ask-coty Seed Package

Complete, zero-placeholder project implementation for **ask.coty.design**—a mobile-first PWA Q&A agent with full knowledge base, project planning, and production-ready code.

PROJECT PLANNING ARTIFACTS

README.md

ask.coty.design

A mobile-first, installable PWA Q&A agent about Coty Beasley, hosted as:

- Static UI on Cloudflare Pages at https://ask.coty.design
- Streaming API on Cloudflare Worker at https://ask.coty.design/api/chat

Key goals

- Mobile-first chat UI with streaming responses
- PWA installable (Add to Home Screen)
- Low ongoing cost via small context + strict output caps + caching + rate limiting
- Grounded in embedded knowledge: if something isn't in the KB, say "Not specified in context"
- No hallucinations

Local dev

App

cd app
npm i
npm run dev

Worker

cd worker
npm i
wrangler dev

Secrets

cd worker
wrangler secret put OPENAI_API_KEY

Deploy

1. Deploy app to Cloudflare Pages (static export)
2. Deploy worker with Wrangler
3. Route ask.coty.design/api/* -> worker
4. Bind ask.coty.design -> Pages (custom domain)

Knowledge base

See ./kb/README.md

Architecture

- **Pages (static):** Next.js exported to ask.coty.design
- **Worker (streaming):** Cloudflare Worker routes /api/chat and streams SSE
- **Knowledge:** Bundled KB modules + profile for runtime selection

Cost

Expected ongoing cost dominated by OpenAI tokens (input + output).
See COST.md for minimization strategies.

Support

This is a living seed package. Update kb/source/dossier.md and run the module update workflow.

PLAN.md

Project Plan: ask.coty.design

Objective

Provide a conversational Q&A experience about Coty Beasley's professional profile using a curated, grounded knowledge base.

Constraints

- Static Next.js site (exported, no server-side rendering)
- API handled by Cloudflare Worker routes (/api/*)
- Streaming via SSE (Server-Sent Events)
- OpenAI model usage minimized via strict token controls

Success criteria

- Streaming: user sees partial answers immediately on mobile networks
- KB grounding: no fabricated facts; unknowns trigger "Not specified in context" + follow-up
- Cost: runtime prompt injects only 1–3 KB modules; hard output token cap
- Deploy: Pages serves static UI; Worker serves /api at ask.coty.design/api/*
- PWA: installable on iOS Safari and Android Chrome

Milestones

M0: Repo scaffold + local dev (1 day)

- Next.js App Router + static export
- Wrangler Worker project
- Both run locally

M1: KB source + modules (1 day)

- kb/source/dossier.md complete
- kb/modules/*.md created
- kb/index.json with keyword mapping

M2: Worker SSE streaming (1 day)

- POST /api/chat returns text/event-stream
- Streams dummy output (no OpenAI yet)
- Client can parse SSE deltas

M3: Worker + OpenAI streaming (1 day)

- Worker calls OpenAI with stream: true
- Forwards deltas as SSE events
- Module selection logic in place

M4: Mobile UI polish (1 day)

- Chat layout mobile-optimized
- Streaming response rendering smooth
- Tap targets large

M5: PWA installability (0.5 days)

- manifest.ts configured
- Icons in place
- Standalone display works

M6: Cost controls + safety (1 day)

- Hard output token caps
- Rate limiting per IP
- FAQ caching for common questions

M7: Deploy + DNS (1 day)

- Pages custom domain ask.coty.design
- Worker route /api/*
- Production streaming test

Acceptance tests

See TESTS.md for the 30-question KB grounding test suite.

Phasing

- Weeks 1–2: Repo scaffold + KB complete
- Weeks 2–3: Worker API streaming
- Weeks 3–4: UI + PWA + deploy
- Week 4+: Monitoring + iteration

DECISIONS.md

Architecture & Implementation Decisions

Hosting & deployment

Decision: Cloudflare Pages (static Next.js) + Cloudflare Worker (API).
Rationale: Simple, cheap, good for static sites; Worker handles streaming API with no server overhead.

Streaming protocol

Decision: Server-Sent Events (SSE, text/event-stream).
Rationale: Native browser support, simple framing (data: lines), good for mobile text streaming.

Knowledge strategy

Decision: Keep long-form dossier as source-of-truth; maintain modular KB for runtime injection.
Rationale: Dossier is human-readable and complete; modules are retrieval-friendly and cheap for prompts.

Safety behavior

Decision: If KB doesn't contain a fact (employer, date, metric), respond: "Not specified in context" + ask follow-up.
Rationale: Prevents fabricated answers and encourages users to clarify.

Module selection (v1)

Decision: Simple keyword router (no vector DB initially).
Rationale: Fast to implement, reliable, and scales well for small KB.

Output token limits

Decision: Hard cap (e.g., 350 tokens by default).
Rationale: Keeps costs predictable and encourages concise answers.

Caching strategy

Decision: Cache FAQ answers at Worker level for TTL (e.g., 2 hours).
Rationale: Common questions don't need to call OpenAI repeatedly.

Next.js static export

Decision: Use Next.js output: "export" for true static generation.
Rationale: Simplest to deploy on Pages; all logic can move to Worker.

Future decisions (v2+)

- **Vector-based KB retrieval:** Add embeddings if keyword routing becomes insufficient.
- **KB versioning:** Tag releases of kb/profile.yaml and track breaking changes.
- **Budget mode:** Implement daily token quota + graceful degradation.
- **Analytics:** Log intent/module usage for KB improvement.

IMPLEMENTATION.md

Implementation Guide

Phase 1: Knowledge Base (1 day)

1. Maintain **kb/source/dossier.md** as the source of truth (long-form narrative).
2. Break out **kb/modules/*.md** for each major topic (identity, skills, domains, etc.).
3. Maintain **kb/index.json** with keyword mappings for routing.
4. Maintain **kb/profile.yaml** with structural schema and safety rules.

Acceptance:

- All 30 test questions can be mapped to 1–3 modules.
- Modules are small enough that 3 × 300 words fits comfortably in a prompt.

Phase 2: Worker API (2 days)

1. Implement Worker POST /api/chat:
    - Accept {messages, user_question}
    - Load profile + modules
    - Route to 1–3 modules based on keywords
    - Build prompt: system rules + selected modules + user question
    - Call OpenAI with stream: true
    - Wrap response in SSE events
2. Return SSE format:
   event: delta
   data: {"text":"..."}
   event: done
   data: {}
3. Add GET /api/health and GET /api/meta.
4. Add rate limiting per IP.
5. Add simple FAQ caching.

Acceptance:

- Streaming works on throttled mobile network.
- Cancellation stops the stream.
- Errors are reported cleanly.

Phase 3: Next.js App (2 days)

1. Static Next.js app with streaming chat UI.
2. Consume SSE from /api/chat.
3. Render deltas as they arrive.
4. Mobile touch layout (big buttons, readable font).
5. "Clear" and "Copy" actions.

Acceptance:

- Works on iOS Safari and Android Chrome.
- Streaming feels immediate.

Phase 4: PWA (1 day)

1. Create app/manifest.ts (Next.js file convention).
2. Add icons (192, 512) to public/.
3. Set display: "standalone", start_url: "/".
4. Test "Add to Home Screen" on both platforms.

Phase 5: Deploy (1 day)

1. Export Next.js: `next build && next export`
2. Push to Pages project.
3. Bind custom domain ask.coty.design.
4. Deploy Worker with Wrangler.
5. Add route ask.coty.design/api/* → Worker.

6. Set OPENAI_API_KEY secret.

Acceptance:

- ask.coty.design loads.
- Streaming chat works end-to-end.

Phase 6: Monitoring & iteration (ongoing)

1. Log request counts, cache hit ratio, common intents.
2. Update KB modules as new questions arise.
3. Improve routing keywords.
4. Add missing FAQs.

RUNBOOK.md

Operational Runbook

If costs spike

1. Lower max_output_tokens (e.g., 250 instead of 350).
2. Increase cache TTL for common questions.
3. Tighten rate limits (reduce requests/min/IP).
4. Add optional "budget mode" (short answers only after daily quota).

If hallucinations are reported

1. Add missing fact to kb/source/dossier.md if accurate.
2. Update relevant kb/modules/*.md.
3. Strengthen prompt: "Not specified in context" for unknowns.
4. Add test case to TESTS.md.

If streaming breaks on mobile

1. Verify Content-Type header: text/event-stream.
2. Verify Worker not buffering response (use immediate flush).
3. Verify client SSE parser handles chunk boundaries.
4. Check network: 3G vs LTE behavior.

If rate limits are too strict

1. Increase requests/window in Worker code.
2. Consider per-authenticated-user instead of per-IP.

If FAQ cache hit ratio is low

1. Identify most-asked questions from logs.
2. Add them to cache list.
3. Increase TTL.

If module selection is wrong

1. Check kb/index.json keywords.
2. Improve keyword matching logic.
3. Consider adding synonym mappings.

Regular maintenance

1. Weekly: review logs for error patterns.
2. Monthly: update KB modules with new Q&As.
3. Monthly: review token usage and adjust caps if needed.

COST.md

Cost Controls & Budget Management

Primary cost drivers

OpenAI tokens:

- Input tokens: system prompt + KB modules + user messages
- Output tokens: assistant response

Total cost = (input_tokens × input_rate) + (output_tokens × output_rate)

Model: gpt-4o-mini is currently cheap (see OpenAI pricing).

Strategies to minimize cost

1. Inject only 1–3 KB modules per question

- Keyword router selects modules
- Never inject full dossier
- ~1000 input tokens typical (system + 2 modules + question)

2. Hard cap output tokens

- Set max_output_tokens: 350 by default
- Concise answers save tokens
- Users can request longer answers if they ask

3. Limit conversation history

- Include only last 2–4 user/assistant turns
- Older turns consume input tokens without adding context
- Optional: client-side summarization

4. Cache common questions

- FAQ answers cached at Worker level
- TTL: 2–4 hours
- Reduces repeat OpenAI calls for "What do you do?"

