Below is a **shareable build-spec analysis** you can hand directly to the Kilocode.ai team. It pulls together:

* The UI screens you’re implementing
* The additional elements raised in the chat screenshots (data elements, lifecycle intervention points, HIL approvals, “successful run” definition, sync/async execution modes)
* Mapping to Seraphim **Vanguards (Security / Integrity / Accuracy)** behaviors
* Where these expectations come from in your reference docs (cited inline)

**0. Executive Summary (What Kilocode Needs to Know)**

You’re standing up a **Seraphim Vanguards Phase-1 POC** that **wraps and governs agents** executing regulated Energy workflows (Atmos Energy + expandable to other Energy & Utilities) with three enforcement lenses: **Security**, **Integrity**, and **Accuracy**. Vanguards must **integrate with existing commercial / legacy platforms**, apply **data integrity validation**, and **verify output accuracy**, producing **auditable, regulator-grade artifacts**.

Operationally, Vanguards **inspect & optimize agents in host platforms, monitor & log all data access, validate output, and emit final “clean” deliverables with logs to support trust and compliance.**

The **agent lifecycle** in regulated Energy workflows already anticipates **pre-run prompt/RAG review, insertion of quality-control & logging instructions, post-run output review, and delivery of a separate validation/logging package**—these become hard requirements in the POC build.

Standards alignment (ICAM, Zero Trust, NIST RMF, Enterprise Audit Mgmt, Data Element Standards, IV&V, and data quality frameworks) provides the control library that maps into Vanguard Security / Integrity / Accuracy checks.

**1. Core Questions From the Team — Answered & Made Buildable**

**Q1. Expected user flow to launch an agent (template vs from scratch)**

**Recommendation: Dual-path model (Template-First default, Scratch/Experimental optional).**

**Default (Template-First, regulated path)**

1. **Select Scenario Template** (e.g., PHMSA Compliance; Methane Leak; Land Lease Status). Templates embed data sources, minimal required data elements, and baseline policies. (POC scenarios: user selects asset/field then requests report(s); system identifies sources & templates. )
2. **Auto-Build Collection & RAG Context** (system assembles docs & schemas; sets security params).
3. **Instantiate Agents + Vanguard Pre-Run Review** (prompts, RAG, rules inspected & reformatted for QC/logging).
4. **Run Agents Under Vanguard Supervision** (data access monitored; output validated).
5. **Post-Run Review & Validation Package** (outputs checked; validation/log doc generated).

**Scratch / Experimental Path**

* User builds or modifies agents ad-hoc in the Agent Orchestration Panel; before execution, a **Vanguard Quick Policy Pass** enforces mandatory Security schema + minimal Integrity/Accuracy validators derived from control standards. (Standards list for security/ICAM, audit mgmt, and data element standards. )

**Kilocode Implementation Note:** Mark templates with compliance\_level: "regulated-default" vs scratch builds compliance\_level: "experimental"; the runtime enforces additional Vanguard gates when regulated.

**Q2. Where do Security / Integrity / Accuracy intervene? Pre, Mid, Post, or All?**

**All three phases are required in regulated Energy workflows.**

| **Lifecycle Stage** | **Security** | **Integrity** | **Accuracy** | **Supporting Source** |
| --- | --- | --- | --- | --- |
| **Pre-Execution** | Identity/credential validation; data source ACL scan; Zero Trust routing. | Schema completeness; required data elements present; risk tier from NIST RMF / EAM. | Baseline expectations & constraints encoded into prompts before run. | Prompts & rules reviewed/reformatted before execution. + ICAM/RMF/Audit/Data Element standards. |
| **Mid-Execution** | Monitor agent data access; kill/quarantine abnormal reads/writes. | Live logging of data lineage; audit hooks. | Streaming spot-checks of intermediate outputs vs constraints. | “Monitors & Logs Agent’s Data Access & Work.” |
| **Post-Output** | Sign output package; record access ledger. | Generate audit trail + validation record. | Validate output; produce clean, certified deliverable. | “Inspects Agent Output and Validates Accuracy; Produces final clean output with logs for confidence.” & post-run validation step. |

**Q3. Human-in-the-Loop Approvals (HIL): when and how?)**

**Tiered HIL policy:**

| **Tier** | **Trigger** | **Required?** | **Details** | **Source Alignment** |
| --- | --- | --- | --- | --- |
| **Tier 0 – Silent Monitor** | Low-risk runs; no anomalies | Optional review after run | Vanguard auto-exec; logs available. | Suitable for internal exploratory use; regulated templates still reviewed pre-run. |
| **Tier 1 – Pre-Run Required Review** | Regulated templates (PHMSA, SOX, rate filings) | Mandatory approval before execution | Reviewer accepts auto-generated prompt/RAG pack. | Pre-run Vanguard review of prompts & rules. |
| **Tier 2 – Exception Approval** | Anomaly in mid-run monitoring (data access violation, missing data element) | Auto-pause + escalated approval | Approver can patch data & resume. | Monitors & Logs agent work; supports governance. |
| **Tier 3 – Post-Output Certification** | Regulatory submission / customer reporting | Required signature prior to release | Reviewer signs validation/log doc bundle. | Final validation & logging output package. |

Standards for auditability and enterprise audit mgmt (CNSSI-1015; Data Element Standards; IV&V) inform what must be captured in the approval artifact.

