Cosolvent Roadmap

> **Source whitepaper:** `MarketTheoryWP.md` — *Thin Markets: A Deep Dive into Market Physics and Engineering*
> **Codebase:** Cosolvent (Python/FastAPI)
> **Date:** 2026-02-19 (updated 2026-03-08)
> **Author:** Generated by Antigravity Agent for DeeperPoint

Preamble

This roadmap targets the Cosolvent codebase — a Python/FastAPI backend with working implementations of many core features described in the whitepaper. The document preserves the original roadmap's whitepaper coverage while:

1. Crediting what Cosolvent already has (no need to rebuild)
2. Identifying what needs to be **added** to reach whitepaper alignment
3. Flagging **unresolved architectural decisions** — these appear inline where they arise, and are collected in Appendix A for consolidated review

---

Executive Summary

The whitepaper defines Cosolvent as an open-source framework for thin market automation comprising eight core modules. Cosolvent already provides working implementations of several foundational features — marketplace configuration via YAML compiler, dynamic profile schemas, three-tier visibility, facilitator participant types, AI-assisted onboarding, vector search, admin oversight, communication, permissions, and document processing.

Scope Restriction: Cosolvent is deliberately scoped to be the *unopinionated matching engine* (handling AI extraction, vector matching, and deal architecture). It explicitly defers frontend UIs, detailed trust/verification protocols, payment/monetization engines, and digital twin simulation (like ClientSynth) to proprietary vertical customizations and parallel ecosystem tools. For detailed boundary definitions, see `Cosolvent-Vertical-Boundaries.md`.

The remaining work falls into three categories:

Category	Description	Sections
Already built	Features Cosolvent has that align with the roadmap	§1
Extension work	Roadmap features to add on the existing foundation	§2–§17
Unresolved decisions	Architectural choices requiring judgment before proceeding	Inline + Appendix A

Table of Contents

---

1. Foundation Already in Place

Cosolvent provides working implementations of the following. **No rebuild is needed** — these are the foundation the remaining roadmap builds on.

1.1 YAML-Driven Marketplace Configuration ✅

`marketplace.yaml` → `MarketplaceConfig` Pydantic model → deterministic compiler pipeline (normalize → hash → render → write → manifest). Includes drift detection (`compile --check`), managed output zones, and spec hashing. Three built-in presets.

Roadmap equivalent: Section 21 (Slots Architecture) — Cosolvent has a more opinionated and auditable implementation than earlier designs described.

1.2 Dynamic Profile Schemas ✅

`schema_engine.py` generates runtime Pydantic models from `marketplace.yaml` field definitions via `pydantic.create_model()`. Profile data is validated against these dynamic models. Includes completeness calculation.

1.3 Three-Tier Visibility ✅

`visibility_engine.py` enforces `public` / `protected` / `private` visibility per field, based on viewer authentication status and ownership.

Roadmap equivalent: Section 4 (Three-Layer Architecture) — functionally equivalent for the profile layer. Extensions needed for the Source Documents layer (see §2).

1.4 Facilitator Participant Type ✅

`ParticipantType` model includes `role: Literal["supply", "demand", "facilitator"]`. Config supports 2–3 participant types.

1.5 AI-Assisted Onboarding ✅

`ai_extraction.py` extracts structured fields from document text via LLM. `ai_generation.py` generates profile summaries. Both wired into onboarding config with per-participant-type flags.

1.6 Vector Search with pgvector ✅

`vector_service.py` implements cosine-distance search with metadata filtering, profile field filtering, and two retrieval modes (`hybrid` and `rag_strict`). `indexer.py` generates embeddings from searchable profile fields via OpenAI `text-embedding-3-small`.

1.7 Admin Oversight ✅

`admin/service.py` provides dashboard stats, user management, application approval/rejection, conversation oversight, LLM settings management, prompt management, document management, and FAQ CRUD.

1.8 Prompt Management ✅

`prompt_manager.py` implements a template system with DB-backed custom prompts and fallback defaults. Four default prompt intents: `rag_query`, `follow_up`, `profile_generation`, `document_extraction`.

1.9 Communication System ✅

`communication/` module with create/accept/reject/close lifecycle, message send/edit/delete, WebSocket support, asset sharing, and config-driven communication rules. Human-to-human only — extensions needed for AI-mediated and deal-scoped communication (see §3, §7).

