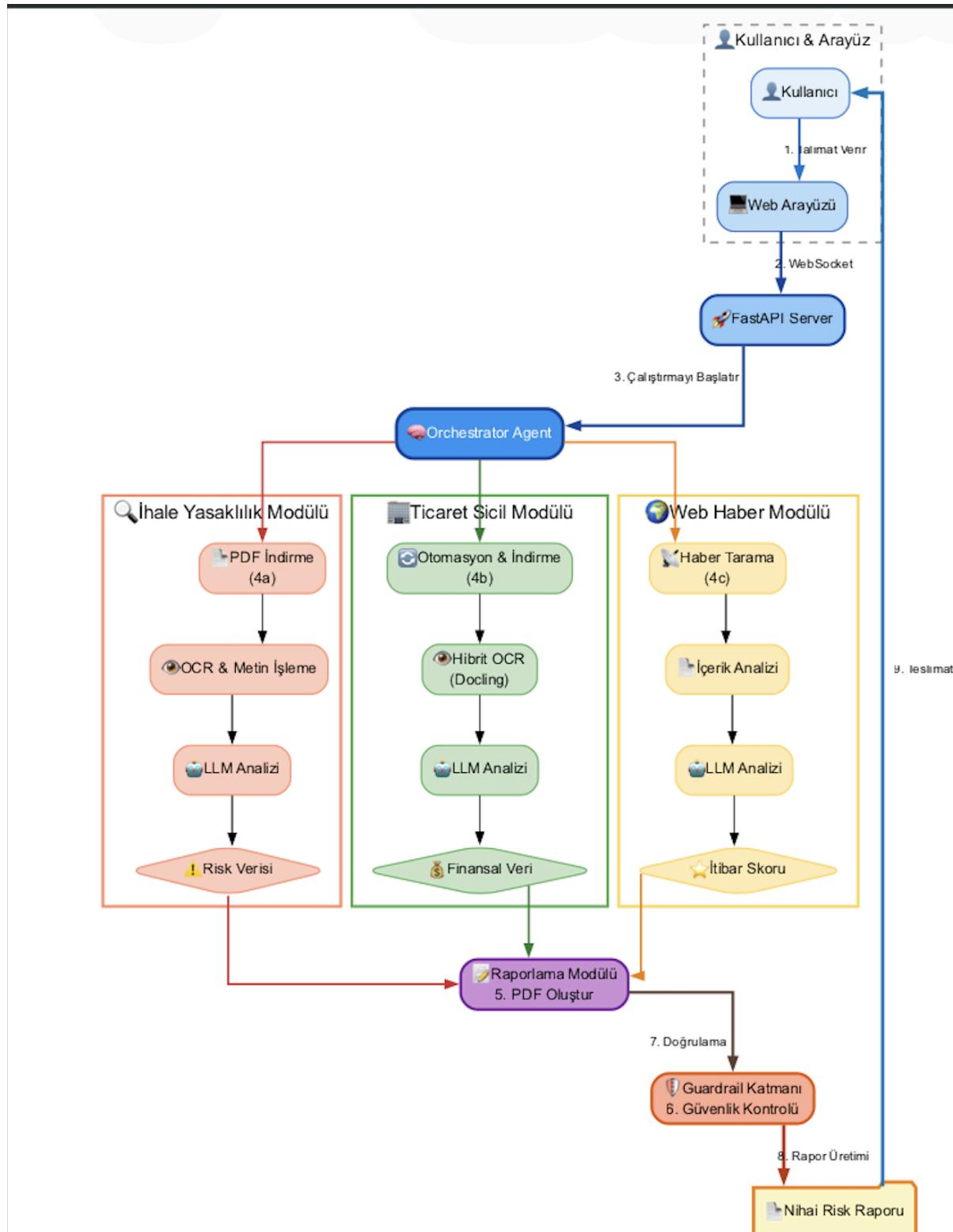


Merhaba, Şimdi size **Turkcell Risk Analyzer** projesini teknik olarak adım adım anlatacağım. Öncelikle size **Mermaid** grafiğini gösterece



TEKNOLOJİ
LİDERLERİ

ğim, sonrasında da adım adım rapor hazırlayacağım





Kullanıcı Arayüzü, WebSocket ve Sunucu Mimarisi

Bu rapor, sistemin dış dünya ile temas ettiği **Presentation Layer (Sunum Katmanı)** ve bu katmanı yöneten **Application Layer (Uygulama Katmanı)** arasındaki akışı detaylandırmaktadır. Diyagramdaki 1. adım (Talimat), Web Arayüzü kutusu, 2. adım (WebSocket) ve Server kutusunu kapsar.

1. Web Arayüzü (Presentation Layer)

Kullanıcının sistemle etkileşime girdiği noktadır. Modern, hızlı ve tepkisel bir deneyim sunmak için tasarlanmıştır.

1.1. Teknoloji Yığını (Tech Stack)

- HTML5 & CSS3:** Semantic HTML yapısı ve CSS değişkenleri (var(--turkcell-blue)) kullanılarak, harici CSS kütüphanelerine (Bootstrap, Tailwind) ihtiyaç duyulmadan, tamamen projeye özel ve hafif bir tasarım sistemi geliştirilmiştir.
- Vanilla JavaScript (ES6+):** Ağır Framework'ler (React, Angular) yerine, tarayıcının doğal DOM API'leri kullanılarak milisaniye seviyesinde yükleme ve tepki süresi elde edilmiştir.
- Event-Driven UI:** Arayüz, sunucudan gelen olaylara (Event) göre anlık olarak kendini günceller (Örn: Yeni bir log geldiğinde terminalin kayması, işlem bitince rapor butonunun açılması).

