1 Motorsport Network - Consulting Opportunities

Date: January 13, 2026

1.1 Executive Summary

Motorsport Network operates a **modern, un-bloated tech stack** (React/Next.js, AWS CloudFront, unified platform). The opportunity isn't "modernize your legacy CMS" - it's **amplifying their core domain** with AI, data unification, and knowledge management.

Strategic Assessment: - Core platform is sound (React/Next.js/Tailwind) - incremental optimization, not rebuild - Video platform (Motorsport.tv) is the exception - appears dated, separate stack - Real opportunity: Data/knowledge unification and AI-powered content operations

---

1.2 Root Cause Analysis: The Missing Domain Layer

The frontend complexity (`msnt-*`) and maintenance burden are symptoms. The upstream problem is: no unified domain catalog.

1.2.1 What's Missing

Without an authoritative domain catalog, the frontend must answer questions that should be resolved upstream:

Question	With Catalog	Without Catalog
"Is this an F1 article?"	`article.series.id === 'f1'`	Parse tags, check categories, guess
"What drivers are mentioned?"	`article.entities.drivers[]`	NER on text, hope for the best
"What ads apply to this property?"	`property.adConfig`	Hardcode per property in component

Question	With Catalog	Without Catalog
“What’s Hamilton’s current team?”	`driver.currentTeam`	Query separate stats system
“How should this display?”	`displayConfig` from backend	500 lines of conditionals

The frontend is doing data integration that should happen upstream.

1.2.2 What They Need

1.2.3 Evidence They Don’t Have This

From org chart analysis: - **MS Stats Operations** is separate (Forix, Wildsoft) - stats are siloed - **No “Data Platform” or “Entity Services” team** visible - **Multiple Product Managers** for different areas but no “Data/Platform PM”

From tech analysis: - `window.msConfigVars` suggests config injection, but probably property-level, not entity-level - No GraphQL or unified API visible (would expect to see it if it existed) - Component logic handles entity resolution (the `msnt-*` problem)

1.2.4 The Real Pitch

> “Your frontend is doing too much work because there’s no authoritative source for ‘what is this article about?’ and ‘how should it display?’
>
> Build an **Entity Catalog + Resolution Layer**, and: - Your frontend becomes a simple renderer - Your maintenance burden drops dramatically - Your AI agents can work with structured data - Your 100+ property/edition combinations become config, not code - New features (prediction markets, personalization) become possible”

