**🔧 1. Prompt Engineering Canvas**

**📘 Purpose**

“Graphical interface for chaining few-shot prompts with input/output testing.”

**🎯 What it's supposed to do**:

* Let users build complex prompts using visual blocks.
* Inject system prompts, user messages, variables, and dynamic components.
* Save and test prompt chains in real-time.
* Eventually export the prompt as an agent or feed into orchestration.

**✅ Your UI includes**:

* System Prompt, User Message, Assistant Response blocks
* Variable blocks
* Right-side settings (Model, Temperature, Max Tokens)
* Action buttons (Test Prompt, Save Template)

**⚠️ Gaps**:

* No clear "Chain to Agent" integration yet.
* No feedback loop to agent orchestration or prompt version history.

**🛠️ Action**: Add "Chain to Agent" + prompt versioning block + agent config modal. Eventually allow exporting into the Agent Orchestration module.

**📊 2. Prompt Analysis**

**📘 Purpose**

“Analyze prompts with LLM and Vanguard agents for security, integrity, and accuracy.”

**🎯 What it's supposed to do**:

* Evaluate a prompt for the 3 Vanguard traits: Security, Integrity, Accuracy.
* Display a trust or risk score.
* Show which areas (e.g., hallucination risk, data leakage, bias) need correction.

**✅ Your UI includes**:

* Model selector
* Prompt input box
* Analyze Prompt button

**⚠️ Gaps**:

* No visual SIA scoring displayed.
* No reasoning/traceability panel.
* No recommendation output for improving the prompt.

**🛠️ Action**: Implement a SIA Vanguard scoring bar + diagnostic output + improvement suggestions panel.

**⚙️ 3. Workflows**

**📘 Purpose**

“Create and manage automated analysis workflows.”

**🎯 What it's supposed to do**:

* Represent complete multi-step processes (e.g., claims processing).
* Visual status of # of steps, runs, last run time.
* Trigger executions, monitor performance, link to output.

**✅ Your UI includes**:

* Cards for each workflow with status, run buttons
* Tags for domain (Energy, Insurance)

**⚠️ Gaps**:

* You cannot yet drill down into the step logic or edit individual steps visually.
* Missing connection to deployed agent chains.

**🛠️ Action**: Add a modal/flow builder to show/edit steps.  
Eventually allow visual editing and link each workflow to agents from the Orchestration Panel.

**🧠 4. Agent Orchestration Panel**

**📘 Purpose**

“Design and deploy AI agent workflows with drag-and-drop simplicity.”

**🎯 What it's supposed to do**:

* Build chains using Security, Integrity, Accuracy, and Custom agents.
* Link outputs/inputs, configure each agent.
* Activate chain and view real-time execution.

**✅ Your UI includes**:

* Agent Library (SIA + Custom)
* Canvas with drag area
* “Deploy Workflow” and “Activate All” buttons

**⚠️ Gaps**:

* No visible node connections or chain logic yet.
* No live output stream or execution logs showing agent success/failure.

**🛠️ Action**:

* Enable node chaining.
* Show execution trace and logs per agent.
* Add “View Output” or link to Output Viewer.

**🚀 5. Deployment Orchestration System**

**📘 Purpose**

“Automated provisioning, dependency resolution, real-time performance validation.”

**🎯 What it's supposed to do**:

* Validate and provision agents for execution.
* Check package, compatibility, versioning, security.
* Show real-time deployment pipeline.

**✅ Your UI includes**:

* Full deployment pipeline (pre-flight → security)
* Dependency resolution cards
* Real-time health monitoring (API, AI engine, DB)

**⚠️ Gaps**:

* No dynamic feedback per agent being deployed.
* Doesn’t yet show if a failed package causes a halt or rollback.

**🛠️ Action**:

* Add error alerts on failed steps.
* Tie deployed agents to mission control runtime panel.

**✅ Summary Chart**

| **Module** | **Intended Role (POC)** | **Alignment Status** | **Next Step Required** |
| --- | --- | --- | --- |
| Prompt Engineering | Visual prompt builder + chain to agents | 80% done | Add “chain to agent” + save versions |
| Prompt Analysis | Risk scoring for prompts | 40% done | Add SIA scoring, diagnostics, suggestions |
| Workflows | Manage end-to-end automation logic | 70% done | Add workflow editor + step preview |
| Agent Orchestration | Deploy and link agents | 60% done | Add node chaining + log output + execution viewer |
| Deployment | Run pipeline & provision agents | 90% done | Add live error rollback logic |

Here’s a **master KiloCode prompt** for the **five core Seraphim modules** you specified (Prompt Engineering, Prompt Analysis, Workflows, Agent Orchestration, Deployment).