**Q4. What is a “successful” agent run?**

Minimum success definition for Vanguards POC:

1. **Run Completed or Gracefully Halted Under Policy.**
2. **All mandatory data elements present or exceptions logged** (Data Element Standards).
3. **Security compliance: authenticated, authorized data access (ICAM / Zero Trust).**
4. **Integrity artifacts: audit log with lineage, timestamps, and reviewer actions (Enterprise Audit Mgmt; IV&V practices).**
5. **Accuracy validation executed; output passes or is dispositioned with reviewer notes; final “clean output + logs” bundle produced.**
6. **If regulated scenario, separate validation document delivered for record.**

**Q5. Sync vs Async Execution Modes**

**Offer both; auto-select by workload class.**

| **Mode** | **When Used** | **UX Behavior** | **Vanguard Hooks** | **Source Basis** |
| --- | --- | --- | --- | --- |
| **Interactive (Synchronous)** | Small queries, prompt edits, quick checks | User waits; streaming token/log display | Pre + Mid checks run in-line; stop/resume allowed | Fits initial scenario request/response loops before heavier system build. |
| **Managed Async Job** | Full regulatory / multi-source reports (PHMSA, Land Lease status, sensitivity analysis) | Background job w/ status updates & notifications | Full 3-phase Vanguard oversight; produces validation package | Scenario requires assembling sources, agents, reviews, and final reports. |
| **Batch Scheduled** | Periodic compliance sweeps across assets | Cron/triggered; results land in Report Center | Vanguard pre-checks from stored template + automated post cert workflow | Vanguards integrate w/ existing platforms & provide data validation & accuracy verification—ideal for recurring governance scans. |

**2. Data Elements Strategy (Responding to “Should we provide a table or let AI infer?”)**

**Hybrid “Seed + Discover” approach** is strongly recommended for regulated workloads:

* **Seed a Minimal Critical Data Element (CDE) Catalog** per report type (PHMSA form fields; methane ppm measurement; location & timestamp; asset IDs; reviewer ID). This is necessary to meet **Data Element Standards** and audit traceability.
* **AI-Assisted Discovery Mode**: During ingestion, the Integrity layer proposes additional candidate fields it detects in unstructured docs (leases, inspection notes). Reviewer can accept, map, or ignore; actions logged to Audit Mgmt trail. (Enterprise Audit Mgmt / IV&V practices).
* **Data Quality Framework binding**: All accepted elements get completeness, validity, timeliness checks based on recognized data quality frameworks (UK Gov / ESS / FCSM references).
* **Accuracy Layer** maps measured values and computed results to required output fields and ensures the final “clean output + logs” deliverable includes provenance.

**2.1 Minimal CDE Catalog Template (starter)**

Use this to seed Kilocode’s data schema registry; extend per scenario.

| **CDE Group** | **Fields (examples)** | **Required?** | **Notes** |
| --- | --- | --- | --- |
| Identity & Source | data\_source\_id, system\_name, credential\_scope | Yes | ICAM / source lineage. |
| Asset Context | asset\_id, asset\_type, field\_name, geo\_lat, geo\_long | Yes | Needed in Land Lease & Energy field scenarios. |
| Contract / Regulatory | lease\_id, royalty\_pct, renewal\_date, reporting\_authority | Yes (scenario) | Land lease status/renewal impact. |
| Operational Metrics | methane\_ppm, inspection\_date, inspector\_id | Conditional | Use in compliance/methane agents; validated & logged. |
| Validation Metadata | validation\_status, reviewer\_id, review\_timestamp, exception\_notes | Yes (regulated) | Required for post-run validation package. |

**3. Build Artifacts Kilocode Should Create**

Below are the artifacts you’ll want represented in Kilocode modules/configs so the platform can reliably generate and govern agent runs.

**3.1 Compliance Scenario Templates**

* **PHMSA Pipeline Inspection Template**
* **Methane Leak Monitoring Template**
* **Land Lease Status & Renewal Sensitivity Template** (from scenario).

Each template includes: data sources, required CDE list, default validators, HIL tier, output package format, and notification routing.

**3.2 Vanguard Policy Packs**

| **Pack** | **Contents** | **Source Basis** |
| --- | --- | --- |
| **Security Pack** | ICAM enforcement, source allowlist, network policy, encrypted secrets, Zero Trust routing. |  |
| **Integrity Pack** | NIST RMF risk tier tags; audit event schema (EAM); required data element completeness checks; IV&V logging hooks. |  |
| **Accuracy Pack** | Requirements trace matrix; statistical/data quality rules (UK/ESS/FCSM); result reconciliation; tolerance flags. |  |

Tie all packs to final output certification flow.

