Apify Integration PRD & Task List

Project Context

Product: LinkedIn Ads Intelligence Platform — SaaS tool for agencies to research competitor LinkedIn ads.
Tech stack: Next.js 14, Supabase (PostgreSQL), Clerk (auth), Prisma, Tailwind CSS v3, shadcn/ui, Trigger.dev
Current state: Project initialized, Prisma schema exists with Advertiser and Ad models, Clerk auth configured, database tables pushed to Supabase.

We are integrating the **Apify LinkedIn Ad Library Scraper** (Actor ID: `AdwCDVyhFcWXQF9tg`, slug: `silva95gustavo/linkedin-ad-library-scraper`) to scrape ads from LinkedIn's Ad Library. This replaces the originally planned custom Playwright scraper.

---

1. Apify Actor Details

Input Parameters

The actor accepts the following input:

```
json

{
  "startUrls": [
    "https://www.linkedin.com/ad-library/search?companyIds=2027242"
  ],
  "resultsLimit": null,
  "skipDetails": false,
  "proxy": {
    "useApifyProxy": true,
    "apifyProxyGroups": ["RESIDENTIAL"],
    "apifyProxyCountry": "US"
  }
}
```

- `startUrls` (required): Array of LinkedIn Ad Library search URLs. Each URL targets one advertiser using `companyIds=<LINKEDIN_COMPANY_ID>`.
- `resultsLimit` (optional): Max number of ads to return. Leave null/empty for all.
- `skipDetails` (optional, boolean): If true, skips fetching full ad details (less data, cheaper). We want `false`.
- Proxy: Datacenter-Automatic with US country works fine.

Run Configuration

- Timeout: 3600 seconds (default)
- Memory: 512 MB (default)

- Cost: ~~$2.19 for 547 ads from one advertiser (~~$0.004/ad)
- At MVP scale (5-10 advertisers, ~2000 ads/month): estimated $8-20/month

API Endpoint

To start a run via API:

```
POST https://api.apify.com/v2/acts/silva95gustavo~linkedin-ad-library-scraper/runs
```

To get dataset results after run completes:

```
GET https://api.apify.com/v2/datasets/{datasetId}/items?format=json
```

Authentication: Pass `token` query parameter with your Apify API token.

---

2. Apify Output Schema (Real Data)

Below is the complete field inventory based on a real scrape of 547 ads. Fields are categorized by availability across ad formats.

Universal Fields (present in ALL 9 formats)

Apify Field	Type	Example	Notes
`adId`	string	`"1145280734"`	Unique ad identifier from LinkedIn
`adLibraryUrl`	string	`"https://www.linkedin.com/ad-library/detail/1145280734"`	Direct link to ad on LinkedIn
`advertiserName`	string	`"Simplicate"`	Company name
`advertiserUrl`	string	`"https://www.linkedin.com/company/2027242"`	LinkedIn company page URL
`advertiserLogo`	string	`"https://media.licdn.com/dms/image/..."`	Logo image URL
`format`	string	`"SINGLE_IMAGE"`	One of 9 format values (see below)
`availability.start`	string (date)	`"2026-02-12"`	Ad start date (YYYY-MM-DD)
`availability.end`	string (date)	`"2026-02-17"`	Ad end date (YYYY-MM-DD)

Apify Field	Type	Example	Notes
`impressions`	string	`"1k-5k"`	Impression range as text
`paidBy`	string	`"Simplicate"`	Entity that paid — sometimes differs from advertiser (e.g. agency)
`ctas`	string[]	`["Meer informatie"]`	Call-to-action buttons. Can be empty array.
`targeting`	object	`{"language": "Nederlands", "location": "Netherlands"}`	Has `language` (optional) and `location` (optional)
`startUrl`	string	`"https://www.linkedin.com/ad-library/search?companyIds=2027242"`	The search URL used to find this ad

Common Fields (present in most but not all formats)

Apify Field	Type	Present In	Notes
`body`	string	All except FOLLOW_COMPANY	Ad copy text. Can be very long with emojis/unicode.
`clickUrl`	string	All except MESSAGE, DOCUMENT	Destination URL when ad is clicked.
`headline`	string	SINGLE_IMAGE, VIDEO, DOCUMENT, SPOTLIGHT	Short headline text.
`impressionsPerCountry`	array	SINGLE_IMAGE, CAROUSEL, VIDEO, MESSAGE, DOCUMENT, EVENT, TEXT	Array of `{country: string, impressions: string}`