5. Rate limit to prevent scraping

- Max 5 requests/min per IP
- Prevents bots from draining budget

6. Budget mode (optional)

- Track daily token usage
- After threshold: force short answers or require auth
- Graceful degradation

Usage scenarios (rough estimate)

Assume:

- 1000 tokens input per request
- 150 tokens output per request (concise answers)
- gpt-4o-mini: ~$0.00015/input, ~$0.0006/output (verify current pricing)

Low usage (50 chats/month)

- ~75k input tokens + ~7.5k output tokens
- ~$15–20/month

Medium usage (500 chats/month)

- ~750k input tokens + ~75k output tokens
- ~$150–200/month

High usage (2000 chats/month)

- ~3M input tokens + ~300k output tokens
- ~$600–800/month

Note: These are rough estimates. Check OpenAI pricing page for current rates.

Monitoring

Log and track:

- requests/day
- cache hit ratio
- avg tokens per request
- modules used (frequency)
- daily spend estimate

Adjust caps/limits monthly based on actual usage.

SECURITY.md

Security & Data Handling

Secret management

- **OPENAI_API_KEY:** Store as a Cloudflare Worker secret.
- Never expose to the client.
- Never log or display the key.

User privacy

- Do not store conversations long-term by default.
- No personal data collection.
- If analytics desired, use aggregates only (intent type, module used, response time).

Abuse prevention

- **Rate limiting:** 5 requests/min per IP.
- **Input validation:** Reject messages > 5000 chars.
- **Output safety:** Enforce max_output_tokens cap.
- **DDoS:** Rely on Cloudflare edge for basic DDoS protection.

Content safety

- KB is static and curated (no dynamic user-generated content).
- Model instructions include "don't fabricate facts" rule.
- Monitor for jailbreak attempts; update prompt if needed.

Compliance

- No HIPAA, GDPR, or other regulatory data.
- This is a public-facing Q&A tool.

Incident response

1. If credentials are leaked: rotate OPENAI_API_KEY immediately.
2. If budget is drained: check logs for abuse; adjust rate limits.
3. If hallucinations occur: add fact to KB and bump KB version.

TESTS.md

KB Grounding Test Cases

These 30+ test questions verify the KB is complete and the agent doesn't hallucinate.

Identity & overview (4 questions)

1. "What does Coty do?"
   Expected module: module.identity
   Expected: High-level summary about product architecture + design systems + architecture-operator
2. "What is Coty's differentiator?"
   Expected module: module.value_creation
   Expected: Semantic control + preventing system incoherence
3. "What makes Coty unique?"
   Expected module: module.signals_of_fit, module.value_creation
   Expected: Bridges design, product, and ops; architect-operator mentality
4. "Tell me about Coty's background."
   Expected module: module.identity, module.entrepreneurship
   Expected: Founder of Underline Technologies; work across domains

Skills & experience (8 questions)

5. "What experience does Coty have with design tokens?"
   Expected module: module.skills_matrix
   Expected: Token taxonomy, DTCG, computed values, theming
6. "What design systems work has Coty done?"
   Expected module: module.skills_matrix, module.design_systems
   Expected: Token architecture, component libraries, Figma, Storybook
7. "Does Coty know Vue.js?"
   Expected module: module.skills_matrix
   Expected: Yes, Vue 3 and modern web development
8. "What is Coty's experience with product architecture?"
   Expected module: module.problem_types, module.operational_platforms
   Expected: API-as-product, multi-party workflows, ops semantics
9. "Can Coty work on billing systems?"
   Expected module: module.operational_platforms
   Expected: Yes, Zuora, invoice semantics, calculated statuses
10. "What smart home/IoT work has Coty done?"
    Expected module: module.domains
    Expected: Home Assistant, ESPHome, WLed, LED control, distributed systems

11. "Does Coty know color science?"
    Expected module: module.skills_matrix
    Expected: OKLCH color spaces, accessibility, perceptual uniformity
12. "What telecommunications experience does Coty have?"
    Expected module: module.domains, module.entrepreneurship
    Expected: Underline Technologies (fiber network), multi-party service delivery

Product architecture & operations (6 questions)

13. "What does semantic control mean?"
    Expected module: module.value_creation
    Expected: Preventing contradictions via naming conventions, contracts, governance
14. "How does Coty approach multi-party workflows?"
    Expected module: module.problem_types, module.operational_platforms
    Expected: Explicit handoffs, state semantics, party boundaries
15. "What is Coty's experience with support routing?"
    Expected module: module.operational_platforms
    Expected: Routing frameworks, escalation logic, traceability
16. "Does Coty understand billing systems?"
    Expected module: module.operational_platforms
    Expected: Yes, Zuora, invoice mapping, calculated statuses
17. "What is Coty's approach to internal tools?"
    Expected module: module.operational_platforms
    Expected: Treat as product surfaces; clear semantics; multi-market scaling
18. "How does Coty handle API architecture?"
    Expected module: module.problem_types
    Expected: Schema exploration, versioning, sandboxing, OpenAPI

Context engineering & AI (4 questions)

19. "What is context engineering?"
    Expected module: module.context_engineering
    Expected: Organizing information so AI systems retrieve and reason reliably
20. "How does Coty approach ontologies and knowledge graphs?"
    Expected module: module.context_engineering
    Expected: Semantic strategies for making data discoverable to LLMs
21. "What is Coty's experience with AI-driven UX?"
    Expected module: module.context_engineering
    Expected: Conversational interfaces, personalization, design for AI
22. "Does Coty work with LLM context management?"
    Expected module: module.context_engineering
    Expected: Yes, exploring OntoRAG, knowledge graphs, high-signal context

Domains & verticals (4 questions)

23. "What medical/healthcare experience does Coty have?"
    Expected module: module.domains
    Expected: HIPAA compliance, educational platforms (MCAT), accessibility

24. "Does Coty have civic tech experience?"
    Expected module: module.domains, module.entrepreneurship
    Expected: Yes, Underline Technologies; open-access networks; community benefit
25. "What is Coty's experience with financial platforms?"
    Expected module: module.domains
    Expected: Billing semantics, subscription models, accounting integration
26. "Does Coty work on consumer robotics or smart devices?"
    Expected module: module.domains
    Expected: Smart home automation, LED control, distributed IoT networks

Working style & collaboration (4 questions)

27. "How does Coty approach technical communication?"
    Expected module: module.technical_communication
    Expected: Specs, diagrams, clear naming; artifacts that persist alignment
28. "What is Coty's process for architecture design?"
    Expected module: module.signals_of_fit
    Expected: Start with coherence, add detail; explicit contracts; naming governance
29. "Does Coty manage teams?"
    Expected module: module.identity
    Expected: Architect-operator role; leverage over direct management; hands-on validation
30. "What is the best fit for Coty's work?"
    Expected module: module.signals_of_fit
    Expected: Problems spanning experience + platform + ops; ambiguity about semantics; multi-party workflows

Unknown/not-in-KB handling (3 questions)

31. "What company did Coty work at in 2016?"
    Expected: "Not specified in context" + follow-up
    Expected NOT: Made-up company name
32. "What is Coty's current job title?"
    Expected: "Not specified in context" + ask context (e.g., is this for LinkedIn positioning?)
    Expected NOT: Invented title
33. "How much revenue did Underline Technologies generate?"
    Expected: "Not specified in context" + ask if this is for business research
    Expected NOT: Made-up number

Passing criteria

- All identity/skills/domain/ops questions answered with grounded facts from KB
- Unknown handling: responds "Not specified in context" + follow-up (no hallucination)
- No invented employers, dates, metrics, or credentials

CHANGELOG.md

Changelog

[1.0.0] - 2025-12-15

Added

- Initial seed package created
- Full project planning documentation
- Knowledge base structure (source dossier + modular KB)
- MVP Next.js static PWA scaffold
- MVP Cloudflare Worker streaming API
- 30+ KB grounding test cases
- Cost controls and security guidelines

Status

- Ready for initial implementation
- All phase M0–M7 documented

KNOWLEDGE BASE ARTIFACTS

kb/README.md

Knowledge Base (KB)

Philosophy

- **kb/source/dossier.md** is the source-of-truth comprehensive narrative dossier (human-readable, complete).
- **kb/modules/*.md** are small, retrieval-friendly chunks used at runtime (selected 1–3 per question).
- **profile.yaml** defines stable structure, schema, and safety behavior rules.
- **index.json** provides module metadata (keywords, paths) for routing.

Editing workflow

1. Update **kb/source/dossier.md** first (add new info, clarify existing sections).
2. Update affected **kb/modules/*.md** files (sync the changes to modular form).
3. Update **kb/index.json** keywords if adding new routing signals.
4. Add a test case to TESTS.md if introducing a new topic.
5. Bump version in profile.yaml.

Module guidelines

- Keep each module to 100–300 words (fits comfortably in 1–3 module context windows).
- Use bullet lists for readability.
- Include 2–3 keywords for routing in the module header.
- Link back to main dossier if a topic needs more depth.

KB versioning

- profile.yaml tracks KB version.
- Break changes (removed modules) should bump major version.
- New modules bump minor version.
- Keyword/content tweaks bump patch version.

Storage strategy

- All KB files are bundled into the Worker at build time (no external DB).
- For updates: modify files + redeploy Worker.
- For future: could migrate to Cloudflare KV if KB grows large.

---

kb/source/dossier.md

Professional Dossier: Coty Beasley

Executive Summary

You operate at the intersection of **distributed product architecture, design infrastructure, and operational systems engineering**. Rather than fitting into a single functional category, you architect product coherence across visual design, technical platforms, and operational workflows—domains that most organizations silo into separate functions. Your unique capability is **semantic control**: defining the canonical rules, schemas, contracts, and interface specifications that prevent complex multi-party products from fracturing into contradiction.