1.10 Permission Engine ✅

`permission_engine.py` checks participant permissions against config, validates conversation initiation rights, and enforces onboarding requirements.

1.11 Document Processing Pipeline ✅

`document_processor.py` implements text chunking (1000 chars, 200 overlap), embedding via OpenAI, and indexing to `ai_document_chunks`. Background processing via ARQ workers. Text only — VLM and STT not yet implemented (see §6).

1.12 File Management, Notifications, Testing, Deployment, CLI ✅

- **Files:** S3 backend, public/private privacy, presigned URLs, 25MB limit.
- **Notifications:** Basic CRUD plumbing in place.
- **Tests:** 36 unit tests covering config, compiler, permissions, visibility, schemas, discovery, profiles, communication, files, auth, database, Redis, queue.

- **Deployment:** Docker Compose, Makefile, health checks, graceful shutdown, Redis-optional startup.
- **CLI:** 7-step wizard plus browser-based setup panel with presets, validation, YAML preview.

---

2. Three-Layer Information Architecture — Extensions

What Cosolvent has

Three-tier field visibility (`public` / `protected` / `private`). Binary file privacy (`public` / `private`).

What's needed

2.1 — Separate Gallery Profile from Matching Profile

Add `gallery_profile` and `matching_profile` separation to profile data. The gallery stores curated, user-approved information for browsing. The matching profile stores richer data (budget ranges, capacity constraints, pricing flexibility, inferred preferences) used by the AI but never displayed. Cosolvent's visibility engine can be extended: `public` → gallery, `protected`/`private` → matching profile only.

2.2 — Expand Document Privacy Levels

Expand file privacy from binary to multi-level: `public`, `gallery`, `match-only`, `ai-only`, `private`. Add `ai_extractable: bool` and `extraction_target` (`gallery`, `matching`, `both`, `none`) fields.

2.3 — Privacy-Aware Extraction Routing

Route extracted data to the correct profile layer based on document privacy settings. Re-route when privacy changes.

2.4 — Dual Embedding Pipeline

Gallery search uses gallery-only embeddings. Matching search uses richer embeddings. `profile_vectors` may need an `embedding_type` column.

> ⚠️ **CONFLICT C4 — Embedding Model Lock-In** applies here. The embedding dimension (1536) is hardcoded in `db_schema.py` and `indexer.py`. See Appendix A, C4.

2.5 — Gallery Profile Editor

UI for users to review, edit, and approve their public gallery profile.

> ⚠️ **CONFLICT C5 — Frontend Strategy** applies here. Cosolvent has no user-facing frontend. See Appendix A, C5.

---

3. Multilateral Marketplace — Extensions

What Cosolvent has

Three participant roles (`supply`, `demand`, `facilitator`). Config-driven participant types. Conversation system with lifecycle management.

> ⚠️ **CONFLICT C3 — Participant Type Limit** applies here. Cosolvent enforces a maximum of 3 participant types. See Appendix A, C3.

What's needed

3.1 — Deal Entity

> ⚠️ **CONFLICT C1 — JSONB vs. Relational** applies here. The Deal data model needs referential integrity (linking principals, facilitators, role slots, progression states). See Appendix A, C1.

Introduce a Deal data model with: principals involved, product/service transacted, route, volume/value/timeline, quality requirements, and **role slots** for facilitator roles (status: `needed`, `searching`, `proposed`, `confirmed`, `not-needed`).

3.2 — Deal-Triggered Facilitator Search

When a buyer-seller match progresses to deal structuring, analyze deal requirements and search for facilitators whose capability profiles match. A second matching pattern distinct from participant-to-participant.

3.3 — Deal Assembly UI

> ⚠️ **CONFLICT C5 — Frontend Strategy** applies here.

Deal view with parameters, role slot status, AI-recommended facilitators, attached facilitators.

3.4 — Facilitator Dashboard

Deal feed, active engagements, capacity management.

3.5 — Communication Extensions

- **Match-scoped messaging** — AI creates a scoped channel for introductions.
- **Deal-scoped messaging** — multi-party communication within a deal context.
- **Progressive trust gating** — channels unlock as trust stages progress (see §9).

3.6 — Handoff Artifact

The platform's primary deliverable — a structured output assembled from profiles, matching signals, conversation context, shared documents, facilitator recommendations, and regulatory flags. Template is admin-configurable per marketplace deployment, connected to the YAML configuration.