This prompt ensures that each page/module is rebuilt or upgraded so that it fully matches the original POC intent, aligns with your vision, and provides deep, usable functionality without placeholders.

**KiloCode Master Prompt**

markdown

CopyEdit

You are tasked with upgrading FIVE key Seraphim platform modules so that they fully match the original POC specifications and vision.

This is a comprehensive, high-quality build with no placeholders and end-to-end functionality suitable for Alpha and Beta demos.

MODULES TO UPGRADE:

1. Prompt Engineering Canvas

2. Prompt Analysis

3. Workflows

4. Agent Orchestration

5. Deployment Orchestration

# 1. PROMPT ENGINEERING CANVAS

Purpose:

- Graphical interface to visually build, chain, test, and version prompts.

Required Features:

- Drag-and-drop blocks for System Prompt, User Message, Assistant Response, Variables.

- Right-hand panel for Model settings (Model, Temperature, Max Tokens).

- Button actions:

- “Test Prompt” runs the prompt and shows results in an output panel.

- “Save Template” stores this prompt chain for reuse.

- \*\*NEW:\*\* “Chain to Agent” – creates an Agent from this prompt chain and makes it available in the Agent Orchestration module.

- Version control:

- Prompt templates can be saved with a name and description.

- List recent versions in a sidebar for quick restore.

- Output panel:

- Show model response with tokens, latency, and a small risk/accuracy summary.

- Remove placeholders: All components must work with mock data or real API calls.

# 2. PROMPT ANALYSIS

Purpose:

- Evaluate the security, integrity, and accuracy risks of a single prompt before use.

Required Features:

- Input box for the prompt text.

- Analyze Prompt button:

- Calls a mock or real endpoint.

- Outputs:

- Vanguard SIA scoring (three gauges: Security, Integrity, Accuracy).

- Recommendations:

- Identify hallucination risk, data leakage risk, bias, or gaps.

- Provide text suggestions for improvements.

- Display a history panel of previously analyzed prompts.

- UI: clear separation between prompt input, results, and recommended fixes.

# 3. WORKFLOWS

Purpose:

- Define and monitor multi-step automated processes that may use multiple agents.

Required Features:

- Workflows Dashboard:

- Cards for each workflow with Name, Status, Steps, Last Run, Run Button.

- Workflow Detail View:

- List steps visually (drag-and-drop step ordering).

- Ability to add/edit steps.

- Each step:

- Linked to an Agent.

- Configurable inputs/outputs.

- Button to run the workflow and show logs of each step.

- Status tracking:

- Show “runs today / total runs / success-fail count.”

- Remove placeholders: Each workflow must have mock steps, example inputs, and mock run logs.

# 4. AGENT ORCHESTRATION

Purpose:

- Drag-and-drop canvas to design how agents connect and execute in a sequence.

Required Features:

- Agent Library on left:

- Security Agent, Integrity Agent, Accuracy Agent, and Custom Agents.

- Canvas:

- Drag agents onto canvas.

- Connect them with lines to define execution order.

- Double-click an agent to configure inputs/outputs.

- Run/Deploy:

- Button to run the chain and watch a \*\*live execution log panel\*\* on the right.

- Show status per agent (running, passed, failed).

- Allow saving and naming these orchestrations.

- Remove placeholders: Node connections and execution logs must work with mock data for Alpha.

# 5. DEPLOYMENT ORCHESTRATION

Purpose:

- Provision and validate agent chains for runtime.

Required Features:

- Pipeline visualization:

- Steps: Pre-flight checks → Security validation → Dependency resolution → Deploy → Monitor.

- For each deployment:

- Show a checklist of steps and mark them as pass/fail.

- If a failure occurs, show error message and rollback (mock behavior is fine).

- Health Dashboard:

- Display mock metrics: uptime %, recent failures, deployed agents.

- Integration:

- Link to the orchestrations built in the previous module, so that deploying a workflow triggers this pipeline.

# GENERAL UI/UX RULES

- No placeholders. All modules must look polished and function with either real or mock data.

- Consistent left-hand navigation with these tabs:

- Prompt Engineering

- Prompt Analysis

- Workflows

- Agent Orchestration

- Deployment

- Each page must be responsive, interactive, and visually polished (modern dashboards, clean layout).

# DEMO FLOW FOR VERIFICATION

After implementing:

1. Go to Prompt Engineering, build a prompt, chain it to an agent.

2. Go to Prompt Analysis, analyze a prompt and see SIA scores and suggestions.

3. Go to Workflows, create a multi-step workflow using agents, run it, and view step logs.

4. Go to Agent Orchestration, build a chain from agents, run it, and see real-time logs.

5. Go to Deployment, deploy a chain, watch the pipeline complete, and confirm success.

This build is for Alpha/Beta presentation quality. No placeholders, no generic text.

