Assignment: Senior Solutions Engineer (AI Agents)

Overview

We're looking for an engineer with a strong commitment to engineering best practices, high-quality code, and thoughtful prompt design for LLM-powered agents. In this exercise, we'll evaluate your ability to:

- Design and implement a **small configurable AI agent**
- Use **prompt engineering** to control the agent's behavior (personality, greetings, escalation)
- Produce **maintainable, robust, and well-structured code** that follows modern software engineering principles

Your goal is to build a minimal **RESTful Web API** that exposes a **configurable customer-support intake agent**. This agent conducts a short conversation to collect information and then **escalates** to a human with a clear summary—either after all fields are collected or when an escalation trigger fires.

Product Requirements

Build a small AI agent that:

1. Collect user information

Base required fields:

- Name
- Address
- Phone
- Email

Must support additional configurable fields, defined via JSON/YAML.

2. Escalate to a human

- After *all required fields* are collected
- **OR earlier**, if a configured escalation trigger appears (e.g., "urgent," "refund request," "complex billing issue," "angry customer")

3. Be fully configurable

Loaded from simple JSON/YAML files in the repository.

Configurable items include:

- Agent personality (tone, style, formality)
- Initial greeting message
- Agent name and title
- Company/brand name
- List and order of fields to collect
- Allowed escalation reasons

User Story

As a user, I want to chat with an AI intake agent that politely gathers my contact information and escalates my case after collecting all required information or when needed.

Example Conversation

Configuration (simplified):

- Greeting: "Hola, soy Kora, tu asistente. Te ayudo a registrar tus datos para que podamos contactarte."
- Personality: "Warm, concise, helpful coordinator"
- Fields to collect: name, phone, email
- Escalation: Enabled with reasons: ["urgent", "complex_question"]

Conversation:

```
User: Hola

Agent: Hola, soy Kora, tu asistente. Te ayudo a registrar tus datos
para que podamos contactarte. ¿Cuál es tu nombre?

User: Soy Oscar Argueta.

Agent: Mucho gusto, Oscar. ¿Me compartes tu número de teléfono?

User: +506 8888 9999

Agent: Gracias. ¿Y tu correo electrónico?

User: oscar@example.com

Agent: Perfecto. Ya tengo tus datos. En unos momentos un miembro de
nuestro equipo se pondra en contacto para ayudarte
```

Behavioral Requirements

1. Always start with the custom greeting

Every conversation must begin with the greeting defined in configuration.

2. Field collection flow

- Ask **one field at a time**
- Minimal validation (email contains @, phone has digits)
- Allow corrections
- After all fields → escalate

3. Small-talk handling (Extra Credit)

Short unrelated questions may be answered briefly, then return to the collection flow.

Note: For simplicity assume the chat is turn-by-turn — the client must wait for the agent's reply before sending the next message.

Functional Requirements

Build an API that handles the following operations:

1. **Create a new conversation**
POST /conversations/
2. **Send a message to the agent**
POST /conversations/{conversation_id}/messages/
3. **Get the chat history**
GET /conversations/{conversation_id}/messages/
4. **Retrieve conversation state**
GET /conversations/{conversation_id}

Implementation Requirements

- **Runtime:** Python 3 with an asynchronous concurrency model
- **Distribution:** A proper Python package, version-controlled on GitHub.

Please grant `mahaddad`, `oargueta3`, `zubenkoivan`, and `dalazx` access to your repository.

- **Code quality:**
- Keep the codebase as close to a production state as is reasonable for an exercise.
- Prefer clear separation of concerns:
* Agent logic (state machine + prompt builder)
* Configuration loading
* API layer
* LLM client abstraction

- **Tests:**
- Tests are expected

- Use your judgment to write tests for the most critical components.

• **State:**

- For simplicity, keep the state in memory, but provide an abstraction that could be swapped for a database.

• **Agentic 3rd-party Libraries:**

- Do **not** use any 3rd-party agentic services/libraries except for an LLM API provider of your choice.

Extra Credit

Add basic **quality checks/evals**:

For example, a small script that runs through a few predefined conversations and prints whether the agent:

- Collected all fields
- Responded in the correct language / tone
- Check escalation happened correctly