**AEO Competitive Intelligence Tool – Updated Build Plan (v2)**

**Revision note (28 Jul 2025):** Only the sections flagged as *gaps and risk areas* in my review have been modified. All other content remains identical to the original document you supplied.

**0. Key Revisions at a Glance**

| **Gap / Risk** | **Change implemented** |
| --- | --- |
| Async/Celery mismatch | Celery tasks call **sync wrappers**; asyncio.run encapsulates async audit loop. |
| Distributed rate‑limiting | Replaced process‑local list with **Redis token‑bucket** via aiolimiter & redis.asyncio. |
| Multilingual coverage | Added **language detection** → per‑lang spaCy / transformer NER pipelines. |
| Sentiment accuracy | Switched to **cardiffnlp/twitter‑roberta‑base‑sentiment** via 🤗 Transformers. |
| Raw LLM storage compliance | Raw payloads trimmed to minimal fields + **30‑day TTL** in Redis; metadata persisted. |
| Cost control | Two‑pass architecture: **Llama‑3‑8B‑Instruct (local)** first; premium models only for unanswered/high‑priority queries. |
| White‑labeling depth | Introduced **report\_themes** table + tenant asset bucket (S3/GCS). |
| Security / GDPR | Option A: per‑tenant Postgres schema. Option B: RLS on shared tables + row‑level tenant\_id. |
| Embedding storage | Added **pgvector** column (answer\_embedding) for semantic diffing. |
| Timeline realism | Extended from **16 → 18 weeks** (+QA & onboarding sprint). |

**1. Technology Stack (additions)**

* **Rate limiting:** aiolimiter + redis.asyncio (shared buckets).
* **Language detection:** langdetect or fasttext.
* **NER & sentiment models:** 🤗 transformers (cardiffnlp/twitter-roberta-base-sentiment, xlm-roberta-base for multilingual NER).
* **Vector search:** pgvector extension in Postgres.

**Updated requirements.txt delta**

aiolimiter==1.1.0 # distributed rate limits

redis[async]==5.0.1 # asyncio client

fasttext-wheel==0.9.2 # lang detection (or langdetect==1.0.9)

pgvector==0.2.2 # Postgres vector bindings

# sentiment / NER handled by existing transformers==4.35.2

**2. Rate Limiter (replaces AIRateLimiter)**

# app/utils/rate\_limiter.py

from aiolimiter import AsyncLimiter

import redis.asyncio as redis

import time

class RedisRateLimiter:

"""Shared token bucket across workers."""

def \_\_init\_\_(self, name: str, rate: int, period: int = 60):

self.name = f"aeo:rate:{name}"

self.rate = rate

self.period = period

self.redis = redis.from\_url(settings.REDIS\_URL)

async def acquire(self):

now = int(time.time())

key = f"{self.name}:{now // self.period}"

remaining = await self.redis.incr(key)

if remaining == 1:

await self.redis.expire(key, self.period)

if remaining > self.rate:

sleep\_for = self.period - (now % self.period)

await asyncio.sleep(sleep\_for)

BasePlatform now composes a RedisRateLimiter instead of the list‑based limiter.

**3. Multilingual Brand Detection**

# app/services/brand\_detector.py (excerpt)

from langdetect import detect

from transformers import AutoModelForTokenClassification, AutoTokenizer

from transformers import pipeline

class BrandDetector:

def \_\_init\_\_(self):

self.en\_nlp = spacy.load("en\_core\_web\_sm")

self.other\_nlp = pipeline("ner", model="xlm-roberta-base", aggregation\_strategy="simple")

self.lang\_cache = {}

def \_get\_doc(self, text: str):

lang = self.lang\_cache.get(text[:100]) or detect(text[:200])

self.lang\_cache[text[:100]] = lang

if lang == "en":

return self.en\_nlp(text), lang

return self.other\_nlp(text), lang # list of dicts

(Downstream code adapts to list‑dict NER format for non‑English.)

**4. Sentiment Analyzer**

# app/services/sentiment.py

from transformers import pipeline

sentiment\_pipe = pipeline("sentiment-analysis", model="cardiffnlp/twitter-roberta-base-sentiment")

def score\_sentiment(texts: list[str]) -> float:

if not texts: return 0.0

scores = []

for res in sentiment\_pipe(texts):

label = res["label"] # POSITIVE / NEGATIVE / NEUTRAL

score = {"POSITIVE": 1, "NEGATIVE": -1, "NEUTRAL": 0}[label]

scores.append(score)

return sum(scores)/len(scores)

BrandDetector.\_calculate\_sentiment now delegates to score\_sentiment.

