

## Overview

This architecture represents a **multi-step AI workflow** for evaluating user eligibility for **social and economic support**, from document upload to AI decisioning and persistent storage. The system consists of a **Streamlit front-end**, **LLM-enhanced parsing and decision engines**, and **asynchronous Flask API microservices** that run inside Docker containers, with final responses stored in a PostgreSQL database.


### **NOTE: LLM Flexibility and Alternative Setup with Ollama**

The current application flow utilizes **OpenAI's GPT-4** as the core LLM for all LangChain agents and supervisors. GPT-4 is responsible for summarization, structured reasoning, vector-based retrieval, and deterministic decision-making via function (tool) calls.

However, this design is **LLM-agnostic** and can be easily swapped to run **fully locally** using [Ollama](#) with models like **LLaMA 3**, as long as the chosen model supports:

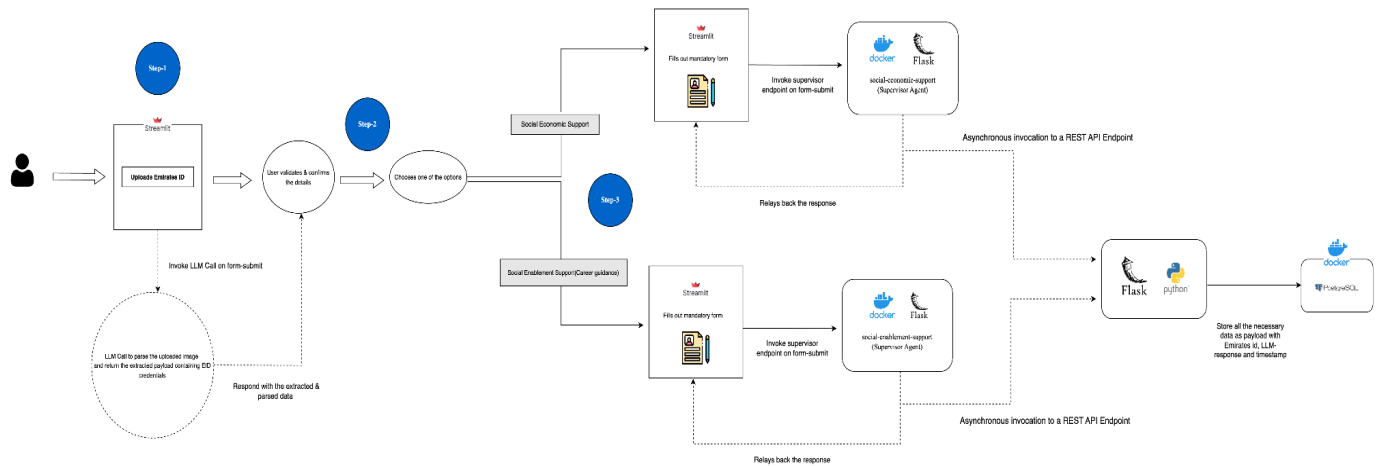
- **Function calling / tool use**
- **Structured JSON response formatting**
- **Consistent streaming output**

A **Jupyter notebook** version of this entire flow (using the same supervisors and logic) with **LLaMA 3 + Ollama** has been included in the same GitHub repository at .

 **Note:** Due to intermittent **response formatting issues with LLaMA 3** or any other local models , the Ollama version has **not been used in the main production flow**. Implementing a locally hosted model is feasible but would require additional research and experimentation to integrate it effectively with LangGraph or any orchestration framework that supports multi-agent stateful execution.

---

## High Level Flow Diagram :



**Langsmith** - has been used throughout the app flow for debugging and observability.

## Step-by-Step Flow

### ✓ Step 1: Emirates ID Upload via Streamlit

- A user uploads an Emirates ID image through a **Streamlit-based UI**.
- On form submission:
  - A **LLM call** is triggered to **parse the image** using multimodal GPT4 , extracting structured fields such as:
    - Emirates ID number
    - Full name
    - DOB, expiry date, nationality, etc.

🔄 The extracted data is returned and displayed on the form for **user validation and confirmation**.

---

## ✅ Step 2: Option Selection

- Once the user confirms the extracted data, they are prompted to choose one of two evaluation flows:
  1. **Social Economic Support**
  2. **Social Enablement Support (Career Guidance)**

Each branch invokes a different **supervisor agent** via REST APIs.

---

## 🧠 Decision Branch A: Social Economic Support

### ♦ Step 3A: Mandatory Form Filled via Streamlit

- User fills out additional information (e.g., income, marital status, number of children).
- On form submission, a call is made to:
  - `social-economic-support supervisor agent`
  - Hosted inside a Docker container running a **Flask app**

## 🔄 LLM + Agent Logic

- The supervisor internally invokes:
    - `credit_risk_evaluator_agent`
    - `financial_burden_evaluator_agent`
  - Final decision is returned (e.g., “Approved”, “Soft Decline”).
- 

## 🧠 Decision Branch B: Social Enablement Support

### ♦ Step 3B: Mandatory Career Guidance Form

- User fills in career-related metadata (last drawn salary, employment domain, resume).
- On submit, a REST API call is made to:
  - `social-enablement-support supervisor agent` via Docker+Flask

### LLM + Agent Logic

- The supervisor invokes:
  - `career_readiness_agent`
  - `upskilling_program_agent`
- Returns a **narrative career plan** including:
  - Employment gap
  - Rehire potential
  - Suggested learning paths

---

### Final Step: Data Storage in PostgreSQL

- Both branches (`social-economic-support` & `social-enablement-support`) return structured JSON results.
  - A separate **Flask service**:
    - Receives the LLM outputs
    - Enriches it with metadata (Emirates ID, timestamps, LLM Output )
    - Stores the final result in a **PostgreSQL database**
-

## Infrastructure Highlights

Layer	Description
Frontend	Streamlit app handles UI interaction and initial uploads
Model Inference	LangGraph + LangChain agents running in Flask Docker containers
API Comm	REST API calls between UI and backend (asynchronous supervisor invocation running in an isolated container)
Persistence	PostgreSQL instance running inside an isolated Container stores all final outcomes and payloads for audit, review, and reporting

---

## Summary

This architecture delivers an **end-to-end AI decision pipeline**, combining:

- User-friendly document upload,
- LLM-driven form parsing,
- Agent-based decision engines (social support + career),
- Seamless REST-based API orchestration,
- Robust containerized deployment and persistence via PostgreSQL.