Game Patch Notes Intelligence API (GPNAI) - Kullanıcı El Kitabı

Sürüm: 3.0 (Faz 3 Entegrasyonu Tamamlanmış)

Tarih: 25 Ekim 2025

İçindekiler:

---

1. Giriş: GPNAI Nedir ve Hangi Sorunu Çözer?

Game Patch Notes Intelligence API (GPNAI), çeşitli oyun geliştiricileri tarafından düzensiz formatlarda yayınlanan **yama notu metinlerini** otomatik olarak analiz eden, önemli değişiklikleri (dengeleme, yeni içerik vb.) ayırt eden ve bu bilgileri **standartlaştırılmış, makine tarafından okunabilir JSON formatında** sunan bulut tabanlı bir API hizmetidir.

Temel Sorun: Oyun dünyasında (e-spor, içerik üretimi, haber siteleri) yama notlarını takip etmek kritik öneme sahiptir. Ancak bu notlar genellikle yapılandırılmamış metinlerdir ve her oyun için farklı formatta yayınlanır. Bu bilgileri manuel olarak işlemek; zaman alıcı, maliyetli ve hataya açıktır.

GPNAI'nin Çözümü: GPNAI, bu süreci **tamamen otomatize eder**. Veriyi çeker, yapay zeka (Google Gemini) ile analiz eder, standart bir JSON formatına dönüştürür ve güvenilir bir API aracılığıyla sunar. Bu sayede kullanıcılar, manuel çaba harcamadan en güncel meta verilerine anında erişebilirler.

---

2. Sistem Mimarisi: Büyük Resim

GPNAI, modern bulut platformlarının (PaaS) prensiplerine uygun olarak **"Stateless" (Durumsuz)** bir mimari üzerine kurulmuştur. Bu, sistemin her bir parçasının (API sunucusu, veri işleyici) birbirinden bağımsız çalıştığı ve veriyi paylaşılan, harici bir depolama alanında (Cloudflare R2) sakladığı anlamına gelir.

Ana Bileşenler:

* **Veri İşleyici (`patch-scraper` & `patch-health-check`):** Render üzerinde çalışan iki ayrı `Cron Job` (Zamanlanmış Görev). `scrape.py` betiğini farklı modlarda çalıştırırlar.
    * `patch-scraper`: Belirlenen aralıklarla (örn: 4 saatte bir) oyun verilerini çeker, analiz eder ve sonucu Cloudflare R2'ye yazar.
    * `patch-health-check`: Daha seyrek (örn: günde bir kez) çalışarak veri kaynaklarının (web siteleri) yapısının bozulup bozulmadığını kontrol eder ve Slack'e uyarı gönderir.
* **API Sunucusu (`game-patch-api`):** Render üzerinde çalışan bir `Web Servisi`. FastAPI ile yazılmıştır. Müşterilerden gelen API isteklerini alır, Cloudflare R2'den ilgili veriyi okur ve JSON olarak yanıtlar.
* **Yapay Zeka (Google Gemini):** Veri İşleyici tarafından çağrılan harici bir servis. Ham metni analiz edip yapılandırılmış JSON'a dönüştürür.
* **Depolama (Cloudflare R2):** Analiz edilmiş JSON verilerinin (`_latest.json`) ve veri bütünlüğünü

kontrol eden hash değerlerinin (`_latest.hash`) saklandığı S3 uyumlu harici depolama alanı.

* **Bildirim (Slack):** Sistemde kritik bir hata oluştuğunda (örn: analiz başarısız, scraper bozuldu) anında bildirim gönderilen platform.

---

3. Klasör ve Dosya Yapısı Ağacı

Projenin temel yapısı aşağıdaki gibidir:

```
/game-patch-api
|
├── .github/                # GitHub Actions vb. için (şu an boş olabilir)
├── venv/                   # Yerel Python sanal ortamı (Git'e gönderilmez)
|
├── main.py                 # FastAPI Web API Sunucusu
├── scrape.py               # Ana Veri İşleyici Motoru (Cron Job'lar tarafından çalıştırılır)
├── utils.py                # Gemini AI ile iletişim ve analiz mantığı
├── scrapers.py             # Her oyun için özel veri çekme (scraping) fonksiyonları
|
├── sources.yaml            # Oyunların ve veri kaynaklarının yapılandırma dosyası
├── render.yaml             # Render.com için altyapı ve dağıtım planı
├── requirements.txt        # Projenin Python bağımlılıkları listesi
|
```

```
├── .env                    # Yerel geliştirme için gizli anahtarlar (Git'e gönderilmez)
├── .gitignore              # Git tarafından takip edilmeyecek dosya/klasörler
│
└── scraper.log             # Yerel testler sırasında oluşturulan log dosyası (Git'e gönderilmez)
```