1.2. Temel Bileşenler

- Dashboard (dashboard.html):** Operasyonel özet ekranıdır. İzlenen şirketler, risk durumu ve sistem sağlığı burada görselleştirilir.
- Operasyon Merkezi (analysis.html):**
 - Canlı Terminal:** Agent'in "Düşünce Zinciri" (Chain of Thought - CoT) burada anlık olarak akar. Kullanıcı, yapay zekanın o an neye karar verdiği şeffaf bir şekilde görür.
 - Dinamik Yol Haritası:** Agent planlama yaptığından, yapılacak işler listesi (Roadmap) UI'da belirir ve tamamlandırmakla işaretlenir.

2. İletişim Protokolü: WebSocket

Klasik HTTP (İstek-Cevap) modelinin aksine, sistem **Gerçek Zamanlı Çift Yönlü İletişim (Full-Duplex)** kullanır.

2.1. Neden WebSocket?

Bu sistemde bir analiz işlemi 2-5 dakika sürebilir. Klasik HTTP isteği zaman aşımına (Timeout) uğrayabilir veya kullanıcıyı "Yükleniyor..." ekranında bekletip deneyimi kötülestirebilirdi. WebSocket sayesinde:

- Bağlantı sürekli açık kalır.
- Sunucu, işlemi bitirmeyi beklemeden, işlem **devam ederken** kullanıcıya veri gönderebilir (Streaming).

2.2. Veri Akış Şeması

1. **Handshake:** Kullanıcı "Başlat" butonuna bastığında tarayıcı /ws/run adresine bağlanır.
2. **Instruction:** Kullanıcı parametreleri (Şirket adı, tarih vb.) JSON formatında sunucuya gider.
3. **Streaming Events:** Sunucu, Agent'tan gelen her logu anında UI'ya iletir:
 - {"type": "thought", "content": "..."} -> Arayüzde düşünce baloncuğu oluşturur.
 - {"type": "tool_start", "tool": "ocr"} -> "OCR Başlatılıyor" ikonunu yakar.

3. Sunucu Katmanı (Application Layer)

Web arayüzünden gelen talepleri karşılayan ve iş mantığını (Business Logic) tetikleyen katmandır.

3.1. Sunucu Çatısı: FastAPI (server.py)

Python tabanlı **FastAPI** framework'ü kullanılmıştır.

- **Asenkron (async/await):** Binlerce eşzamanlı WebSocket bağlantısını, sistemi kilitlemeden yönetebilir.
- **Performans:** Node.js ve Go ile yarışan yüksek performansa sahiptir.

3.2. İş Parçacığı Yönetimi (Threading)

WebSocket sunucusu async (asenkron) çalışırken, arka planda çalışan Agent genelde CPU yoğunluklu (OCR, PDF işleme) işler yapar. Bu yüzden Agent, ana sunucuyu kilitlememesi için **Ayrı bir Thread** içinde başlatılır.

Python

```
# Sunucu Mantığı Özeti

@app.websocket("/ws/run")

async def websocket_endpoint(ws):

    await ws.accept()

    config = await ws.receive_json()

    # Agent'ı ayrı çekirdekte/thread'de başlat
    agent_thread = threading.Thread(target=run_agent, args=(config,))
```

```
agent_thread.start()
```

```
# Agent çalışırken gelen mesajları UI'a köprüle
while True:
    event = event_queue.get()
    await ws.send_json(event)
```

Bu mimari, arayüzün asla donmamasını ve kullanıcının her zaman sistemi kontrol edebilmesini sağlar.

4. Orchestrator Agent Mimarisi (core/orchestrator_agent.py)

Agent başlatıldığı anda körüköprüne iş yapmaz. Önce "Düşünür", sonra "Planlar", en son "Yapar".

2.1. Agentic Planner (Planlama Motoru)

Agent'ın ilk işi, kullanıcının ne istedğini anlamaktır (_agentic_planner).

- **Senaryo:** Kullanıcı "X firmasının sadece ihale yasaklarını bul" dedi.
- **Karar:** Agent, Ticaret Sicil ve Web Haber modüllerini **devre dışı bırakır**.
- **Çıktı:** Dinamik bir yol haritası (Roadmap) oluşturur ve bunu UI'ya gönderir.

2.2. Tool Registry (Yetenek Havuzu)

Agent, modüler bir yapıya sahiptir. Ne yapacağını kod içinde `if/else` ile değil, "Araç Çantası"na (Registry) bakarak seçer.

- `download_ihale_rg`: İhale verisini indiren araç.
- `ocr_mnemonics`: Belgeleri okuyan araç.
- `search_news`: Haber tarayan araç.

Bu yapı sayesinde sisteme yeni bir modül (Örn: "Borsa İstanbul Modülü") eklemek sadece 2 satır kod gerektirir; Agent'ın beynine dokunmaya gerek kalmaz.

2.3. Execution Engine (Yürütmeye Motoru)

Agent, planladığı adımları `run_generator` fonksiyonu içinde işler.

- **Akıllı Cache:** Önce "Ben bu şirketi daha önce inceledim mi?" diye bakar. Varsa, saniyeler içinde eski raporu sunar.
- **Hata Toleransı (Self-Correction):** Eğer bir adım başarısız olursa (Örn: Web sitesi açılmadı), Agent bunu raporlar ve alternatif yolları dener (Örn: Google Cache'e bak).