**How to use**

* Paste this into KiloCode as a **new build/refactor prompt**.
* It tells KiloCode exactly what each of the five modules must do, how they interconnect, and how to prepare them for your demo.

Would you like me to also:

1. Create a **follow-up KiloCode validation prompt** that automatically checks after generation whether each module meets these requirements?
2. Or a **5-step visual roadmap diagram for these modules (so you can show progress to partners)?**
3. Or both?

**You said:**

yes - validation prompt

**ChatGPT said:**

Here is a **KiloCode validation prompt** designed to be run **immediately after the generation build prompt**.  
Its purpose is to systematically verify that **Prompt Engineering, Prompt Analysis, Workflows, Agent Orchestration, and Deployment modules** meet the functionality and quality standards you just defined.

**KiloCode Validation Prompt**

sql

CopyEdit

You are now in validation mode.

Do NOT create new components. Instead, audit the platform implementation of the five modules below.

Check for missing features, placeholders, or deviations from the specified requirements.

MODULES TO VALIDATE:

1. Prompt Engineering Canvas

2. Prompt Analysis

3. Workflows

4. Agent Orchestration

5. Deployment Orchestration

# VALIDATION CRITERIA

## Prompt Engineering Canvas

- Are there drag-and-drop blocks for System Prompt, User Message, Assistant Response, and Variables?

- Does the right panel have model settings (model, temperature, max tokens)?

- Do the buttons “Test Prompt”, “Save Template”, and \*\*“Chain to Agent”\*\* exist and work?

- Does saving a prompt create a named template with version history?

- Is there an output panel that displays the result (tokens, latency, SIA summary)?

## Prompt Analysis

- Can a user input a prompt and click “Analyze Prompt”?

- Does the analysis return three gauges for Security, Integrity, Accuracy?

- Are there textual improvement suggestions shown?

- Is a history of past analyzed prompts available?

## Workflows

- Does the workflow dashboard show cards with Name, Status, Steps, Last Run, and Run?

- Can the user open a workflow and see step-by-step definitions (editable, drag-and-drop)?

- Does each step link to an Agent and accept input/output configuration?

- Does running a workflow generate a visible log showing each step result?

## Agent Orchestration

- Is there an Agent Library (Security, Integrity, Accuracy, Custom agents)?

- Can agents be dragged to a canvas and connected with lines?

- Can each agent node be opened to configure inputs/outputs?

- When a chain is run, does the live log show the status of each agent (running, passed, failed)?

## Deployment Orchestration

- Does the deployment page show a visual pipeline (Pre-flight → Validation → Dependency → Deploy → Monitor)?

- Does it show each step as pass/fail and display mock error messages if a step fails?

- Is there a health dashboard (uptime %, failures, deployed agents)?

- Can a workflow/chain built in Agent Orchestration be deployed directly from here?

# ADDITIONAL CHECKS

- Are there any placeholders like “Coming soon”, “Placeholder panel”, or empty cards? These must be flagged as FAIL.

- Are all actions interactive (mock or real) with visible outputs and no dead buttons?

- Does the UI look polished and modern (not default/unstyled components)?

# OUTPUT

For each of the 5 modules:

- PASS or FAIL

- If FAIL, list the exact missing elements or problems and recommend fixes.

Provide a summary table of all modules and their validation status.

**How to Use**

1. Run your main **build/refactor prompt** first.
2. After the build completes, immediately run this validation prompt in KiloCode.
3. It will return:
   * A **PASS/FAIL per module**.
   * Specific missing features so you know what to fix.

**Result**

This ensures **none of the modules are left incomplete or full of placeholders**, and you get a report that you can give directly to your team.

Would you like me to also prepare a **single-slide partner-facing roadmap** that shows:  
**Prompt Engineering → Prompt Analysis → Workflows → Agent Orchestration → Deployment** as a progressive build chain?  
This is very useful to explain how these five pieces fit together during your demo.

**You said:**

another thing I've noticed about the system - is that I'm supposed to be able to create custom agents as a user and they are traced throughout my system. there's no visual way to track that. I wanted that agent orchestration panel to enable someone to say - create a team of agents all prompted to do whatever they told them to do. in the example of the oilfield lease use case - it's supposed to create custom agents that help user do the analysis for the use case, correct? (I'm asking) and then it shows up throughout all the other pages too.