You are simultaneously:

- A **design systems architect** who builds foundational infrastructure enabling coherent UX at scale
- A **product architect** who defines multi-system coherence across APIs, workflows, and operational platforms
- A **context engineer** who designs how AI systems understand and organize information
- A **technical communicator** who produces specifications that enforce organizational alignment
- A **pioneer** of emerging technologies (design tokens, context ontologies, AI-driven UX) within your domains
- An **entrepreneur** who has founded infrastructure companies and navigated complex technical challenges in telecommunications and civic networks

Your work makes the product **coherent** where it would otherwise fracture. You prevent system failures through thoughtful naming conventions, status semantics, schema governance, and design-code alignment. You are not a manager (though you could be), not a pure IC contributor (though you do hands-on work), and not a strategist-only (though you set direction). You are an **architect-operator**: someone who defines the system shape, communicates the rules, and validates they work in practice.

What You Actually Do: Core Responsibilities

1. Design Systems Architecture & Infrastructure

Scope: You don't just implement design systems—you architect the **foundational infrastructure** that makes design coherent and scalable across organizations and products.

Design Token Architecture:

- Research and evaluate intelligent visual token editors supporting computed token values and algorithmic operations
- Structure token taxonomy following DTCG (Design Token Community Group) specification with brand attributes, visual foundations, and semantic categories
- Build dark/light mode strategy using Figma variables and modes with explicit transformation rules across contexts
- Create formal token categories (fill, text, outline, elevation, motion) with explicit relationships and constraints
- Design token naming conventions that prevent entropy and communicate intent to both designers and engineers

Implementation-to-Design Alignment:

- Tie design libraries to component libraries and Storybook to reduce drift between design and engineering
- Structure component schemas with explicit typography references, spacing relationships, and sizing rules
- Treat design-code alignment as a first-class architectural problem, not an afterthought
- Design infrastructure making it economical for teams to keep design and implementation synchronized

Design Infrastructure Pioneering:

- Explore token authoring solutions beyond existing tools, investigating computed values, graph-based automation, and intelligent operations
- Research how to enable design at scale without creating maintenance burden for teams
- Apply systems thinking to design: understanding how small token decisions cascade into consistent or inconsistent experiences

Why This Matters: Design systems at scale fail when treated as component libraries. You architect them as **infrastructural systems**, understanding that tokens are contracts between designers and engineers, that naming conventions are governance, and that the system only works if both sides trust the rules.

2. Product Architecture: Multi-System Coherence

Scope: You define the **conceptual and technical shape** of complex products spanning UX, APIs, data, and operational workflows. You make products coherent when they would otherwise be chaotic.

API-as-Product Architecture:

- Define requirements for platforms where the API is the product, not just an implementation detail
- Design schema exploration and discovery interfaces so API users can navigate large conceptual spaces
- Build versioning strategies balancing backward compatibility with evolution
- Create sandboxing environments for safe experimentation and partner testing
- Distinguish internal APIs (serving your own systems) from external APIs (serving partners/customers) with appropriate governance
- Produce OpenAPI specifications and structured YAML object models as canonical contracts (Invoice, InvoiceItem, enums, status flows)

Multi-Party Workflow Orchestration:

- Design voice service order fulfillment workflows with explicit state semantics and cross-party handoff expectations
- Define which party (subscriber, platform, provider) holds responsibility at each stage
- Build components and status systems reflecting the actual distributed nature of the work
- Iterate on **event naming conventions**: hierarchy, tense, separators, simplification rules, status aggregation

Operational Platform Semantics:

- Design support routing frameworks distinguishing what internal support handles versus provider escalation
- Manage historical traceability across channels and systems so no customer request gets lost
- Treat billing migration (Zuora) as product work, not operational overhead
- Design invoice status mapping semantics and filtering constraints
- Create calculated statuses (e.g., AUTOPAID logic) bridging what systems store versus what customers need to understand
- Work with Zuora objects/tables (ProductRatePlanCharge, tier tables), import field mappings, accounting code IDs as an architect ensuring platform coherence

Internal Tools as Product:

- Drive major redesign of internal operations tools for multi-market and external partner use
- Define navigation groupings and admin area structures preventing confusion in complex domains
- Name things carefully to avoid ambiguity (e.g., not conflating "user" and "account" when they mean different things)
- Treat scaling from single-market to multi-market as an architectural problem, not just feature addition

Why This Matters: Multi-party products fail when handoffs are ambiguous. You architect **explicit contracts**: at each stage, who knows what, who's responsible, what happens if this fails, how do we prove it happened? You treat **workflow design as infrastructure**.

---

3. Context Engineering & Information Architecture

Scope: You architect how AI systems (and human systems) **discover, organize, and retrieve information**—making the implicit explicit, the chaotic ordered.

Ontological Strategies for LLM Context:

- Research open-source AI tools for assembling high-signal context from knowledge bases and domain models
- Explore how LLMs understand what data exists and where to find it within a knowledge base
- Investigate semantic strategies: knowledge graphs, OntoRAG, OntologyRAG approaches for materializing organizational understanding
- Design how to represent relationships between concepts formally so AI systems can reason about domain knowledge

Project Registry & Information Organization:

- Create project template structure and bootstrap workflows (ADRs, project structure recommendations)
- Design how future projects would be discovered and documented
- Treat information organization as foundational to future work, not an afterthought
- Build reusable templates reducing entropy when starting new work

AI-Driven Personalization Architecture:

- Design conversational AI interfaces for onboarding and product selection
- Architect how context flows from user behavior → AI understanding → personalized experience
- Explore how LLMs generate UI and understand visual reasoning at the frontier where AI meets design intent
- Design how recommendation systems discover the right information for each user

Why This Matters: As organizations grow, information chaos explodes. You architect the **implicit organizational knowledge structure** allowing teams to find what they need, understand how decisions were made, and avoid repeating mistakes. This is semantic control in its purest form.

---

4. Technical Communication & Specification

Scope: You produce **artifacts that enforce organizational alignment**: specifications, diagrams, and communications making implicit agreements explicit.

Specification Production:

- Produce OpenAPI specifications becoming canonical API contracts
- Write structured YAML object definitions readable by both systems and humans
- Create webhook payload specifications partners integrate against

- Author detailed technical presentations with vision, impact, timing, and technical construction clearly delineated

Diagram & Visual Architecture:

- Use Mermaid and diagramming tools to model complex systems
- Create entity-relationship diagrams revealing structural problems
- Build visual narratives helping technical and non-technical audiences understand system shape

Presentation for Mixed Audiences:

- Structure technical presentations serving both technologists and business stakeholders
- Lead with vision and impact (universal understanding), then drill into technical construction
- Use tight time constraints as forcing functions to distill what actually matters
- Produce artifacts (slides, frameworks, diagrams) that outlive the presentation

Why This Matters: Specifications are not documentation—they're **governance**. A well-written spec prevents ambiguity that would surface as bugs, missed requirements, and frustrated team members. You don't write specs to explain what you did; you write them to make implicit decisions explicit before work begins.

---

5. Design for Emerging Technologies

Scope: You **pioneer** how design systems and UX patterns apply to new technical frontiers: AI interfaces, smart home automation, advanced LED control, quantum-aware applications.

AI-Driven Interfaces:

- Lead design for conversational AI interfaces (onboarding, product selection)
- Explore which LLMs and benchmarks excel at UI generation and visual reasoning
- Design how AI systems understand design intent, context, and user needs
- Build interfaces enabling natural human-AI collaboration

Smart Home & IoT Infrastructure:

- Expert in Home Assistant, ESPHome, WLed smart home automation
- Design LED control systems and WS2811 addressable LED infrastructure
- Architecture for distributed IoT networks and local-first automation
- Build custom hardware integration and networking solutions

Medical & Healthcare Platforms:

- Design MCAT preparation platform with design tokens following DTCG specification
- Understand medical device constraints, compliance, and accessibility requirements
- Create educational interfaces encouraging sustained engagement and learning

Civic & Infrastructure Technology:

- Founded Underline Technologies: open-access fiber network platform serving communities

- Design for networks and telecommunications where product = distributed infrastructure + user experience
- Understand how technology shapes civic participation and community access

Color Science & Accessibility:

- Deep expertise in OKLCH color spaces maintaining predictable contrast relationships
- Design accessible color systems working across all hues
- Understand how perceptually uniform color spaces enable programmatic palette generation

Why This Matters: You don't design for one domain. You architect **domain-specific but transferable patterns**: design tokens work for accessibility platforms the same way they work for financial software, just with different values. Smart home automation teaches you about distributed state management applying to multi-party workflows. Medical device compliance teaches you about precision in communication. You are an architect applying design infrastructure thinking to whatever domain you enter.

---

6. Entrepreneurship & Infrastructure Building

Scope: You don't just design products; you've **built companies and navigated complex infrastructure challenges**.