---

4. Semantic Matching Engine — Extensions (Module 1)

What Cosolvent has

pgvector search with cosine distance, metadata filtering, `hybrid` and `rag_strict` modes. Single embedding from searchable fields.

What's needed

- **4.1 — Bidirectional matching** — mutual preference matching considering both parties' profiles.
- **4.2 — Multi-signal embedding enrichment** — structured templates blending multiple profile signals, different templates per participant type.
- **4.3 — Match rationale generation** — LLM-generated "why this match" explanations respecting privacy boundaries.
- **4.4 — Generative preference elicitation** — conversational discovery of requirements replacing free-text search.
- **4.5 — Three search modes** — gallery search (public profiles), participant-to-participant match search (deep, private signals), deal-to-facilitator search (deal requirements → service providers).

---

5. Trusted Intermediary Protocol (Module 2)

What Cosolvent has

Nothing. Participants communicate directly.

What's needed

Operationally implemented through the three-layer architecture (§2):

- **Confidential matching pipeline** — LLM reads from both gallery and matching profiles, evaluates fit, reveals only *that* a match exists, not the underlying sensitive data.
- **Structured disclosure gates** — parties progressively share information by elevating document privacy levels.
- **Privacy-respecting rationale** — match explanations must not leak private signals.

---

6. Multimodal Input Pipeline — Extensions (Module 3)

What Cosolvent has

Text-based document processing (chunking + embedding + indexing). AI extraction from text. No voice or image.

What's needed

- **6.1 — VLM integration** — image-based document extraction (certificates, invoices, product photos).
- **6.2 — STT integration** — Whisper or equivalent, multilingual support.
- **6.3 — Natural language listing creation** — voice/text → structured listing.
- **6.4 — Privacy-aware extraction routing** — connect multimodal extraction to the three-layer model (§2.3).

---

7. Asynchronous Brokerage Agents (Module 4)

What Cosolvent has

Nothing. Human-to-human conversation only.

What's needed

- **User Agent entity** — each participant configures an AI agent with negotiation parameters, authority levels, persona, tool access.
- **Asynchronous conversation engine** — multi-turn, state-persisted across days, time-zone-aware, with human escalation.
- **Deal progression workflow** — inquiry → qualification → negotiation → deal structuring → human approval → Handoff Artifact.
- **Notification integration** — leverages existing notification plumbing.

---

8. Memory and Context Management (Module 5)

What Cosolvent has

RAG chat threads (basic conversation history). No per-user preference memory, no interaction logging, no anticipatory matching.

What's needed

- **User interaction log** — search queries, match views, rejections, conversation turns, linked to users.
- **Preference evolution analysis** — LLM-driven pattern detection over interaction history.
- **Anticipatory matching** — proactive notifications when new listings match inferred needs.
- **Deal outcome data** — record outcomes for completed/failed transactions.
- **Institutional memory** — interaction knowledge persists across participant turnover.

---

9. Trust Gradation Framework — Extensions (Module 6)

What Cosolvent has

Binary trust: admin approves or rejects applications. No graduated model, no verification, no reputation.

What's needed

- **9.1 — Verification pipeline** — auto-analyze documents for consistency, cross-reference external data, assign verification confidence.
- **9.2 — Reputation system** — track transaction outcomes, response times, dispute rates, counterparty ratings.
- **9.3 — Progressive trust stages:**
    1. Anonymous browsing → public gallery only
    2. Verified profile → AI begins using `ai-only` documents
    3. Guided introduction → AI-mediated contact; gallery profiles shared

4. Structured information exchange → selective document disclosure
5. Protected transaction → escrow/insurance/dispute resolution
6. Post-transaction evaluation → bidirectional ratings

- **9.4 — Transparent matching** — surface AI reasoning while respecting privacy boundaries.

---

10. Dynamic Pricing Module (Module 7)

What Cosolvent has

Nothing. No pricing concepts in the data model.

What's needed

> ⚠ **CONFLICT C1 — JSONB vs. Relational** applies here. Transaction history and price comparables need relational integrity. See Appendix A, C1.

- **Pricing fields** — asking prices, ranges, currency, unit, conditions on listings.
- **Transaction history model** — completed deals with actual prices, quality attributes, ratings.
- **Fair-value estimation service** — comparable analysis, confidence-banded estimates.
- **Pricing in match results** — fair-value estimates alongside recommendations.