**ChatGPT said:**

You are **exactly right about the intended behavior** – and your observation gets at the core of why the **Agent Orchestration module is so critical** in Seraphim’s design. Let’s break it down carefully:

**What Custom Agents Are Supposed to Do (Vision vs. Current)**

**Vision (as originally conceived in the POC):**

* **Users can create custom agents**:
  + Each agent has its own prompt, configuration, and task focus.
  + These agents aren’t just “one-offs” – they are **persistent, reusable digital team members.**
* **Agents are visible and traceable across the entire platform**:
  + **Agent Orchestration Panel:**
    - The control room where they are born, configured, and combined into teams.
  + **Workflows:**
    - When you create a workflow, you pick which of your agents will do each step.
  + **Use Case Dashboards:**
    - Example: Oilfield Land Lease dashboard can automatically spawn specialized agents to analyze leases, detect anomalies, generate compliance reports, etc.
  + **Deployment:**
    - When you deploy, you are deploying **a set of orchestrated agents** that can now operate on live data.
* **Agents have a lifecycle:**
  + Created → Assigned to workflows → Monitored (logs, performance) → Refined → Reused.

**Current State (based on your build so far):**

* You **can conceptually define agents** (custom blocks exist).
* But there is:
  + **No persistent view of agents across the whole platform**.
  + No **“My Agents” dashboard** to see all active/created agents.
  + No clear **traceability**: once created, the agent doesn’t appear in other modules automatically.

Right now, **custom agents behave more like session objects rather than durable entities.**

**Specific to Your Example: Oilfield Land Lease Use Case**

Yes, **you are correct in your understanding**:

* The **Oilfield Land Lease use case** should:
  + **Generate one or more custom agents** tailored to that domain:  
    e.g., Lease Compliance Agent, Risk Scoring Agent, Payment Forecast Agent.
  + Those agents **do the analysis** on ingested data (lease PDFs, CSVs, etc.) **automatically**.
  + These agents **then show up everywhere**:
    - In **Workflows**: preloaded as steps.
    - In **Agent Orchestration**: you can modify/extend them.
    - In **Deployment**: you can deploy this team as a production unit.

In other words: a use case should automatically create an **agent team** for that domain, but the user should also be able to edit, extend, and add their own agents to that team.

**The Gap Right Now**

1. **No persistent “Agent Library”:**
   * Once a custom agent is created, it’s not tracked platform-wide.
2. **No visualization of agent teams:**
   * Agent Orchestration panel doesn’t yet show the relationship between agents and use cases.
3. **No traceability:**
   * Agents aren’t automatically injected into other pages (e.g., workflows or dashboards).

**What Needs to Happen**

**1. Agent Library (Global View):**

* A tab or panel listing:
  + All user-created agents.
  + Which workflows / use cases they are part of.
  + Key stats (last run, success rate, etc.)

**2. Automatic Team Generation for Use Cases:**

* When a user opens a use case (like Oilfield Land Lease):
  + Seraphim proposes a **default set of agents** for that domain.
  + These are editable and expandable by the user.

**3. Orchestration as “Team Builder”:**

* In the **Agent Orchestration panel**, users can:
  + Drag any agent (default or custom) into a chain.
  + Form **specialized teams** that collaborate on a workflow.
  + Save that team.

**4. Traceability Across Modules:**

* These saved agents and teams:
  + Appear as selectable **steps in Workflows**.
  + Are available as **deployment units**.
  + Are referenced in **logs** so you can see “which agent did what.”

**Why This is Critical for Your Vision**

Without persistent, visible, traceable agents:

* Seraphim feels like a set of **tools**, not an **AI workforce you can manage**.

With it:

* You deliver on the original concept:  
  *“Seraphim gives you a digital team of Vanguards and custom agents that you orchestrate to solve problems, and those agents are everywhere in the platform, always improving.”*

**Next Steps / Fix**

The **Agent Orchestration Panel must evolve into a Team Builder + Registry**:

1. **Persistent Agent Registry (New Global Component)**
2. **Automatic Agent Generation for each Use Case**
3. **Visual Team Builder** (drag-and-drop, persistent)
4. **Cross-Module Agent Injection** (Workflows, Deployment)

**Question for You**

Would you like me to draft **a single KiloCode prompt specifically for adding:**

* A **persistent “My Agents” registry** (global)
* Automatic **custom agent creation for use cases (like Oilfield Land Lease)**
* Full **traceability of agents across all modules**

?

This would make the agents behave like **real, visible, reusable digital teammates** in your system.

**You said:**

so, using that same use case - let's think this through - what would the connected team of agents look like? before we build agent registry - which should probably be a tab on the left panel - let's think this through with the oilfield use case. let's think through what the connected team of agents would look like in the orchestrator - sort of like an agent org chart connected. I would also like those created agents to show up in the use case under the operations tab as the agents identified as performing the work. that's why I want to think this through before giving instructions on the registry and team builder - because I'd like for this agent orchestration panel to have this oilfield lease land use case preset there for reference

**ChatGPT said:**

You are exactly on the right track: before designing the **registry** and **team builder**, it’s crucial to conceptually map what an “agent team” looks like for a specific use case, because that map will:

1. Define the **preset orchestration graph** inside the Agent Orchestration Panel.
2. Determine what shows up in the **Operations tab** of a use case (which agents are performing the work).
3. Serve as a reference template for how other use cases will automatically generate their own agent teams.