**5. Compliance‑Friendly Response Storage**

# app/models/response.py (delta)

raw\_payload BYTEA -- encrypted, purged via TTL job

text\_excerpt TEXT -- first 1 000 chars, kept

answer\_embedding VECTOR(768) -- pgvector

A nightly Celery beat task deletes raw\_payload older than 30 days to satisfy OpenAI/Anthropic retention terms.

**6. Two‑Pass Cost Optimisation**

**Config**

# app/config/settings.py (delta)

LOW\_COST\_MODEL = "local-llama3-8b"

PREMIUM\_MODEL\_MAP = {"openai": "gpt-4o", "anthropic": "claude-3-opus"}

**Audit flow excerpt**

if question\_priority < 8: # low/medium

response = await local\_llama.safe\_query(question)

if brand\_detector.detect\_brands(...):

return response # good enough

# else escalate to premium models

**7. White‑Label Theming**

CREATE TABLE report\_themes (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

tenant\_id UUID NOT NULL,

name TEXT,

primary\_color VARCHAR(7),

secondary\_color VARCHAR(7),

logo\_url TEXT,

footer\_html TEXT,

created\_at TIMESTAMP DEFAULT NOW()

);

ReportGenerator accepts a theme\_id to inject colours/logos per agency.

**8. Security & Multi‑Tenancy**

* **Option A – separate schemas**: each tenant has its own schema (CREATE SCHEMA tenant\_123) with the same table set, managed via Postgres search\_path.
* **Option B – shared schema + RLS**: add tenant\_id on every row, enable ALTER TABLE ... ENABLE ROW LEVEL SECURITY; and attach a SET rls.tenant\_id at connection time.

Either way, add **bcrypt‑hashed API keys**, enforce HTTPS, and store secrets in Vault or AWS Secrets Mgr.

**9. Vector Storage for Trend Diffing**

ALTER TABLE responses ADD COLUMN answer\_embedding VECTOR(768);

Populate with OpenAIEmbeddings or sentence-transformers/all-MiniLM-L6-v2 to support semantic change tracking between audit runs.

**10. Revised Timeline**

| **Phase** | **Original** | **Revised** |
| --- | --- | --- |
| Core infrastructure | 2 wks | 2 wks |
| AI integration & rate‑limit infra | 2 wks | 2.5 wks |
| Brand detection & sentiment | 2 wks | 2.5 wks |
| Question engine | 2 wks | 2 wks |
| Audit processing | 2 wks | 2 wks |
| Reporting & theming | 2 wks | 2.5 wks |
| API & Front‑end | 2 wks | 2 wks |
| Deployment, QA & onboarding sprint | **2 wks (new)** |  |
| **Total** | 16 wks | **18 wks** |

**11. Celery Worker Wrapper (sync‑safe)**

# app/tasks/audit\_tasks.py (excerpt)

@celery\_app.task(bind=True)

def run\_audit\_task(self, audit\_config\_id: str):

db = SessionLocal()

try:

processor = AuditProcessor(db)

# ... register platforms ...

audit\_run\_id = asyncio.run(processor.run\_audit(uuid.UUID(audit\_config\_id)))

return {"status": "completed", "audit\_run\_id": str(audit\_run\_id)}

except Exception as e:

self.retry(countdown=60, max\_retries=3, exc=e)

finally:

db.close()

**12. Monitoring Additions**

* **Redis & Celery boards**: Flower + custom Prometheus metrics for rate‑limit buckets.
* **Vector usage**: track pg\_stat\_user\_indexes for pgvector performance.
* **Sentiment/NER latency**: histogram in Prometheus.

**13. Updated Success Metrics**

* **Average audit cost**: < USD 15 (medium tier) thanks to two‑pass model.
* **Multilingual accuracy (F1)**: > 0.80 across top 5 EU languages.
* **White‑label theme adoption**: 70 % of agencies customise within first month.

*(All untouched sections – code blocks, schema details, testing strategy, docker files, etc. – are identical to the original version.)*