1.2.5 How This Connects Everything

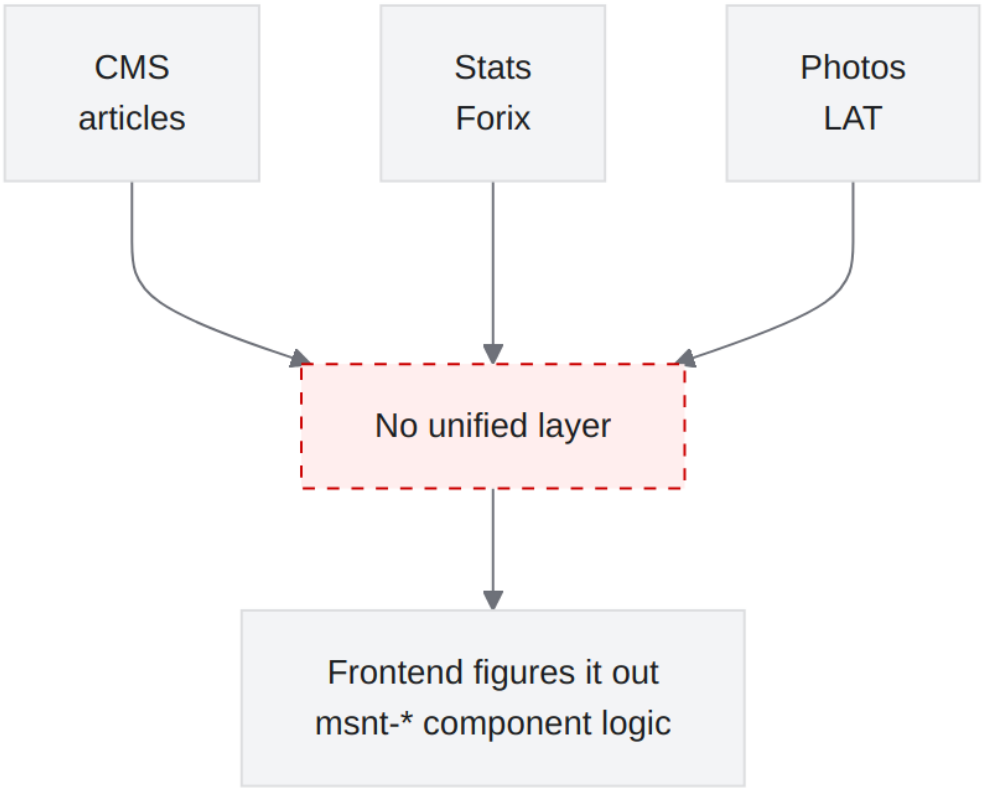

Figure 1: Diagram

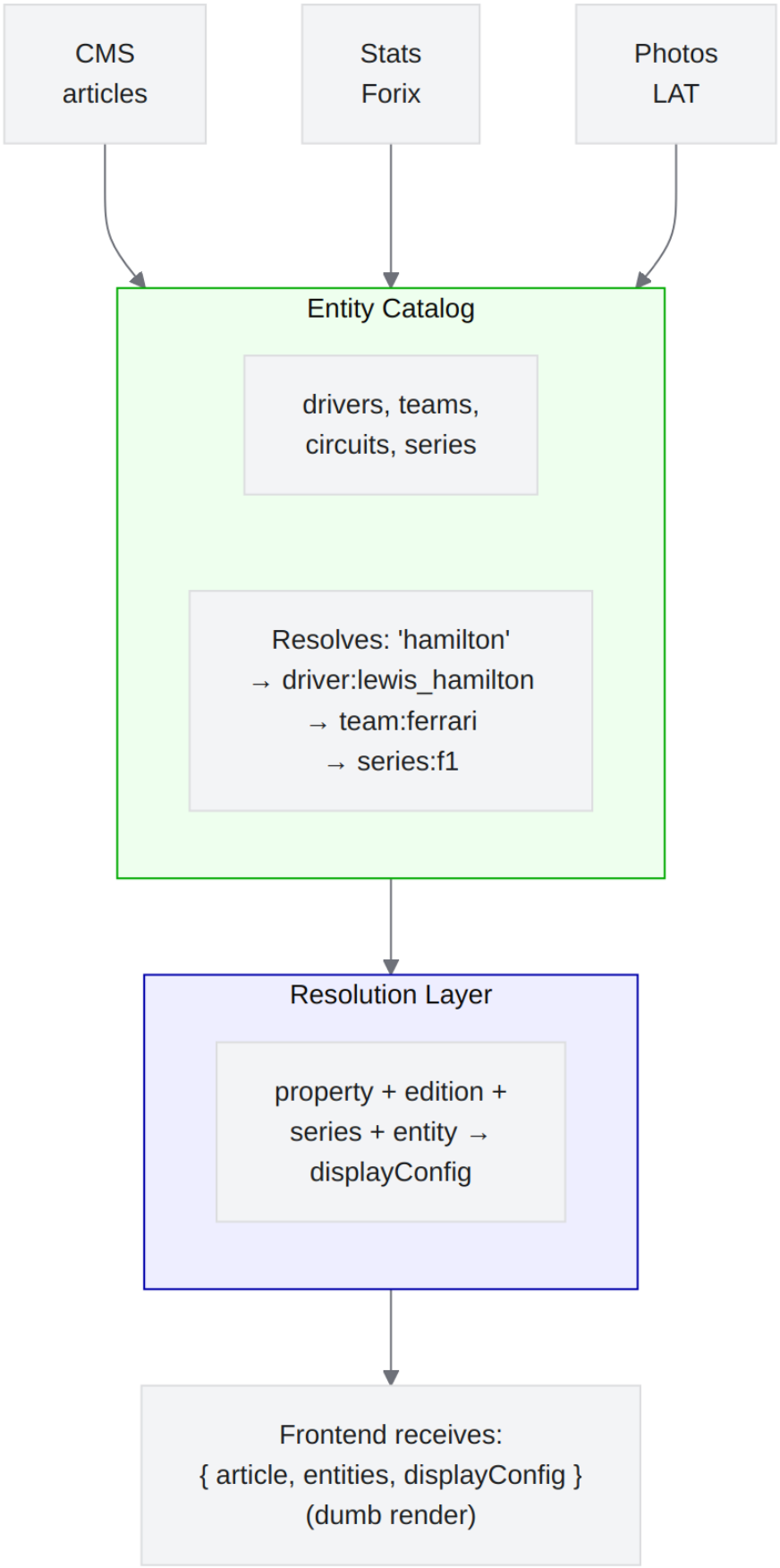

Figure 2: Diagram

Opportunity	Actually Solving
#2 Knowledge Graph	The entity catalog
#5 Entity Catalog	Same thing, API layer
#6 Maintenance/msnt-*	Symptom of missing catalog
#3 AI Content Strategy	Needs entities to enrich content
#4 Data Unification	Analytics tied to entities

They're all the same problem: no domain layer.

1.3 Strategic Design Framework

Applying the **10-15% Innovation Principle**: copy patterns for 85-90%, innovate on differentiators.

1.3.1 For Motorsport Network:

Domain Type	Their Domain	Assessment
Core (Innovate)	Racing data, series expertise, historical stats	This is their differentiator - no one else has 75 years of Autosport archives
Supporting	Publishing platform, ad serving, subscriptions	Modern stack, well-executed - maintain, don't rebuild
Generic	CDN, auth, email, monitoring	Off-the-shelf - nothing to do here

Key Insight: Their competitive advantage is **domain knowledge** (motorsport expertise,

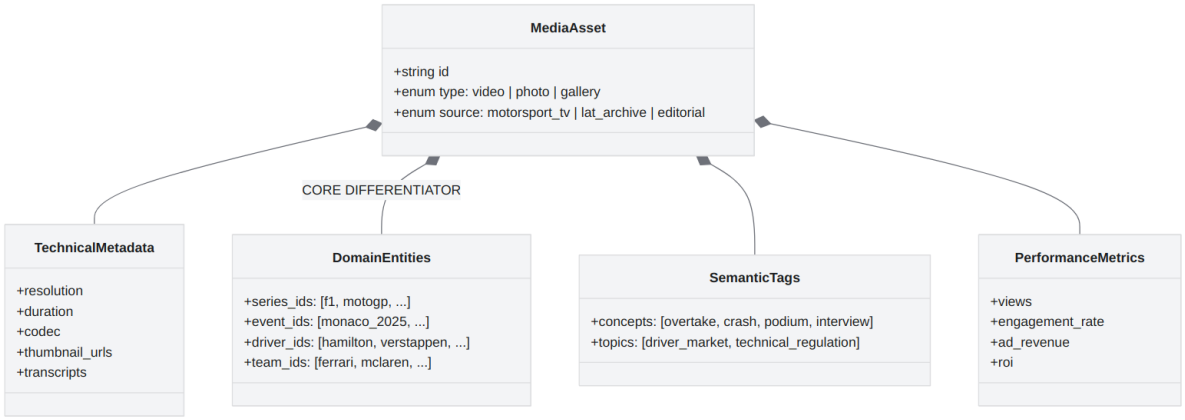

Figure 3: Diagram

historical data, LAT photo archive, Forix stats). Consulting should amplify this, not reinvent their publishing platform.