---

11. Dispute Resolution Pipeline (Module 8)

What Cosolvent has

Nothing.

What's needed

> ⚠ **CONFLICT C1 — JSONB vs. Relational** applies here. Dispute records with evidence chains need referential integrity. See Appendix A, C1.

- **Disputes data model** — claims, evidence, descriptions.
- **AI triage** — classify, suggest resolutions for minor issues, escalate complex cases.
- **Predictive risk scoring** — flag high-risk in-progress deals.

---

12. User Aggregation & Cooperatives

What Cosolvent has

Nothing. Participants are individuals only.

What's needed — three implementation tiers

Tier 1 — Group as marketplace participant: Add a `group` participant type with membership list and designated manager. Group participates in matching as a single entity.

Tier 2 — Member-aware aggregation: Members submit data via low-bandwidth channels (SMS, WhatsApp, field agent). Group profiles computed from aggregated member data. Supply schedules and

order allocation tracking.

Tier 3 — Cooperative management (vertical-specific): Revenue distribution, governance, seasonal planning, certification management.

---

13. Psychological Framing & Personalization

What Cosolvent has

Nothing. All users see the same content.

What's needed

- **Behavioural analytics tracking** for interaction patterns.
- **Psychographic classification module** (primarily a prompt-engineering task on top of existing LLM service).
- **Dynamic message framing** in chatbot, match notifications, and listing displays.

---

14. Sales & Proactive Outreach

What Cosolvent has

Nothing. Entirely passive.

What's needed

- **Proactive matching notifications** — alert users when new listings match their needs.
- **Outreach generation** — LLM-crafted personalised messages explaining match relevance.
- **External signal monitoring** (future) — RSS, news, procurement feeds to identify potential participants.

---

15. Geographic & Temporal Distance Modelling

What Cosolvent has

Profile schemas support a `location` field type. No geo-aware search, distance calculation, or temporal matching.

What's needed

- **Geolocation data** (lat/long). Logistics cost estimation and distance filtering.
- **Temporal availability models** — production/availability windows on listings, desired delivery windows for buyers.
- **Temporal matching** — supply/demand window overlap scoring.
- **Time-zone-aware communication** for brokerage agents (§7).
- **Economic shipping radii** as configurable parameters per product category.

---

16. Framework Generalization — Extensions

What Cosolvent has

YAML-driven configuration, dynamic schemas, deterministic compiler, managed output zones, drift detection, prompt management, admin API, setup wizard.

What's needed

16.1 — Multi-Provider AI Abstraction

> ⚠️ **CONFLICT C4 — Embedding Model Lock-In** applies here. See Appendix A, C4.

Partially built. Cosolvent now has a provider-agnostic abstraction layer supporting OpenAI, OpenRouter, and Google Gemini:

- ✅ `providers.py` — `PROVIDER_REGISTRY` with per-provider `base_url`, `supports_embeddings`, `default_embedding_model`, `default_embedding_dimensions`, and `api_key_env_name`.
- ✅ `client_factory.py` — `get_chat_client()` and `get_embedding_client()` resolve provider dynamically from DB config.
- ✅ `embedding_client.py` — provider-agnostic `get_embedding()` and `get_embeddings_batch()`.
- ✅ `model_fetcher.py` — dynamic model list fetching with TTL cache for all three providers.
- ✅ `llm_client.py` — `generate()` resolves provider/model from DB settings, no longer hardcoded to OpenAI.

Remaining work:

- Task-level routing — different models for different tasks within the same pipeline.
- Prompt-to-model binding — prompts specify preferred model.
- Fallback chains — ordered fallback lists per service.
- Additional providers — Anthropic, HuggingFace, Ollama, custom REST endpoints.

16.2 — Knowledge Slot (Reference Library)

`ai_document_chunks` provides general RAG. A separate reference library is needed for sponsor-curated domain knowledge.

- **Separate reference docs from participant docs.** New `reference_library` table.
- **Vertical-specific metadata schema** — tag vocabulary defined per vertical.
- **Document curation workflow** — admin uploads, tags, describes, versions documents.
- **Metadata-filtered vector search** — pre-filter by metadata, then rank by similarity.
- **User-context scoping** — inject participant metadata as implicit retrieval filters.
- **Domain Q&A integration** — chatbot supports "domain knowledge" mode.