Özet

Diyagramdaki "Orchestrator Agent" kutusu, basit bir fonksiyon değil; kendi hafızası, planlama yeteneği ve hata yönetimi olan sanal bir çalışandır. "Başlatır" oku ise bu çalışmaya dosyayı verip "İşini yapmaya başla" talimatının verildiği anıdır.

5. Bölüm: Veri Toplama ve İşleme Hatları (Ultra Deep Dive)

Bu rapor, sistemin dış dünyaya açılan kapıları olan veri toplama modüllerinin **kaynak kodu seviyesindeki** çalışma mantığını, kullanılan algoritmaların **Matematiksel/Mantıksal temellerini** ve **İstisna Yönetimi (Exception Handling)** stratejilerini enince detayına kadar açıklar.

1. Akış: İhale Yasaklılık ve Risk Analizi

Teknik Hedef: Resmi ve yapısal olmayan (Unstructured) PDF verisini, sorgulanabilir (Queryable) JSON/TXT formatına dönüştürmek.

[Phase 1] Veri Kaynağı ve İndirme Motoru

Dosya: tender_ban_module/download_tender_ban_pdfs.py

- **İndirme Algoritması (Date-Based Partitioning):** Sistem, verilen tarih aralığını (Start -> End) gün gün parçalar (timedelta(days=1)).
- **Thread Pool:** concurrent.futures.ThreadPoolExecutor(max_workers=30) kullanılarak her gün ayrı bir thread'de, aynı anda taranır. Bu, G/C darboğazını (I/O Bottleneck) aşmak için yapılmıştır.
- **Kaynak Kod Detayı (Feature Flag):** Kod içinde *Brute-force denemesi kullanıcı isteğiyle kapatıldı* şeklinde bir yorum satırı ve logic switch bulunmaktadır. Bu, Resmi Gazete sunucularını yormamak için **bilinçli olarak** kapatılmış bir özelliktir. Sadece HTML içinde açıkça linki verilen (`href.endswith(".pdf")`) dosyalar indirilir.

[Phase 2] OCR ve Görüntü İşleme (The "Race" Architecture)

Dosya: ocr_module/tender_ban_ocr.py

- **Tasarım Deseni:** Parallel Race to First Result
- Standart bir döngü (for pdf in files) yerine, işlemci çekirdeği kadar Worker Process (multiprocessing.Pool) başlatılır.
- **IPC (Inter-Process Communication):** multiprocessing.Manager().Event() nesnesi (stop_event) tüm işçiler arasında paylaştırılır.

Algoritma:

Python

```
if is_match:  
    stop_event.set() # Bayrağı kaldır  
  
    pool.terminate() # Diğer tüm işçileri ÖLDÜR
```

Bu yaklaşım, aranan şirketin dosyanın başında bulunması durumunda, kalan 1000 sayfanın taranmasını engelleyerek $O(1)$ - $O(N)$ arasında değişen ama ortalama çok hızlı bir performans sağlar.

- **Fuzzy Matching (Bulanık Mantık):** OCR hatalarını (ZETA -> 2ETA) tolere etmek için Levenshtein Distance tabanlı fuzzywuzzy kütüphanesi kullanılır.
- **Eşik Değer (Threshold):** ≥ 85 . (Deneysel olarak belirlenmiş optimum değer).
- **Preprocessing:** clean_name fonksiyonu ile şirket ismindeki LTD, ŞTİ, SANAYİ gibi gürültü kelimeleri regex ile temizlenir.

[Phase 3] LLM Analizi (Asenkron Parçalama)

Dosya: llm_ocr_analyzers/analyze_tender_ban.py

- **Prompt Mühendisliği:** Modelin "dikkat penceresi" (Context Window) dağılmasının diye tek bir devasa prompt yerine 3 parçalı (**Chunking**) yapı kurulmuştur.
- **Parallel Request:** Python asyncio.create_task ve asyncio.gather kullanılarak bu 3 parça (İdare, Yasaklı, Detay) aynı anda API'ye gönderilir. Toplam bekleme süresi **3x değil, Max(Task1, Task2, Task3)** kadardır.

2. Akış: Ticaret Sicil ve Headless Browser

Teknik Hedef: Güvenlik önlemleriyle (CAPTCHA, Login) korunan dinamik bir SPA (Single Page Application) sitesinden veri kazmak.

[Phase 1] Otomasyon ve Ağ Müdahalesi

Dosya: trade_registry_module/download_trade_registry_pdfs.py

- **Teknik: CDP (Chrome DevTools Protocol) Bağlantısı:** Script, launch() komutu ile kendi tarayıcısını açmaz. Bunun yerine connect_over_cdp("http://localhost:9222") ile **9222 portunda** halihazırda çalışan (ve kullanıcının E-Devlet girişi yaptığı) Chrome instance'ına kanca atar.
- **Network Interception (Ağ Dinleme):** Playwright'in page.on("response") dinleyicisi bir "Sniffer" gibi çalışır. content-type: application/pdf yanıtı görüldüğünde, URL yakalanır ve işlem iptal edilip indirme curl ile yapılır. Bu, tarayıcının PDF görüntüleyicisini (Viewer) bypass eder ve hızı artırır.

[Phase 2] Hibrit Döküman Analizi (Fallback Chain)

Dosya: ocr_module/trade_registry_ocr.py

- **Zincirleme Hata Toleransı (Chain of Responsibility):** Kod, hiyerarşik bir try-except bloğu içinde çalışır:
 1. **Düzey 1 (High Fidelity): IBM Docing:** Belgeyi piksellerine değil, mantıksal yapısına (Layout) göre ayırtırır. Tabloları **Markdown Table** formatında çıkarır.
 2. **Düzey 2 (Fast Text): PyMuPDF (Fitz):** Eğer Docing çökerse, doğrudan PDF'in içindeki metin katmanını (Text Layer) okur. Çok hızlıdır ama tablo yapısını bozar.
 3. **Düzey 3 (Last Resort): Tesseract OCR:** Eğer PDF bir "resim" ise (Text Layer yoksa), görüntü işleme ile karakter tanıma yapılır. En yavaş yöntemdir.

