## Executive Summary

This project presents a prototype AI-driven solution designed to automate the social support application workflow for a government social security department. The goal is to reduce the current processing time from 5–20 days to a matter of minutes by leveraging locally hosted AI models, advanced machine learning, and agent-based orchestration.

The system features a multimodal, end-to-end architecture that ingests documents (PDFs, images, Excel files), extracts and validates structured data, performs eligibility checks, and delivers real-time decisions and recommendations via a user-facing GenAI chatbot. Built using Python and a modular architecture, the solution incorporates OCR, document parsing, ML classifiers, and vector databases to ensure accuracy and scalability.

Key technologies include Streamlit for the frontend, Scikit-learn for eligibility classification, Qdrant for vector similarity matching, PostgreSQL/MongoDB for data storage, and LangGraph/Crew.AI for agent orchestration. Observability is managed through Langfuse, ensuring transparency and traceability of AI outputs.

This solution demonstrates how AI can significantly streamline government workflows, reduce human bias, and provide more equitable, efficient access to economic and social support for citizens.

### **- Problem Statement**

The social support application process within the government social security department is currently slow, error-prone, and heavily reliant on manual effort. Applicants often wait between 5 to 20 working days for decisions, due to:

* Manual data entry from scanned or handwritten documents
* Inconsistent and unverified applicant information across forms
* Bottlenecks caused by multi-step human review across departments
* Subjective and potentially biased decision-making processes
* Lack of automation in determining eligibility and recommending support actions

This leads to inefficiencies, delays in delivering aid, and inconsistent service to citizens in need of timely economic support.

### **- AI-Driven Opportunity**

The aim is to build an AI-based system that:

* Automates up to 99% of the end-to-end application workflow
* Uses a GenAI chatbot to guide and interact with users in real time
* Processes multimodal data (text, images, spreadsheets)
* Evaluates eligibility using a combination of LLMs, ML classifiers, and rule-based logic
* Recommends decisions and enablement paths (e.g., upskilling, jobs)
* Ensures observability and auditability for fairness and compliance

### **- Assumptions**

1. Synthetic or mock data is allowed for prototyping (no real PII is used).
2. All applicants provide the following documents digitally:
   * Emirates ID (scanned image)
   * Bank statements (PDF)
   * Resume (text or PDF)
   * Excel file for assets and liabilities
   * Credit bureau report
3. Basic eligibility criteria are available for classification:
   * Income threshold, family size, employment history, asset limits, etc.
4. Local models (LLMs and ML) are used instead of external APIs due to data privacy needs.
5. Government workflows allow system recommendations to be reviewed before final decision.
6. Document formats are semi-structured and OCR-capable.
7. The decision engine must be transparent and explainable.
8. Multiple departments may review output via a central system interface.

## **SOLUTION STRATEGY OVERVIEW**

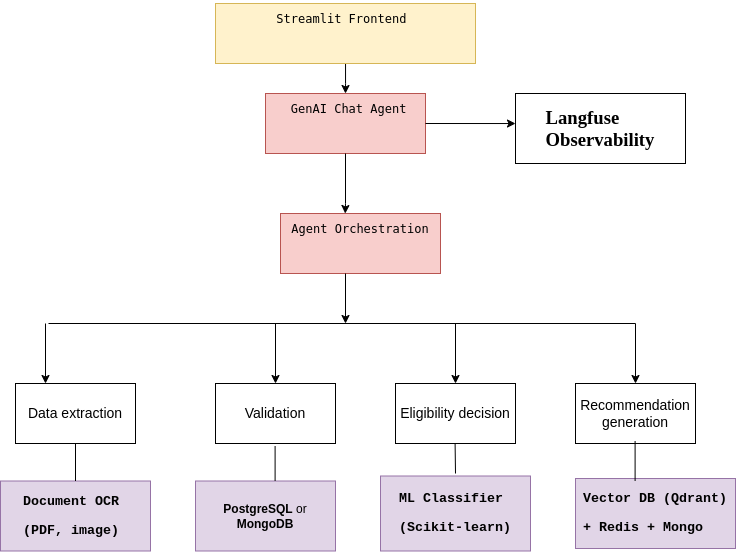
### **Objective**

Automate decision-making in social support applications (99% automated) using GenAI + ML + RAG + agentic orchestration.