16.3 — UI Customization Layers

- **Terminology** — `ui_labels` mapping framework concepts to vertical-specific names.

- **Language translation** — LLM-assisted locale file generation; runtime translation for dynamic content.
- **Visual branding** — CSS custom properties for theme configuration.

16.4 — AI-Assisted Market Configuration (future)

AI assistant in admin that generates config suggestions from conversational market description.

> ⚠️ **CONFLICT C2 — Build-Time vs. Runtime Config** applies here. This feature's feasibility depends on whether structural config can be modified at runtime. See Appendix A, C2.

16.5 — MCP Client Integration

MCP client for external data source/tool connectivity.

---

17. Infrastructure & Operations — Extensions

What Cosolvent has

Docker Compose, Makefile, Redis + ARQ workers, Postgres + pgvector, S3, health checks, graceful shutdown.

What's needed

- **Structured logging** — shared module, JSON formatter, environment-variable log levels, request correlation IDs.
- **LLM call observability** — model, token counts, latency, cost estimate, service name, success/failure per call.
- **Notification service expansion** — email, SMS, push. Extend existing `notifications/` plumbing.
- **Scheduled job runner** — reputation recalculation, preference evolution analysis, predictive risk scoring.
- **Vector database scaling** — HNSW indexing, performance monitoring.
- **Multi-tenancy** — tenant isolation or vertical scoping.
- **API gateway** — rate limiting, API keys, usage tracking.
- **Encryption at rest** — for confidential matching profile data.
- **Privacy audit log** — track document privacy changes for compliance.

---

18. Prioritised Implementation Phases

This phasing assumes Cosolvent's existing codebase and architecture. Conflict decisions (Appendix A) are noted as gating items.

Phase 0 — Observability & Hygiene

No dependencies. Start immediately. ~1 week.

#	Item
0.1	Structured logging module with JSON formatter and env-variable log level
0.2	Request correlation IDs via middleware
0.3	Replace any `print()` with structured `logger` calls
0.4	LLM call observability (model, tokens, latency, cost per call)
0.5	Privacy audit log for document privacy changes

Phase 1 — Three-Layer Architecture & Multilateral Foundation

Builds on Cosolvent's existing profile and visibility infrastructure. 4–6 weeks.

#	Item	Dependency
1.1	Add gallery/matching profile separation	None
1.2	Expand file privacy to multi-level	None
1.3	Build gallery profile editor UI	1.1, **C5**
1.4	Privacy-aware extraction routing	1.2, §1.5
1.5	Dual embedding pipeline (gallery + matching)	1.1, **C4**
1.6	Bidirectional matching	1.5
1.7	Match rationale generation	1.6
1.8	**Resolve C1** — architectural decision before Deals	—
1.9	Deal data model with role slots	1.8
1.10	Deal-triggered facilitator search	1.9
1.11	Handoff Artifact generator (admin-configurable template)	1.9, 1.7

Phase 2 — Trust, Verification & Admin

3–5 weeks.

#	Item	Dependency
2.1	Participant verification pipeline	Phase 1
2.2	Reputation/trust score model	Phase 1
2.3	Progressive trust stages with disclosure gating	2.1, 2.2, 1.2
2.4	Transparent matching (privacy-respecting AI reasoning)	1.7
2.5	~~Multi-provider AI abstraction~~ (§16.1) — **partially complete** (OpenAI, OpenRouter, Gemini). Remaining: task routing, additional providers	None
2.6	Prompt-to-model binding + fallback chains	2.5

#	Item	Dependency
2.7	**Resolve C2** if runtime structural config desired	—

Track A — Marketplace Depth

After Phase 2. Makes the platform better at facilitating deals.