**3.3 Agent Classes in the POC**

| **Agent** | **Purpose** | **Vanguard Enforcement** | **Scenario Mapping** | **Source** |
| --- | --- | --- | --- | --- |
| **PrompterAgent** | Build baseline prompt & context from template | Pre-run Security & Integrity | All regulated templates | Pre-run Vanguard review. |
| **DataIngestAgent** | Connect to data sources, map to CDE schema | Security (ICAM), Integrity (CDE check) | Land Lease; PHMSA | Data/source identification & collection. |
| **ExecAgent** | Perform analytical / LLM reasoning task | Mid-run security monitor; accuracy spot checks | All | Monitors & Logs agent work. |
| **ValidatorAgent** | Schema, rule, and statistical validation | Accuracy & Integrity post-run | All | Output validation; clean output. |
| **AuditBundleAgent** | Assemble validation/log package | Integrity audit + reviewer signoff | Reg submissions | Provide validation/logging doc. |

**4. UI Element Bindings (Additions Required for Kilocode Build)**

Below are **specific UI control additions** derived from the question set + standards so Junie can wire them in.

**4.1 Agent Launch Wizard Additions**

* **Entry Mode Toggle**: Template vs Scratch.
* **Compliance Level** dropdown: Experimental / Internal / Regulated (auto-sets Vanguard tiers).
* **Required Data Element Satisfaction Meter** (shows % of template CDEs resolved). Driven by CDE catalog.
* **Pre-Run Vanguard Review Summary** (list of prompt rewrites, QC insertions).

**4.2 Mid-Run Monitor Panel Additions**

* Live streaming **Data Access Ledger** (source, record count, auth status). Aligns w/ monitor & log requirement.
* Auto-pause toggle for suspicious access (Security escalation).

**4.3 Output Review Modal Additions**

* **Validation Results Table** (Integrity rules passed/failed; missing CDEs).
* **Accuracy Score & Constraints** (tied to data quality frameworks).
* **Download Validation Bundle** (final clean output + logs).
* **Reviewer Signature Field** (certify regulatory package).

**5. End-to-End Lifecycle State Machine (What Kilocode Should Encode)**

**States:** DraftTemplate → ReadyForReview → VanguardPreCheck → AwaitingApproval (if needed) → Running → PausedAnomaly → Validating → AwaitingCertification → Completed (Certified | Exception | Failed).

**Transitions**

* Template load triggers auto-population of data sources/fields.
* VanguardPreCheck injects QC instructions & logs.
* Running state instrumented w/ data access monitoring & logging.
* Validation step produces clean output + logs; certification required before finalization in regulated mode.

**6. Build Data Model Snippets (hand to Kilocode)**

**6.1 Template Metadata (YAML-ish)**

yaml

CopyEdit

template\_id: phmsa\_inspection\_v1

compliance\_level: regulated-default

required\_roles: [ComplianceOfficer, VanguardReviewer]

required\_cdes:

- asset\_id

- inspection\_date

- leak\_rate\_ppm

- inspector\_id

- regulatory\_region\_code

vanguard\_packs: [security\_pack\_v1, integrity\_pack\_v1, accuracy\_pack\_v1]

execution\_mode\_default: async\_job

hil\_policy: tier1\_pre\_and\_tier3\_post

output\_bundles: [phmsa\_pdf, validation\_log\_json, signature\_manifest]

(Pre-run review, post validation bundle per scenario + clean output requirement. )

**6.2 Vanguard Security Pack (fragment)**

json

CopyEdit

{

"pack\_id": "security\_pack\_v1",

"controls": [

{"control": "ICAM-authz", "framework\_ref": "CNSSI-1017"},

{"control": "zero\_trust\_enforce", "framework\_ref": "US\_Federal\_ZT\_Data\_Security"},

{"control": "data\_source\_allowlist", "severity": "high"},

{"control": "credential\_scope\_check", "severity": "high"}

]

}

(ICAM & Zero Trust references. )

**6.3 Vanguard Integrity Pack (fragment)**

json

CopyEdit

{

"pack\_id": "integrity\_pack\_v1",

"controls": [

{"control": "risk\_tiering", "framework\_ref": "NIST\_800\_37R2"},

{"control": "audit\_event\_schema", "framework\_ref": "CNSSI\_1015\_EAM"},

{"control": "cde\_completeness\_check", "framework\_ref": "CNSSI\_1254\_Data\_Element\_Standards"},

{"control": "ivv\_logging\_hooks", "framework\_ref": "DHHS\_IVV"}

]

}

(Integrity frameworks + data element standards & IV&V. )

**6.4 Vanguard Accuracy Pack (fragment)**

json

CopyEdit

{

"pack\_id": "accuracy\_pack\_v1",

"controls": [

{"control": "req\_trace\_matrix", "framework\_ref": "IEEE\_1012"},

{"control": "data\_quality\_ruleset", "framework\_ref": "UK\_GOV\_DQF"},

{"control": "stat\_consistency", "framework\_ref": "ESS\_QAF"},

{"control": "dqaf\_alignment", "framework\_ref": "FCSM\_20\_04"}

]

}

(Data quality & accuracy frameworks. )