3. Akış: Web Haber ve Çok Katmanlı İstihbarat

Teknik Hedef: İnternet üzerindeki gürültülü veriden (Reklam, Menü, Spam) saf bilgi damıtmak.

[Phase 1] Arama Stratejisi ve Veri Temizliği

Dosya: web_search/WebSearch.py

- **Şirket İsmi Normalizasyonu (clean_company_name):**
 - **Stopword Removal:** search_term_stopwords listesinde imalat, taahhüt, enerji gibi kelimeler bulunur.
 - **Logic:** "MARDİVA İNŞAAT ENERJİ SANAYİ" gibi bir isimden ENERJİ ve SANAYİ atılır, geriye MARDİVA İNŞAAT kalır. Bu temiz isim arama motoruna gönderilir.
- **Domain Filtering:** _first_token_in_url fonksiyonu: Eğer bulunan URL mardivainsaat.com ise (şirketin kendi token'ı URL'de geçiyorsa), bu sonuç elenir. Amaç sadece **objektif dış kaynakları** bulmaktır.

[Phase 2] LLM Duygu Analizi (Sentiment Pipeline)

- **Prompt:** LLAMA_STRICT_JSON_PROMPT
- **JSON Schema Enforcement:** LLM'den serbest metin yerine, kod ile parse edilebilir katı bir JSON istenir:

JSON

```
{  
  "is_relevant": true,  
  "sentiment": "NEGATIVE",  
  "reason": "..."  
}
```

- **Neden Sonuç İlişkisi:** LLM sadece "Negatif" deyip geçmez, "Neden?" sorusunun cevabını (reason sahası) kanıt olarak sunar. Bu veri, rapordaki grafiklerin dayanağını oluşturur.

6. Bölüm: Raporlama Motoru ve Güvenlik Mimarisi (Guardrail & Reporting Engine)

6.1. Yönetici Özeti ve Teknik Kapsam

Bu bölüm, Turk3cell Ajan Platformu'nun çıktı üretim (reporting_module) ve güvenlik denetim (guardrail) süreçlerini atomik seviyede analiz eder.

Büyük Dil Modelleri (LLM), doğaları gereği stokastik (olasılıksal) yapılardır. Bu durum, modellerin zaman zaman "halüsinsiyon" veya "toksik içerik" üretmesine yol açabilir. Geliştirilen mimari, bu olasılıksal riskleri minimize etmek için **Deterministik Raporlama** ve **Intercepting Filter (Araya Giren Filtre)** desenlerini kullanır. Hedef, kurumsal kimlik standartlarına (CI/CD) uygun vektörel çıktılar üretmek ve bu çıktıları son kullanıcıya sunmadan önce katı bir güvenlik protokolünden geçirmektedir.

6.2. Modül 1: Dinamik PDF Motoru (reporting_module)

Teknik Hedef: Farklı ajanlardan gelen asenkron veri akışını senkronize etmek ve ReportLab kütüphanesi üzerinde çalışan, görsel açıdan zenginleştirilmiş bir PDF oluşturmaktır.

6.2.1. Layout & Styling Engine (Görsel İşleme Çekirdeği)

Raporun görsel iskeleti, `generate_risk_report.py` dosyası içerisinde tanımlanan "Style Matrix" ile yönetilir. Kurumsal kimlik, kod içinde sabitlenmiş (Hard-Coded) renk uzayları ile korunur:

- **Renk Paleti (HEX Codes):**
 - NAVY_BLUE (#0E2B4B): Ana Başlıklar, Header ve Footer çizgileri.
 - Turk3cell_ORANGE (#F8931F): Vurgu alanları, seperatörler ve kritik uyarılar.
 - LIGHT_GREY (#F2F4F8): Tablo satırlarında okunabilirliği artırmak için "Zebra Stripe" arka planı.
 - DARK_GREY (#333333): Gövde metni (Body Text) standart rengi.
- **Tipografi ve Font Yönetimi (pdfmetrics):**

Standart Helvetica fontunun Türkçe karakterlerde (ş, ğ, ī) yarattığı bozulmayı (mojibake sorunu) önlemek amacıyla, sistem işletim sistemi seviyesinden TrueType font yüklemesi yapar:

 - Python

```
font_path = "/System/Library/Fonts/Supplemental/Arial.ttf"
```

```
pdfmetrics.registerFont(TTFont('Arial', font_path))
```

-
-

6.2.2. "Lazy" Tablo Oluşturma Algoritması

Metin akışı sırasında tablolar statik şablonlar yerine, veri içeriğine göre dinamik olarak oluşturulur. Sistem, satır bazlı bir okuma (stream reading) yaparak tablo yapılarını algılar:

1. **Regex Detection:** Her satır if ":" in line mantıksal operatörü ile taranır. Anahtar-Değer çifti yakalandığında tablo modu aktifleşir.
2. **Buffering:** Eşleşen satırlar anlık işlenmez, `table_buffer` listesinde belleğe alınır.
3. **Flush Trigger:** Boş bir satır veya yeni bir başlık algılandığında `flush_table()` fonksiyonu tetiklenir.
4. **Style Application:** Tablo nesnesine otomatik stil matrisi uygulanır:
5. Python

```
('ROWBACKGROUNDS', (0,0), (-1,-1), [colors.white, LIGHT_GREY]), # Zebra Deseni
```

```
('GRID', (0,0), (-1,-1), 0.5, colors.lightgrey) # Hücre Sınırları
```

- 6.
- 7.