### **Key Users**

* Government officers
* Applicants (residents/citizens)

## PHASE 1: **ARCHITECTURE (High-Level)**



Phase 2 architecture diagram represents the high-level design of the AI workflow for automating social support application processing. It shows how different components interact to process user data, perform validations, and generate decisions using AI agents.

### **1. Streamlit Frontend**

* Acts as the user interface for applicants and government agents.
* Allows users to:
  + Upload documents (e.g., bank statements, ID, Excel files)
  + Chat with the AI assistant
  + View application status

### **2. GenAI Chat Agent**

* This is the main conversational interface (LLM-powered).
* Users interact with it in natural language.
* It parses intent and routes data or requests to the orchestration layer.

### **3. Langfuse Observability**

* Connected to the chat agent to provide real-time monitoring.
* Tracks:
  + Prompt responses
  + Agent performance
  + Error handling
* Ensures transparency and debugging ability for AI workflows.

### **4. Agent Orchestration**

* This is the "brain" of the system.
* Orchestrates tasks between various AI agents using tools like LangGraph, Crew.AI, or Synthetic Kernel.
* Based on user inputs or uploaded documents, it triggers:
  + Data extraction
  + Validation
  + Eligibility decision
  + Recommendation generation

## **Modular AI Agents**

Each agent has a specific responsibility, promoting modularity and scalability:

### **A. Data Extraction Agent**

* Extracts data from:
  + PDFs (bank statements)
  + Images (Emirates ID)
  + Excel (assets/liabilities)
* Uses OCR tools (e.g., Tesseract or PyMuPDF)

### **B. Validation Agent**

* Compares data between different documents (e.g., form vs. credit report)
* Checks for:
  + Inconsistent addresses
  + Conflicting family info
* Stores clean data in PostgreSQL or MongoDB

### **C. Eligibility Agent**

* Uses Scikit-learn ML models to:
  + Evaluate income, employment, demographics
  + Predict whether the applicant qualifies for support

### **D. Recommendation Agent**

* Suggests economic enablement actions like:
  + Upskilling
  + Job matches
  + Counseling
* Leverages Qdrant (Vector DB) for semantic similarity search (e.g., resume → matched job profiles)

## **Data Flow Summary**

1. User uploads files or asks questions via Streamlit
2. GenAI Chat Agent routes the task
3. Agent Orchestration coordinates multiple specialized agents
4. Each agent processes different data types
5. Outputs are monitored via Langfuse and returned to the user

## PHASE **2**: **KEY MODULES TO BE IMPLEMENT**

### 1. **Data Ingestion Module**

* Handle inputs: scanned Emirates ID, bank statements, resumes, Excel sheets
* Use OCR (e.g., Tesseract for image/text), pandas for Excel
* Store structured data in PostgreSQL / MongoDB

### 2. **Multimodal Processing**

* Text: GPT-4 or local LLM (Ollama) for summarization/extraction
* Image: OCR with layout-aware parsing (e.g., LayoutLM or simple PyMuPDF)
* Tabular: Pandas → insert into PostgreSQL or Redis

### 3. **Validation Agent**

* Compare extracted vs. declared fields (e.g., income in bank vs. form)
* Highlight inconsistencies using LLM + rule-based checks

### 4. **Eligibility Classifier**

* Use Scikit-learn (e.g., RandomForestClassifier) for decision:
  + Inputs: family size, income, employment history, liabilities, etc.
  + Output: Approve / Soft Decline

### 5. **Economic Enablement Recommender**

* Recommend based on skills/resume: job, upskilling, career counseling
* Use embedding search via Qdrant (similar resumes → suggest action)

### 6. **Agentic Orchestration**

* Use LangGraph or Crew.AI:
  + Chain: Extraction → Validation → Eligibility → Recommendation