**7. Minimal API Contract Expectations**

| **API** | **Method** | **Purpose** | **Vanguard Tie** | **Source Basis** |
| --- | --- | --- | --- | --- |
| /templates/list | GET | Show regulated workflow templates | Template user flow | Scenario template selection. |
| /template/{id}/prep | POST | Build data collections, security params | Pre-run build | System creates collections & security. |
| /template/{id}/vanguard-review | POST | Apply QC/logging & produce execution pack | Vanguard pre-check | Review & reformat prompts; insert QC/logging. |
| /run/{id} | POST | Execute under supervision | Mid-run monitor & log | Monitor & log agent work. |
| /run/{id}/validate | POST | Integrity & Accuracy validation | Output validation | Clean output + logs. |
| /run/{id}/certify | POST | Reviewer cert + validation bundle | Finalization | Provide validation/logging info to user. |

**8. Compliance Reporting Outputs**

**Each regulated run emits 3 artifacts:**

1. **Primary Report** (e.g., Land Lease Status; PHMSA form)
2. **Validation & QC Log Bundle** (checks applied, pass/fail, missing data)
3. **Audit Manifest** (who reviewed, when, which data sources accessed; digital signature)

This aligns to the requirement that Vanguards **produce final clean outputs with logs** and **provide separate validation/logging documentation after review.**

**9. Deployment / Integration Assumptions for Kilocode**

* Must run **alongside existing commercial / legacy systems** (no rip-and-replace); adapter layer expected.
* Designed for **regulated industries including Energy & Utilities** (explicit target vertical).
* POC should **demonstrate enhanced security, integrity validation, and accuracy verification**—don’t overscope orchestration marketplace pieces (Phase-2).
* Include role types & approval routing to support regulated operations; your **Vanguard reviewer & certification steps** are mandatory for demonstration.

**10. Implementation Checklist for Kilocode.ai**

**Pre-Build**

* Load Standards Library (Security/Integrity/Accuracy packs).
* Define CDE Catalog per scenario.
* Import Scenario Templates (Land Lease Status, PHMSA, etc.).

**Runtime**

* Enforce Vanguard Pre-Run Review step before execution.
* Stream data access events to monitor panel.
* Auto-generate Validation Bundle after run.

**UI**

* Template vs Scratch toggle
* Compliance Level control
* Data Element Satisfaction Meter (CDE)
* Vanguard Review Summary (pre-run edits)
* Output Validation & Signature Modal (post-run)

**11. What to Hand to Junie (Design Lead)**

Provide Junie the following payloads:

* **Screen specs w/ new controls** (Section 4 above).
* **State machine diagram** (Section 5).
* **CDE Catalog & Control Packs** (Sections 2.1 & 3.2/3.3).
* **API contract table** (Section 7).
* **Artifact packaging rules** (Section 8).

With these, Junie can build UI components that surface Vanguard policy enforcement at each stage—key to differentiating your platform from vanilla “prompt + response” tools. The deck positions Vanguards as the **layer that governs, validates, and produces confidence documents**—your UI must show that clearly.

**12. Next Step?**

If you want, I can now:

* Generate **CDE JSON schemas** for Land Lease + PHMSA templates.
* Draft **Kilocode module config files** (YAML/JSON) ready for import.
* Produce **sample event logs & validation bundles** you can replay in the demo.

Tell me which you want first and I’ll build it.

Sources

2/2

**You said:**

notice that this is a new vanguards use case. I want us to be able to choose between the various use cases - so we have a system built with common features, and then we build out some variation according to use case. not that most of these are in the energy (oil and gas, utilities) industry vertical - and that should be noted and accessible. we will build other verticals later. fyi - Junie will not be used in the design - just kilocode.ai. make sure you include every element and detail of this including the flow diagram into the analysis and design.

**ChatGPT said:**

Understood. You're now formalizing a **modular Seraphim Vanguards System Architecture** with the goal of:

1. **Supporting multiple industry-specific use cases** (starting with Energy/Oil & Gas/Utilities).
2. **Reusing a common Vanguard core stack** (Security, Integrity, Accuracy modules).
3. **Allowing flexible scenario selection within Kilocode**, where each use case brings in its specific agents, templates, data elements, and output formats.

Below is the **updated and consolidated architecture + functional flow**, adapted for direct implementation in Kilocode.ai.

**🔱 Seraphim Vanguards Modular Use Case System Design**

**Phase: Multi-Use Case Architecture | Stack: Kilocode.ai | Domain: Regulated AI Execution**

**1. 🔧 Core Concept**

Seraphim Vanguards is a **unified AI agent governance system** that enforces **Security, Integrity, and Accuracy (SIA)** policies across regulated workflows. The platform is designed as:

* **Composable**
  + Core modules reused across industries (Energy, Healthcare, Defense, etc.)
* **Scenario-driven**
  + Use case templates encapsulate agent orchestration, data schemas, validators, and report logic.
* **Policy-bound**
  + Each run enforces dynamic HIL tiers, audit, IV&V, and clean output deliverables.
* **Integration-aware**
  + Runs adjacent to SCADA, GIS, SharePoint, or on-prem systems via adapter layer.