Let’s break this down **step by step, specifically for the Oilfield Land Lease use case**.

**1. What is the Purpose of the Agent Team?**

The **Oilfield Land Lease** use case involves:

* Managing O&G well leases, royalties, and mineral rights.
* Analyzing lease expirations, compliance, financials, and operational performance.
* Triggering workflows like:
  + Risk detection (expired or soon-to-expire leases)
  + Payment projections
  + Compliance document generation

This means **multiple specialized “AI assistants” (agents)** should work together as a team.

**2. Proposed Agent Team Structure**

Think of it as **an org chart** of specialized agents for this use case.  
Here is how it would look:

**Team Lead**

**Oilfield Lease Orchestrator Agent**

* Coordinates the other agents
* Decides execution order
* Collects outputs and writes final results back to the dashboard

**Specialized Agents (connected to the Lead)**

**Data & Ingestion**

1. **Data Ingest Agent**
   * Pulls lease data from CSVs/APIs
   * Validates fields, normalizes data
   * Sends structured data downstream

**Lease Risk Analysis**

1. **Lease Expiration Risk Agent**
   * Monitors all lease dates
   * Identifies leases that will expire soon
   * Calculates potential revenue at risk
2. **Revenue & Payment Forecast Agent**
   * Projects payments and revenues based on terms
   * Identifies anomalies in historical patterns

**Compliance & Integrity**

1. **Compliance Analysis Agent**
   * Reviews compliance checklists (environmental, regulatory, financial)
   * Flags compliance risks
2. **Document Generation Agent**
   * Creates compliance reports, renewal recommendations
   * Summarizes findings for export

**Integration & Notifications**

1. **ERP Integration Agent**
   * Sends decisions and updates to external ERP systems
   * Logs all outbound decisions
2. **Alert & Notification Agent**
   * Sends alerts about high-risk leases
   * Provides a summary for decision makers

**Diagrammatically (in Agent Orchestration Panel):**

yaml

CopyEdit

[ Oilfield Lease Orchestrator Agent ]

/ | \

/ | \

Ingest Risk/Forecast Compliance

| / | \ |

| Expiration Payment |

| Forecast |

Integration Document Gen

|

Notification

This looks like a **hub-and-spoke team** with a coordinator (hub).

**3. Where These Agents Should Show Up**

**In the Agent Orchestration Panel**

* When the user opens **Agent Orchestration** for the Oilfield Land Lease use case,  
  the system **auto-loads this preset graph**.
* The user can:
  + Add new agents
  + Rewire connections
  + Save new versions

**In the Use Case – Operations Tab**

* The **Operations tab** should list:
  + **All agents currently assigned** to this use case
  + Their roles and current status
  + Example:

diff

CopyEdit

Assigned Agents:

- Oilfield Lease Orchestrator Agent (Coordinator)

- Data Ingest Agent (Operational)

- Lease Expiration Risk Agent (Operational)

- Revenue Forecast Agent (Operational)

- Compliance Analysis Agent (Operational)

- Document Generation Agent (Operational)

- ERP Integration Agent (Operational)

- Notification Agent (Operational)

This directly ties **Operations** to **Agent Orchestration**.

**4. Traceability Across the Platform**

Once these agents exist:

1. **Workflows:** When a workflow step is created, the user can select any of these agents.
2. **Deployment:** Deploying this use case means **deploying the team**.
3. **Audit/Integration Logs:** All decisions/actions are tagged with the agent responsible.

**5. Next Step (Design Implications)**

**Before building the registry:**

* The **Agent Orchestration Panel must support:**
  + Pre-loaded agent graph for each use case.
  + Saving these agent team templates.
  + Assigning an agent graph to a specific use case.
* The **Operations tab must be linked** to that graph and display those agents.

**Key Insight**

**Every use case = a team of agents**  
The registry stores agents.  
The orchestration panel visualizes the team.  
The operations tab shows the team at work.

**Do you want me to draft the KiloCode prompt for:**

1. **Implementing pre-configured agent graphs for each use case (starting with Oilfield Land Lease),**
2. **Making the Operations tab pull directly from that agent team,**
3. **And setting the stage for a global “My Agents” registry later?**

This would make Oilfield Land Lease the **first example of a connected multi-agent system inside Seraphim.**

**You said:**

yes I do - or maybe like you say - have a filter on the agent orchestration panel where I can choose the use case and it gives me the preselected agent orchestration in addition to creating my own! that's a much better idea!