6.2.3. Asenkron Görsel İşleme ve Caching

Rapor içeresine eklenecek şirket logoları ve grafikler, I/O darboğazını (bottleneck) engellemek amacıyla ana süreçten izole edilmiştir.

- **Thread Pool:** Resim indirme işlemleri `concurrent.futures.ThreadPoolExecutor(max_workers=8)` yapılandırması ile paralel havuzda işlenir.
- **In-Memory Caching:** Ağ trafiğini optimize etmek için, indirilen görseller `image_cache = {}` sözlüğünde (URL -> BytesIO) saklanır. Böylece aynı logo rapor içinde tekrarlandığında sunucudan yeniden indirilmez.

6.3. Modül 2: Güvenlik ve Denetim Katmanı (Guardrail Architecture)

Guardrail sistemi, `core/orchestrator_agent.py` üzerinde çalışan merkezi orkestrasyon biriminin **Post-Processing (Son İşlem)** fazında konumlandırılmıştır.

6.3.1. Hibrit Metin Madenciliği (Hybrid Text Extraction Strategy)

Sistem, PDF içeriğini analiz edebilmek için "Fail-Safe" (Hata Toleranslı) bir strateji izler:

1. **Optimistic Path (Sidecar TXT):** Rapor oluşturulurken, PDF ile eş zamanlı olarak ham metin verisini içeren bir .txt dosyası kaydedilir. Sistem öncelikle bu dosyayı okur; bu yöntem I/O maliyetini minimuma indirir.
2. **Pessimistic Path (Binary PDF Parsing):** Eğer yan dosya mevcut değilse, `pypdf` kütüphanesi devreye girer ve PDF dosyası sayfa bazlı (page-by-page) taranarak içerik belleğe alınır.

6.3.2. LLM Entegrasyonu ve JSON Enforcement

Sistemin bilişsel katmanı olan `core/llm_client.py` dosyasında, "Provider-Agnostic" (Sağlayıcı Bağımsız) güvenlik önlemleri uygulanır:

- **JSON Zorlama:**
 - **OpenAI:** `response_format={"type": "json_object"}` parametresi ile.
 - **Gemini:** `generation_config={"response_mime_type": "application/json"}` ayarı ile modelin sohbet tabanlı metin üretmesi engellenir.
- **Robustness Loop (Hata Toleransı):**
`generate_risk_report.py` içindeki `robust_llm_json` fonksiyonu, API'den dönen veriyi temizler. Eğer model veriyi Markdown blokları (json) içine hapsederse, `.split("json")[1]` metodu ile temizlik yapılır ve `json.loads` `retry`` mekanizmasıyla (`max_retries=2`) garanti altına alınır.

6.3.3. İş Mantığı ve Veri Hijyeni

Sistemin "Hatalı Pozitif" üretmesini engelleyen ve veri güvenliğini sağlayan son katman kontrolleri:

- **Domain-Specific Override (İhale İptal Mantığı):**
Veritabanında yasak kararı olsa bile, dosya sisteminde cancellation_verdict.txt (İptal Kararı) dosyası tespit edilirse, raporun sonucu dinamik olarak değiştirilir:
- Python

if "YASAK KALDIRILDI" in verdict:

```
return f"CANCELLATION_FOUND: {verdict}"
```

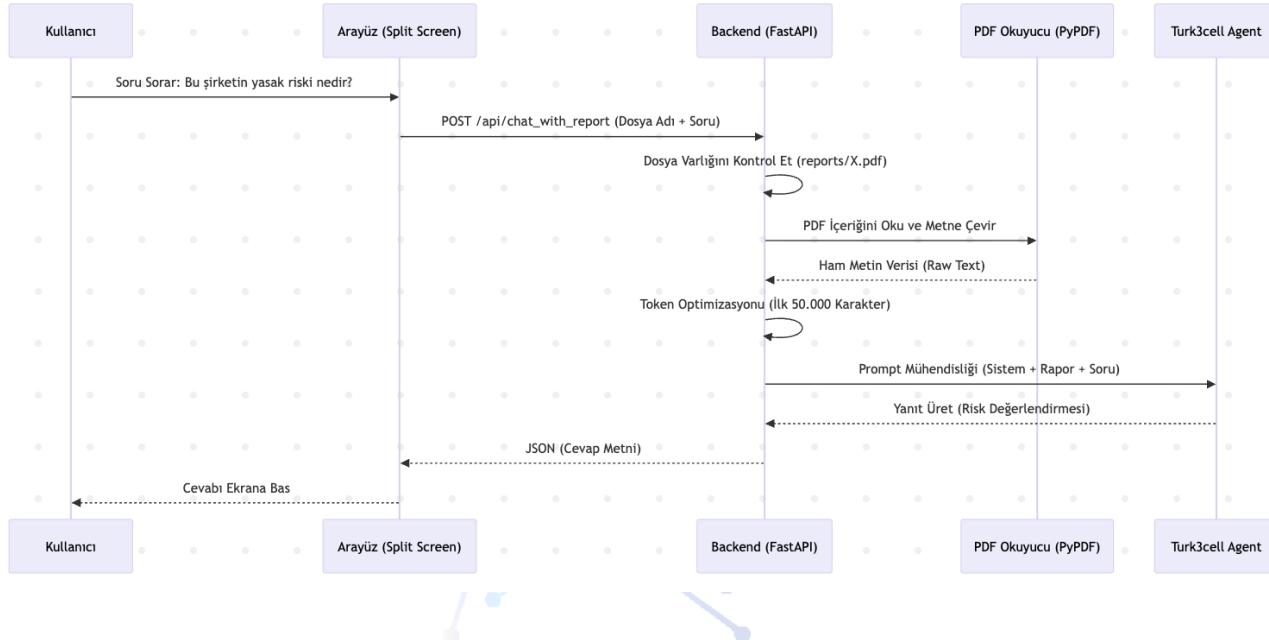
-
-
- **Artifact Sanitization (Dosya Temizliği):**
Rapor teslim edildikten sonra, shutil.rmtree kullanılarak trade_registry_module/downloaded_pdfs (Ticari sırlar) ve ocr_results (Ham metinler) klasörleri fiziksel olarak imha edilir.
- **Filename Sanitization:**
sanitize_filename fonksiyonu ile Türkçe karakterler (ı->i, ş->s) bir haritalama (mapping) sözlüğü ile ASCII formatına dönüştürülür, böylece işletim sistemleri arası dosya ismi uyumsuzlukları giderilir.