---

4. Bileşenlerin Detaylı Açıklaması (Dosyalar ve Görevleri)

* **`main.py`**:

    * **Teknoloji:** FastAPI, Boto3, functools (lru_cache).
    * **Görev:** Müşteriye dönük API sunucusudur. Dış dünyadan gelen istekleri karşılar.
    * **İşleyiş:**
        * `/` endpoint'i: Temel API bilgisi döndürür.
        * `/health` endpoint'i: Sistemin "canlı" olup olmadığını kontrol etmek için basit bir yanıt verir (Öneri 4.2).
        * `/patches` endpoint'i:
            * Gelen isteğin `X-API-Key` başlığını `API_KEY` ortam değişkeniyle doğrular (Öneri 3.2).
            * `game` parametresini alır. Eksikse 400 hatası verir.
            * Oyun adına göre R2'deki dosya adını (`oyunadı_latest.json`) belirler.
            * `fetch_from_s3` fonksiyonunu çağırır. Bu fonksiyon, `lru_cache` sayesinde sonucu hafızada (cache) tutarak R2'ye giden istekleri azaltır (Öneri 3.3).
            * `fetch_from_s3`, `boto3` kullanarak R2'den dosyayı okur. Dosya yoksa `None` döndürür.

* Sonuç `None` ise 404 hatası verir. Veri varsa JSON olarak müşteriye döndürür.

* **`scrape.py`**:

* **Teknoloji:** Python, YAML, Requests, Boto3, concurrent.futures, hashlib, sys.

* **Görev:** Arka planda çalışan ana veri işleme motorudur. Render'daki Cron Job'lar tarafından `--run=scrape` veya `--run=health` argümanlarıyla çalıştırılır.

* **İşleyiş (`--run=scrape` modu):**

1. `sources.yaml` dosyasını okur.
2. `ThreadPoolExecutor` kullanarak `scrapers.py` içindeki tüm oyunların `fetch_*` fonksiyonlarını **paralel** olarak çalıştırır (Öneri 2.4).
3. Her bir çekilen ham verinin `sha256` hash'ini hesaplar.
4. `get_hash_from_s3` ile R2'deki son kaydedilen hash'i okur.
5. Hash'ler aynıysa, o oyunu atlar (`SKIPPED`) (Öneri 1.3).
6. Hash farklıysa veya ilk kez çalışıyorsa, ham veriyi `utils.py`'deki `analyze_with_gemini`'ye **sırayla** (API limitini korumak için aralarda bekleme yaparak) gönderir.
7. Başarılı analiz sonucu (`result`) gelirse, `save_json_to_s3` ile JSON'u R2'ye (`_latest.json`), `save_hash_to_s3` ile de yeni hash'i R2'ye (`_latest.hash`) kaydeder.
8. Herhangi bir kritik hata (API hatası, S3 hatası) olursa `send_alert` ile Slack'e bildirim gönderir (Öneri 4.1).

* **İşleyiş (`--run=health` modu):**

1. `sources.yaml` dosyasını okur.
2. `scrapers.py` içindeki her `fetch_*` fonksiyonunu çalıştırır.
3. Eğer bir fonksiyon `None` döndürürse (veri çekemezse), bunun bir scraper bozulması olabileceğini varsayar.
4. Bozuk olduğu düşünülen scraper'ların listesini `send_alert` ile Slack'e gönderir (Öneri 1.1).

* **`utils.py`**:

* **Teknoloji:** Google GenAI Kütüphanesi.

* **Görev:** Gemini AI ile iletişimi yönetir.

* **İşleyiş:** `analyze_with_gemini` fonksiyonu:

* `GEMINI_API_KEY` ortam değişkenini okur.

* Verilen ham metin ve oyun adı ile prompt'u oluşturur (JSON formatı ve önemli değişiklikleri çıkarma talimatı içerir).

* Gemini API'ye isteği gönderir (`response_mime_type="application/json"` zorlamasıyla).

* Gelen yanıtı `json.loads` ile ayrıştırmaya çalışır. Başarısız olursa veya API hatası alırsa, hatayı loglar, `send_alert` ile Slack'e bildirir ve `None` döndürür.

* **`scrapers.py`**:

* **Teknoloji:** Requests, BeautifulSoup4, lxml.

* **Görev:** Her bir oyun için spesifik veri çekme mantığını içerir.