1.4 Opportunity 1: Video Platform Modernization (Motorsport.tv)

1.4.1 The Problem

Motorsport.tv appears to run on a separate, older tech stack: - Simpler vanilla JavaScript (not React/Next.js) - Different CDN patterns - Likely separate DAM and metadata systems - Disconnected from main editorial workflow

1.4.2 Proposed Solutions

1.1 DAM Integration & Metadata Enrichment **Pattern:** Unified asset catalog with rich metadata linking to domain entities.

Value: Any content creator can query “all Hamilton overtakes at Monaco” and get video, photos, and stats in one result.

1.2 Post-Publishing Workflow **Pattern:** Metrics feedback loop that ties performance back to assets.

Capabilities: - Auto-tag social shares with UTM parameters for attribution - Aggregate engagement by asset/series/driver/topic - Surface “what’s working” patterns to editorial team - Feed learnings back to recommendation engine

Figure 4: Diagram

1.3 AI-Assisted Curation **Pattern:** Agent-assisted video compilation and highlight generation.

NOT creation - curation and assembly: - Auto-generate highlight reels from tagged segments - Suggest "related videos" based on domain entities - Identify archive footage relevant to current news - Generate video metadata from audio transcripts (speech-to-text + NER)

1.5 Opportunity 2: Knowledge Graph & Domain Catalog

1.5.1 The Problem

Motorsport Network likely has **fragmented domain knowledge**: - Editorial knows recent driver quotes - Stats team has historical performance data - Photo team has LAT archive metadata - Ad sales knows advertiser relationships

No unified system to answer: "Everything about Lewis Hamilton - recent quotes, career stats, best photos, related advertisers, social posts."

1.5.2 Proposed Solution: Motorsport Knowledge Graph

Pattern: Unified knowledge base with domain entities as first-class citizens.

2.1 Entity Catalog

2.2 Query Examples Once unified, the system can answer:

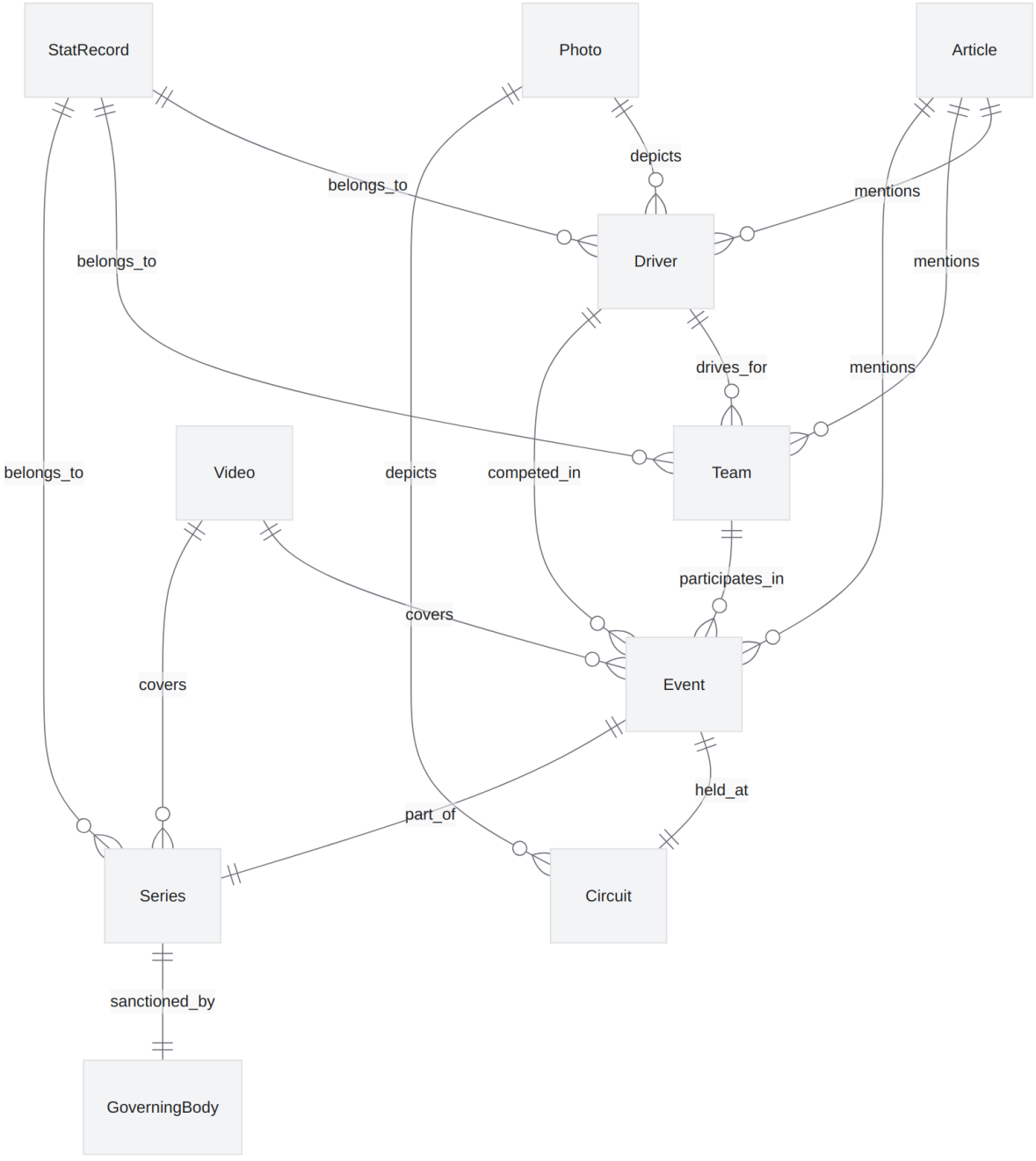

Figure 5: Diagram

Query	Data Sources Required
“Hamilton’s Monaco history”	Stats, photos, articles, videos
“Ferrari sponsors”	Ad sales, team metadata
“Trending drivers this week”	Analytics, social, engagement
“What did Verstappen say about Red Bull?”	Article quotes, transcripts
“Best-performing F1 content last month”	Analytics aggregated by series tag

2.3 RAG Pipeline for Content Enhancement **Pattern:** Knowledge-grounded content generation.

1.6 Opportunity 3: AI-Powered Content Strategy (Ahrefs Integration)

1.6.1 The Problem

Ahrefs provides content gap, backlink, and keyword data - but it requires manual analysis to turn into content decisions.

