Assignment: Software Engineer, Agents

Take-Home Assignment: Configurable Conversational Agent

Goal

Build a small **conversational agent** that collects user information, handles light interruptions, and **escalates conversations to a human** when appropriate.

We are evaluating:

- Software design and code quality (would we merge this?)
- Configuration system quality
- Agent behavior control
- Agentic patterns and extensibility
- Testing strategy
- Practical judgment

Requirements

1) Agent Runtime Interface

Build an asynchronous (`async`) runtime that can execute multiple agent instances and manage their state. We want to see a clear separation between:

- Agent logic
- Orchestration

- Infrastructure

---

2) Conversation Flow

The agent must:

1. Always begin a new conversation with a **configured custom greeting**.
2. Collect configured fields **one at a time**.
3. Allow **corrections** (e.g., “No, my email is…”).
4. Perform **minimal validation** (basic format checks are sufficient).
5. Escalate when:
    - all required fields are collected, **or**
    - the user triggers an escalation reason based on a configured policy.
6. Handle off-topic responses and redirect the user back to the intended goal.

---

3) Configuration Requirements

All configuration must be external (YAML or JSON) and validated.

1. Agent Personality

Configurable attributes should include:

- tone
- style
- formality
- emoji usage
- customizable emoji list

2. Custom Greeting Message

A configurable greeting message that is always used at the start of a new conversation.

3. Fields to Collect

Base required fields:

- Name
- Address
- Phone
- Email

The system **must support N configurable fields**, defined entirely via configuration. Consider how multiple collection attempts are handled.

4. Escalation

Each escalation policy must define:

- `enabled` — whether the escalation is active
- `reason` — a free-text description of the policy

State Model

Keep state in memory for this exercise, but implement a **state store abstraction** that can be swapped for a database later.

State should track:

- messages
- collected fields, including confidence and validation status
- the current field being requested
- escalation status and reason

No global state.

Constraints

- Python 3
- Async-friendly design
- Avoid heavy frameworks unless clearly justified

Allowed Libraries

Use good judgment. A small set of common utilities is acceptable (e.g., `pydantic`, `pyyaml`, `httpx`, `pytest`). Do not rely on external SaaS services beyond your chosen LLM provider.

If in doubt, please confirm with the team.

Testing Expectations

We are looking for tests that validate **behavior and state transitions**.

Guidelines:

- Mock the **LLM boundary only**.
- Avoid tests that primarily assert mock call details.
- Include tests for:
  - configuration validation errors
  - field collection order
  - correction handling
  - escalation payload correctness
  - clear reporting of pass/fail statistics

Evaluation Rubric (High Level)

We will score:

1. **Architecture and separation of concerns**
2. **Configuration design and validation**
3. **Agent behavior quality and state handling**
4. Agentic patterns and extensibility
5. **Test quality**
6. **Code clarity and maintainability**
7. **Judgment** (questions asked, scope decisions, tradeoffs)

Clarifying Questions

If you have questions or need clarification, please feel free to reach out to oscar@konko.ai and cc michael@konko.ai

Repository & Submission

- **Distribution:** A proper Python package, version-controlled on GitHub.
- Grant access to: `mahaddad`, `oargueta3`, `zubenkoivan`, `dalazx`.
- Include:
  - a `README` with run instructions
  - sample configuration files
  - a `DECISIONS.md` describing:
    - architecture choices
    - tradeoffs
    - what you would do with one additional day
    - how (if at all) you used AI coding tools