* **İşleyiş:** Her `fetch_*` fonksiyonu, `scrape.py`'den gelen `session` objesini kullanarak ilgili web sitesine veya RSS akışına istek atar, HTML/XML'i ayrıştırır (`html.parser` veya `lxml-xml`), ilgili metin içeriğini bulur, temizler ve string olarak döndürür. Hata durumunda `None` döndürür.

* **`sources.yaml`**:

* **Teknoloji:** YAML.

* **Görev:** Desteklenen oyunları ve bunlara karşılık gelen `scrapers.py` fonksiyon adlarını listeler. Yeni oyun eklemeyi kolaylaştırır (Öneri 1.2).

* **İşleyiş:** `scrape.py` tarafından okunarak hangi oyunların işleneceğini ve hangi fonksiyonların çağrılacağını belirler.

* **`render.yaml`**:

    * **Teknoloji:** YAML (Render Blueprint formatı).

    * **Görev:** Render.com için altyapı tanımıdır. Üç servisi (`web`, `cron`, `cron`) ve bunların yapılandırmalarını (build komutu, start komutu, zamanlama, ortam değişkenleri) tanımlar.

    * **İşleyiş:** Render, GitHub'a push yapıldığında bu dosyayı okur ve altyapıyı buna göre günceller.

* **`requirements.txt`**:

    * **Görev:** Projenin çalışması için gereken tüm Python kütüphanelerini listeler. Render `buildCommand: pip install -r requirements.txt` komutuyla bu kütüphaneleri otomatik olarak kurar.

* **`.env`**:

    * **Görev:** Yerel makinede geliştirme yaparken kullanılan gizli anahtarları (API Key, S3 credentials, Slack URL, Gemini Key) içerir. **Asla GitHub'a gönderilmemelidir.**

* **`.gitignore`**:

    * **Görev:** Git'in hangi dosya ve klasörleri (örn: `venv/`, `__pycache__/`, `.env`, `scraper.log`) takip etmemesi gerektiğini belirtir.

* **`scraper.log`**:

    * **Görev:** `scrape.py` betiği yerel makinede çalıştırıldığında logların yazıldığı dosyadır. Render üzerindeki loglar için Render'ın kendi arayüzü kullanılır.

---

5. Platform Entegrasyonları

Sistem, işlevselliğini sağlamak için aşağıdaki harici platformlarla entegre çalışır:

* **GitHub (`github.com/tradevaultshop-prog/game-patch-api`)**:

    * **Rol:** Kaynak Kod Yönetimi.

    * **Entegrasyon:** Projenin tüm kodu burada barındırılır. Render'a "Blueprint" olarak bağlanmıştır. `main` dalına yapılan her `push` işlemi, Render'da otomatik dağıtımı tetikler.

    * **Ayarlar:** Render entegrasyonu için GitHub hesabınızdan Render uygulamasına izin vermeniz gerekir.

* **Render (`dashboard.render.com`)**:

    * **Rol:** Uygulama Barındırma (PaaS).

    * **Entegrasyon:** `render.yaml` dosyasını okuyarak altyapıyı (1 Web Servisi, 2 Cron Job) oluşturur ve yönetir. GitHub'dan kodu çeker, `requirements.txt` ile bağımlılıkları kurar ve `startCommand`'ları çalıştırır.

    * **Ayarlar:**

        * "Blueprint" servisi GitHub deposuna bağlanır (Dal: `main`).

        * Her üç servis (`game-patch-api`, `patch-scraper`, `patch-health-check`) için "Environment" (Ortam Değişkenleri) bölümüne gerekli API anahtarları ve gizli bilgiler (GEMINI_API_KEY, S3_*, API_KEY, SLACK_WEBHOOK_URL) eklenir.

* **Google AI Studio (`aistudio.google.com/api-keys`)**:

    * **Rol:** Yapay Zeka Analizi.

* **Entegrasyon:** `utils.py` dosyası, `google-genai` kütüphanesi aracılığıyla bu platformun API'sini kullanır.

* **Ayarlar:** Platformdan bir `API Anahtarı` (`GEMINI_API_KEY`) alınır ve bu anahtar Render'daki ilgili servislere (`patch-scraper`, `patch-health-check`) ortam değişkeni olarak eklenir. (API limitlerine takılmamak için faturalandırmanın etkinleştirilmesi önerilir).

* **Cloudflare (`dash.cloudflare.com`)**:

* **Rol:** Harici Dosya Depolama (R2).

* **Entegrasyon:** `scrape.py` (`boto3` ile) analiz edilmiş JSON ve hash dosyalarını R2'ye yazar. `main.py` (`boto3` ile) bu dosyaları R2'den okur.

* **Ayarlar:**

    * `game-patch-data` adında bir R2 Bucket oluşturulur.