1.6.2 Proposed Solution: Automated Content Opportunity Pipeline

Pattern: Data-driven content prioritization with AI analysis.

3.1 Pipeline Architecture

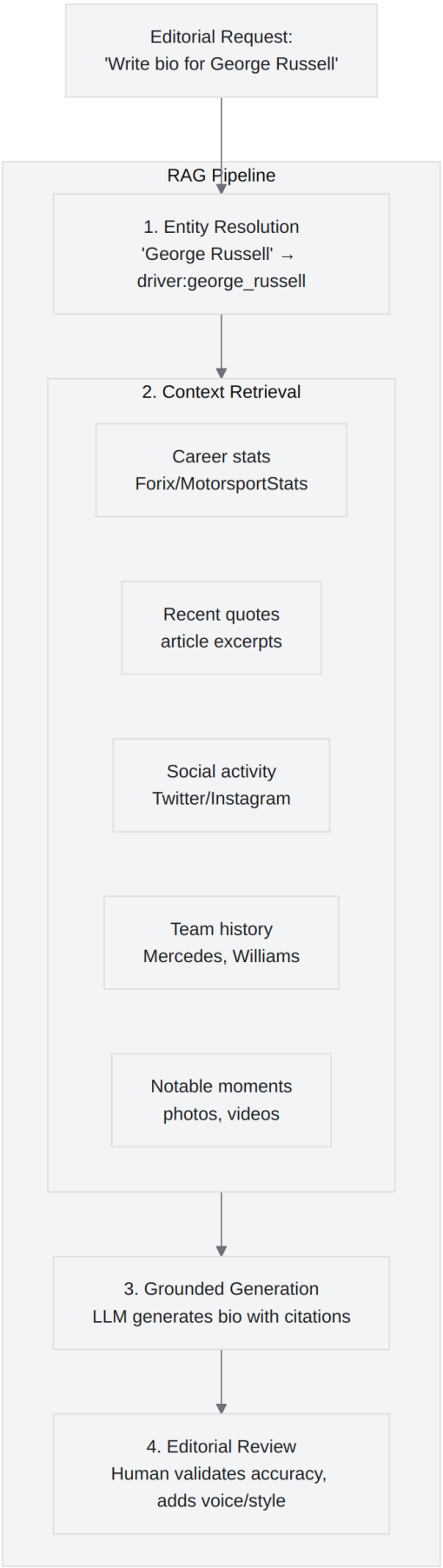

Figure 6: Diagram

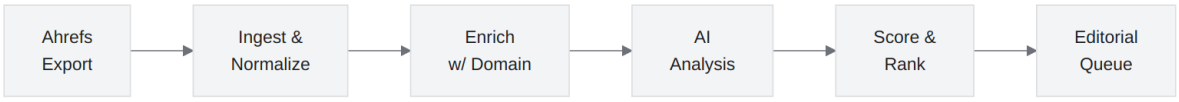

Figure 7: Diagram

Figure 8: Diagram

3.2 Processing Stages **Stage 1: Ingest** - Import Ahrefs CSVs (content gap, competitors, keywords) - Normalize to common schema - Deduplicate and clean

Stage 2: Domain Enrichment - Tag keywords by series (F1, MotoGP, NASCAR) - Link to existing content inventory - Identify competing articles in-house

Stage 3: AI Analysis

Stage 4: Score & Rank - Composite score: Volume × Difficulty × Authority × Relevance - Filter by editorial capacity - Surface to planning dashboard

Stage 5: Output - Prioritized content queue with AI-generated briefs - Links to existing content (for updates vs. new articles) - Recommended writers based on topic expertise

3.3 Example Workflow

1.7 Opportunity 4: Data Unification & Analytics Transformation

1.7.1 The Problem

Likely data silos: - **Google Analytics** - Web traffic, behavior - **Google Ad Manager** - Ad performance, revenue - **Piano** - Subscription metrics - **Social platforms** - Engagement metrics - **CRM** (if exists) - Advertiser relationships, deals

No unified view of: Content → Audience → Revenue.