Format-Specific Fields

Apify Field	Type	Format	Example
`imageUrl`	string	SINGLE_IMAGE	Single image URL
`videoUrl`	string	VIDEO	Video file URL (.mp4)
`documentUrl`	string	DOCUMENT	PDF document URL
`imageUrls`	string[]	DOCUMENT	Array of page preview image URLs
`slides`	array	CAROUSEL	Array of `{imageUrl: string}` objects

Format Values Found in Real Data

Format Value	Count (547 ads)	Has Media	Media Field(s)
`SINGLE_IMAGE`	475 (87%)	Yes	`imageUrl`
`CAROUSEL`	21	Yes	`slides[].imageUrl`
`VIDEO`	15	Yes	`videoUrl`
`MESSAGE`	13	No	— (InMail ad, no creative shown)
`TEXT`	10	No	— (text-only ad)
`DOCUMENT`	6	Yes	`documentUrl`, `imageUrls[]`
`EVENT`	5	No	— (links to LinkedIn event)
`SPOTLIGHT`	2	No	— (dynamic ad, personalized per viewer)
`FOLLOW_COMPANY`	1	No	— (dynamic follower ad)

Not seen in this dataset but may exist: `LEAD_GEN_FORM`, `CONVERSATION`. Handle these as unknown/other.

Impression Values (string ranges)

The `impressions` field is always a string. These are the possible values observed:

```
"< 1k", "<\u200a1k", "1k-5k", "5k-10k", "10k-20k",
"20k-30k", "30k-50k", "50k-100k", "100k-150k", "150k-200k"
```

Note: Some values contain a Unicode thin space (`\u200a`) before "1k". Normalize these when storing.

---

3. Prisma Schema Updates

Update the existing Prisma schema. The `Advertiser` model stays mostly the same. The `Ad` model needs significant updates to match real Apify data.

Updated Advertiser Model

```prisma
```

```
model Advertiser {
  id                String   @id @default(cuid())
  name              String
  linkedinCompanyId String   @unique          // e.g. "2027242"
  linkedinUrl       String?                  // e.g. "https://www.linkedin.com/company/2027242"
  logoUrl           String?                  // from Apify advertiserLogo field
  status            String   @default("approved") // "pending" | "approved" | "rejected"
  lastScrapedAt     DateTime?
  totalAdsFound     Int      @default(0)
  createdAt         DateTime @default(now())
  updatedAt         DateTime @updatedAt

  ads               Ad[]
  scrapeRuns        ScrapeRun[]
}
```

Updated Ad Model

prisma

```
model Ad {
  id                  String    @id @default(cuid())
  externalId          String    @unique       // Apify "adId" — LinkedIn's ad ID
  advertiserId        String
  advertiser          Advertiser @relation(fields: [advertiserId], references: [id], onDelete: Cascade)

  // Core content
  format              String                 // "SINGLE_IMAGE" | "CAROUSEL" | "VIDEO" | etc.
  bodyText            String?                // Apify "body" — ad copy
  headline            String?                // Apify "headline"
  callToAction        String?                // First element of Apify "ctas" array
  destinationUrl      String?                // Apify "clickUrl"

  // Media — primary URL for quick access
  mediaUrl            String?                // Primary image/video URL (imageUrl, videoUrl, or first slide)
  mediaData           Json?                  // Format-specific media data (slides array, imageUrls array, documentUrl, etc.)

  // Dates and performance
  startDate           DateTime?              // Apify "availability.start"
  endDate             DateTime?              // Apify "availability.end"
  impressions         String?                // Raw string like "1k-5k"

  // Targeting
  targetLanguage      String?                // Apify "targeting.language"
  targetLocation      String?                // Apify "targeting.location"

  // Metadata
  paidBy              String?                // Apify "paidBy" — may differ from advertiser
  adLibraryUrl        String?                // Direct link to LinkedIn Ad Library detail page
  impressionsPerCountry Json?                 // Array of {country, impressions} objects

  // Timestamps
  firstSeenAt         DateTime @default(now()) // When we first scraped this ad
  lastSeenAt          DateTime @default(now()) // Updated each scrape where ad still appears
  createdAt           DateTime @default(now())
  updatedAt           DateTime @updatedAt

  @@index([advertiserId])
  @@index([format])
  @@index([startDate])
  @@index([endDate])
}
```