    * Bu bucket için "Object Read & Write" izinli bir R2 API Token oluşturulur. Token'ın `Access Key ID` ve `Secret Access Key` değerleri alınır.

    * Bucket'ın `S3 API Endpoint URL`'si kopyalanır.

    * Bu 4 bilgi (`S3_BUCKET_NAME`, `S3_ENDPOINT_URL`, `S3_ACCESS_KEY_ID`, `S3_SECRET_ACCESS_KEY`) Render'daki ilgili servislere (`game-patch-api`, `patch-scraper`, `patch-health-check`) ortam değişkeni olarak eklenir.

    * Bucket ayarlarından "Object Versioning" (Nesne Sürümleme) özelliğinin etkinleştirilmesi önerilir (Faz 3).

* **Slack (`hooks.slack.com/services/...`)**:

* **Rol:** Anlık Hata Bildirimi.

* **Entegrasyon:** `scrape.py` dosyası, kritik hatalar oluştuğunda `send_alert` fonksiyonu aracılığıyla bu platforma bir mesaj gönderir.

* **Ayarlar:** Belirli bir Slack kanalı için "Incoming Webhook URL" oluşturulur ve bu URL, Render'daki ilgili servislere (`patch-scraper`, `patch-health-check`) `SLACK_WEBHOOK_URL` ortam değişkeni olarak eklenir.

---

6. Veri Akışı: Girdiden Çıktıya Yolculuk

1. **Girdi:** Oyunların web sitelerinden/RSS akışlarından alınan ham, yapılandırılmamış **HTML** veya **XML** metni.

2. **Temizleme:** Metin, HTML/XML etiketlerinden arındırılır ve kısaltılır.

3. **Hash Kontrolü:** Temiz metnin hash'i hesaplanır ve R2'deki son hash ile karşılaştırılır. Değişiklik yoksa işlem durur.

4. **Yapay Zeka Analizi:** Değişiklik varsa, temiz metin Google Gemini'ye gönderilir.

5. **Yapılandırma:** Gemini, metni analiz eder ve prompt'taki talimatlara göre önemli değişiklikleri içeren bir **JSON objesi** döndürür.

6. **Harici Depolama:** Bu JSON objesi R2'ye `_latest.json` olarak, yeni hash değeri de `_latest.hash` olarak kaydedilir.

7. **API Sunumu:** Müşteri API'ye istek attığında, `main.py` R2'den ilgili `_latest.json` dosyasını okur.

8. **Çıktı:** Okunan **JSON verisi** müşteriye yanıt olarak döndürülür.

---

7. Çıktı Verisi: JSON Formatı ve Yorumlanması

API'nin `/patches` endpoint'inden dönen temel JSON yapısı şöyledir:

```json
{
  "game": "Oyun Adı",
```

```
  "patch_version": "Tespit Edilen Versiyon veya 'unknown'",
  "date": "Tespit Edilen Tarih veya 'unknown'",
  "changes": [
    {
      "type": "nerf | buff | new | fix",
      "target": "Etkilenen Öğe (örn: Karakter, Silah)",
      "ability": "Etkilenen Yetenek (varsa)",
      "details": "Değişikliğin Açıklaması"
    },
    // ... başka değişiklikler ...
  ]
}
```

* **`game`, `patch_version`, `date`:** Yama notu hakkındaki temel meta verilerdir. AI bunları metinden çıkaramazsa "unknown" döner.
* **`changes`:** En önemli kısımdır. Bir dizi (array) olarak o yamadaki önemli değişiklikleri listeler.
    * `type`: Değişikliğin kategorisini belirtir.
    * `target`, `ability`: Değişikliğin neyi etkilediğini belirtir.
    * `details`: AI tarafından oluşturulan, değişikliğin özet açıklamasıdır.
* **Önemli Not - `changes: []`:** Eğer `changes` dizisi boş gelirse, bu bir hata değildir. Sistem veriyi çekmiş, analiz etmiş ancak prompt'taki kriterlere uyan (nerf, buff, new, fix) raporlanacak önemli bir değişiklik bulamamıştır.

---

8. Sistem Kullanımı (Sahip ve Müşteri Perspektifi)

* **Sistem Sahibi (Yönetici) Olarak:**

    * **Kurulum:** Sistemi kurmak için bu kılavuzdaki adımları (platform hesapları oluşturma, anahtarları alma, kodu GitHub'a yükleme, Render'ı yapılandırma) takip edersiniz.

    * **İzleme:** Render loglarını (`patch-scraper` ve `patch-health-check` için) ve Slack'teki `#gpnai-alerts` kanalını düzenli olarak kontrol edersiniz. `/health` endpoint'ini bir uptime monitörüne eklersiniz.