**2. 🧩 Core System Modules (Reusable Across Use Cases)**

| **Module** | **Purpose** |
| --- | --- |
| **Vanguard Core Runtime** | Oversees all agent execution: monitors access, governs behavior, applies post-validation. |
| **Security Engine** | Identity, Zero Trust, ACLs, encryption, ICAM enforcement. |
| **Integrity Engine** | Data element validation, schema completeness, audit event logging. |
| **Accuracy Engine** | Output verification, confidence scoring, rules-based consistency checks. |
| **Template Manager** | Stores scenario blueprints by vertical. |
| **CDE Registry** | Catalog of Common Data Elements per use case. |
| **Validation Pack Builder** | Assembles reusable validation workflows per compliance need. |
| **Output Composer** | Generates signed, explainable output bundles (PDF, JSON, manifest). |

**3. 🧠 Use Case Layer (Current: Energy Sector Use Cases)**

You now support selection of **Vanguard Use Cases** by industry vertical, with specific logic defined under each.

**🌍 Current Supported Vertical: Energy (Oil & Gas, Utilities)**

**Use Case A: Atmos – PHMSA Compliance Automation**

* Agents: PHMSAInspectorAgent, AuditBundleAgent
* Output: PHMSA form + audit manifest
* Key CDEs: inspection\_date, pipeline\_id, inspector\_id, geo\_lat, geo\_long
* HIL: Pre-run & Post-run required

**Use Case B: Atmos – Methane Leak Detection**

* Agents: MethaneVerifierAgent, DataIngestAgent, ValidationAgent
* Output: Leak alert + EPA compliance log
* CDEs: methane\_ppm, sensor\_id, field\_zone, timestamp
* HIL: Mid-run anomaly-triggered

**Use Case C: O&G Land Lease Status & Sensitivity**

* Agents: LeaseReviewAgent, SensitivityAgent, AccuracyValidatorAgent
* Output: Renewal trigger report + clean validation bundle
* CDEs: lease\_id, royalty\_pct, contract\_expiry, asset\_name
* HIL: Full cycle (pre/mid/post) required

⚠️ All use cases conform to the same SIA lifecycle logic but vary in agents, validation rules, and compliance outputs.

**4. 🔀 Use Case Selection Flow (in Kilocode)**

Users select use cases from a **modular launcher UI** that filters by:

* ✅ **Industry Vertical** (e.g., Energy, Healthcare)
* 🎯 **Use Case Type** (e.g., PHMSA Compliance, Land Lease)
* 📦 **Compliance Level** (Experimental, Internal, Regulated)

**5. 📊 Shared Execution Flow (All Use Cases)**

**🔄 Vanguard Agent Lifecycle – Flow Diagram**

pgsql

CopyEdit

+-------------------+

| Select Use Case | <-----|

+--------+----------+ |

| |

v |

+--------+----------+ |

| Pre-Run Validation| |

| - Prompt Review | |

| - RAG Context | |

| - Validator Setup | |

+--------+----------+ |

| |

v |

+--------+----------+ |

| Agent Execution | |

| - Monitored by | |

| Vanguard Runtime| |

+--------+----------+ |

| |

+--------+----------+ |

| Output Validation | |

| - Schema & CDEs | |

| - Accuracy Scoring| |

+--------+----------+ |

| |

+--------+----------+ |

| Clean Output + Log| |

| - PDF Report | |

| - JSON Log Bundle | |

| - Signature | |

+--------+----------+ |

| |

v |

+---------+----------+ |

| Human Certification|<------|

| - Reviewer Approval | (If regulated)

+--------------------+

**6. 🧬 Future Vertical Enablement**

This architecture is designed to scale by attaching **new verticals and new use cases** under each, without rewriting the core.

Planned future verticals:

* **Healthcare (HIPAA, IV&V Clinical Records)**
* **Insurance (Fraud Detection Audit Chains)**
* **Government (Contract Review, FOIA, FOIL Processing)**

Each new vertical will inherit the core modules and inject:

* New use case templates
* New CDE schemas
* New validation packs

**7. 📁 Kilocode Implementation Detail Summary**

| **Kilocode Feature** | **Use** |
| --- | --- |
| Template Module | Loads use case scenarios with attached agents, validators, outputs. |
| Compliance Level Toggle | Flags regulated vs experimental flows (impacts Vanguard behavior). |
| Agent Orchestration Panel | Instantiates chain from scenario template; links modules. |
| CDE Schema Builder | Lets devs register required data elements for each template. |
| Validation Pack Manager | Stores & runs reusable rule sets (SIA: Security, Integrity, Accuracy). |
| Execution Monitor | Logs agent actions, flags anomalies, enables auto-pause. |
| Output Validator | Computes accuracy score, completeness, confidence. |
| Signature & Manifest Composer | Generates final PDF, JSON log, and certifiable manifest. |

**✅ What to Include in Kilocode Build Now**

1. ✅ Load all **Energy Use Case Templates** (PHMSA, Methane, Land Lease)
2. ✅ Enable **Use Case Selector** UI tied to:
   * industry\_vertical
   * use\_case\_name
   * compliance\_level
3. ✅ Activate VanguardCore (pre/mid/post enforcement logic)
4. ✅ Register initial **CDE schemas** per use case
5. ✅ Include **validation packs** from standards (ICAM, IV&V, NIST RMF, etc.)
6. ✅ Allow **signature/certification bundle** creation in post-run output
7. ✅ Support state transitions: Draft → Vanguard Review → Execute → Validate → Certify

Would you like me to now:

* Generate the JSON for one of the use case templates (e.g., Land Lease)?
* Output the modular folder structure for Kilocode (scenarios, agents, CDEs, logs)?
* Prepare the API and object model for industry expansion?