New ScrapeRun Model (for tracking/debugging)

```
prisma
```

```
model ScrapeRun {
  id              String    @id @default(cuid())
  advertiserId    String
  advertiser      Advertiser @relation(fields: [advertiserId], references: [id], onDelete: Cascade)
  apifyRunId      String?                    // Apify's run ID for reference
  apifyDatasetId  String?                    // Apify's dataset ID
  status          String    @default("pending") // "pending" | "running" | "completed" | "failed"
  adsFound        Int       @default(0)
  adsNew          Int       @default(0)       // Ads not previously in our database
  adsUpdated      Int       @default(0)       // Ads that already existed and were refreshed
  costUsd         Float?                     // Estimated cost of this run
  errorMessage    String?
  startedAt       DateTime  @default(now())
  completedAt     DateTime?

  @@index([advertiserId])
  @@index([status])
}
```

---

4. Field Mapping: Apify → Prisma

This is the single transformation layer. All Apify field names should ONLY appear in the mapping function. The rest of the codebase only uses our Prisma field names.

typescript

```ts
// lib/apify/transform.ts

interface ApifyAd {
  adId: string;
  adLibraryUrl: string;
  advertiserName: string;
  advertiserUrl: string;
  advertiserLogo: string;
  format: string;
  body?: string;
  headline?: string;
  clickUrl?: string;
  ctas: string[];
  availability: { start: string; end: string };
  impressions: string;
  paidBy: string;
  targeting: { language?: string; location?: string };
  imageUrl?: string;
  videoUrl?: string;
  documentUrl?: string;
  imageUrls?: string[];
  slides?: Array<{ imageUrl: string }>;
  impressionsPerCountry?: Array<{ country: string; impressions: string }>;
  startUrl: string;
}

// Returns data ready for prisma.ad.upsert()
function transformApifyAd(raw: ApifyAd, advertiserId: string) {
  return {
    externalId: raw.adId,
    advertiserId,
    format: raw.format,
    bodyText: raw.body ?? null,
    headline: raw.headline ?? null,
    callToAction: raw.ctas?.[0] ?? null,
    destinationUrl: raw.clickUrl ?? null,
    mediaUrl: getMediaUrl(raw),
    mediaData: getMediaData(raw),
    startDate: raw.availability?.start ? new Date(raw.availability.start) : null,
    endDate: raw.availability?.end ? new Date(raw.availability.end) : null,
    impressions: normalizeImpressions(raw.impressions),
    targetLanguage: raw.targeting?.language ?? null,
    targetLocation: raw.targeting?.location ?? null,
    paidBy: raw.paidBy ?? null,
    adLibraryUrl: raw.adLibraryUrl ?? null,
    impressionsPerCountry: raw.impressionsPerCountry ?? null,
```

```
    lastSeenAt: new Date(),
  };
}

// Extract the primary media URL based on format
function getMediaUrl(raw: ApifyAd): string | null {
  switch (raw.format) {
    case 'SINGLE_IMAGE': return raw.imageUrl ?? null;
    case 'VIDEO': return raw.videoUrl ?? null;
    case 'CAROUSEL': return raw.slides?.[0]?.imageUrl ?? null;
    case 'DOCUMENT': return raw.imageUrls?.[0] ?? null;
    default: return null;
  }
}

// Store format-specific media as structured JSON
function getMediaData(raw: ApifyAd): object | null {
  switch (raw.format) {
    case 'CAROUSEL':
      return { slides: raw.slides ?? [] };
    case 'DOCUMENT':
      return { documentUrl: raw.documentUrl, imageUrls: raw.imageUrls ?? [] };
    case 'VIDEO':
      return { videoUrl: raw.videoUrl };
    case 'SINGLE_IMAGE':
      return { imageUrl: raw.imageUrl };
    default:
      return null;
  }
}

// Normalize Unicode thin spaces and whitespace in impression strings
function normalizeImpressions(raw: string): string {
  return raw?.replace(/[\u200a\u200b\u00a0]/g, '').trim() ?? null;
}
```