### 7. **LLM Interface (Chat)**

* Powered by Ollama (Mistral, Phi) + OpenWebUI
* Interact via Streamlit frontend

### 8. **Observability**

* Use Langfuse to track agent outputs, errors, decision trees

## PHASE **3**: **TECH STACK SUMMARY**

| Component | Tool |
| --- | --- |
| LLM | Ollama (Mistral / Phi-2) |
| Frontend | Streamlit |
| ML Classification | Scikit-learn |
| OCR / Document parsing | Tesseract / PyMuPDF / pdfplumber |
| Chat Interface | OpenWebUI / FastAPI wrapper |
| Orchestration | LangGraph / Crew.AI |
| Reasoning Framework | ReAct / Reflexion |
| Vector Store | Qdrant + Redis |
| Structured Data | PostgreSQL / MongoDB |
| Graph Data (Optional) | Neo4j (family relations) |
| Observability | Langfuse |
| Source Code | GitHub (private repo) |

## PHASE **4**: **DELIVERABLES**

### .

├── agents

│   ├── data\_extraction.py

│   ├── data\_generation.py

│   ├── eligibility.py

│   ├── recommendation.py

│   └── validation.py

├── all-files.txt

├── App

│   ├── app.py

│   └── new\_env

│   ├── bin

│   ├── etc

│   ├── lib

│   ├── lib64 -> lib

│   ├── pyvenv.cfg

│   └── share

├── Arabic\_injesting.py

├── arabic\_ocr\_output.txt

├── data\_injesting.py

├── diagrams

│   └── architecture.png

├── docs

│   └── solution\_summary.pdf

├── main.py

├── model\_download.py

├── models

│   └── random\_forest\_model.pkl

├── project\_structure.txt

├── rag\_app.py

├── rag\_pipeline.py

├── README.md

├── requirements.txt

├── scraping.py

├── Screenshot from 2025-05-29 21-37-37.png

├── Semantic\_search.py

├── streamlit\_app.py

├── test\_data

│   ├── Bank-Statement.pdf

│   ├── confusion\_matrix\_rf.png

│   ├── emirates\_id.jpg

│   ├── mock\_test\_data.csv

│   ├── mock\_test\_data\_validated.csv

│   └── Scanned\_Resume.png

├── test\_ocr\_utils.py

└── utils

├── data\_cleaning.py

└── ocr\_utils.py

13 directories, 34 files

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

**Mock-up data generation**

Generating rich mock applicant data for simulating real-world use cases.

* Determine if a person qualifies for economic social support (approval / soft decline)
* Recommend economic enablement options (e.g., job matching, training)

### **1: Define the Eligibility Rules**

Based on your dataset structure, we can define simple rules (you can adjust these later):

| Field | Meaning | Eligibility Threshold |
| --- | --- | --- |
| employment\_status | 0 = Unemployed | Unemployed eligible |
| family\_size | 0 = ≤3, 1 = >3 | Family > 3 has higher priority |
| assets | 0 = <20,000 | Low assets preferred |
| liabilities | 1 = High liabilities | Considered vulnerable |
| credit\_score | 0 = <650 | Low score increases priority |
| name mismatch | Flag mismatched names | Risky / fraud filter |
| Nationality | 1 = UAE national  0 = Non-UAE national | Only UAE are accepted |

**Age : based on UAE laws**

| Law / Decree | Description |
| --- | --- |
| Federal Decree-Law No. 33 of 2021 | Labour Law – regulates employment, minimum age |
| Federal Law No. 7 of 1999 | Pensions and Social Security Law for Emiratis |
| Federal Decree-Law No. 57 of 2023 | Updated pension regulations |
| Federal Decree-Law No. 23 of 2022 | Social support for low-income Emirati citizens |