6.4. Gelecek Vizyonu (Future Scope)

Mevcut mimari metinsel analize odaklanmış olup, sonraki fazlarda PDF içindeki grafiklerin **Vision Transformers (ViT)** modelleri ile taraması (Multimodal Guardrail) ve KVKK kapsamında kişisel verilerin (TCKN, Telefon) otomatik maskelenmesi (PII Redaction) planlanmaktadır.

7. Bölüm: RAG Tabanlı Etkileşimli Analiz Modülü (Interactive Intelligence)

(Rapor oluşturduktan sonra sohbet ettiğimiz yer.)



7.1. Yönetici Özeti ve Paradigma Değişimi

Geleneksel raporlama sistemleri, "Statik Tüketim" (Static Consumption) prensibine dayanır; kullanıcı PDF'i okur ve süreç biter. Turk3cell Ajan Platformu, bu tek yönlü akışı yıkarak "Dinamik Sorgulama" (Dynamic Interrogation) paradigmmasına geçiş yapmıştır.

Bu modül, **Retrieval-Augmented Generation (RAG)** mimarisinin hafifletilmiş (lightweight) bir varyasyonunu kullanarak, statik PDF raporlarını yaşayan, sorgulanabilir ve bağlam farkında (Context-Aware) birer bilgi tabanına dönüştürür.

7.2. Mimari Genel Bakış

Sistem, klasik "Rule-Based" chatbotların aksine, "In-Context Learning" (Bağlam İçi Öğrenme) yöntemini benimsenir. Kullanıcı bir soru yönelttiğinde, sistem sadece soruyu değil, ilgili raporun **bilgi uzayını** (knowledge space) da LLM'in çalışma belleğine (Working Memory) enjekte eder.

Veri Akış Şeması (Data Flow):

- User Query:** Kullanıcı arayüzden spesifik bir soru sorar (Örn: "Şirketin finansal risk skoru neden 70?").
- Payload Construction:** Frontend, dosya ismini ve soruyu JSON formatında paketler.

3. **Secure Retrieval:** Backend, dosya sisteminden ilgili PDF'i güvenli bir şekilde okur.
4. **Context Injection:** PDF içeriği metne dönüştürülür ve LLM'in "System Prompt"una eklenir.
5. **Inference:** Model, genel internet bilgisiyle değil, *sadece* verilen metinle (Closed-Domain) cevap üretir.

7.3. Teknik Bileşenler ve İmplementasyon

7.3.1. Backend Servis Mimarisi (server.py)

Sohbet mantığı, FastAPI üzerinde çalışan `chat_with_report` uç noktası (endpoint) ile yönetilir.

- **Endpoint:** `/api/chat_with_report`
- **Method:** POST
- **Request Model (ChatReportRequest):**
 - `filename`: Hedef raporun diskteki kimliği.
 - `message`: Kullanıcının doğal dildeki sorusu (NLQ).

7.3.2. PDF Parsing ve Veri Madenciliği

Raporlar diskte binary formatta saklandığı için, LLM'in anlayabileceği semantik yapıya dönüştürülmesi gereklidir.

- **Motor:** pypdf kütüphanesi.
- **Algoritma:** PdfReader.pages döngüsü kullanılarak, doküman sayfa sayfa taranır (`extract_text()`).
- **Güvenlik Protokolü (Path Traversal Prevention):**
Kötü niyetli kullanıcıların `.../etc/passwd` gibi yollarla sunucu dosyalarına erişmesini engellemek için, dosya yolları `os.path.join("reports", filename)` ile kilitlenmiş ve `os.path.exists` ile validasyon sağlanmıştır.

7.3.3. Prompt Mühendisliği ve Rol Atama

Sistemin zeka katmanı, LLM'e (GPT-4o) gönderilen "**System Instruction**" ile şekillendirilir. Model, "Yaratıcı Yazar" modundan çıkarılıp "Finansal Analist" moduna sokulur.

Prompt Şablonu:

Python

```
system_prompt = """
```

Rol: Sen Finansal Risk Raporlarını analiz eden kıdemli bir asistansın.

Kısıtlar:

1. Sadece sana verilen "REPORT CONTENT" içindeki bilgileri kullan.

- Raporda olmayan bir bilgi sorulursa "Bu bilgi raporda yer almıyor" de. (No Hallucination)
- Cevapların profesyonel, net ve Türkçe olsun.

.....