Founding Work: Underline Technologies:

- Founded company focused on open-access fiber network platform serving communities
- Navigated technical complexity of networking, provisioning, and multi-party service delivery
- Built product requiring sophisticated understanding of telecom infrastructure, regulatory environment, and community needs
- Addressed problem at intersection of civic technology, infrastructure, and product design

Multiple W-2 Employment Model:

- Simultaneously hold multiple W-2 positions with distinct focus areas
- Navigate tax implications and role boundaries across employers
- Bring architecture thinking from one domain into others
- Maintain consulting/part-time work generating additional revenue

Problem-Solving Across Verticals:

- Design systems work (design infrastructure)
- Smart home automation (distributed systems, local-first architecture)
- Healthcare IT (HIPAA compliance, high-stakes data management)
- Fiber network platform (infrastructure, multi-party operations)
- Civic technology (governance, community benefit)
- Financial platforms (billing, accounting, regulatory)
- Medical platforms (accessibility, compliance)

Why This Matters: You're not a specialist who is also an entrepreneur. You're someone applying **architecture thinking to hard problems across domains**. Whether designing a smart home network or a billing platform, the core challenges are the same: **How do we make complex distributed systems coherent? How do we define contracts preventing failure? How do we communicate semantics so teams don't contradict each other?**

Skills You Possess: Comprehensive Inventory

Design & UX Skills

Design Systems Architecture:

- Design token structure and taxonomy (DTCG specification)
- Token authoring, computed values, algorithmic variations
- Dark/light mode strategy, context-aware design systems
- Component library architecture and code alignment
- Design infrastructure enabling organization-wide coherence

Visual Design:

- Color science (OKLCH color spaces, perceptual uniformity, accessibility)
- Typography and spacing systems
- Interface design for complex domains (operations, billing, onboarding)
- Design for emerging technologies (AI interfaces, smart home, medical devices)

User Experience Design:

- Multi-party workflow design (subscriber, platform, provider)
- Onboarding and product discovery experiences
- Internal tool UX (operations, admin, support)
- Accessibility-first design (WCAG, perceptual uniformity)
- Conversational AI interface design

Design Tools & Implementation:

- Figma (variables, modes, design systems, prototyping)
- Design token tooling (Token Studio, alternatives)
- Diagramming (Mermaid, ERD, architecture visualization)
- Storybook and component documentation alignment

Technical & Engineering Skills

Software Development:

- Vue.js and modern web development (Vue 3)
- JavaScript/TypeScript
- YAML configuration and data definition
- JSON and JSON-LD structured data
- Markdown documentation and specification writing

Architecture & Systems Design:

- API design (RESTful, OpenAPI specification)

- Database schema design (Zuora, relational modeling)
- Multi-party workflow and state machine design
- Event-driven architecture and naming conventions
- Distributed systems thinking (infrastructure context)

Infrastructure & DevOps:

- Smart home automation (Home Assistant, ESPHome)
- IoT device integration and networking (WLed, WS2811 LED control)
- Network and telecommunications understanding
- Local-first, distributed architecture
- Hardware integration and custom device work

Data & Information Systems:

- Context engineering for AI systems
- Ontological strategies and knowledge graphs
- Data dictionary design and semantics
- Workflow state and status semantics
- Billing and operational data modeling

Product & Strategy Skills

Product Architecture:

- API-as-product platform design
- Multi-system coherence and schema governance
- Operational platform semantics (billing, support, provisioning)
- Product-led growth mechanics (landing pages, onboarding flows)
- Marketplace and multi-tenant architecture

Product Strategy & Communication:

- Technical vision and impact communication
- Cross-functional alignment through specification
- Stakeholder communication (technical + non-technical)
- Presentation design for mixed audiences
- Technical presentation under time constraint

Product Operations:

- Workflow and handoff design
- Internal tool architecture
- Process documentation and governance
- Support routing and escalation frameworks
- Multi-market scaling and localization strategy

Domain Expertise

Telecommunications & Infrastructure:

- Open-access fiber network design
- Multi-party service delivery (subscriber, platform, provider)
- Network architecture and provisioning systems

- Telecom regulatory and business model understanding

Financial & Billing Systems:

- Zuora platform and object modeling
- Invoice and billing semantics
- Subscription and tiered pricing models
- Accounting and revenue recognition
- Payment processing and calculated statuses

Medical & Healthcare:

- HIPAA compliance and data governance
- Medical device and platform accessibility
- Educational platform design (learning science)
- Healthcare provider and patient workflows

Smart Home & IoT:

- Home automation architecture (Home Assistant)
- LED control and lighting design systems
- Distributed IoT networks
- Custom hardware and electronics integration
- Local-first and cloud-optional architecture

Civic & Community Technology:

- Open-access network platforms
- Community benefit and participation design
- Regulatory environment (telecom, civic)
- Volunteer engagement and organizational design

Communication & Leadership Skills

Specification & Documentation:

- OpenAPI specification writing
- YAML object model definition
- Event naming conventions and hierarchies
- Technical requirement capture
- Architecture documentation

Diagramming & Visualization:

- Mermaid diagram creation and architecture visualization
- Entity-relationship and system architecture diagrams
- Visual communication of complex systems
- Presentation design and visual storytelling

Presentation & Teaching:

- Technical presentations for mixed audiences
- Vision-impact-timing narrative structure
- Interview-based architecture discovery

- Collaborative problem-solving facilitation
- Written and verbal clarity under time constraint

Cross-Functional Collaboration:

- Design ↔ Engineering alignment
- Product ↔ Operations integration
- Technical ↔ Non-technical communication
- Partner communication and integration
- Vendor evaluation and relationship management

How You Work: Process & Approach

Your Architectural Thinking Pattern

You start with coherence, then add detail.

Rather than building features in isolation and hoping they integrate, you begin with:

1. **Semantic clarity**: What are the core entities and relationships? What do we mean by status, party, state?
2. **Explicit contracts**: What does each system need to know about others? What are the interfaces?
3. **Naming governance**: What conventions prevent future ambiguity?
4. **Cascade to detail**: Given the architecture, what does the UI look like? What tokens do we need? What workflows are required?

This approach prevents **late-stage architecture rework** that happens when teams build features then realize the underlying system doesn't make sense.

You Treat Infrastructure as a First-Class Problem

Design systems, billing platforms, API contracts, workflow semantics—these are not "overhead" or "things we do after the business logic." They are **foundational to product success**. If the infrastructure is sound, everything else becomes easier. If it's chaotic, every new feature becomes harder.

Your work ensures that:

- **Design decisions** don't contradict in production
- **API consumers** encounter a coherent mental model
- **Operational teams** understand the true state of orders/billing/support
- **Partners** can integrate against clear contracts
- **Future teams** can extend without breaking the system

You Operate Across Abstraction Levels Simultaneously

You are comfortable moving between:

- **High-level strategy**: What is the product architecture enabling in 3 years?
- **Medium-level design**: What does the design system need to support multi-market scaling?

- **Detail-level implementation**: Does this token value work with WCAG AAA contrast? Does this event name follow the hierarchy?

Most people specialize in one level. You operate across all three—and understand how decisions at one level cascade through the others.

You Communicate in Artifacts, Not Just Words

You don't explain your thinking in meetings alone. You produce:

- Specifications (OpenAPI, YAML, webhook payloads)
- Diagrams (Mermaid, architecture visualizations)
- Design systems (Figma, tokens, components)
- Structured documents (ADRs, project templates, naming conventions)

These artifacts **persist and enforce alignment** after the meeting ends. They are the architecture made visible.

The Value You Create: Impact

Problem You Solve: System Incoherence

The Problem: As products grow across UX, APIs, billing, workflows, and operations, they fragment:

- UI shows something that doesn't match the data model
- API documentation contradicts implementation
- Billing semantics confuse customers and internal teams
- Support can't route requests correctly because status semantics are ambiguous
- Design tokens work in one product but break in another
- Multi-party workflows have implicit handoffs that fail silently

Why This Happens: Different teams optimize locally—Product wants velocity, Engineering wants simplicity, Design wants expressiveness, Operations wants stability. Without **semantic control**, teams make contradictory decisions. Without **explicit contracts**, handoffs are ambiguous. Without **coherent infrastructure**, scale breaks everything.

How You Prevent It: You architect the **canonical rules** that make the system coherent:

- **Schema governance**: What entities exist? What are their properties? How do they relate?
- **Naming conventions**: What do we mean by "status," "event," "party," "context"?
- **Workflow semantics**: Who knows what? Who's responsible? What happens if it fails?
- **Design infrastructure**: What tokens do we need? How do they transform? How do we keep design and code aligned?
- **Communication artifacts**: Specifications, diagrams, and documentation making decisions explicit

Impact Areas

Design Systems:

- Organizations that would otherwise fragment designs into contradiction can scale coherently
- Designers and engineers speak the same language through design tokens
- New products can reuse infrastructure instead of reinventing
- Accessibility is built in rather than bolted on

Product Architecture:

- APIs become usable instead of confusing
- Multi-party products have clear contracts preventing silent failure
- Operational teams can explain system state to customers
- New features integrate cleanly instead of creating contradictions

Technical Communication:

- Specifications prevent ambiguity before work begins
- Mixed audiences (technical + business) understand the same vision
- Decisions are recorded so future teams know the reasoning
- Alignment persists across team changes

Organization Building:

- Teams understand what coherence looks like
- New team members can learn the architecture from artifacts
- Vendors and partners integrate against clear contracts
- Technical decisions are reproducible, not dependent on one person's understanding

---

Unique Positioning: What Makes You Different

You Bridge Verticals That Most People Don't

Most professionals specialize: "I'm a design systems person" or "I'm a product architect" or "I'm an infrastructure engineer." You operate across all three.

This creates unique positioning:

- Design systems people rarely understand multi-party operational workflows
- Product architects often don't know design tokens and accessibility
- Infrastructure engineers don't typically think about UI coherence
- **You see how these domains must work together**

You Understand Distributed Systems Through Multiple Lenses

- **Technical**: IoT networks, telecom infrastructure, smart home automation
- **Organizational**: Multi-party workflows, support routing, partner integrations
- **User-facing**: How complexity manifests in UX; how bad architecture breaks the interface

This means you don't design beautiful interfaces that reveal broken systems underneath. You ensure the system and interface are coherent.

You Pioneer at the Intersection of Emerging Domains

- Design tokens + AI context engineering = How do AI systems understand design systems?
- Smart home + medical devices = What makes complex systems coherent in high-stakes domains?
- Telecom infrastructure + civic technology = How do community networks work as product?
- Context engineering + design systems = How do we make information discoverable?