A1 — Deals, Agents & Memory

#	Item	Dependency
A1.1	User interaction logging	Phase 1
A1.2	Preference evolution analysis	A1.1
A1.3	Anticipatory matching notifications	A1.2
A1.4	Match-scoped and deal-scoped communication	1.9, §1.9
A1.5	User Agent entity and configuration	Phase 2
A1.6	Asynchronous conversation engine	A1.5
A1.7	Deal progression workflow → Handoff Artifact	A1.6, 1.11
A1.8	Notification service expansion (email, SMS, push)	§1.12

A2 — Pricing, Aggregation & Disputes

#	Item	Dependency
A2.1	Transaction history data model	Phase 1
A2.2	Fair-value estimation service	A2.1
A2.3	Pricing in match results and Handoff Artifact	A2.2
A2.4a	Collective Participant Tier 1 (group as participant)	Phase 1
A2.4b	Collective Participant Tier 2 (data aggregation)	A2.4a
A2.5	Dispute data model and AI triage	Phase 2
A2.6	Predictive risk scoring	A2.5, A2.1

Track B — Platform Breadth

After Phase 2. Makes the platform accessible to more markets and participants.

B1 — Input, Framework & Knowledge

#	Item	Dependency
B1.1	VLM integration	None

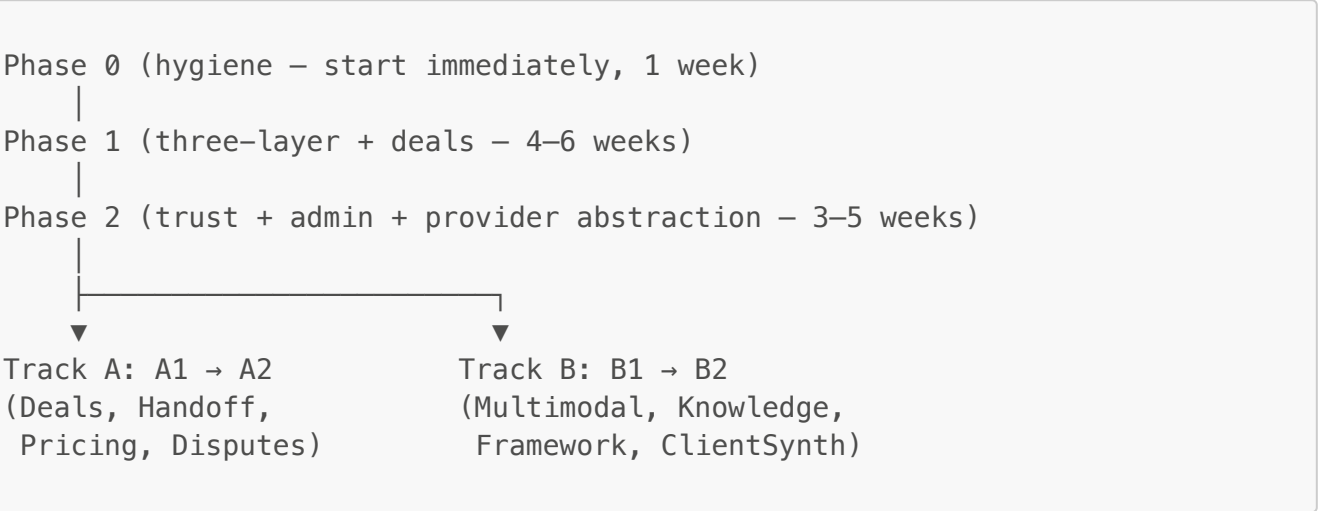
#	Item	Dependency
B1.2	STT integration (Whisper)	None
B1.3	Natural language listing creation	B1.1, B1.2
B1.4	Knowledge Slot reference library (§16.2)	Phase 2
B1.5	Metadata-filtered vector search for reference docs	B1.4
B1.6	UI customization layer (terminology, i18n, theme)	Phase 2
B1.7	MCP client integration	Phase 2
B1.8	Market Physics Scorecard model	None

B2 — ClientSynth, Digital Twins & Global Scale

> ⚠️ **CONFLICT C6 — ClientSynth Integration** applies here. See Appendix A, C6.

#	Item	Dependency
B2.1	Define Cosolvent ↔ ClientSynth API contract	B1.6
B2.2	Synthetic mode (clearly labelled synthetic participants)	B2.1
B2.3	Digital Twin simulation harness	B2.2, B1.8
B2.4	Geolocation and logistics estimation	None
B2.5	Regulatory context module	B1.7
B2.6	WhatsApp/SMS/USSD interface layer	B1.3, A1.8
B2.7	Multi-tenancy	Phase 2

Critical Path



Estimated calendar time to demo-able state: Phase 0 + Phase 1 + Phase 2 + max(A1, B1) = **14–20 weeks**

This is shorter than earlier estimates because Cosolvent starts with a significantly more complete foundation.