1.7.2 Proposed Solution: Unified Analytics Data Model

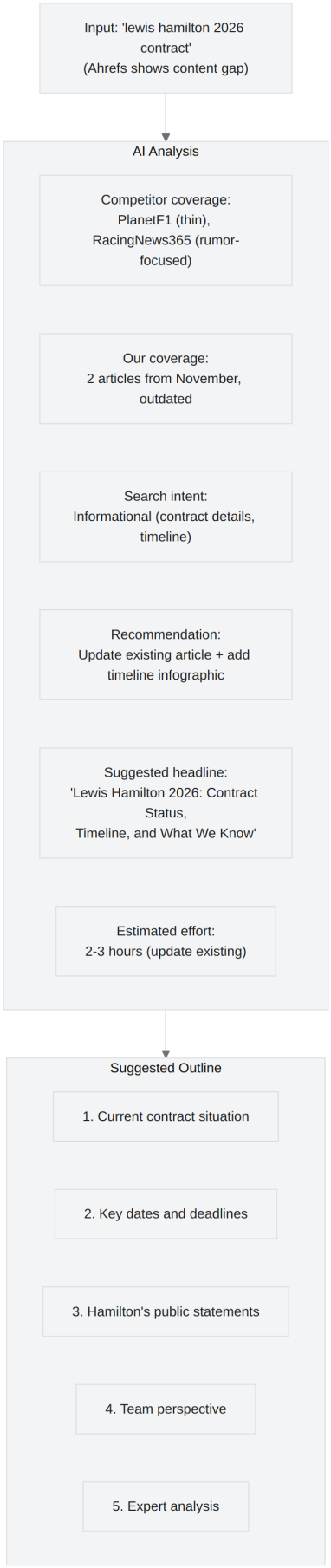

Figure 9: Diagram

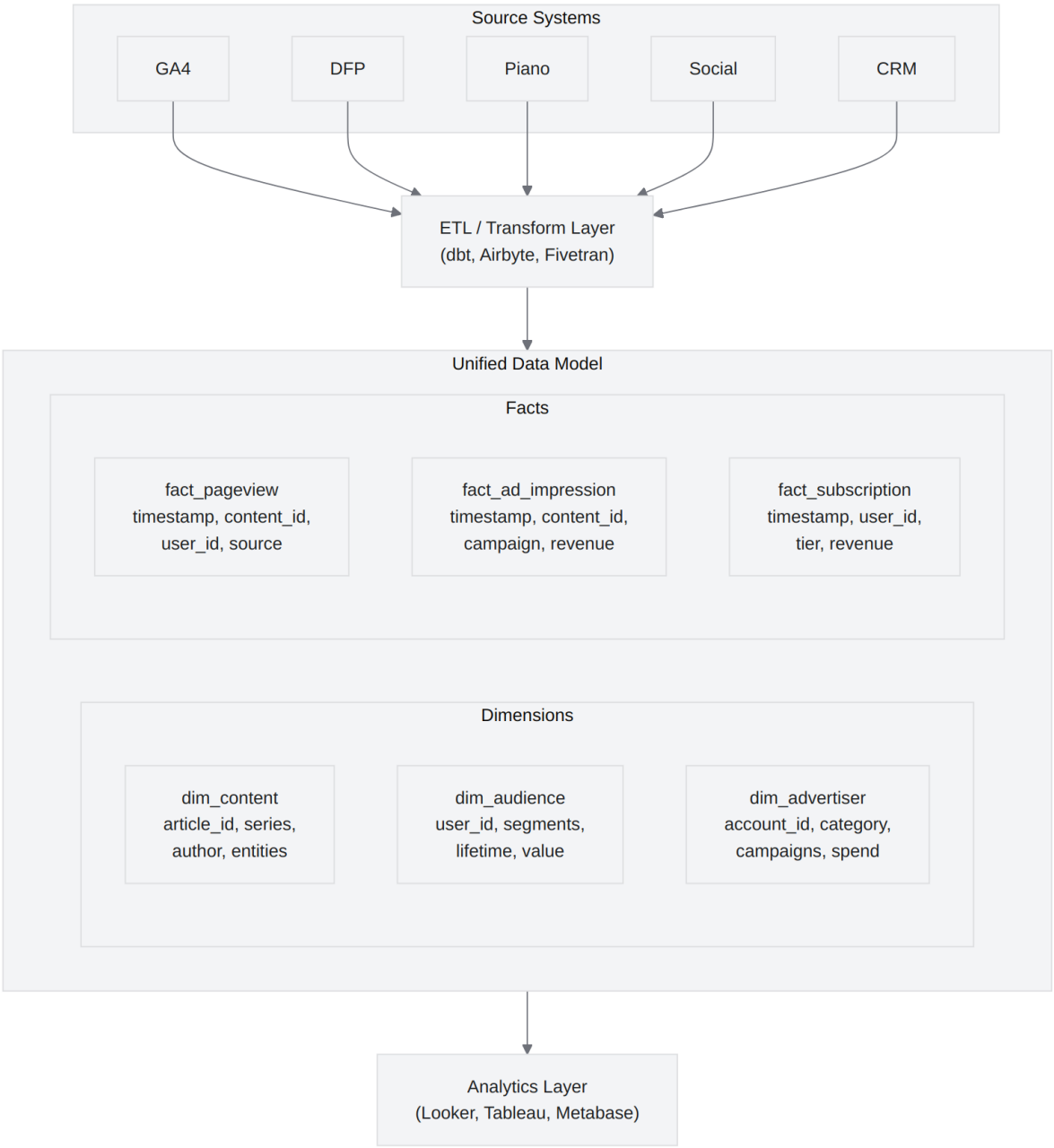

Figure 10: Diagram

4.1 Data Architecture

4.2 Key Metrics Unlocked

Metric	Sources Combined
Content ROI	Pageviews (GA4) + Ad Revenue (DFP) + Sub Conversion (Piano)
Author Performance	Articles × Views × Revenue × Engagement
Series Value	Traffic + Revenue + Audience segments per series
Advertiser Fit	Campaign performance × Audience overlap × Content affinity
Content Velocity	Time to publish × Performance by speed

4.3 Transformation Examples Missing aggregation #1: Content → Revenue attribution

```
-- What's each article actually worth?
SELECT
  content_id,
  SUM(ad_revenue) as ad_revenue,
  SUM(sub_revenue_attributed) as sub_revenue,
  (ad_revenue + sub_revenue) / pageviews as revenue_per_view
FROM unified_content_metrics
GROUP BY content_id
```

Missing aggregation #2: Audience → Content affinity

```
-- Which series do high-value users prefer?
SELECT
  series,
  COUNT(DISTINCT user_id) as users,
  AVG(lifetime_value) as avg_ltv
FROM pageviews
JOIN audience ON user_id
```

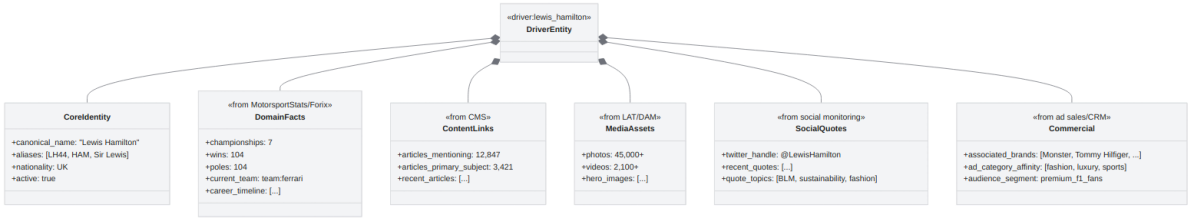

Figure 11: Diagram

```
WHERE lifetime_value > percentile(75)
GROUP BY series
```

1.8 Opportunity 5: Domain Entity Catalog (The Pattern)

1.8.1 The Problem

No single source of truth for “who is Lewis Hamilton” with all associated data.