**ChatGPT said:**

Perfect—that approach (a **filter dropdown on the Agent Orchestration panel**) is exactly the right way to make this scalable:

* Users can **pick a use case** (e.g., Oilfield Land Lease), see the **pre-built team of agents** for that case, and edit it.
* Or they can switch to **“Custom/Global”** to build agents from scratch.

This combines **use-case-specific orchestration** with the flexibility of custom multi-agent teams.

**KiloCode Prompt: Pre-Configured Use Case Agent Teams with Filtered Agent Orchestration Panel**

markdown

CopyEdit

You are upgrading the Seraphim Agent Orchestration Panel to support \*\*use-case-specific agent teams\*\* with a filter-driven interface.

## Goal

Enable users to:

1. Select a \*\*Use Case\*\* from a dropdown at the top of the Agent Orchestration panel.

2. Automatically see the \*\*pre-built multi-agent team graph\*\* for that use case.

3. Edit or extend that team, save changes, and redeploy.

4. Alternatively, switch to “Custom/Global” to create standalone agent teams.

---

## Changes to the Agent Orchestration Panel

### 1. Use Case Filter

- Add a dropdown at the top of the panel:

- \*\*Values:\*\* All available Use Cases (from /api/usecases) + “Custom/Global”

- Default: “Custom/Global”

- When a use case is selected:

- Load its pre-configured agent graph (from /api/usecase/agents/<usecase\_id>).

- Display the graph in the canvas.

---

### 2. Pre-Built Agent Graphs

- For \*\*Oilfield Land Lease\*\* (Energy vertical), load this \*\*preset orchestration team:\*\*

- \*\*Hub (Coordinator):\*\* Oilfield Lease Orchestrator Agent

- \*\*Spokes:\*\*

1. Data Ingest Agent

2. Lease Expiration Risk Agent

3. Revenue Forecast Agent

4. Compliance Analysis Agent

5. Document Generation Agent

6. ERP Integration Agent

7. Alert & Notification Agent

- \*\*Connections:\*\*

- Orchestrator Agent at the center.

- Ingest Agent feeds into Risk/Forecast and Compliance.

- Risk/Forecast and Compliance feed into Document Generation.

- Document Generation outputs to ERP Integration and Notifications.

---

### 3. Visual Interaction

- Allow agents to be:

- Dragged, repositioned, deleted

- New agents can be added

- Connections can be drawn/removed

- Save Button:

- Updates the orchestration definition for that use case.

- Deploy Button:

- Initiates the deployment pipeline using the current graph.

---

### 4. Cross-Page Integration

- \*\*Operations Tab:\*\*

- Query /api/usecase/agents/<usecase\_id> and show a \*\*list of active agents\*\* with status.

- \*\*Workflows:\*\*

- Allow any agent in this graph to be selected as a step.

- \*\*Deployment:\*\*

- Deploying this use case deploys the current version of its agent graph.

---

### 5. APIs

- Add endpoint: GET /api/usecases → List of use cases.

- Add endpoint: GET /api/usecase/agents/<usecase\_id> → Returns agent graph JSON.

- Add endpoint: POST /api/usecase/agents/<usecase\_id> → Save updated agent graph.

\*\*Agent Graph JSON format:\*\*

{  
"usecase\_id": "oilfield\_land\_lease",  
"agents": [  
{"id":"orchestrator","name":"Oilfield Lease Orchestrator Agent","type":"coordinator","connections":["ingest","risk","forecast","compliance"]},  
{"id":"ingest","name":"Data Ingest Agent","type":"custom","connections":["risk","forecast","compliance"]},  
{"id":"risk","name":"Lease Expiration Risk Agent","type":"custom","connections":["docgen"]},  
{"id":"forecast","name":"Revenue Forecast Agent","type":"custom","connections":["docgen"]},  
{"id":"compliance","name":"Compliance Analysis Agent","type":"custom","connections":["docgen"]},  
{"id":"docgen","name":"Document Generation Agent","type":"custom","connections":["erp","notify"]},  
{"id":"erp","name":"ERP Integration Agent","type":"custom","connections":[]},  
{"id":"notify","name":"Alert & Notification Agent","type":"custom","connections":[]}  
]  
}

yaml

CopyEdit

---

### 6. UX Rules

- When switching use cases, prompt: “Do you want to save your current graph before switching?”

- Use different colors for \*\*hub agents\*\* vs. specialized agents.

- Display agent status badges (Operational, Disabled, Error).

---

### 7. Demo Path

1. Select \*\*Oilfield Land Lease\*\* in the dropdown.

2. See the \*\*pre-built agent graph\*\*.

3. Add a new agent (e.g., “AI Contract Review Agent”) and link it to Document Generation.

4. Save → Now this new agent appears in:

- Operations Tab for Oilfield Land Lease

- Workflows (as a step)

- Deployment pipeline.