| Age Range | Meaning in UAE Law |
| --- | --- |

|  |  |
| --- | --- |
| 15–17 | Minor – limited work rights |

|  |  |
| --- | --- |
| 18–59 | Working age – eligible for jobs/support |

|  |  |
| --- | --- |
| 60+ | Retirement age – may qualify for social support |

**Eligibility Rules Based on Age**

**This is considered the working-age population. Eligibility for financial support is based on:**

| Factor | Why it Matters | Points (Score) |
| --- | --- | --- |
| employment\_status == 0 | Unemployed people need support | +1 |
| family\_size == 1 | Larger families (size > 3) need more | +1 |
| assets == 0 | Low assets = financially vulnerable | +1 |
| liabilities == 1 | High debts = more at risk | +1 |
| credit\_score == 0 | Poor credit = possible financial strain | +1 |

### **Age Group: 60 years and above**

This group is considered retirement age. UAE laws support seniors especially if they are low-income.

* The person is a UAE national
* AND has low assets OR poor credit

Then they automatically qualify for strong support.

This reflects the intent of Federal Decree-Law No. 23 of 2022, which offers elderly citizens monthly support even if they’re not working.

### **Eligibility Scoring Logic**

Use a simple scoring system (you can later train a model):

* +1 for unemployed
* +1 for family size > 3
* +1 for low assets
* +1 for high liabilities
* +1 for low credit score

Scoring :5 = Strong approve

* 3–4 = Approve
* 1–2 = Soft decline
* 0 = Reject

## **Enablement Recommendations**

These are non-cash support options to help people become financially independent or more secure. They’re based on age + condition:

| Condition | Recommendation |
| --- | --- |
| age 18–59 and employment\_status == 0 | Job matching or training |
| credit\_score == 0 | Financial counseling |
| full\_name != name\_in\_bank | KYC verification support |
| age ≥ 60 and nationality == 1 | Retirement financial assistance |

## Step 2 :Validation

Clean the mock dataset (mock\_test\_data.csv) by applying these validation rules:

1. Remove any non-UAE nationals (nationality != 1)
2. Remove rows where full\_name ≠ name\_in\_bank
3. Remove rows where full\_name ≠ name\_in\_resume
4. Remove rows where age < 18

**Step 2: Recommendation System**

* Using mock\_test\_data\_validated.csv
* Shuffle and split into 80% train, 20% test
* Train a classifier (I am using RandomForestClassifier)
* Predict enablement\_recommendation
* Evaluate with a confusion matrix

## Assumptions

Simplify enablement\_recommendation into categories:

| Recommendation Label | Class |
| --- | --- |
| Job matching / training | 0 |
| Financial counseling | 1 |
| Retirement financial assistance | 2 |
| No recommendation | 3 |

## "No recommendation" means:

* Applicant is employed
* Credit score is good (credit\_score == 1)
* Age is under 60 (so not a retiree)
* Name matches across all documents (no KYC flag)
* Everything looks fine — no vulnerability or support need detected

**Results**

After trained a on half of the data and testing on the other half, Random Forest model to predict enablement\_recommendation, and the results are excellent — at least according to the current test data.

[[117 0 0 0]

[ 0 152 0 0]

[ 0 0 76 0 ]

[ 0 0 0 36 ]]

📋 Classification Report:

precision recall f1-score support

Financial counseling 1.00 1.00 1.00 117

Job matching / training 1.00 1.00 1.00 152

No recommendation 1.00 1.00 1.00 76

Retirement financial assistance 1.00 1.00 1.00 36

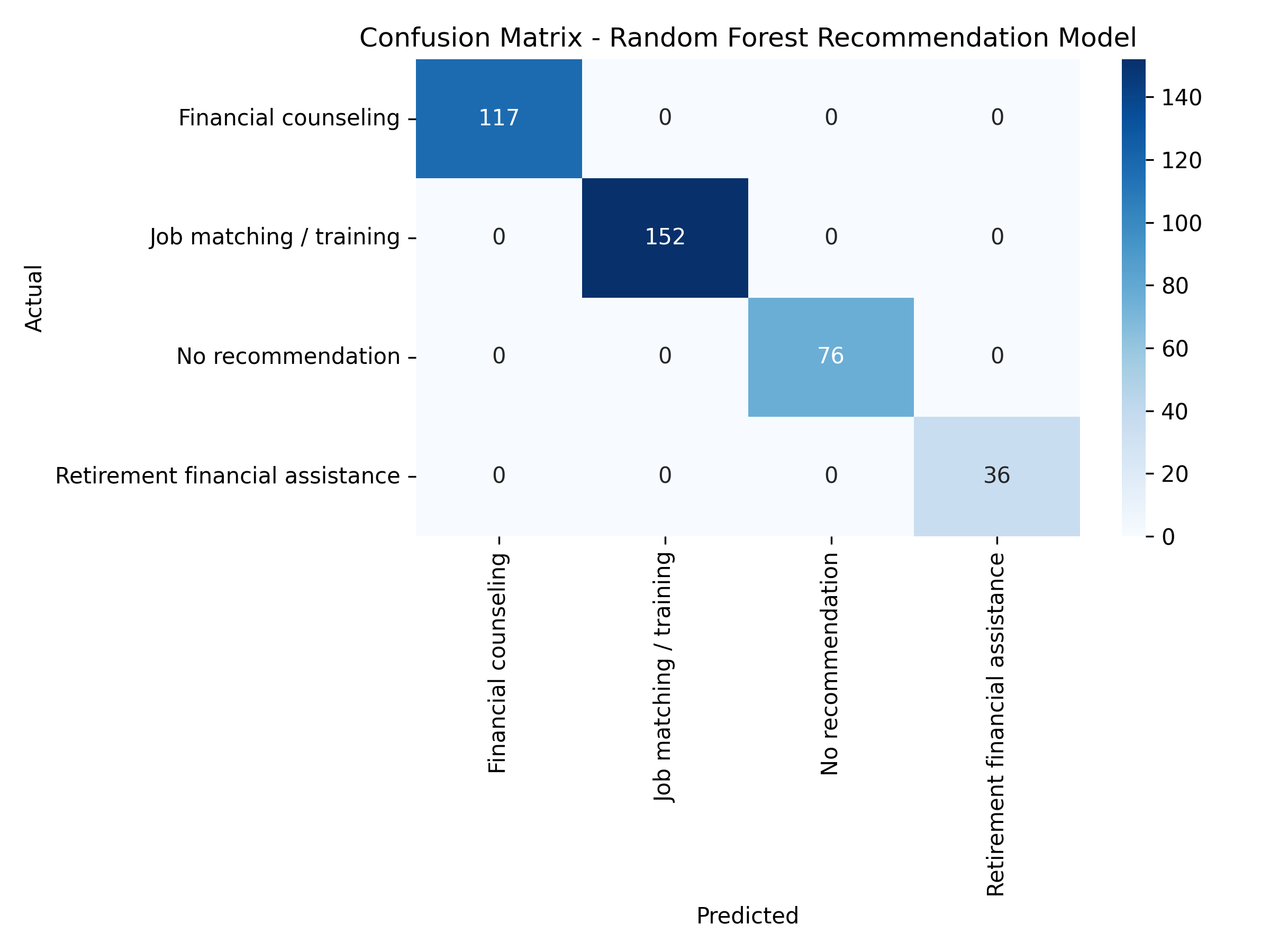
accuracy 1.00 381

macro avg 1.00 1.00 1.00 381

weighted avg 1.00 1.00 1.00 381

| Metric | Meaning |
| --- | --- |
| Precision | Of all the times a class was predicted, how often was it correct? |
| Recall | Of all the true instances of a class, how many were correctly found? |
| F1-score | Harmonic mean of precision and recall |
| Support | Number of actual instances of the class |

| Actual Class | Predicted As | Count |
| --- | --- | --- |
| Financial counseling | Financial counseling | 117 |
| Job matching / training | Job matching / training | 152 |
| No recommendation | No recommendation | 76 |
| Retirement financial assistance | Retirement financial assistance | 36 |



## 

## 

## Confusion Matrix All Metrics = 1.00 (100%)

This means:

* Every prediction made by the model was correct.
* There were no false positives or false negatives.
* This is perfect classification on this dataset.