1.8.2 The Pattern: Entity-Centric Knowledge Management

5.1 Catalog Architecture

5.2 Access Patterns API Examples:

```
// Get entity with all facets
GET /entities/driver/lewis_hamilton
GET /entities/driver/lewis_hamilton?facets=stats,quotes,photos

// Search across entities
GET /entities/search?q=hamilton&type=driver,team

// Get content for entity
GET /entities/driver/lewis_hamilton/content?limit=10&sort=recent

// Get quotes/social
GET /entities/driver/lewis_hamilton/quotes?topic=ferrari&since=2024-01

// Get related entities
```

```
GET /entities/driver/lewis_hamilton/related
→ Returns: team:ferrari, driver:leclerc, circuit:monaco, ...
```

Editorial Use Cases:

Request	Entity Catalog Query
"Hamilton bio for article sidebar"	`/entities/driver/lewis_hamilton?facets=core,stats`
"Recent Hamilton quotes about Ferrari"	`/entities/driver/lewis_hamilton/quotes?topic=ferrari`
"Best Hamilton photos at Monaco"	`/entities/driver/lewis_hamilton/photos?circuit=monaco&sort=engage`
"Hamilton's tweets about his dog Roscoe"	`/entities/driver/lewis_hamilton/social?topic=roscoe`
"Advertisers who want Hamilton content"	`/entities/driver/lewis_hamilton?facets=commercial`

1.9 Summary: Strategic Consulting Positioning

1.9.1 Where We Add Value

Opportunity	Their Status	Our Value-Add
Publishing Platform	Modern (React/Next.js)	Minor optimization only
Video Platform	Dated (separate stack)	**Major opportunity** - DAM, metadata, workflow
Knowledge Graph	Likely fragmented	**Major opportunity** - unified domain catalog

Opportunity	Their Status	Our Value-Add
Content Strategy	Manual (Ahrefs → articles)	**Major opportunity** - AI-powered pipeline
Data/Analytics	Likely siloed	**Major opportunity** - unified model
Entity Catalog	Probably doesn't exist	**Major opportunity** - foundational capability

1.9.2 Engagement Options

Option A: Quick Win (2-4 weeks) - Ahrefs → Content pipeline proof-of-concept - Demonstrate AI-assisted content prioritization - Deliverable: Working prototype + 10 content briefs

Option B: Data Foundation (6-8 weeks) - Analytics data model design - Source system mapping - Deliverable: Data architecture + implementation roadmap

Option C: Knowledge Platform (12-16 weeks) - Entity catalog design + pilot implementation - Integration with existing systems (CMS, DAM, stats) - Deliverable: Working entity catalog for top 100 drivers

Option D: Video Transformation (16-20 weeks) - Motorsport.tv platform assessment - DAM integration architecture - AI metadata enrichment pipeline - Deliverable: Modernization roadmap + pilot implementation

1.9.3 Core Message

> "Your publishing platform is solid. Your competitive advantage is **75 years of motorsport expertise** - the data, relationships, and domain knowledge. We help you unlock that advantage with unified knowledge systems, AI-powered content operations, and data-driven decision making."

1.10 Opportunity 6: Maintenance Burden & Agentic Development

1.10.1 The Problem: "Hard to Maintain"

Based on org chart analysis, several structural factors contribute to maintenance challenges:

6.1 Engineering Concentration Risk

Location	Engineers	Percentage	Risk Level
Ukraine (Dnipro/Lviv)	~30+	~65%	High - active conflict zone
Russia	2	~4%	High - sanctions complexity
Western Europe	~5	~11%	Low
Remote/Other	~10	~20%	Low

45+ engineers concentrated in Ukraine creates: - Geopolitical continuity risk - Timezone gaps with US/UK stakeholders (8-10 hour offset) - Knowledge silos (if key people leave, knowledge leaves) - Communication overhead across 15+ countries

6.2 Custom Component Tax (`msnt-*`)

What `msnt-*` Is Solving The custom component library handles publishing-specific problems that standard UI libraries don't:

Problem	What `msnt-*` Does
Multi-property theming	Same component, different brand colors/fonts per property
Series-specific logic	F1 articles show driver tags, NASCAR shows sponsor logos

Problem	What `msnt-*` Does
Ad slot integration	Components know where to inject ad placements
Edition handling	Language/locale-specific rendering (20+ editions)
CMS integration	Components consume their proprietary CMS data shapes

The core need: An "Article Card" must work on Autosport (UK), Motorsport.com (German), Motor1 (Italian), and InsideEVs (US) - with different theming, ad rules, and content structures.

Why Standard Libraries Don't Solve This Standard component libraries (MUI, Chakra, Radix) solve **generic UI problems**: - Buttons, inputs, modals, tabs - Design system tokens - Accessibility

They **don't solve** publishing-specific problems: - "Show this ad unit after the 3rd paragraph on mobile" - "On F1 articles, show driver entity cards in sidebar" - "Autosport gets Piano paywall, Motor1 gets different paywall"

So they built `msnt-*` to encapsulate **business logic + UI** together.

The Problem: Knowledge Locked in Code

```
msnt-article-card
├── Renders article preview ✓
├── Handles 6 property themes ✓
├── Handles 20+ language editions ✓
├── Injects ad slots per property ✓
├── Shows series-specific metadata ✓
└── Maintained by... the team that built it
    └── Who are mostly in Ukraine
        └── And if they leave, the knowledge leaves too
```

The component works, but the knowledge is locked in the code.