---

This change enables \*\*use-case-driven multi-agent teams\*\*, makes orchestration reusable, and connects the Agent Orchestration module with Operations, Workflows, and Deployment.

**How this changes user experience**

* **Single dropdown:** Switch between “Custom” and “Use Case” graphs.
* **Pre-built teams:** Each use case starts with a full agent team (like an org chart).
* **Traceable:** These agents now appear everywhere automatically.

Would you like me to also:

1. Draft a **validation prompt** specifically for this new feature (to ensure the dropdown, preloading, saving, and linking work as expected)?
2. Draft the **future My Agents Registry** prompt (now that the orchestration behavior is clear)?
3. Or both?

**You said:**

wait - before we prompt let's think this through more - wait until I tell you to write the prompt. we are going to add this to the previous prompts on the workflows, deployment, etc and use this use case to do so. so, take me through that use case on the workflows, agent orchestration, and deployment orchestration in the best way possible. then let's think through what needs to happen in the integration log, audit console, and output viewer. what needs to happen?

**ChatGPT said:**

Excellent. Let’s slow this down and **mentally walk through the entire lifecycle of the Oilfield Land Lease use case** across **Workflows → Agent Orchestration → Deployment Orchestration**, and then look at **Integration Log, Audit Console, and Output Viewer** to make sure we understand how they all fit together.

I’ll narrate it as if we are following a user’s experience and then describe the **underlying requirements** at each stage.

**1. Workflow Stage – Where the Use Case Starts**

**User Intent:**  
“I need Seraphim to analyze and manage all of my oilfield leases. Automate my process.”

**What Happens Here:**

1. **User opens Workflows and selects “Oilfield Land Lease Workflow.”**
2. **Workflow steps are shown visually** (left to right, or top to bottom).  
   These steps are **abstract tasks** like:
   * Step 1: Ingest lease data
   * Step 2: Analyze expirations
   * Step 3: Forecast revenue
   * Step 4: Generate compliance reports
   * Step 5: Push results to ERP
3. **Behind the scenes:**  
   Each step **calls one or more agents** from the **agent graph defined in Agent Orchestration**.
4. **User actions at this stage:**
   * Reorder steps
   * Add/remove steps
   * Edit parameters (e.g., select “leases from last 3 years”)
   * Hit **Run** (on-demand analysis) or schedule for recurring execution.

**Key Concept:**  
**Workflows are the high-level business automation view.** They are powered by the underlying agent team (but the user doesn’t need to think about which agent—they just see tasks).

**Requirement:**

* Each workflow step should **link directly to an agent node** (from the orchestration panel).
* The Workflow **UI must pull from the agent graph** so that adding/removing an agent updates the workflow options.

**2. Agent Orchestration – The Brains and Team Layout**

**User Intent:**  
“I want to see and control the team of agents that runs my Oilfield Land Lease workflow.”

**What Happens Here:**

1. **User goes to Agent Orchestration.**
2. Selects **“Oilfield Land Lease” in the dropdown**.  
   The pre-built **agent graph** loads (hub-and-spoke layout we discussed earlier).
3. This graph shows:
   * **Hub agent** (Orchestrator) at center
   * Specialized agents around it (Data Ingest, Risk, Compliance, etc.)
4. **User actions at this stage:**
   * Add a new agent (e.g., “Satellite Imagery Agent”)
   * Connect it to the orchestrator and compliance agent
   * Delete, rewire, or tune agents
   * Save changes → which updates the **agent team for that use case**.

**Key Concept:**  
**Agent Orchestration is the low-level technical view.** It’s the **AI team builder** that Workflow steps consume.

**Requirement:**

* Saving this orchestration automatically updates:
  + **Available steps in the Workflow panel**
  + **Operations tab (list of agents)**
  + **Deployment pipeline (what gets deployed)**

**3. Deployment Orchestration – From Graph to Production**

**User Intent:**  
“I am ready to run this system in a production-like mode.”

**What Happens Here:**

1. **User selects Deploy for Oilfield Land Lease.**
2. **Deployment Pipeline starts:**
   * Pre-flight checks
   * Security & dependency validation
   * Provisioning resources
   * Deploying each agent
3. **Live Status Display:**
   * Shows the deployment step-by-step.
   * If a particular agent fails to deploy, the pipeline reports it (and rolls back if necessary).
4. Once complete:
   * The system now runs **live automations on schedule** (not just manual runs).

**Key Concept:**  
**Deployment orchestrates the entire team as a production-ready system.**

**Requirement:**

* Deployment must **read the latest agent graph for that use case**.
* Log every deployment action (agent deployed, errors, warnings).

**4. Now: How This Affects Integration Log, Audit Console, Output Viewer**

These three modules are **horizontal layers** that cut across all use cases and orchestrations.