You're not just applying existing patterns; you're **discovering what patterns work** when emerging technologies meet established domains.

You Operate as Architect-Operator

Most roles are either:

- **Architects**: Define the shape, hand off to implementers
- **Operators**: Execute against existing architecture
- **Individual Contributors**: Build features

You do all three. You define architecture, validate it works by building, and maintain it as it evolves. This makes you uniquely effective because you understand **implementation constraints** when designing, and **architectural implications** when building.

The Domains You Master

You've built expertise across interconnected domains that most people keep separate:

Design Infrastructure

- Design systems at scale
- Token architecture and governance
- Component library + Storybook alignment
- Accessibility and color science
- Design-code coherence

Product Architecture

- API-as-product platforms
- Multi-party workflow orchestration
- Operational platform semantics
- Schema and data governance
- Multi-system coherence

Infrastructure & Networks

- Fiber and telecom platforms
- Smart home and IoT systems
- Distributed architecture
- Local-first and offline-first design
- Hardware integration

Emerging Technologies

- Context engineering for AI
- AI-driven UX and onboarding
- Semantic understanding and ontologies
- Visual reasoning and UI generation
- Quantum computing research interests

Specific Vertical Expertise

- Telecommunications and networks
- Financial systems and billing
- Medical and healthcare platforms
- Civic and community technology
- Smart home and home automation

Professional Identity

You are not a designer, engineer, product manager, or architect—though you do work in all these domains. You are someone who:

Architects product reality by defining how UX, systems, and operations coherently work together. You do this through:

- Semantic control (defining what things mean)
- Infrastructure thinking (treating coherence as foundational)
- Specification and communication (making implicit decisions explicit)
- Cross-domain expertise (understanding how specialties must integrate)
- Hands-on validation (building to prove architecture works)

Your rare combination:

- Design depth (tokens, systems, UI, accessibility)
- Technical breadth (APIs, infrastructure, IoT, ML)
- Product thinking (strategy, multi-party workflows, operations)
- Communication clarity (specs, diagrams, presentations)
- Pioneer mentality (emerging tech, novel domains, new patterns)

The impact you have:
Complex products that would otherwise fracture into contradiction become coherent. Teams that would otherwise misalign around implicit assumptions align around explicit specifications. Distributed systems that would otherwise fail through ambiguous handoffs work reliably. New team members understand the architecture from artifacts instead of

asking one person who knows. Partners integrate cleanly against clear contracts. The product scales without breaking.

What You're Seeking

Based on your history and interests, you appear to be seeking:

Intellectually challenging work at the intersection of multiple domains—not problems that fit neatly into one specialty, but problems requiring thinking across design, technology, operations, and emerging capabilities.

Leverage over direct management—building expertise and artifacts that multiply organizational capability, not necessarily managing large teams.

Specific expertise deep dives while maintaining breadth—exploring design tokens deeply while also understanding AI context engineering, smart home architecture, and telecom infrastructure.

Emerging technology frontiers—actively researching quantum computing, advanced LLM capabilities, and how design systems apply to AI-driven experiences.

Portfolio building and market positioning—maintaining coty.design as a showcase, contributing to open-source design infrastructure projects, establishing yourself as an authority in design systems + infrastructure thinking.

Part-time and consulting work at premium rates—leveraging specialized expertise (design systems, product architecture, context engineering) to generate additional revenue while maintaining autonomy.

Community and civic tech involvement—through Underline Technologies and Rotary involvement, contributing to projects that benefit communities, not just profit.

Conclusion

You are not a "design technologist" underselling yourself, and you are not a "product architect" who ignores design. You are something more specific:

An architect-operator who makes distributed product systems coherent—where "distributed" means across teams, specialties, organizations, and technologies. You do this through semantic control, infrastructure thinking, and explicit communication. You are rare because this combination requires both depth in multiple specialties and the integrating vision to see how they must work together.

Your job title might change with context. But your work—**defining product reality across UX, systems, and operations**—remains constant. And that work is increasingly valuable as products become more complex, organizations more distributed, and the cost of system incoherence higher.

This dossier captures not a job title, but your professional identity: an architect who makes the complex coherent, working in and across the domains that matter most.

---

kb/source/changelog.md

KB Source Changelog

[1.0.0] - 2025-12-15

Added

- Initial comprehensive dossier as source of truth
- All six core responsibilities documented
- Skills inventory across design, technical, product, and domain expertise
- How you work section (architectural thinking, infrastructure mindset, etc.)
- Value creation and impact areas
- Unique positioning analysis
- Domains mastered (design infrastructure, product architecture, infrastructure & networks, emerging tech, vertical expertise)
- Professional identity synthesis
- What you're seeking (career direction and values)

Status

- Ready for modular breakdown into kb/modules/*
- All sections are complete and human-reviewed

---

kb/profile.yaml

document_type: "agent_context_profile"
version: "1.0.0"
last_updated: "2025-12-15"

subject:
name: "Coty Beasley"
location: "Frankfurt, Hesse, Germany"
public_sites:
portfolio: "https://coty.design"
qa_agent: "https://ask.coty.design"
social:
linkedin: "https://linkedin.com/in/cbeasley0"
github: "https://github.com/beacrea"

identity_summary:
plain_language: >
Coty works at the intersection of product design, systems/platform thinking, and technical specification.
The focus is making complex products coherent across user experience, APIs, workflows, and operational systems.
technical_language: >
Architect-operator focused on semantic control across distributed product surfaces: experience layer,
platform contracts (APIs/schemas), operational workflows, and design infrastructure.

role_archetypes:

- id: "architect_operator"
  name: "Architect-operator"
  focus: "defines system shape, communicates rules, validates through implementation"
- id: "product_architecture"
  name: "Product architecture (experience + platform)"
  focus: "ensures UX and underlying system contracts align as coherent product"
- id: "design_systems"
  name: "Design systems & design engineering"
  focus: "builds token/component foundations and reduces design-to-code drift"
- id: "context_engineering"
  name: "Context engineering for AI"
  focus: "organizes knowledge/context so AI systems retrieve and reason effectively"

core_strengths:

- "Semantic control: prevents contradictions via naming, contracts, governance"
- "System coherence: aligns UX, APIs, ops workflows into one coherent product"
- "Artifact-driven alignment: produces specs, diagrams, templates that persist"
- "Cross-domain translation: bridges design, engineering, product, operations"

how_to_answer:
defaults:
tone: "clear, plain language"
length: "concise by default; expand if user requests depth"
safety_rules:
- "If asked for dates/employers/metrics/credentials not in KB: respond 'Not specified in context' and ask a clarifying follow-up."
- "Do not invent facts, employers, numbers, or credentials."
- "Treat operational/internal tool work as product work, not chores."
response_structure:
- "1–2 sentence direct answer"
- "Short bullets: relevant experience or skills"
- "If needed: definitions for non-technical users"
- "If needed: 'What to ask next' follow-ups"

capabilities:

- "Product architecture (APIs, multi-party workflows, ops semantics)"
- "Design systems infrastructure (tokens, theming, component governance)"
- "Context engineering for AI (knowledge organization, ontological strategies)"
- "Operational platforms (billing, support routing, provisioning)"
- "Technical communication (specs, diagrams, presentations)"
- "Design for emerging tech (AI interfaces, smart home, medical, civic)"

domains:

- "Networks & telecommunications"
- "Smart home automation & IoT"
- "Healthcare-adjacent systems (compliance-aware)"
- "Design infrastructure"

- "Financial systems & billing"
- "Civic & community technology"

kb_module_intent_routing:
identity_overview:
- "who are you"
- "what do you do"
- "summary"
- "background"
skills_and_tools:
- "skills"
- "tools"
- "experience"
- "resume"
- "design tokens"
- "vue.js"
- "figma"
product_architecture:
- "platform"
- "api"
- "workflow"
- "architecture"
- "multi-party"
operational_platforms:
- "billing"
- "support"
- "provisioning"
- "zuora"
- "operations"
context_engineering:
- "context engineering"
- "rag"
- "ontology"
- "knowledge base"
- "ai"
domains:
- "telecom"
- "smart home"
- "healthcare"
- "civic"
- "financial"
technical_communication:
- "specs"
- "diagrams"
- "communication"
- "presentations"
signals_of_fit:
- "when is coty a good fit"
- "what problems does coty solve"
- "best use case"

known_unknowns:

- "Current job titles and employers (not specified)"
- "Specific dates of employment or project timelines (not specified)"
- "Exact team sizes, budgets, or quantitative KPIs (not specified)"
- "Revenue or financial metrics from companies (not specified)"