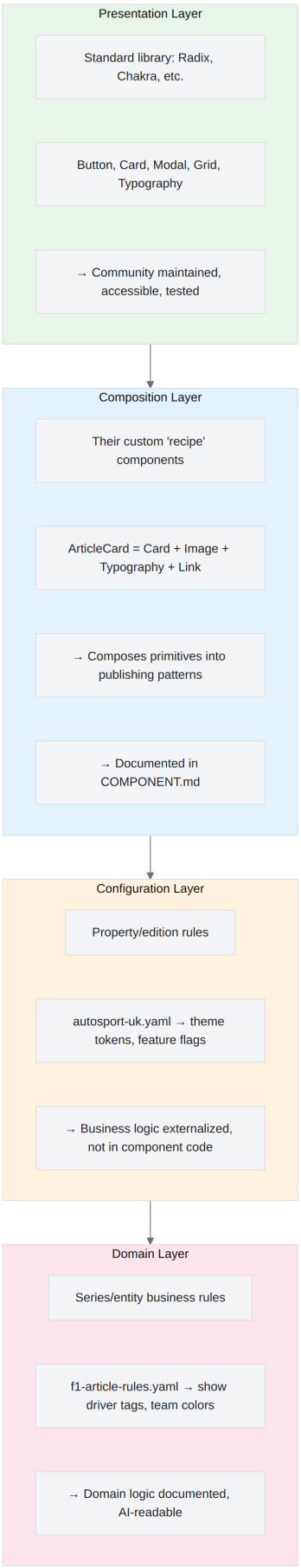

Figure 12: Diagram

The Right Way: Separation of Concerns

Concrete Example: Current vs. Better Current (likely):

```
// msnt-article-card - all logic in one component
export function MsntArticleCard({ article, property, edition }) {
  // 500+ lines mixing:
  // - UI rendering
  // - Property-specific theming
  // - Series-specific logic
  // - Ad slot injection
  // - Edition-specific formatting
  // - Paywall checks
  // ... tribal knowledge embedded in conditionals
}
```

Better approach:

```
// 1. Primitive (from Radix/Chakra) - community maintained
import { Card, Image, Text } from '@radix-ui/themes'

// 2. Composition (their layer) - simple, focused
export function ArticleCard({ article, config }) {
  return (
    <Card className={config.theme.cardClass}>
      <Image src={article.image} />
      <Text>{article.title}</Text>
      {config.features.showAuthor && <AuthorByline />}
      {config.features.showSeriesBadge && <SeriesBadge />}
    </Card>
  )
}

// 3. Configuration (external YAML) - no code changes needed
// autosport-uk.yaml
features:
  showAuthor: true
  showSeriesBadge: true

// 4. Domain rules (external, AI-readable)
// series/f1.yaml
```

```
article_display:
  show_driver_tags: true
  show_team_colors: true
```

Migration Path They don't need to throw away `msnt-*`. They need to **extract the knowledge**:

Phase	Action	Outcome
1. Document	Create COMPONENT.md for each component	Knowledge captured
2. Extract config	Move hardcoded values to YAML	Config changes don't need deploys
3. Swap primitives	Gradually replace custom UI with standard library	Community maintains base

Why This Matters

Aspect	Current (`msnt-*`)	Separated Approach
UI bugs	Team fixes them	Community fixes them
Business logic	Hidden in components	Explicit in YAML
New properties	Copy code, modify	Copy config, modify
AI assistance	Must understand code	Reads config files
Onboarding	"Read the code"	"Read the docs"

The Maintenance Math

Standard Library	Custom Library
Community maintains components	Internal team maintains everything

Standard Library	Custom Library
Bug fixes from OSS contributors	All bugs are your bugs
Accessibility built-in (often)	Accessibility is custom work
New hires know the patterns	Every hire needs onboarding
Documentation exists	Documentation must be written

If they have 50 custom components and each needs 2-4 hours/month of maintenance, that's 100-200 engineering hours/month just on component upkeep - before any feature work.

6.3 Multi-Property Scale

```
6 properties × 20+ language editions = 100+ configurations

Each configuration needs:
├── Locale-specific content rules
├── Regional ad configurations
├── Local SEO optimization
├── Currency/date formatting
├── Legal compliance (GDPR, etc.)
└── Property-specific features
```

Even with a unified platform, the **configuration matrix** creates maintenance burden.

6.4 Leadership Gaps

Open Position	Impact
General Manager - Americas	No US business leadership
Global Head of Programming	No content strategy ownership

Open Position	Impact
Global Head of Commercial Products	No product-market fit ownership
CFO/COO (interim)	Financial/ops decisions delayed
Multiple Product Managers	Features lack clear ownership

Without clear ownership, engineering works on whatever comes in - reactive maintenance instead of proactive improvement.

1.10.2 The Solution: Agentic Development Patterns

The maintenance burden isn't primarily a **technology problem** - it's a **knowledge management problem**. The logic, patterns, and decisions are trapped in people's heads and scattered code.

Agentic coding can help by: 1. **Encapsulating domain logic** in documented, executable patterns 2. **Reducing knowledge silos** through shared context 3. **Automating repetitive tasks** across properties 4. **Maintaining consistency** despite distributed teams

6.5 Pattern: Documented Component Library Instead of `msnt-*` as code-only, create **self-documenting components**:

```
components/
├── msnt-article-card/
│   ├── index.tsx              # The component
│   ├── COMPONENT.md           # Human-readable documentation
│   ├── variants.json          # All supported configurations
│   ├── tests/                 # Test cases
│   └── examples/              # Usage examples
```