---

5. Apify Client Service

typescript

```ts
// lib/apify/client.ts

const APIFY_TOKEN = process.env.APIFY_API_TOKEN;
const ACTOR_ID = 'silva95gustavo~linkedin-ad-library-scraper';
const BASE_URL = 'https://api.apify.com/v2';

// Start a scrape run for one advertiser
async function startScrapeRun(linkedinCompanyId: string): Promise<{ runId: string }> {
  const response = await fetch(
    `${BASE_URL}/acts/${ACTOR_ID}/runs?token=${APIFY_TOKEN}`,
    {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        startUrls: [
          `https://www.linkedin.com/ad-library/search?companyIds=${linkedinCompanyId}`
        ],
        skipDetails: false,
      }),
    }
  );
  const data = await response.json();
  return { runId: data.data.id };
}

// Poll until run finishes (or use webhook — see Task 4)
async function waitForRun(runId: string): Promise<{ datasetId: string; status: string }> {
  // Poll every 10 seconds, timeout after 20 minutes
  const maxAttempts = 120;
  for (let i = 0; i < maxAttempts; i++) {
    const response = await fetch(
      `${BASE_URL}/actor-runs/${runId}?token=${APIFY_TOKEN}`
    );
    const data = await response.json();
    const status = data.data.status;

    if (status === 'SUCCEEDED') {
      return { datasetId: data.data.defaultDatasetId, status };
    }
    if (status === 'FAILED' || status === 'ABORTED' || status === 'TIMED-OUT') {
      throw new Error(`Apify run ${runId} ended with status: ${status}`);
    }
    await new Promise(resolve => setTimeout(resolve, 10_000));
  }
  throw new Error(`Apify run ${runId} timed out waiting`);
}
```

```
// Fetch all results from a completed run's dataset
async function getDatasetItems(datasetId: string): Promise<ApifyAd[]> {
  const response = await fetch(
    `${BASE_URL}/datasets/${datasetId}/items?format=json&token=${APIFY_TOKEN}`
  );
  return response.json();
}
```

---

6. Upsert Strategy

When storing ads, use `upsert` on `externalId` to handle re-scraping the same advertiser:

```typescript
```

```
async function storeAds(ads: ApifyAd[], advertiserId: string): Promise<{ new: number; updated: number }> {
  let newCount = 0;
  let updatedCount = 0;

  for (const raw of ads) {
    const data = transformApifyAd(raw, advertiserId);

    const existing = await prisma.ad.findUnique({
      where: { externalId: data.externalId },
      select: { id: true },
    });

    await prisma.ad.upsert({
      where: { externalId: data.externalId },
      create: {
        ...data,
        firstSeenAt: new Date(),
      },
      update: {
        ...data,
        // Don't overwrite firstSeenAt on update
      },
    });

    if (existing) {
      updatedCount++;
    } else {
      newCount++;
    }
  }

  return { new: newCount, updated: updatedCount };
}
```

---

7. Environment Variables

Add to `.env.local`:

```
APIFY_API_TOKEN=your_apify_api_token_here
```

Add to `.env.example`:

```
APIFY_API_TOKEN=
```

8. Task List (Ordered)

Complete these tasks in order. Each task should be a working increment — test before moving to the next.

Task 1: Update Prisma Schema

Files to modify: `prisma/schema.prisma`

1. Update the `Advertiser` model to match the schema in Section 3. Add fields: `linkedinCompanyId` (unique), `linkedinUrl`, `logoUrl`, `status`, `lastScrapedAt`, `totalAdsFound`.
2. Replace the existing `Ad` model with the schema in Section 3. Key: `externalId` must be `@unique`.
3. Add the new `ScrapeRun` model from Section 3.
4. Run `npx prisma db push` to apply changes.
5. Run `npx prisma generate` to update the Prisma client.

Verify: Open Prisma Studio (`npx prisma studio`) and confirm all three tables exist with correct columns.

Task 2: Create Apify Type Definitions

Files to create: `lib/apify/types.ts`

1. Define the `ApifyAd` interface matching the raw JSON output (see Section 4).
2. Define an `ApifyRunStatus` type: `'READY' | 'RUNNING' | 'SUCCEEDED' | 'FAILED' | 'ABORTED' | 'TIMED-OUT'`.
3. Define an `AdFormat` type with all known values: `'SINGLE_IMAGE' | 'CAROUSEL' | 'VIDEO' | 'MESSAGE' | 'TEXT' | 'DOCUMENT' | 'EVENT' | 'SPOTLIGHT' | 'FOLLOW_COMPANY'`.
4. Export all types.