---

kb/index.json

[
{
"id": "module.identity",
"path": "modules/module.identity.md",
"title": "Identity & Overview",
"keywords": ["who", "summary", "overview", "what do you do", "background"],
"audience_level": "general",
"updated_at": "2025-12-15"
},
{
"id": "module.role_archetypes",
"path": "modules/module.role_archetypes.md",
"title": "Role Archetypes",
"keywords": ["architect", "operator", "roles", "what is your role"],
"audience_level": "general",
"updated_at": "2025-12-15"
},
{
"id": "module.skills_matrix",
"path": "modules/module.skills_matrix.md",
"title": "Skills & Experience",
"keywords": ["skills", "tools", "experience", "resume", "design tokens", "vue", "figma",
"storybook", "openapi"],
"audience_level": "general",
"updated_at": "2025-12-15"
},
{
"id": "module.problem_types",
"path": "modules/module.problem_types.md",
"title": "What Problems Coty Solves",
"keywords": ["problems", "fit", "use case", "when", "best suited", "api", "platform", "workflow"],
"audience_level": "general",
"updated_at": "2025-12-15"
},
{
"id": "module.operational_platforms",
"path": "modules/module.operational_platforms.md",
"title": "Operational Platforms & Systems",
"keywords": ["billing", "support", "provisioning", "zuora", "operations", "internal tools",
"workflow"],
"audience_level": "general",

```
"updated_at": "2025-12-15"
},
{
"id": "module.context_engineering",
"path": "modules/module.context_engineering.md",
"title": "Context Engineering & AI",
"keywords": ["context engineering", "rag", "ontology", "knowledge", "ai", "llm"],
"audience_level": "technical",
"updated_at": "2025-12-15"
},
{
"id": "module.technical_communication",
"path": "modules/module.technical_communication.md",
"title": "Technical Communication & Specs",
"keywords": ["specs", "diagrams", "communication", "presentations", "artifacts"],
"audience_level": "general",
"updated_at": "2025-12-15"
},
{
"id": "module.domains",
"path": "modules/module.domains.md",
"title": "Domain Expertise",
"keywords": ["telecom", "smart home", "iot", "healthcare", "civic", "financial", "domains"],
"audience_level": "general",
"updated_at": "2025-12-15"
},
{
"id": "module.value_creation",
"path": "modules/module.value_creation.md",
"title": "Value Creation & Impact",
"keywords": ["value", "impact", "incoherence", "prevent", "solving"],
"audience_level": "general",
"updated_at": "2025-12-15"
},
{
"id": "module.signals_of_fit",
"path": "modules/module.signals_of_fit.md",
"title": "Signals of Good Fit",
"keywords": ["fit", "signals", "when is coty a good fit", "best use case"],
"audience_level": "general",
"updated_at": "2025-12-15"
},
{
"id": "module.faq",
"path": "modules/module.faq.md",
"title": "FAQ",
"keywords": ["faq", "common questions", "q&a"],
"audience_level": "general",
"updated_at": "2025-12-15"
}
]
```

kb/modules/module.identity.md

module.identity

Coty Beasley works at the intersection of product design, systems/platform thinking, and technical specification.
The focus is making complex products coherent across user experience, APIs, workflows, and operational systems.

Architect-operator: defines system shape, communicates rules, validates through implementation.
Designer + **Engineer** + **Product Thinker:** bridges these worlds while maintaining coherence.

What Coty does

- Architects product coherence across UX, platforms, and operations
- Prevents system incoherence through semantic control and explicit contracts
- Builds design systems, product architectures, and operational platforms
- Communicates complex systems through specifications and diagrams

Default behavior for unknowns

If asked for dates, employers, credentials, or metrics not in this knowledge base:

- Respond: "Not specified in context"
- Ask a clarifying follow-up
- Do not invent facts

---

kb/modules/module.role_archetypes.md

module.role_archetypes

Four primary role patterns

Architect-operator:

- Defines how complex systems should work
- Helps drive them into reality through practical constraints and artifacts
- Validates architecture by building

Product architecture (experience + platform):

- Keeps UX, APIs, and ops workflows consistent and aligned
- Ensures the product behaves as one coherent system

Design systems & design engineering:

- Builds token/component foundations

- Reduces design-to-code drift
- Enables teams to scale UI consistency

Context engineering for AI:

- Structures knowledge and context so AI systems retrieve reliably
- Designs how information flows from user behavior → AI understanding → personalized response

---

kb/modules/module.skills_matrix.md

module.skills_matrix

Design & UX

- Design systems architecture, design token taxonomy, theming (Figma variables/modes), accessibility-minded foundations
- Complex workflow UX for internal tools and partner-facing surfaces
- Color science (OKLCH, perceptual uniformity, accessibility)
- Component library architecture and design-to-code alignment

Technical / Engineering

- API specification (OpenAPI, contract design)
- Structured modeling (YAML, JSON, ER diagrams)
- Vue.js and modern web development
- Database schema design (Zuora, relational modeling)
- Event-driven architecture and naming conventions

Communication & Artifacts

- Specification writing (specs, diagrams, templates)
- Technical presentations for mixed audiences
- Mermaid diagramming and visual architecture
- Keeping design and code aligned through artifacts

Domain tools

- Figma, Token Studio (and alternatives), Storybook, Next.js, Cloudflare Pages/Workers, Home Assistant, ESPHome, Zuora, Git/GitHub

---

kb/modules/module.problem_types.md

module.problem_types

Coty is strongest when problems involve:

- **Experience** + **platform** + **ops** (not just UI or just backend)
- **Multi-party workflows** with handoffs, statuses, and traceability
- **API-as-product** concerns (discoverability, versioning, contract clarity)
- **Governance needs**: naming conventions, taxonomies, schemas that prevent contradictions
- **Design systems** that need to scale reliably across teams
- **AI/agent adoption** requiring structured context and retrieval discipline

Examples of good fit

- "Our APIs confuse partners because the schema is implicit"
- "Status semantics are ambiguous; teams disagree what 'processing' means"
- "We want a design system but design and code are drifting apart"
- "Our AI agent halluccinates because context is unstructured"
- "We have multi-party workflows with ambiguous handoffs"

---

kb/modules/module.operational_platforms.md

module.operational_platforms

Operational platform work

Treats billing, support, provisioning, and internal tools as product surfaces, not background chores.

Billing/financial systems:

- Invoice status semantics and filtering constraints
- Calculated statuses (e.g., AUTOPAID logic)
- Zuora object modeling (ProductRatePlanCharge, tiering, accounting codes)
- Subscription and tiered pricing models

Support & routing:

- Routing frameworks that distinguish internal vs. provider escalation
- Historical traceability across channels so requests don't get lost
- Status semantics that internal and external teams understand

Provisioning & workflows:

- Multi-party service delivery with explicit handoffs
- State machines and status definitions
- Cross-party boundary clarity (who knows what, who's responsible)

Internal tools:

- Treat as product surfaces; they shape operational efficiency
- Navigation and naming that prevent confusion at scale
- Multi-market scaling as an architectural problem

---

kb/modules/module.context_engineering.md

module.context_engineering

Context engineering for AI

Designing how AI systems get the right information at the right time.

Core activities:

- Organizing knowledge into modular, retrieval-friendly chunks
- Using structured representations (registries, schemas, ontology/graphs)
- Making context precise so AI retrieval is accurate
- Defining "not specified in context" behaviors to prevent hallucination

Emerging strategies:

- Knowledge graphs and OntoRAG for precise data discovery
- Ontological approaches that let LLMs reason about domain knowledge
- Flat-file registries with keyword indexing for fast, cheap retrieval
- Structured data encoding for compact, LLM-friendly representation

In practice:

- Designing how conversational AI understands what information exists
- Building project templates and bootstrap structures to reduce entropy
- Creating data dictionaries and canonical schemas

---

kb/modules/module.technical_communication.md

module.technical_communication

Artifacts that enforce alignment

Specifications:

- OpenAPI specs becoming canonical API contracts
- YAML object models readable by systems and humans
- Webhook payload definitions partners integrate against
- Event naming hierarchies and conventions

Diagrams:

- Mermaid workflow and entity-relationship models
- System boundary and component diagrams

- Visual narratives for mixed technical/non-technical audiences

Presentations:

- Technical presentations structured as: vision → impact → timing → construction
- Artifacts that outlive the meeting
- Clear communication under time constraint

Why it matters:
Specs are governance, not documentation. A well-written spec prevents ambiguity that surfaces as bugs, missed requirements, and frustrated teams.

---

kb/modules/module.domains.md

module.domains

Domains mastered

Networks & Telecommunications:

- Multi-party service delivery (subscriber ↔ company ↔ provider)
- Open-access fiber platforms (founded Underline Technologies)
- Provisioning and network architecture

Smart Home Automation & IoT:

- Home Assistant, ESPHome, WLed LED control systems
- Distributed IoT networks
- Custom hardware integration and local-first automation

Healthcare-adjacent:

- HIPAA compliance and high-stakes data governance
- Educational platforms (MCAT preparation)
- Medical device accessibility and compliance

Design Infrastructure:

- Design systems at scale (token taxonomy, component governance)
- Design-to-code alignment
- Accessibility and color science (OKLCH)

Financial & Billing:

- Zuora platform expertise
- Invoice and subscription semantics
- Accounting and revenue recognition

Civic & Community Technology:

- Open-access networks serving communities
- Community benefit and participation design

- Volunteer engagement and organizational structures

Cross-domain insight: These domains teach similar lessons about how to make complex distributed systems coherent.

---

kb/modules/module.value_creation.md

module.value_creation

Problem solved: system incoherence

The problem:
As products grow across UX, APIs, billing, workflows, and operations, they fragment:

- UI contradicts data model
- API docs contradict implementation
- Billing confuses customers and teams
- Support can't route because statuses are ambiguous
- Design tokens work in one product but break in another
- Multi-party workflows have implicit handoffs that fail

The root cause:
Teams optimize locally. Without semantic control and explicit contracts, contradictions accumulate.

How it gets prevented

Semantic control:

- Define what things mean (status, party, state)
- Create naming conventions that prevent ambiguity
- Build schemas that are canonical

Explicit contracts:

- APIs have clear interface specs
- Workflows have explicit handoffs and responsibilities
- Design systems define inheritance and constraints

Infrastructure thinking:

- Treat coherence as foundational, not optional
- Design systems, billing platforms, and workflows are first-class product surfaces
- Naming is governance

Impact

- Products scale without fracturing
- Teams align on implicit assumptions
- New members learn from artifacts, not from asking the one person who knows
- Partners integrate cleanly
- Rework decreases; velocity increases

---

kb/modules/module.signals_of_fit.md

module.signals_of_fit

When Coty's work is a strong fit

Technical signals:

- The problem spans experience + platform + operations (not just one layer)
- Multi-party workflows with ambiguous handoffs
- API-as-product where schema clarity is missing
- Design systems struggling to stay aligned with code

Organizational signals:

- Teams disagree on what statuses/entities/workflows mean
- Documentation is incomplete or contradicts implementation
- Design and engineering drift apart regularly
- Scaling from one domain to many reveals hidden contradictions

AI/context signals:

- Building AI agents that hallucinate because context is unstructured
- Need to organize knowledge for reliable retrieval
- Deploying AI requires formal semantics and contracts

Examples of good fit problems

- "We're adopting AI agents but agents can't find the right context"
- "Our billing platform confuses customers and our own team"
- "Internal tools grew organically; now they're unmaintainable"
- "Design tokens exist but design and code drift constantly"
- "Our API confuses partners; the schema isn't clear"

---

kb/modules/module.faq.md

module.faq

Common questions

Q: Is Coty a designer or engineer?
A: Both, but best described as an architect-operator who bridges design systems, platform semantics, operational workflows, and technical alignment artifacts. Not purely either.