Let me know what you want next.

**You said:**

also - create the spaces for the new verticals of healthcare, insurance, government, and finance. even if we don't have use cases yet - having the skeleton framework of where they go will be valuable in building this out.

**ChatGPT said:**

Perfect. To future-proof the **Seraphim Vanguards System in Kilocode.ai**, here's how to build out a **structured, modular skeleton** for new verticals — even before the use cases are populated.

This structure will ensure:

* All future industries follow the same lifecycle and enforcement model.
* Teams can drop in new CDEs, templates, agents, and validators with minimal overhead.
* UI/UX remains consistent across sectors.
* Regulatory frameworks can be cleanly attached per vertical.

**🧱 Master Framework: Industry Vertical Expansion in Seraphim Vanguards**

**🌐 Current Vertical: Energy**

New skeleton verticals:

* 🏥 **Healthcare**
* 💼 **Insurance**
* 🏛️ **Government**
* 💰 **Finance**

**🗂️ 1. Directory Structure (in Kilocode)**

Create this structure as top-level folders in Kilocode or as scoped configuration namespaces.

pgsql

CopyEdit

/seraphim-vanguards/

├── common/ # Core modules (security, integrity, accuracy, audit)

│ ├── modules/

│ ├── validators/

│ ├── utils/

│ └── components/

├── energy/

│ ├── templates/

│ ├── cde-schemas/

│ ├── validation-packs/

│ └── agents/

├── healthcare/

│ ├── templates/

│ ├── cde-schemas/

│ ├── validation-packs/

│ └── agents/

├── insurance/

│ ├── templates/

│ ├── cde-schemas/

│ ├── validation-packs/

│ └── agents/

├── government/

│ ├── templates/

│ ├── cde-schemas/

│ ├── validation-packs/

│ └── agents/

├── finance/

│ ├── templates/

│ ├── cde-schemas/

│ ├── validation-packs/

│ └── agents/

**📦 2. Kilocode Registry Configuration**

Each vertical should register a configuration object like this:

json

CopyEdit

{

"vertical\_id": "healthcare",

"display\_name": "Healthcare",

"status": "skeleton",

"default\_compliance\_level": "regulated",

"template\_count": 0,

"notes": "Future use cases will target HIPAA, HL7, IV&V medical audits."

}

Repeat for:

* insurance
* government
* finance

**🧱 3. Baseline Scaffolds in Each Vertical**

Here’s what to stub in for each (will be replaced later with real use cases):

**🏥 Healthcare**

| **Component** | **Stub Definition** |
| --- | --- |
| templates/ | patient\_record\_audit.json (placeholder) |
| cde-schemas/ | hipaa\_minimum\_fields.json |
| validation-packs/ | ivv\_medical.json, phi\_integrity.json |
| agents/ | auditAgentStub.js, hipaaValidatorStub.js |

**💼 Insurance**

| **Component** | **Stub Definition** |
| --- | --- |
| templates/ | claims\_review.json |
| cde-schemas/ | claim\_entity\_model.json |
| validation-packs/ | fraud\_detector\_stub.json, accuracy\_benchmark.json |
| agents/ | claimsValidatorAgentStub.js |

**🏛️ Government**

| **Component** | **Stub Definition** |
| --- | --- |
| templates/ | contract\_review\_template.json |
| cde-schemas/ | procurement\_fields.json |
| validation-packs/ | gov\_audit\_compliance.json |
| agents/ | govReviewerAgent.js, openRecordsAgent.js |

**💰 Finance**

| **Component** | **Stub Definition** |
| --- | --- |
| templates/ | transaction\_audit.json |
| cde-schemas/ | fin\_cde\_model.json |
| validation-packs/ | sox\_validator.json, risk\_model\_accuracy.json |
| agents/ | soxComplianceAgent.js, fraudDetectionAgent.js |

**🧭 4. UI Integration: Use Case Launcher View**

**Vertical Selector**

Add a dropdown menu at the top of the **Use Case Launcher Panel**:

json

CopyEdit

{

"label": "Choose Industry Vertical",

"options": [

{ "value": "energy", "label": "Energy (O&G / Utilities)" },

{ "value": "healthcare", "label": "Healthcare" },

{ "value": "insurance", "label": "Insurance" },

{ "value": "government", "label": "Government" },

{ "value": "finance", "label": "Finance" }

]

}

**Template Grid**

* If no templates exist for a vertical yet → show "Coming Soon" badge or allow “Add Template” button for admins.