COMPONENT.md contains:

```
# Article Card Component

## Purpose
Displays article preview in list/grid contexts.

## Props
| Prop | Type | Required | Description |
|------|------|----------|-------------|
| article | Article | Yes | Article entity |
| variant | 'compact' | 'hero' | 'sidebar' | No | Display variant |
| showAuthor | boolean | No | Show author byline |

## Business Rules
- Hero variant only for featured articles
- Compact variant max 2 lines of title
- Author hidden on mobile for compact

## Series-Specific Overrides
- F1: Show driver tags prominently
- NASCAR: Include sponsor logos
- MotoGP: Motorcycle class badge

## Accessibility
- ARIA labels for screen readers
- Keyboard navigation support
- Color contrast meets WCAG AA
```

Why this helps: - New engineers read docs, not reverse-engineer code - AI agents can use docs to make correct changes - Business rules explicit, not buried in conditionals - Series-specific logic documented, not tribal knowledge

6.6 Pattern: Configuration as Documentation Instead of scattered config files:

```
# properties/autosport-uk.yaml

property:
  id: autosport-uk
  domain: autosport.com
  language: en-GB

series_focus:
```

```
  primary: [f1, motogp, wec]
  secondary: [indycar, formula-e]

features:
  paywall: true
  paywall_provider: piano
  comments: false
  newsletter_popup: true

advertising:
  dfp_network_id: "6122441"
  floor_cpm: 2.50
  video_enabled: true

seo:
  default_title_template: "{title} | Autosport"
  canonical_domain: autosport.com
  hreflang_editions: [us, de, fr]

# Business context (for humans and AI)
notes: |
  Autosport is the UK flagship property, established 1950.
  Premium positioning - higher ad rates than motorsport.com.
  Paywall strategy: 3 free articles, then Piano prompt.
  F1 content performs best; WEC is niche but high-value audience.
```

Why this helps: - One file per property = easy to audit - Business context alongside technical config - AI agents understand *why* not just *what* - New properties copy template, modify values

6.7 Pattern: Agentic Task Automation For repetitive multi-property tasks, encode the process:

```
# TASK: Deploy New Ad Unit Across Properties

## Context
When Ad Ops requests a new ad placement, it must be added to all
properties with appropriate configuration per property tier.

## Steps
```

```
1. **Identify affected properties**
   - Premium tier: autosport.com, motorsport.com (en)
   - Standard tier: motor1.com, insideevs.com
   - Video: motorsport.tv (separate process)

2. **For each property:**
   - Add placement to DFP
   - Update property config YAML
   - Add component placement in template
   - Test on staging

3. **Verification:**
   - Ad loads on all viewports
   - Viewability tracking fires
   - No CLS (Cumulative Layout Shift)

## Property-Specific Rules
- Autosport: No interstitials (premium UX)
- Motor1: Higher ad density allowed
- InsideEVs: EV-specific advertiser exclusions

## Rollback
If issues detected, revert config YAML and redeploy.
```

An AI agent can: - Read this task definition - Execute steps across properties - Apply property-specific rules automatically - Report completion status

6.8 Pattern: Knowledge-Grounded Code Changes Instead of: “Add feature X” → engineer figures out context

With documented patterns:

```
Request: "Add driver bio widget to article sidebar"

Agent workflow:
1. Read COMPONENT.md for existing sidebar components
2. Read entity catalog for Driver entity structure
3. Read property configs for which properties need this
```

```
4. Check business rules for bio display requirements
5. Generate code following documented patterns
6. Create PR with context from documentation
```

The knowledge is in the docs, not in engineer's heads.

1.10.3 6.9 Implementation: Shared Documentation Layer

```
motorsport-platform/
├── docs/
│   ├── ARCHITECTURE.md           # System overview
│   ├── DOMAIN_MODEL.md           # Entity definitions
│   ├── BUSINESS_RULES.md         # Cross-cutting rules
│   └── decisions/                # ADRs
│
├── components/
│   └── [each with COMPONENT.md]
│
├── properties/
│   └── [each with property.yaml + notes]
│
├── tasks/
│   ├── deploy-ad-unit.md
│   ├── add-new-language.md
│   ├── update-series-branding.md
│   └── [repeatable processes]
│
└── agents/
    ├── content-migration/        # Automated content tasks
    ├── config-sync/              # Keep properties aligned
    └── component-audit/          # Check consistency
```

Benefits: | Before | After | |———|———| | Knowledge in Ukraine team's heads | Knowledge in documented patterns | | New hire: 3-6 month ramp | New hire: days to weeks | | Tribal

knowledge required | AI agents can execute tasks | | Manual multi-property updates | Automated with verification | | “Ask Serhii” for backend questions | Read the docs |

1.10.4 6.10 Consulting Engagement: Platform Documentation Sprint

Scope: Document the undocumented - capture tribal knowledge into executable patterns.

Phase 1: Discovery (2 weeks) - Interview key engineers (especially Ukraine team leads) - Map component library and identify documentation gaps - Catalog property-specific business rules - Identify top 10 repetitive maintenance tasks

Phase 2: Documentation (4-6 weeks) - Create COMPONENT.md for top 20 components - Build property configuration schema - Document business rules by series/property - Create task definitions for common operations

Phase 3: Agent Enablement (2-4 weeks) - Set up agentic development workflow - Test AI-assisted code changes against documentation - Train team on pattern-based development - Establish documentation maintenance process

Deliverables: - Component documentation library - Property configuration system - Task automation templates - Agentic development playbook

Outcome: Engineering velocity increases because knowledge is accessible, not locked in heads or timezones.

1.11 Appendix: Technology Patterns (Not Recommendations)

These are patterns, not product recommendations. Implementation would use their existing stack where possible.

Capability	Pattern	Example Technologies
Knowledge Graph	Property graph	Neo4j, AWS Neptune, TypeDB

Capability	Pattern	Example Technologies
Entity Catalog	API-first data layer	Hasura, PostgREST, custom GraphQL
RAG Pipeline	Vector store + LLM	Pinecone, Weaviate, pgvector
Data Lake	Medallion architecture	Databricks, Snowflake, BigQuery
ETL/Transform	dbt + orchestration	dbt, Airbyte, Dagster
Content Pipeline	Event-driven	Kafka, AWS EventBridge, n8n