## This Too Good to Be True?

Yes... and no.

### Possible Reasons for Perfect Accuracy:

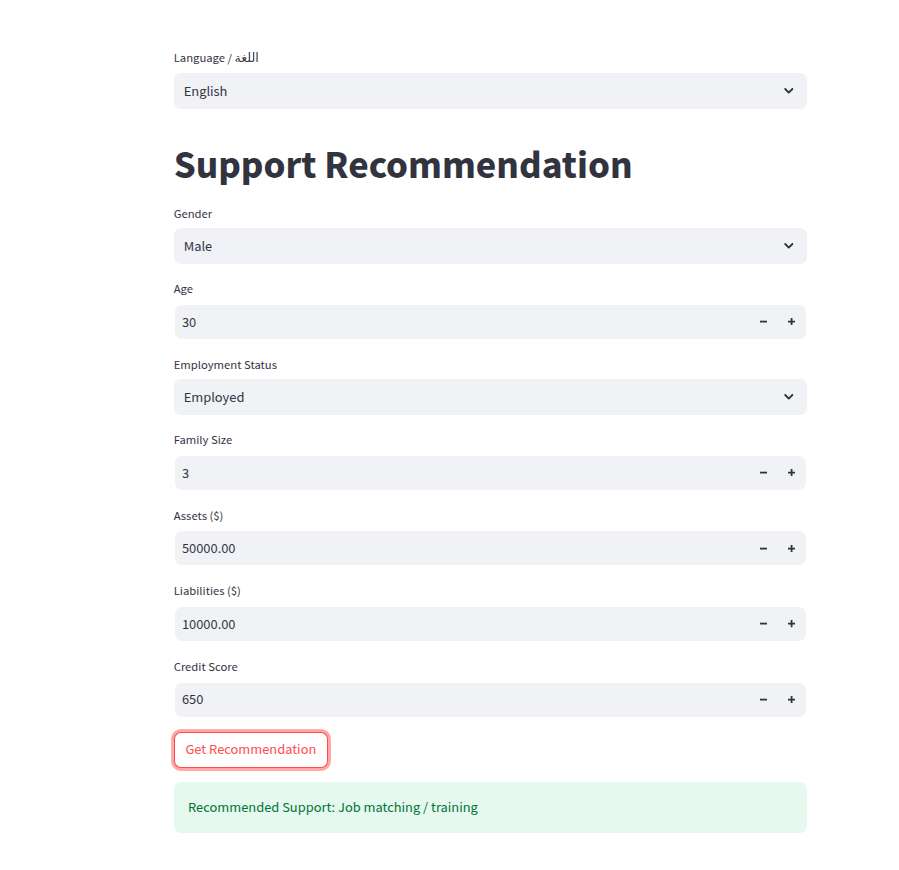
1. The data is clean and synthetic, so patterns are easy to learn.
2. The feature space is highly predictive — e.g., employment\_status, age, and credit\_score might directly map to recommendations.
3. No noise or ambiguity — real-world data is messier.

### In a real production setting:

* We expect less than perfect accuracy.
* We should validate on truly unseen or real-world data.

The trained model be used to predict new data as in streamlit run App/app.py





### Recommendation in Arabic and English

## ================================================

**USING LLM**

## 1- Creating the Vector Store (Knowledge Base)

The vector store is like a searchable memory of all your legal documents. Here's how it's built:

### 1. **Load the English PDFs**

You start by reading all English law documents (PDFs) from your data/laws\_pdf/English folder.

### 2. **Split the Text into Chunks**

To make search efficient, the text is broken into overlapping chunks (e.g., 512 tokens with 125 token overlap). Each chunk becomes an independent searchable unit.

### 3. **Convert Text into Vectors (Embeddings)**

Each chunk is passed through a pre-trained embedding model (like all-MiniLM-L6-v2). This model converts the text into a numerical vector — a list of numbers that captures the meaning of the text.