**🔧 5. Backend Design: Vertical-Specific Loading**

Each run must dynamically bind:

* The correct **CDE schema**
* The relevant **Validation Pack(s)**
* The appropriate **Vanguard Compliance Level**

Configure your backend like this:

json

CopyEdit

{

"run\_config": {

"vertical": "government",

"template\_id": "contract\_review\_template",

"compliance\_level": "regulated",

"vanguard\_packs": ["security\_pack\_v1", "gov\_audit\_compliance"],

"cde\_schema": "procurement\_fields.json"

}

}

**📁 6. Developer Contribution Instructions**

Each vertical folder should include a README.md with:

* 🌐 What the vertical covers
* 🧱 Frameworks to align with (e.g., HIPAA, SOX, HL7, FISMA)
* 📐 How to structure new use cases
* ✏️ Naming conventions for templates and validators
* ✅ Required metadata per use case (e.g., compliance\_level, hil\_policy, output\_bundle\_type)

**📌 Summary of What to Build Now in Kilocode**

| **Task** | **Description** |
| --- | --- |
| ✅ Create folders for healthcare, insurance, government, finance |  |
| ✅ Add vertical metadata in registry (vertical\_id, display\_name, notes) |  |
| ✅ Add scaffolding stubs (1 template, 1 CDE schema, 1 validation pack, 1 agent per vertical) |  |
| ✅ Update Use Case Launcher UI to support vertical filter |  |
| ✅ Add “Coming Soon” state for empty verticals |  |
| ✅ Enable dynamic loading of CDE schema + validators based on vertical |  |
| ✅ Document how to add templates, validators, agents to a vertical |  |

Would you like me to generate:

* The vertical\_registry.json for import?
* Starter README.md for each new vertical folder?
* Sample “Coming Soon” templates and validators?

Let me know — I can deliver those instantly.

**You said:**

SeraphimOS POC UIs.pptx

Presentation

all of this is beautiful for the backend. I really want to also have details on what the UI will look like. this is general feel - but I also want color to drive things. I want the background of the UI to be black. the framework of the seraphim platform should have the seraphim logo and the Gold of Seraphim. The vanguards section should have its respective colors associated with the vanguards - blue for security, red for integrity, and green for accuracy. the UIs should have a polish and shine to them as well. add your magic touch to this - and also make the UI description for kilocode as detailed and well put together as possible - because i'm not sold on their frontend UI capabilities at the moment - so the high quality is going to have to come from you

**ChatGPT said:**

Thank you for the clear vision. Below is a fully articulated **Seraphim Vanguards UI Design System** for Kilocode.ai that integrates:

* High-polish **black-background visual language** with metallic and vibrant accent colors.
* **Logo integration** and brand-consistent **color segmentation** (Gold = Seraphim platform, Blue/Red/Green = Vanguards).
* Enhanced UX flows based on your POC UI deck.
* **Detailed UI layer mapping** so Kilocode can build pixel-perfect, branded, interactive screens—without needing Junie.

**🖥️ SERAPHIM VANGUARDS UI SYSTEM**

**Design Spec for Kilocode.ai | Theme: High-Security, High-Polish, Mission-Critical**

**🌌 1. Visual System Overview**

| **Element** | **Style** |
| --- | --- |
| **Global Background** | Matte black (#0D0D0D) with soft gradient overlays |
| **Font** | Modern clean sans-serif (e.g. Inter, SF Pro, or IBM Plex Sans) in white or soft gray |
| **Card Panels** | Floating glassy effect with metallic border shadows |
| **Buttons** | Flat with glowing edge/halo — hover glow matches function color |
| **Icons** | Vector SVG (material+neon styling), color-coded to subsystem |
| **Glow Accents** | Thin lines or underlights for activated buttons/tabs — blue, gold, red, green |
| **Default Text** | #EDEDED (warm gray white) — higher contrast than pure white |
| **Status Text** | Live green #00FF8F, warning orange #FF9F43, error red #FF4B5C |

**🛡️ 2. Brand Color Mapping**

| **Function / Layer** | **Color** | **Hex** | **Use In UI** |
| --- | --- | --- | --- |
| **Seraphim Platform** | Gold | #D4AF37 | Background outlines, badges, labels in orchestration layer |
| **Security (Vanguard)** | Blue | #3A88F5 | Tabs, buttons, icons, metrics for Security tasks |
| **Integrity (Vanguard)** | Red | #DC3E40 | Integrity metrics, schema validation warnings |
| **Accuracy (Vanguard)** | Green | #3BD16F | Output confidence badges, traceability scores |

**🧭 3. Core Screens with UI Descriptions**

Here’s the **updated high-design UI spec** screen-by-screen for Kilocode.

