Python LLM Agent Designer - Technical Assessment

Prepared by Teams Squared

Total Time: 1 hour

Submission Instructions

1. **Clone this repository** and create a **new branch** using the naming format: `<your_first_name>_ai-agent-engineer`
2. Work on your branch. **Commit frequently** with meaningful messages (e.g., “Add query route”, “Implement RAG pipeline”).
3. Once finished, **submit a pull request to** `main` with a short summary of what you’ve completed.
4. A Teams Squared engineer will review your pull request and provide feedback within 24–48 hours.

If you have any questions or need clarification during the assessment, leave a comment in your pull request.

What are you building?

- You’re building a **Policy Assistant** - an LLM + RAG-powered API that can interpret and answer user queries about a company’s refund and return policies.
- Your task is to:
  - Build a **Flask API** that uses **OpenAI** and **LangChain** to retrieve relevant sections from these policies.
  - Ensure that answers are returned in **structured JSON format** (see below) to support chatbot or dashboard integrations.
  - **For bonus points**, you may also build a simple React (TypeScript) frontend for interacting with this API.
- The policies are stored as Markdown files in `backend/policies/`. The project structure is as follows:

```
/frontend/ # React w/ TypeScript (Vite-based) frontend
/backend/ # Python Flask backend
├── app.py
├── policies/
│ ├── refund.md
│ └── return.md
└── requirements.txt
```

- If you're ready, you may begin each of the following tasks **one-by-one**. *Good luck to you!*

Task 1 - Flask Backend API

- Accepts a JSON payload like:

```
{ "question": "What is the refund deadline for digital products?" }
```

- Uses LangChain with OpenAI to generate a context-aware answer based only on `policies/*.md`.
- Returns a structured JSON response using a **suitable prompt**:

```
{
    "summary": "Refunds for digital products are not allowed once the license is activated.",
    "bullets": [
        "Refund requests must be made within 14 days.",
        "Product must be unused and license unactivated.",
        "Partial refunds are possible after 14 days."
    ]
}
```

- If no relevant information is found:

```
{
    "summary": "insufficient context",
    "bullets": []
}
```

- **Environment Setup:**
  - Create a Python virtual environment in `/backend/`
  - Install necessary packages and freeze them to `requirements.txt`
  - Use `python-dotenv` to manage API keys via a `.env` file
  - Please find an **OpenAI** key via this link and add it to the `.env` file, making sure **not to commit it to the repository**.
- You are encouraged to use techniques such as `StructuredOutput` (s) or `TypedDict` (s) to enforce the JSON output structure.

Task 2 - RAG Pipeline (Retrieval-Augmented Generation)

- **Load, split, and embed** the Markdown documents from `backend/policies/` using **LangChain**.
- Use an in-memory vector store like **FAISS** or **Chroma**.
- Ensure the agent **does not hallucinate** - answers must come from retrieved content only.

Task 3 – Frontend UI (Optional Bonus)

- In `frontend/`, complete the simple React (Vite-based) UI:
  - Add a chat-like interface with a text input and "Ask" button.
  - On submit, send a `POST` request to the Flask `/query` endpoint.
  - Display the returned `summary` and `bullets` **below the input**.
  - No need for conversation history - just display the current response.

Evaluation Criteria

Area	Weight	Notes
Flask API functionality	25%	Route setup, request parsing, JSON response
RAG integration	25%	Proper embedding, retrieval, chunking
Prompt consistency	25%	JSON enforcement, hallucination control
Realism of output	25%	Answers match context, clean bullets
Frontend UI	+25%	Input box + display output