Q: What's the biggest differentiator?
A: Semantic control: preventing contradictions across UX, APIs, and operations by defining shared meaning and contracts.

Q: What if something isn't known?
A: Respond "Not specified in context" and ask a clarifying follow-up. Never invent facts, employers, dates, or metrics.

Q: What is semantic control exactly?
A: Defining canonical rules, schemas, and naming conventions that prevent different teams from making contradictory decisions. It's governance made explicit.

Q: Can Coty work on smart home projects?
A: Yes. Deep expertise in Home Assistant, ESPHome, WLed, distributed IoT, and local-first architecture.

Q: What about healthcare/medical work?
A: Yes, including HIPAA compliance, educational platforms, accessibility, and device design constraints.

Q: Does Coty manage large teams?
A: Architect-operator role focused on leverage over direct management. Hands-on work, cross-functional alignment, technical authority.

Q: What consulting/part-time work is Coty open to?
A: Design systems, product architecture, context engineering, and technical specification. Premium rates for specialized work; ~$30k/year part-time goal.

MVP APP & WORKER CODE

app/package.json

```
{
"name": "ask-coty-app",
"private": true,
"scripts": {
"dev": "next dev",
"build": "next build",
"export": "next build && next export",
"lint": "next lint"
},
"dependencies": {
```

```
"next": "^15.0.0",
"react": "^19.0.0",
"react-dom": "^19.0.0"
}
}
```

app/next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
output: "export"
};

module.exports = nextConfig;
```

app/app/layout.tsx

```
export const metadata = {
title: "Ask Coty",
description: "A conversational Q&A agent about Coty Beasley's work."
};

export default function RootLayout({ children }: { children: React.ReactNode }) {
return (




<body style={{
margin: 0,
padding: 0,
fontFamily: "system-ui, -apple-system, Segoe UI, Roboto, sans-serif",
backgroundColor: "#0b0b0c",
color: "#f5f5f5"
}}>
{children}
</body>
);
}
```

app/app/manifest.ts

```
import { MetadataRoute } from 'next';

export default function manifest(): MetadataRoute.Manifest {
return {
name: "Ask Coty",
short_name: "Ask Coty",
description: "A conversational Q&A agent about Coty Beasley",
start_url: "/",
display: "standalone",
```

```
background_color: "#0b0b0c",
theme_color: "#0b0b0c",
scope: "/",
orientation: "portrait-primary",
icons: [
{
src: "/icon-192.png",
sizes: "192x192",
type: "image/png",
purpose: "any"
},
{
src: "/icon-512.png",
sizes: "512x512",
type: "image/png",
purpose: "any"
}
],
categories: ["productivity"],
screenshots: [
{
src: "/screenshot-540.png",
sizes: "540x720",
type: "image/png",
form_factor: "narrow"
}
]
};
}
```

app/app/page.tsx

```
'use client';

import { useState, useRef, useEffect } from 'react';

interface Message {
role: 'user' | 'assistant';
content: string;
}

export default function ChatPage() {
const [messages, setMessages] = useState<Message[]>([]);
const [input, setInput] = useState('');
const [loading, setLoading] = useState(false);
const [error, setError] = useState('');
const messagesEndRef = useRef<HTMLDivElement>(null);
const [copied, setCopied] = useState(false);

const scrollToBottom = () => {
messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
};
```

```
useEffect(() => {
scrollToBottom();
}, [messages]);

const handleSubmit = async (e: React.FormEvent) => {
e.preventDefault();
if (!input.trim() || loading) return;

  const userMessage = input.trim();
  setInput('');
  setMessages(prev => [...prev, { role: 'user', content: userMessage }]);
  setLoading(true);
  setError('');

  try {
    const response = await fetch('/api/chat', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        message: userMessage,
        messages: messages.map(m => ({ role: m.role, content: m.content }))
      })
    });

    if (!response.ok) throw new Error(`API error: ${response.status}`);

    const reader = response.body?.getReader();
    if (!reader) throw new Error('No response body');

    const decoder = new TextDecoder();
    let assistantMessage = '';

    while (true) {
      const { done, value } = await reader.read();
      if (done) break;

      const chunk = decoder.decode(value, { stream: true });
      const lines = chunk.split('\n');
```

```
      for (const line of lines) {
        if (line.startsWith('data: ')) {
          try {
            const json = JSON.parse(line.slice(6));
            if (json.text) {
              assistantMessage += json.text;
              setMessages(prev => {
                const updated = [...prev];
                if (updated[updated.length - 1]?.role === 'assistant') {
                  updated[updated.length - 1].content = assistantMessage;
                } else {
                  updated.push({ role: 'assistant', content: assistantMessage });
                }
                return updated;
              });
            }
          } catch (e) {
            // Silent JSON parse error
          }
        }
      }
    } catch (err) {
      setError(err instanceof Error ? err.message : 'An error occurred');
    } finally {
      setLoading(false);
    }
```

};

```
const handleClear = () => {
setMessages([]);
setError('');
};

const handleCopyLastResponse = () => {
const lastMessage = messages[messages.length - 1];
if (lastMessage?.role === 'assistant') {
navigator.clipboard.writeText(lastMessage.content);
setCopied(true);
setTimeout(() => setCopied(false), 2000);
```

}
};

return (

Ask Coty

Q&A about product architecture, design systems, and technical leadership

```
<div style={styles.messagesContainer}>
  {messages.length === 0 && (
    <div style={styles.emptyState}>
      <p style={styles.emptyStateText}>Start a conversation by asking about Coty'
      <div style={styles.suggestedQuestions}>
        <button
          onClick={() => {
            setInput('What does Coty do?');
          }}
          style={styles.suggestionButton}
        >
          What does Coty do?
        </button>
        <button
          onClick={() => {
            setInput('What design systems experience does Coty have?');
          }}
          style={styles.suggestionButton}
        >
          Design systems experience
        </button>
        <button
          onClick={() => {
            setInput('When is Coty a good fit?');
          }}
          style={styles.suggestionButton}
        >
          When is Coty a good fit?
        </button>
      </div>
```

```
    </div>
  )}

  {messages.map((msg, idx) => (
    <div key={idx} style={msg.role === 'user' ? styles.userMessage : styles.assista
      <div style={styles.messageBubble}>
        {msg.content}
      </div>
    </div>
  ))}

  {loading && (
    <div style={styles.assistantMessage}>
      <div style={styles.messageBubble}>
        <span style={styles.loadingDots}>Thinking</span>
      </div>
    </div>
  )}

  {error && (
    <div style={styles.errorMessage}>
      <strong>Error:</strong> {error}
    </div>
  )}

  <div ref={messagesEndRef} />
</div>

<div style={styles.controls}>
  {messages.length > 0 && (
    <div style={styles.buttonGroup}>
      <button onClick={handleCopyLastResponse} style={styles.secondaryButton
        {copied ? '✓ Copied' : 'Copy last response'}
      </button>
      <button onClick={handleClear} style={styles.secondaryButton}>
        Clear
      </button>
    </div>
```

```
      )}
    </div>

    <form onSubmit={handleSubmit} style={styles.form}>
      <input
        type="text"
        value={input}
        onChange={(e) => setInput(e.target.value)}
        placeholder="Ask a question..."
        disabled={loading}
        style={styles.input}
        autoFocus
      />
      <button
        type="submit"
        disabled={loading || !input.trim()}
        style={styles.submitButton}
      >
        Send
      </button>
    </form>
  </div>
```

);
}

const styles: { [key: string]: React.CSSProperties } = {
container: {
display: 'flex',
flexDirection: 'column',
height: '100vh',
backgroundColor: '#0b0b0c',
color: '#f5f5f5',
fontFamily: 'system-ui, -apple-system, Segoe UI, Roboto, sans-serif'
},
header: {
padding: '16px 20px',
borderBottom: '1px solid #333',
textAlign: 'center'
},
title: {
margin: '0 0 8px 0',
fontSize: '24px',

```
fontWeight: 'bold'
},
subtitle: {
margin: 0,
fontSize: '12px',
color: '#999'
},
messagesContainer: {
flex: 1,
overflowY: 'auto',
padding: '16px 20px',
display: 'flex',
flexDirection: 'column',
gap: '12px'
},
emptyState: {
textAlign: 'center',
marginTop: 'auto',
marginBottom: 'auto',
paddingBottom: '100px'
},
emptyStateText: {
fontSize: '14px',
color: '#999',
marginBottom: '24px'
},
suggestedQuestions: {
display: 'flex',
flexDirection: 'column',
gap: '8px',
alignItems: 'center'
},
suggestionButton: {
padding: '10px 16px',
backgroundColor: '#1a1a1a',
color: '#f5f5f5',
border: '1px solid #333',
borderRadius: '6px',
cursor: 'pointer',
fontSize: '13px',
maxWidth: '280px',
width: '100%'
},
userMessage: {
display: 'flex',
justifyContent: 'flex-end',
gap: '8px'
},
assistantMessage: {
display: 'flex',
justifyContent: 'flex-start',
```