7.3.4. Token Optimizasyonu ve Truncation Stratejisi

Büyük Dil Modellerinin (LLM) sınırlı bir "Bağlam Penceresi" (Context Window) vardır. 100+ sayfalık raporlar bu sınırı aşabilir.

- Sorun:** Token Overflow (Bağlam Taşması).
- Çözüm:** "Head-Heavy Truncation" stratejisi.
- Algoritma:**
- Python

```
if len(text_content) > 50000:  
    text_content = text_content[:50000] # İlk 50k karakter önceliklendirilir.
```

-
- Bu stratejinin seçilme nedeni, raporların en kritik verilerinin (Yönetici Özeti, Risk Skorları, Sonuç) genellikle dokümanın ilk %20'lik diliminde bulunmasıdır.*

7.4. Kullanıcı Deneyimi (Frontend UX)

Kullanıcı etkileşimi için "Split-Screen Architecture" (Bölünmüş Ekran Mimarisi) tercih edilmiştir (`web_ui/chat_pdf.html`).

- Sol Panel (Interaction Layer):** Chatbot arayüzü. JavaScript fetch API ile asenkron iletişim sağlar.
- Sağ Panel (Reference Layer):** PDF Görüntüleyici (`<embed type="application/pdf">`). Kullanıcı, yapay zekanın cevabını okurken aynı anda kaynak dokümanı teyit edebilir (Verification).

7.5. Avantajlar ve Sınırlılıklar Analizi

Avantajlar (Pros)

- Hallucination-Free (Halüsinsiyonsuz):** Modelin bilgi uzayı sadece PDF içeriği ile sınırlandığı için, uydurma cevap verme riski minimize edilmiştir.
- Low-Latency (Düşük Gecikme):** Karmaşık Vektör Veritabanı (Vector DB) ve Embedding işlemleri (Pinecone/ChromaDB) kullanılmadığı için, cevap süresi milisaniyeler mertebesindedir.

3. **Contextual Integrity:** Kullanıcı "CTRL+F" ile kelime aramak yerine, "Şirketin ana riski nedir?" gibi anlamsal sorular sorabilir.

Mevcut Sınırlılıklar ve Gelecek Vizyonu (Cons & Future Scope)

- **Bağlam Limiti:** 50.000 karakterden uzun raporlarda, dokümanın son sayfaları analize dahil edilemeyebilir. Gelecek versiyonda "Semantic Search & Chunking" (RAG Level 2) mimarisine geçilerek bu sorun aşılacaktır.
- **Multimodal Körlük:** Şu anki sistem sadece metin tabanlıdır; tablolardaki sayısal veriler veya grafikler tam olarak analiz edilememektedir. Gelecek fazda "Vision-Language Models" (VLM) entegrasyonu planlanmaktadır.

8. Bölüm: 7/24 Asenkron Gözetleme Mimarisi (Watchtower Daemon)

8.1. Yönetici Özeti ve Operasyonel Vizyon

Geleneksel istihbarat sistemleri, kullanıcı talebine dayalı (**On-Demand**) çalışır. Ancak finansal ve hukuksal riskler mesai saatleri dışında veya kullanıcının sisteme olmadığı anlarda da gelişebilir.

Turk3cell Ajan Platformu bünyesindeki "**Watchtower**" (**Gözetleme Kulesi**) modülü, "Ayarla ve Unut" (Set and Forget) prensibiyle çalışan, insan müdahalesi gerektirmeyen otonom bir **Sentinel** (**Nöbetçi**) servisidir. Bu modül, hedef şirketleri 7/24 arka planda tarayarak; ihale yasakları, konkordato ilanları veya negatif medya yansımalarını (Negative Sentiment) anlık olarak tespit eder ve "**Erken Uyarı Sistemi**" (Early Warning System) olarak görev yapar.

8.2. Mimari Tasarım: Event Loop vs. Cron Job

Sistemin mimari tasarımda, harici ve ağır yönetim maliyeti olan "Cron Job" veya "Celery Worker" yapıları yerine, Python'un modern **asyncio** kütüphanesine dayalı **Application-Level Event Loop** (Uygulama Seviyesi Olay Döngüsü) tercih edilmiştir.

Neden Bu Mimari Seçildi?

1. **Hafif Sıklet (Lightweight):** Harici bir Redis kuyruğuna veya Scheduler servisine ihtiyaç duymaz. Tek bir konteyner (Docker) içinde hem web sunucusu hem de izleme servisi çalışabilir.
2. **Stateful Operation:** Uygulama belleğine (RAM) doğrudan erişimi olduğu için veritabanı bağlantı havuzunu (Connection Pool) ortak kullanır, kaynak israfını önler.

Akış Diyagramı (Process Flow)

1. **Initialization:** Sunucu başlar (Startup Event).
2. **Task Injection:** İzleme görevi (`monitor_companies`) ana döngüye "Non-Blocking" olarak enjekte edilir.
3. **Polling Cycle:**
 - Veritabanından "Aktif" şirketleri çek.
 - Her şirket için asenkron ajanları (Resmi Gazete, Haberler) tetikle.
 - Risk skoru hesapla.
 - Risk varsa `alerts` tablosunu güncelle.
4. **Sleep:** Konfigüre edilen süre (örn: 3600 sn) kadar `await asyncio.sleep()` moduna geç.

8.3. Teknik İmplementasyon Detayları

8.3.1. Servis Entegrasyonu ve Yaşam Döngüsü (`server.py`)

İzleme döngüsü, FastAPI'nin yaşam döngüsü kancaları (Lifecycle Hooks) kullanılarak yönetilir. `@app.on_event("startup")` dekoratörü, sunucu ayağa kalktığı anda izleme thread'ini başlatır.