What the Handoff Artifact Demo Requires

The Handoff Artifact is the convergence point — the demo showing the platform working end-to-end:

- ✅ Profile infrastructure (existing)
- ✅ Search infrastructure (existing)
- ✅ Communication infrastructure (existing)
- Phase 1: Gallery/matching split, bidirectional matching, deals, rationale
- Phase 2: Trust model, verification
- A1: Agent conversation, deal progression
- A frontend (**C5**)

Cross-Cutting Principles

These principles from the whitepaper guide implementation across all phases.

A — Why thin markets are different

1. **Structural desire must exist.** AI can accelerate discovery but cannot create demand that does not exist.
2. **Test with thin-market dynamics.** Few participants, infrequent transactions, high stakes per transaction.
3. **Trust is the prerequisite, not a feature.** Every new capability: does this increase or decrease willingness to engage?

B — What the platform does

4. **The framework defines structure; the vertical defines content.** Cosolvent's YAML compiler already embodies this.
5. **Deals need more than two parties.** Real transactions require facilitators.
6. **The platform's job is to get parties to the table, not to run the table.** V1 ends at a Handoff Artifact.

C — How information flows

7. **Privacy is a prerequisite.** Fewer participants = more identifiable data.
8. **Gallery is for discovery, matching is for depth.** Never conflate the two.
9. **Users own their information boundaries.** Per-document privacy must be editable.
10. **Communication is scoped, not open.** Match or deal contexts, not general messaging.
11. **Never destroy information through premature standardisation.** Cosolvent's dynamic schemas embody this.

D — How we implement it

12. **Design for cognitive bandwidth constraints.** Curated subsets, not raw data dumps.

13. **Prompt-driven, not code-driven.** Cosolvent's prompt management supports this.

Appendix A — Conflict Register

The following architectural decisions require judgment calls before implementation proceeds. Each is referenced inline in the sections where it applies.

C1: JSONB Document Store vs. Typed Relational Tables

Referenced in: §3.1, §10, §11, Phase 1 item 1.8

Cosolvent's approach: All domain data stored in generic JSONB document tables via `CollectionProxy`. Maximum flexibility — any field without migration — but no referential integrity, no efficient joins, no database-enforced business rules.

Roadmap's assumption: Typed tables for deals, transactions, trust scores, dispute records, group memberships, price comparables — all requiring relational integrity.

Options:

Option	Description	Pros	Cons
A — Hybrid	Keep JSONB for flexible entities (profiles, messages, notifications). Typed relational tables for relational entities (deals, transactions, disputes, trust scores).	Best of both worlds. Cosolvent already does this partially (`profile_vectors`, `ai_document_chunks` are typed).	Two data access patterns to maintain.
B — Full JSONB	Everything stays in the document model.	Maximum flexibility. No schema migrations.	No referential integrity. Complex queries on related entities become expensive application-level joins.
C — Full relational	Migrate all entities to typed tables.	Full database integrity. Standard ORM patterns.	Loses dynamic schema flexibility for profiles. Major migration effort.

Blocking: Deals (§3.1), pricing (§10), disputes (§11), trust scores (§9.2), cooperatives (§12).

C2: Build-Time vs. Runtime Configuration

Referenced in: §16.4, Phase 2 item 2.7

Cosolvent's approach: Configuration is a build-time artifact. YAML compiled into generated code, database migrations, and OpenAPI specs. Changes require recompilation and redeployment.

Roadmap's assumption: Marketplace configuration managed at runtime through admin UI, stored in database, changeable without redeployment.

Already partially resolved: Cosolvent allows runtime changes for some settings — LLM parameters, prompts, admin overrides via admin API. The compiler approach only applies to structural config (participant types, schemas, communication rules).

Options:

Option	Description	Pros	Cons
A — Accept current hybrid	Structural config at build-time (auditable, deterministic). Operational settings at runtime.	Strongest engineering pattern in the codebase. Testable. Auditable via drift detection.	Non-technical operators cannot add profile fields without developer involvement.
B — Admin-triggered recompilation	Admin UI modifies `marketplace.yaml` and triggers recompile + hot-reload.	Preserves compiler pipeline. Enables non-technical operators.	Adds complexity (in-place compilation, hot-reload, rollback).
C — Runtime-only config	Replace compiler with DB-stored config and runtime schema management.	Maximum flexibility for operators.	Loses determinism, drift detection, audit trail. Major architectural change.