    * **Bakım:** Bir scraper bozulduğunda (Slack uyarısı gelince), `scrapers.py` dosyasını günceller ve GitHub'a gönderirsiniz. Yeni bir oyun eklemek için `sources.yaml` ve `scrapers.py` dosyalarını günceller ve GitHub'a gönderirsiniz.

    * **Yönetim:** API anahtarlarını (Gemini, R2, Slack, Kendi API Anahtarınız) yönetirsiniz. Müşterilere dağıtılacak `API_KEY`'i oluşturur ve güvenli bir şekilde iletirsiniz.

* **API Kullanıcısı (Müşteri) Olarak:**

    * **Erişim:** Size sağlanan `API_KEY`'i alırsınız.

    * **Kullanım:** İstediğiniz oyunun verisini almak için API endpoint'ine bir GET isteği yaparsınız. Bu isteğin başlığına (`Header`) `X-API-Key` olarak size verilen anahtarı eklemelisiniz.

        * Örnek İstek (Python `requests` ile):

            ```python

            import requests

            import os


            API_ENDPOINT = "https://game-patch-api.onrender.com/patches"

            API_KEY = "sk-gpnai-..." # Size verilen anahtar


            headers = {"X-API-Key": API_KEY}
            ```

```
params = {"game": "Valorant"}

response = requests.get(API_ENDPOINT, headers=headers, params=params)

if response.status_code == 200:
    data = response.json()
    print(json.dumps(data, indent=2, ensure_ascii=False))
else:
    print(f"Hata: {response.status_code} - {response.text}")
```

* **Yorumlama:** Dönen JSON verisini kendi uygulamanızda (web sitesi, bot, analiz aracı) işlersiniz. `changes` dizisinin boş olabileceğini göz önünde bulundurursunuz.

* **Sağlık Kontrolü:** İsteğe bağlı olarak, API'nin genel durumunu kontrol etmek için `/health` endpoint'ini çağırabilirsiniz (bu endpoint API anahtarı gerektirmez).

---

9. Bakım ve Sorun Giderme

* **Scraper Bozulması:** En sık karşılaşılacak sorundur. `patch-health-check` Slack uyarısı verdiğinde veya `patch-scraper` loglarında bir oyun için sürekli `None` döndüğünü gördüğünüzde, ilgili oyunun web sitesini kontrol edin. HTML yapısı değişmişse, `scrapers.py` dosyasındaki ilgili `fetch_*` fonksiyonunu güncelleyin ve GitHub'a gönderin.

* **API Hataları (Gemini/R2):** Slack uyarısı geldiğinde, hatanın kaynağını (Gemini kotası, R2 erişim sorunu vb.) loglardan tespit edin ve ilgili platformun (Google AI Studio, Cloudflare) ayarlarını kontrol edin. Gerekirse API anahtarlarını yenileyin ve Render ortam değişkenlerini güncelleyin.

* **API Yanıt Vermiyor:** `/health` endpoint'i yanıt vermiyorsa, Render'daki `game-patch-api` (Web) servisinin loglarını kontrol edin. Çökme varsa, nedeni araştırın (`main.py` hatası, bağımlılık sorunu vb.).

---

10. Gelecek Vizyonu ve Potansiyel Geliştirmeler

Bu kılavuzda detaylandırılan Faz 1, 2 ve 3 güncellemeleri, sağlam bir temel oluşturmuştur. Gelecekte sistem, aşağıdaki yönlerde daha da geliştirilebilir:

* **Daha Akıllı Analiz:** Geçmiş verileri (R2 versiyonlama sayesinde) kullanarak trend analizi yapma, değişikliklerin önemini puanlama ("impact score").

* **Kendini İyileştiren Promptlar:** AI çıktılarının kalitesini otomatik olarak analiz edip `utils.py` içindeki prompt'u optimize eden bir mekanizma.

* **Veritabanı Entegrasyonu:** R2 yerine PostgreSQL gibi bir veritabanı kullanarak daha karmaşık sorgular ve analizler yapma yeteneği.

* **Genişletilmiş Kapsam:** Desteklenen oyun sayısını `sources.yaml` üzerinden artırma, hatta oyun dışı veri kaynaklarını (örn: e-spor turnuva sonuçları) ekleme.

* **Kullanıcı Arayüzleri:** API'yi kullanan örnek bir web arayüzü, Discord botu veya mobil uygulama geliştirme.

Bu kılavuz, GPNAI sisteminin mevcut mimarisini, işleyişini ve kullanımını kapsamlı bir şekilde açıklamaktadır.