Python

```
# Background Task Management  
@app.on_event("startup")  
  
async def start_watchtower_daemon():
```

....

Sunucu başlatıldığında Watchtower servisini

**YARININ
TEKNOLOJİ
LİDERLERİ**

ana Event Loop'a 'Fire-and-Forget' task olarak ekler.

....

```
logger.info("Watchtower Daemon: ONLINE")
```

```
asyncio.create_task(monitoring_service.start_periodic_check())
```

8.3.2. Servis Mantığı (`monitoring_service.py`)

MonitoringService sınıfı, "Polling" (Periyodik Sorgulama) mantığını yönetir. Sistem kaynaklarını tüketmemek için "Jitter" (Rastgele Gecikme) stratejisi uygulanabilir, böylece tüm şirketler aynı milisaniyede taranarak API limitlerini zorlamaz.

- **Döngü Yönetimi:** while True: döngüsü içinde, hata durumunda servisin çökmesini engelleyen try-except blokları ile sarmalanan(awaitable) fonksiyonlar çalıştırılır.
- **Durum Güncellemesi:** Her tarama sonrası, hedef şirketin last_check (Son Kontrol Zamanı) damgası güncellenir. Bu, "Stale Data" (Bayat Veri) oluşumunu engeller.

8.3.3. Veritabanı Şeması ve Veri Yapısı

İzleme verileri, ilişkisel bütünlüğü koruyan ancak NoSQL esnekliği sağlayan hibrit bir yapıda saklanır.

| Sütun Adı | Veri Tipi | Teknik Açıklama |
|-----------------|--------------|--|
| id | UUID | Kaydın benzersiz kimliği (Primary Key). |
| user_id | ForeignKey | Şirketi izlemeye alan kullanıcı referansı. |
| company_name | VARCHAR(255) | İzlenen hedef varlık (Entity). |
| status | ENUM | İzleme durumu: 'active', 'paused', 'archived'. |
| check_frequency | INTEGER | Tarama sıklığı (Saniye cinsinden). Varsayılan: 3600. |
| last_check | DATETIME | Son başarılı tarama zamanı (UTC). |
| alerts | JSONB | Tespit edilen risklerin yapılandırılmış listesi. |

Örnek alerts JSON Yapısı:

JSON

[

{

"type": "BAN_DETECTED",

"source": "Resmi Gazete",

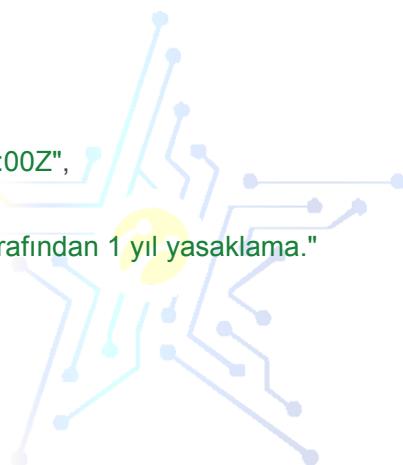
"severity": "CRITICAL",

"timestamp": "2024-02-14T03:00:00Z",

"details": "Kamu İhale Kurumu tarafından 1 yıl yasaklama."

}

]



8.4. Kullanım Senaryoları (Use Cases)

Senaryo 1: "Silent Killer" (İhale Yasağı Tespiti)

Sistem her gece 03:00'te Resmi Gazete modülünü tetikler. İzleme listesindeki "X İnşaat A.Ş." için yeni bir yasak kararı yayınlandığı tespit edilirse:

- Veritabanına CRITICAL seviyesinde alarm kaydı düşür.
- Kullanıcı sabah sisteme girdiğinde Dashboard'da kırmızı bir uyarı kartı görür.

Senaryo 2: "Reputation Watch" (Medya Takibi)

Şirket hakkında ulusal basında "yolsuzluk", "soruşturma" veya "iflas" gibi negatif anahtar kelimeler içeren haberler yayınlandığında, NLP modülü (Sentiment Analysis) bunu negatif skorlar ve sisteme anında bildirim olarak döşer.

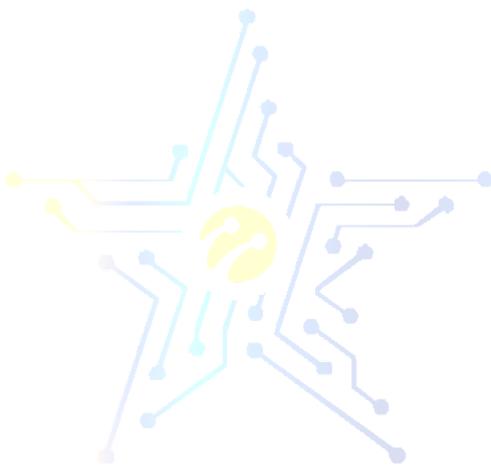
8.5. Performans, Ölçeklenebilirlik ve Kaynak Yönetimi

- Asenkron I/O (Non-Blocking):** Python'un `async/await` yapısı sayesinde, sistem 1000 şirketi tararken ağ isteklerini (Network Request) beklemek zorunda kalmaz. Bir



istek cevabını beklerken diğerine geçer. Bu, tek bir CPU çekirdeği ile binlerce eşzamanlı takibi mümkün kılar.

- **Akıllı Aralıklar (Adaptive Polling):** Sistem, her dakika sorgu atmak yerine, veri kaynağının güncellenme sıklığına göre (örn: Resmi Gazete için günde 1, Haberler için saatte 1) optimize edilmiş aralıklarla çalışır.



**YARININ
TEKNOLOJİ
LİDERLERİ**