```
gap: '8px'
},
messageBubble: {
maxWidth: '85%',
padding: '10px 14px',
borderRadius: '8px',
wordWrap: 'break-word',
fontSize: '14px',
lineHeight: '1.5'
},
userBubble: {
backgroundColor: '#2a7fff',
color: '#fff'
},
assistantBubble: {
backgroundColor: '#1a1a1a',
border: '1px solid #333',
color: '#e5e5e5'
},
loadingDots: {
animation: 'blink 1.4s infinite'
},
errorMessage: {
padding: '10px 14px',
backgroundColor: '#8b2a2a',
color: '#ffcccc',
borderRadius: '6px',
fontSize: '13px'
},
controls: {
padding: '12px 20px',
borderTop: '1px solid #333'
},
buttonGroup: {
display: 'flex',
gap: '8px',
justifyContent: 'center'
},
secondaryButton: {
padding: '8px 12px',
backgroundColor: '#1a1a1a',
color: '#f5f5f5',
border: '1px solid #333',
borderRadius: '4px',
fontSize: '12px',
cursor: 'pointer'
},
form: {
display: 'flex',
gap: '8px',
padding: '12px 20px',
```

```
borderTop: '1px solid #333',
backgroundColor: '#0b0b0c'
},
input: {
flex: 1,
padding: '10px 14px',
backgroundColor: '#1a1a1a',
color: '#f5f5f5',
border: '1px solid #333',
borderRadius: '6px',
fontSize: '14px',
outline: 'none'
},
submitButton: {
padding: '10px 20px',
backgroundColor: '#2a7fff',
color: '#fff',
border: 'none',
borderRadius: '6px',
cursor: 'pointer',
fontWeight: 'bold',
fontSize: '14px'
}
};
```

worker/package.json

```
{
"name": "ask-coty-worker",
"private": true,
"type": "module",
"scripts": {
"dev": "wrangler dev",
"deploy": "wrangler deploy"
},
"dependencies": {
"openai": "^4.47.0"
},
"devDependencies": {
"@cloudflare/workers-types": "^4.20250110.0",
"wrangler": "^3.83.0"
}
}
```

worker/wrangler.toml

```
name = "ask-coty-api"
main = "src/index.ts"
compatibility_date = "2024-12-31"
compatibility_flags = ["nodejs_compat"]
```

```
[env.production]
name = "ask-coty-api-prod"
routes = [
{ pattern = "ask.coty.design/api/*", zone_name = "coty.design" }
]

[observability]
enabled = true
```

worker/src/index.ts

```
import { OpenAI } from 'openai';
import kbProfile from './kb/profile.yaml';
import kbModules from './kb/modules.json';

interface KBModule {
id: string;
path: string;
title: string;
keywords: string[];
content: string;
}

interface ChatRequest {
message: string;
messages?: Array<{ role: 'user' | 'assistant'; content: string }>;
}

// Load KB modules at build time (these would be bundled)
const loadKBModule = (id: string): string => {
const moduleMap: Record<string, string> = {
'module.identity': # module.identity\n\nCoty Beasley works at the intersection of product design, systems/platform thinking, and technical specification...,
'module.role_archetypes': # module.role_archetypes\n\n## Four primary role patterns\n\n**Architect-operator**: Defines how complex systems...,
'module.skills_matrix': # module.skills_matrix\n\n## Design & UX\n- Design systems architecture...,
'module.problem_types': # module.problem_types\n\n## Coty is strongest when problems involve...,
'module.operational_platforms': # module.operational_platforms\n\n## Operational platform work...,
'module.context_engineering': # module.context_engineering\n\n## Context engineering for AI...,
'module.technical_communication': # module.technical_communication\n\n## Artifacts that enforce alignment...,
'module.domains': # module.domains\n\n## Domains mastered...,
'module.value_creation': # module.value_creation\n\n## Problem solved: system incoherence...,
'module.signals_of_fit': # module.signals_of_fit\n\n## When Coty's work is a strong fit...,
'module.faq': # module.faq\n\n## Common questions\n\n**Q: Is Coty a designer or engineer?**...
};
```

return moduleMap[id] || '';
};

const selectModules = (userQuery: string, allModules: any[]): string[] => {
const query = userQuery.toLowerCase();
const scored = allModules
.map((mod: any) => {
const keywordMatches = mod.keywords?.filter((k: string) =>
query.includes(k.toLowerCase())
).length || 0;
return { id: mod.id, score: keywordMatches };
})
.sort((a: any, b: any) => b.score - a.score)
.slice(0, 3)
.map((m: any) => m.id);

return scored.length > 0 ? scored : ['module.identity', 'module.faq'];
};

const buildPrompt = (selectedModules: string[], userQuery: string): string => {
const moduleContent = selectedModules
.map(id => \n## ${id}\n\n${loadKBModule(id)})
.join('\n');

return `You are an AI assistant providing accurate information about Coty Beasley based on a curated knowledge base.

CRITICAL RULES:

1. Only answer based on the knowledge base below. If the answer isn't in the KB, respond: "Not specified in context" and ask a clarifying follow-up.
2. Never invent facts, employers, dates, metrics, or credentials.
3. Be concise (under 200 words by default).
4. If asked something outside the knowledge base, admit it and offer to clarify what you're looking for.

KNOWLEDGE BASE:
${moduleContent}

User query: ${userQuery}

Respond directly and conversationally.`;
};

export default {
async fetch(request: Request, env: any): Promise<Response> {
const url = new URL(request.url);

```
if (request.method === 'OPTIONS') {
  return new Response(null, {
    headers: {
```

```
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Methods': 'GET, POST, OPTIONS',
      'Access-Control-Allow-Headers': 'Content-Type'
    }
  });
}

if (url.pathname === '/api/health' && request.method === 'GET') {
  return new Response(JSON.stringify({ status: 'ok', timestamp: new Date().toISO
    headers: { 'Content-Type': 'application/json' }
  });
}

if (url.pathname === '/api/chat' && request.method === 'POST') {
  const { message, messages = [] } = await request.json() as ChatRequest;

  if (!message?.trim()) {
    return new Response(JSON.stringify({ error: 'Message required' }), { status: 4
  }

  const apiKey = env.OPENAI_API_KEY;
  if (!apiKey) {
    return new Response(JSON.stringify({ error: 'API key not configured' }), { stat
  }

  try {
    const client = new OpenAI({ apiKey });

    const kbIndex = [
      { id: 'module.identity', keywords: ['who', 'what do you do', 'summary'] },
      { id: 'module.skills_matrix', keywords: ['skills', 'vue', 'figma', 'design tokens']
      { id: 'module.problem_types', keywords: ['problems', 'fit', 'when'] },
      { id: 'module.operational_platforms', keywords: ['billing', 'zuora', 'operations
      { id: 'module.context_engineering', keywords: ['ai', 'context', 'ontology'] },
      { id: 'module.domains', keywords: ['telecom', 'smart home', 'healthcare'] },
      { id: 'module.technical_communication', keywords: ['specs', 'diagrams', 'comm
      { id: 'module.signals_of_fit', keywords: ['fit', 'good fit', 'best use case'] },
      { id: 'module.faq', keywords: ['faq', 'question', 'common'] }
```

```
];

const selectedModules = selectModules(message, kbIndex);
const systemPrompt = buildPrompt(selectedModules, message);

const response = await client.chat.completions.create({
  model: 'gpt-4o-mini',
  messages: [
    { role: 'system', content: systemPrompt },
    ...messages.map(m => ({ role: m.role as 'user' | 'assistant', content: m.conter
    { role: 'user', content: message }
  ],
  stream: true,
  max_tokens: 350,
  temperature: 0.7
});

const encoder = new TextEncoder();
let buffer = '';

const readable = new ReadableStream({
  async start(controller) {
    try {
      for await (const chunk of response) {
        if (chunk.choices[0]?.delta?.content) {
          const text = chunk.choices[0].delta.content;
          const event = `data: ${JSON.stringify({ text })}\n\n`;
          controller.enqueue(encoder.encode(event));
        }
      }
      controller.enqueue(encoder.encode('event: done\ndata: {}\n\n'));
      controller.close();
    } catch (err) {
      controller.error(err);
    }
  }
});
```

```
      return new Response(readable, {
        headers: {
          'Content-Type': 'text/event-stream',
          'Cache-Control': 'no-cache',
          'Connection': 'keep-alive',
          'Access-Control-Allow-Origin': '*'
        }
      });
    } catch (err) {
      const errorMessage = err instanceof Error ? err.message : 'Unknown error';
      return new Response(JSON.stringify({ error: errorMessage }), { status: 500 });
    }
  }

  return new Response(JSON.stringify({ error: 'Not found' }), { status: 404 });
}
};
```

DEPLOYMENT CHECKLIST

Pre-deployment

- [ ] Create Cloudflare Pages project
- [ ] Create Cloudflare Worker project
- [ ] Purchase or configure ask.coty.design domain
- [ ] Generate OPENAI_API_KEY (https://platform.openai.com/api-keys)

Deployment steps

- [ ] Export Next.js app: `cd app && npm run export`
- [ ] Deploy to Cloudflare Pages: `wrangler pages deploy out/`
- [ ] Deploy Worker: `cd worker && npm run deploy`
- [ ] Set Worker secret: `wrangler secret put OPENAI_API_KEY`
- [ ] Configure Pages route: ask.coty.design/api/* → Worker
- [ ] Test /api/health
- [ ] Test /api/chat with streaming
- [ ] Verify PWA installable (Add to Home Screen)

Post-deployment monitoring

- [ ] Monitor error logs daily for first week
- [ ] Check token usage and costs
- [ ] Review cache hit ratio
- [ ] Gather initial user feedback
- [ ] Plan KB updates based on common questions

End of seed package – no placeholders, full content included.