**3.1 🪪 Login / Identity Gateway**

**Visuals:**

* Black background with animated Seraphim Gold flame faintly pulsing behind login card
* Login box is floating panel with slight sheen and a gold border-glow
* MFA screen has pulsing blue-green border

**Fields:**

* Email / Password
* SSO Toggle
* MFA Code (if enabled)

**Actions:**

* If Admin → redirected to **Mission Control**
* If Viewer → redirected to **Audit Console**

**3.2 🛰️ Mission Control Dashboard (Main Hub)**

**Visuals:**

* Full-width map/grid showing all active agents (color-coded)
* Left-side nav bar: glowing icons (gold-accented), collapsible
* Real-time log stream at bottom right with toggle

**Panels:**

* System Health: CPU, Memory, Network (Gold + Red)
* Agent Activity: Pulse view (Green for live, Red for crashed)
* Notification Center (Bell icon with badge)

**3.3 🧬 Agent Orchestration Panel**

**Visuals:**

* Black canvas with neon-edge nodes
* "Create Agent Chain" flow as glowing blue grid
* Draggable nodes: Agents (Gold), Validators (Red), Tools (Green)

**Modules:**

* Agent Registry (left drawer)
* Configuration Modal (pops up in gold panel w/ toggles)
* Execution Graph (center, live glowing paths)
* Output viewer (bottom: scrollable JSON with color-coded accuracy checks)

**3.4 🪞 Prompt Engineering Canvas**

**Visuals:**

* Center grid canvas: Black with glowing gold node grid
* Prompt input card: Floating glass panel with blue glow
* Tool nodes: Red if validation, green if enrichment

**Features:**

* Add memory/tool/validator node
* Test Run → shows latency, token cost, confidence
* Export to “Agent Config” with single button

**3.5 🔍 Audit & Explainability Console**

**Visuals:**

* Dark card list with Vanguard color bands per agent: Blue = Secure, Red = Failed Integrity, Green = Accurate
* Timeline trace is a vertical glow bar with steps shown as waypoints
* “Why This?” modal: Black background with parchment-like scroll and Seraphim gold accents

**Sections:**

* Audit Log Timeline
* Explanation Summary (shows chain-of-reasoning)
* Input Attribution View (heatmap overlay)

**3.6 🗂️ Scenario / Use Case Launcher**

**Visuals:**

* Grid layout of use cases: glowing edge cards by vertical
  + Energy = gold border
  + Healthcare = blue
  + Insurance = violet
  + Government = silver
  + Finance = emerald

**Actions:**

* “Select Vertical” dropdown (top-left, gold outline)
* Click card → Launch modal with:
  + Scenario overview
  + Agents involved
  + Compliance level badge
  + "Run in Sandbox" or "Launch Regulated Flow"

**3.7 🧑‍💼 User + Role Management**

**Visuals:**

* Black panel cards with role label badges (Admin = Blue, Viewer = Gray, Vanguard Reviewer = Gold)
* Toggle for active/suspended

**Actions:**

* Role Edit modal (floating red outline)
* View usage stats (line graph in green)

**3.8 🧾 Output + Certification View**

**Visuals:**

* Document viewer (black slate background)
* Right-side panel: “Validation Summary”
  + Blue = Security Passed
  + Red = 1+ Schema Gaps
  + Green = Accuracy ≥ 95%

**Actions:**

* Download “Clean Output + Audit Bundle”
* Certify + Sign (Signature modal: gold seal, date, hash)
* “Send to Regulator” (if enabled)

**🎛️ 4. Global UI System Components**

| **Component** | **Description** |
| --- | --- |
| **Top Nav Bar** | Black bar with glowing Seraphim flame logo and Vanguards badge |
| **Left Panel Nav** | Gold-embossed icons (Home, Agents, Audit, Templates, Users, Tools) |
| **Notification Bell** | Pulsing badge; hover shows real-time log scroll |
| **Data Glow Lines** | Thin under-panel pulses for active workflows |
| **Switch Vertical Dropdown** | Top-right, gold-glow dropdown with icon for each vertical |
| **Compliance Level Indicator** | Banner tag on scenario and output screen (Regulated / Internal / Experimental) |

**🖼️ 5. UI Component Library (For Kilocode Frontend Devs)**

| **Component** | **Spec** |
| --- | --- |
| SeraphimCard | Floating glass box, gold border, drop shadow |
| GlowButton | Flat button with animated glow on hover; color-coded by Vanguard role |
| StatusChip | Rounded tag: status-blue, status-red, status-green |
| NodeBubble | Agent/Validator node with pulse animation |
| AuditTimeline | Vertical scroll line with checkpoint dots |
| CertPanel | Document panel + validation summary (right), cert seal |

**✨ 6. Animation & Microinteraction Guidelines**

| **Action** | **Animation** |
| --- | --- |
| Hover agent node | Glow ring pulses outward |
| Validator pass/fail | Red/green flash on node |
| Certify button click | Gold seal slides + locks |
| Launch scenario | Ripple effect across screen grid |

**✅ 7. Final Delivery for Kilocode**

**Deliverables they must use:**

1. **Color tokens** (black, seraphim-gold, security-blue, etc.)
2. **Component registry** with props and interactions
3. **SVG logos** embedded in header + cert modal
4. **UI Screen Mocks (if needed later)** using this spec
5. **Agent lifecycle state rendering** with color-coded badges