Task 3: Create Transform Layer

Files to create: `lib/apify/transform.ts`

1. Implement `transformApifyAd()` exactly as shown in Section 4.
2. Implement `getMediaUrl()` — extracts the single most relevant media URL per format.
3. Implement `getMediaData()` — stores all format-specific media as a JSON object.
4. Implement `normalizeImpressions()` — strips Unicode thin spaces.
5. Export all functions.

Important: This is the ONLY file that references Apify field names. Every other file works with our Prisma types.

Task 4: Create Apify Client

Files to create: `lib/apify/client.ts`

1. Implement `startScrapeRun(linkedinCompanyId)` as shown in Section 5.
2. Implement `waitForRun(runId)` with polling (10s interval, 20min timeout).
3. Implement `getDatasetItems(datasetId)` to fetch results.
4. Implement a convenience function `scrapeAdvertiser(linkedinCompanyId)` that chains: startScrapeRun → waitForRun → getDatasetItems → return transformed ads.
5. Add proper error handling: wrap all fetch calls in try/catch, check response.ok, throw descriptive errors.

Task 5: Create Ad Storage Service

Files to create: `lib/services/ad-storage.ts`

1. Implement `storeAds(apifyAds, advertiserId)` using the upsert strategy from Section 6.
2. After storing ads, update the `Advertiser` record: set `lastScrapedAt` to now, set `totalAdsFound` to total ad count.
3. Create a `ScrapeRun` record to log the run (status, counts, timing).
4. Return a summary: `{ adsNew, adsUpdated, totalProcessed }`.

Task 6: Create Scrape API Route (Manual Trigger)

Files to create: `app/api/scrape/[advertiserId]/route.ts`

1. Create a POST endpoint that accepts an advertiserId.
2. Look up the advertiser in the database, get their `linkedinCompanyId`.
3. Create a ScrapeRun record with status "running".
4. Call `scrapeAdvertiser()` from the Apify client.
5. Call `storeAds()` to persist results.
6. Update the ScrapeRun record with results (status "completed", ad counts).
7. Return JSON response with scrape results.
8. On error: update ScrapeRun to "failed" with error message, return 500.

Note: This is a long-running request (1-5 minutes). For MVP this is acceptable when triggered manually from admin UI. Later, this moves to Trigger.dev background jobs.

Task 7: Test End-to-End with Real Data

No new files — manual testing.

1. Add a test advertiser to the database: Simplicate, linkedinCompanyId: `2027242`.
2. Trigger the scrape API route.

3. Verify in Prisma Studio:
    - 547 ads were created in the Ad table
    - All 9 format types are represented
    - `mediaUrl` is populated for SINGLE_IMAGE, VIDEO, CAROUSEL, DOCUMENT formats
    - `mediaData` JSON contains the correct format-specific data
    - `startDate` and `endDate` are valid dates
    - `impressions` has no Unicode artifacts
    - The ScrapeRun record shows correct counts
4. Trigger the scrape again and verify upsert works: 0 new, 547 updated, `lastSeenAt` timestamps refreshed.

Task 8: Admin Page — Advertiser Management

Files to create: `app/admin/advertisers/page.tsx`, related components

1. List all advertisers with: name, status, lastScrapedAt, totalAdsFound.
2. "Add Advertiser" form with fields: name, linkedinCompanyId.
3. A "Scrape Now" button per advertiser that calls the API route from Task 6.
4. Show scrape status/progress (loading state while scrape runs).
5. Display last scrape result (new/updated counts) after completion.

Task 9: Budget Tracking (Cost Guard Rail)

Files to modify: `lib/services/ad-storage.ts`, ScrapeRun model

1. After each run completes, estimate cost: `adsFound * 0.004` (based on ~$0.004/ad from real data).
2. Store `costUsd` in the ScrapeRun record.
3. Add a helper `getMonthlySpend()` that sums ScrapeRun costs for the current month.
4. Before starting a new run, check if monthly spend exceeds a configurable limit (default: $50). If exceeded, throw an error instead of starting the run.
5. Show monthly spend on the admin page.