Blocking: AI-assisted market configuration (§16.4). Affects framework generalization (§16) and the "less-skilled developer" deployment goal.

C3: Participant Type Count Limit

Referenced in: §3

Cosolvent's approach: `Maximum 3 participant types for MVP` validation rule in `MarketplaceConfig.cross_validate()`.

Roadmap's assumption: No explicit limit. Multi-party marketplaces with multiple facilitator subtypes (customs broker, shipper, inspector, trade finance, insurance).

Options:

Option	Description	Pros	Cons
A — Relax to 5–8 now	One-line change in validation.	Accommodates whitepaper scenarios. No other code changes needed.	Slightly more config surface to test.

Option	Description	Pros	Cons
B — Keep at 3	MVP constraint prevents scope creep.	Simpler testing. Forces prioritisation.	May force artificial compromises in marketplace design.

Impact if deferred: Low — one-line change when needed.

C4: Embedding Model Lock-In

Referenced in: §2.4, §16.1

Cosolvent's approach (partially resolved): The provider registry (`providers.py`) now defines `default_embedding_dimensions` per provider (1536 for OpenAI, 768 for Gemini), and the embedding client (`embedding_client.py`) resolves the provider/model dynamically from DB config. However, `db_schema.py` still hardcodes `Vector(1536)` for both `ai_document_chunks` and `profile_vectors` tables, meaning switching to a non-1536-dimension provider (e.g., Gemini at 768) would require a database migration and re-embedding.

Roadmap's assumption: Multi-provider AI with configurable embedding models, shared dimensions across all vector stores.

Options:

Option	Description	Pros	Cons
A — Parameterise now	Extract `1536` to config. Add migration path for re-embedding.	Low cost now. Avoids painful re-embedding later.	Minor additional abstraction.
B — Defer	Leave `Vector(1536)` in schema until a provider switch is actually needed.	No work now.	Switching models later requires re-embedding all vectors — a batch operation proportional to dataset size.

Impact if deferred: Moderate. Re-embedding is expensive but not architectural — it's a batch migration. The provider abstraction layer is already in place; only the database column width is locked.

C5: Frontend Strategy

Referenced in: §2.5, §3.3, Phase 1 item 1.3

Cosolvent's approach: No user-facing frontend. API-only backend with operator setup panel.

Alternative considered: Adapting a previous Next.js/React frontend would likely be more work than building new, since API shapes differ significantly.

Options:

Option	Description	Pros	Cons
A — Build new frontend	Choose framework (Next.js, SvelteKit, etc.) and build against Cosolvent's API.	Clean start. API-first design.	Significant effort. Framework choice needed.
B — Adapt existing frontend	Rework a previous Next.js frontend to target Cosolvent's API shapes.	Existing code to build on.	API shapes are completely different. Likely more work than option A.
C — Remain headless	Cosolvent is a backend framework only. Each vertical builds its own frontend.	Minimal framework work. Verticals control UX.	Every user-facing feature is someone else's problem. Demo requires a frontend regardless.

Blocking: Every user-facing feature — gallery browsing, profile management, deal assembly, dashboards, the Handoff Artifact demo.

C6: ClientSynth & Digital Twin Integration

Referenced in: Track B2

Cosolvent's approach: No synthetic user concept. No ClientSynth integration.

Roadmap's assumption: ClientSynth generates synthetic participants for testing, demo, and cold-start bootstrapping. Digital Twins combine Cosolvent + ClientSynth for full market simulation.

Restriction (from GEMINI.md): Synthetic profiles must **never** be used simultaneously with real user profiles. They are for populating prototype websites for testing and demonstration only. A real-world marketplace must be built without synthetic users — recruitment is the sponsor's responsibility.

Options:

Option	Description	Pros	Cons
A — Define API contract now	Specify how ClientSynth generates participants conforming to `marketplace.yaml` schemas. Build integration later.	Enables parallel development. Low effort.	Work product may need revision when integration is actually built.
B — Defer entirely	Build real functionality first. Add synthetic population later.	No premature design. Focus on core.	Demo requires manual test data creation until integration exists.

Impact if deferred: Low for core functionality. Higher for demonstration and investor-facing demos.

This roadmap will be updated as implementation progresses and as the whitepaper framework evolves.