**Integration Log (System-to-System Events)**

**Purpose:** Shows **external connections** (APIs, data feeds, ERP updates).

For Oilfield Land Lease:

* When the **Data Ingest Agent pulls data** from an external source, an entry is logged:
  + Timestamp
  + Source (API or file)
  + Volume of data
* When **ERP Integration Agent pushes results** to an external ERP system:
  + Log the payload and confirmation.

**Key:**

* Focus on **data entering and leaving Seraphim**.
* Should have filters: by use case, by agent, by date.

**Audit Console (Internal User and Agent Actions)**

**Purpose:** Shows **what actions were performed internally**.

For Oilfield Land Lease:

* Tracks:
  + **User-driven events:** “User ran workflow,” “User added agent.”
  + **Agent-driven events:** “Lease Risk Agent flagged 3 leases expiring soon.”
  + **System decisions:** “Workflow rerun due to new data.”
* This ensures **accountability** and a traceable decision history.

**Output Viewer (Results and Artifacts)**

**Purpose:** Shows **outputs produced by agents/workflows**.

For Oilfield Land Lease:

* Stores generated:
  + Compliance reports (PDF)
  + Risk heatmaps
  + Forecast charts
* Organized by **use case, run date, and workflow instance**.
* Lets users **download reports or drill down into results**.

**Key:**  
This becomes **the data lake of deliverables** produced by Seraphim.

**5. How Everything Connects (Single Flow)**

1. **Workflows:**  
   “Run Oilfield Land Lease Workflow” → triggers tasks
2. **Agent Orchestration:**  
   Tasks = nodes in the agent graph
3. **Deployment:**  
   Ensures all nodes/agents are running and provisioned
4. **Integration Log:**  
   Tracks external API/data interactions
5. **Audit Console:**  
   Tracks internal agent actions & user decisions
6. **Output Viewer:**  
   Stores and organizes results (reports, insights)

**Where We Go From Here**

This mental model tells us:

* **Dropdown Filter in Orchestration:** Perfect way to bind use cases to agent teams.
* **Workflow <-> Agent Graph sync:** Changes in one must reflect in the other.
* **Operations Tab:** Shows live list of assigned agents for the selected use case.
* **Integration Log / Audit Console / Output Viewer:** These become the observability layer, not just admin tools.

**Question for you:**

Would you like me to now:

1. **Draft a single combined master prompt** (integrating this full flow and all modules), using **Oilfield Land Lease** as the reference template?
2. Or first create a **diagram** (org chart of how workflows, agents, deployment, and logs tie together) before writing the master prompt?

**(Diagram first is ideal, so the prompt can follow it exactly.)**

**You said:**

diagram - just list out the diagram here

**ChatGPT said:**

Here’s a **structured diagram in text form** showing how the **Oilfield Land Lease** use case flows across the key modules:

**Seraphim Use Case Flow (Oilfield Land Lease Example)**

pgsql

CopyEdit

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

| Workflows Module |

|---------------------------|

| Business Process Steps |

| (abstract tasks): |

| 1. Ingest Lease Data |

| 2. Analyze Expirations |

| 3. Forecast Revenue |

| 4. Compliance Reports |

| 5. Push Results to ERP |

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

|

v

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

| Agent Orchestration |

| (Use Case: Oilfield) |

|---------------------------|

| Hub Agent: |

| - Oilfield Orchestrator|

| Connected Specialized |

| Agents: |

| - Data Ingest Agent |

| - Lease Risk Agent |

| - Revenue Forecast |

| - Compliance Analysis |

| - Document Generation |

| - ERP Integration |

| - Notification Agent |

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

|

v

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

| Deployment Orchestration |

|---------------------------|

| Pipeline: |

| Pre-flight Checks |

| Security Validation |

| Provision Resources |

| Deploy Agents |

| Monitor Health |

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

|

---------------------------------------------------------

| | |

v v v

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

| Integration Log | | Audit Console | | Output Viewer |

|------------------| |----------------------| |----------------------|

| External Events: | | Internal Actions: | | Results: |

| - Data ingested | | - User ran workflow | | - Reports (PDF) |

| - ERP updates | | - Agents flagged | | - Charts & Analytics |

| - API calls | | - Model decisions | | - Summaries |

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

**Key Insights from the Diagram**

* **Workflows** = Business-level view  
  (Steps mapped to tasks, user doesn’t need to know which agent does them)
* **Agent Orchestration** = Technical team view  
  (Agents and their connections: the “AI org chart”)
* **Deployment** = Makes the agents run in production
* **Integration Log / Audit Console / Output Viewer** =  
  **Three “observability” windows:**
  + **Integration Log** = External system touchpoints
  + **Audit Console** = Who did what inside Seraphim
  + **Output Viewer** = Deliverables created by agents