---

9. Example Apify Output (Real Data)

SINGLE_IMAGE

```json
```

```
{
  "adId": "1145280734",
  "adLibraryUrl": "https://www.linkedin.com/ad-library/detail/1145280734",
  "advertiserLogo": "https://media.licdn.com/dms/image/...",
  "body": "[VACATURE] Customer Support Consultant bij Simplicate...",
  "clickUrl": "https://werkenbij.simplicate.nl/o/support-consultant-2?...",
  "ctas": ["Meer informatie"],
  "advertiserName": "Simplicate",
  "advertiserUrl": "https://www.linkedin.com/company/2027242",
  "availability": { "start": "2026-02-12", "end": "2026-02-17" },
  "format": "SINGLE_IMAGE",
  "paidBy": "Simplicate",
  "impressions": "<\u200a1k",
  "targeting": { "language": "Nederlands", "location": "Amsterdam Area, Groningen en omgeving" },
  "headline": "Vacature Support Consultant | Amsterdam of Groningen",
  "imageUrl": "https://media.licdn.com/dms/image/...",
  "startUrl": "https://www.linkedin.com/ad-library/search?companyIds=2027242"
}
```


CAROUSEL


```
json

{
  "adId": "750982734",
  "format": "CAROUSEL",
  "body": "...",
  "impressions": "< 1k",
  "slides": [
    { "imageUrl": "https://media.licdn.com/dms/image/.../ssu-carousel-card-image_lax_1200_1200/..." },
    { "imageUrl": "https://media.licdn.com/dms/image/.../ssu-carousel-card-image_lax_1200_1200/..." }
  ]
}
```


VIDEO


```
json

{
  "adId": "771309874",
  "format": "VIDEO",
  "headline": "👉 Bekijk de Online Masterclass",
  "videoUrl": "https://dms.licdn.com/playlist/vid/v2/.../progressive-servable-video/..."
}
```


DOCUMENT

```json
{
  "adId": "685869434",
  "format": "DOCUMENT",
  "documentUrl": "https://media.licdn.com/dms/document/media/v2/.../document-sliced-pdf_multiton/...",
  "imageUrls": [
    "https://media.licdn.com/dms/image/v2/.../ads-document-images_1920/...",
    "https://media.licdn.com/dms/image/v2/.../ads-document-images_1920/..."
  ]
}
```

MESSAGE (InMail — minimal data)

```json
{
  "adId": "943272674",
  "format": "MESSAGE",
  "body": "He %FIRSTNAME%,\n\nBenieuwd naar het uurtarief van consultancybureaus...",
  "impressions": "5k-10k",
  "targeting": { "language": "English", "location": "Netherlands" }
}
```

SPOTLIGHT (Dynamic Ad)

```json
{
  "adId": "750972514",
  "format": "SPOTLIGHT",
  "body": "Simplicate brengt alle projectinformatie samen op één plek",
  "headline": " 🎯 Het overzicht over projecten en omzet kwijt? 🔥 ",
  "clickUrl": "https://www.simplicate.com/nl/ingenieurs/...",
  "ctas": ["Plan een demo"]
}
```

FOLLOW_COMPANY (Dynamic Ad — note different paidBy)

```json
```

```
{
 "adId": "819500756",
 "format": "FOLLOW_COMPANY",
 "paidBy": "Ideuzo Com",
 "impressions": "100k-150k",
 "clickUrl": "https://www.linkedin.com/company/2027242/life",
 "ctas": ["Visit life"]
}
```


EVENT


```
json

{
 "adId": "1091001534",
 "format": "EVENT",
 "body": "Consultancy bureaus: Meer rust, meer structuur en meer grip op sales...",
 "clickUrl": "https://www.linkedin.com/events/winterofsalesmetrickaarsman.../",
 "ctas": ["View event"],
 "impressionsPerCountry": [
   { "country": "Netherlands", "impressions": "100%" },
   { "country": "Portugal", "impressions": "<1%" }
 ]
}
```


---

10. Key Principles

1. **Isolation:** Only `lib/apify/transform.ts` knows Apify field names. Everything else uses Prisma types.
2. **Upsert, don't duplicate:** Always match on `externalId` (LinkedIn's ad ID). Re-scraping updates existing records.
3. **Store raw where useful:** `mediaData` (JSON) preserves format-specific detail. `impressionsPerCountry` (JSON) keeps geo data.
4. **Budget safety:** Never run a scrape without checking monthly spend first.
5. **Track everything:** Every scrape creates a `ScrapeRun` record for debugging and cost tracking.