### 4. **Store the Vectors in FAISS**

These vectors are stored using FAISS, a fast similarity search engine. It lets you quickly retrieve the most similar chunks later, based on a user’s question.

The final result is a saved vector index on your disk (data/vectorstores/english\_laws).

## 2: RAG Pipeline (Retrieval-Augmented Generation)

RAG stands for Retrieval-Augmented Generation, and it helps your system generate accurate answers based only on trusted legal content.

### 1. **User Asks a Question**

For example: “What is the retirement age for UAE nationals?”

### 2. **Retrieve Relevant Chunks**

The system searches the FAISS vector store using the embedding of the question. It finds the most relevant chunks from your law documents.

### 3. **Combine the Chunks into a Prompt**

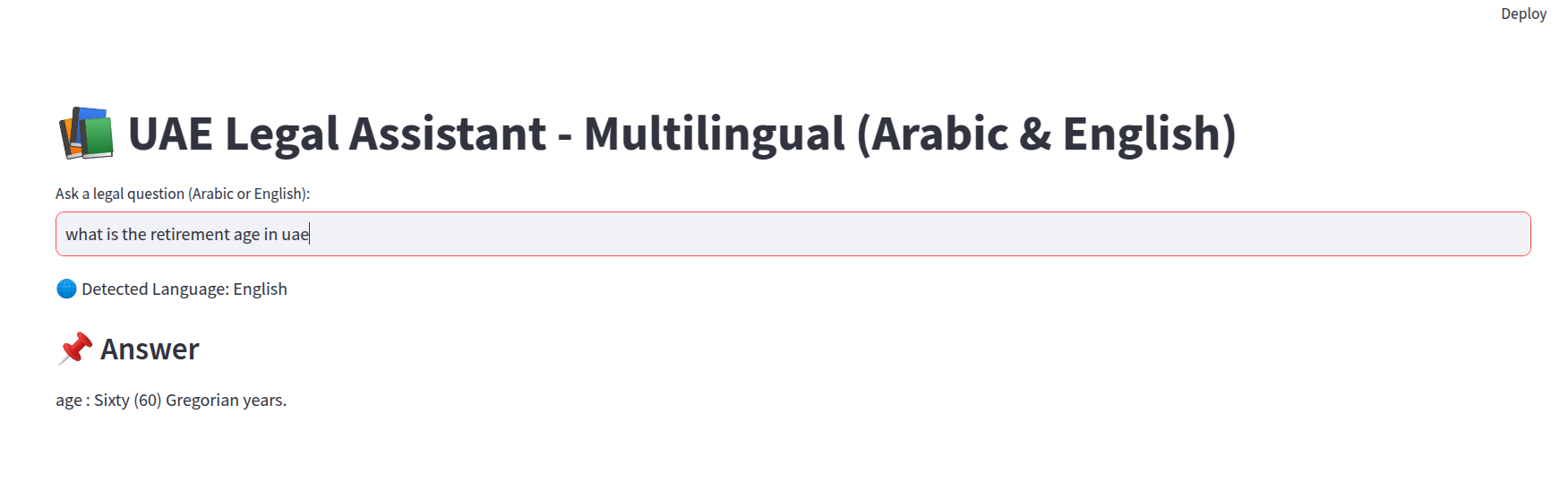
These chunks are inserted into a carefully designed prompt, which is fed to a local language model (e.g., Falcon, Mistral, or Flan-T5). The prompt includes the question and retrieved law excerpts.

### 4. **LLM Generates the Answer**

The language model reads the law excerpts and generates a natural-language answer — based only on what’s in the legal documents.

The laws, decrees, and regulations related to social security and benefits were downloaded into the laws\_pdf folder in both Arabic and English.  
There were no issues with ingesting the English data; however, the Arabic files required OCR processing to convert them into text files, which were then saved in the path:  
social-support-ai/data/laws\_pdf/Arabic/text.

Using FAISS, we were able to create separate vector stores for both English and Arabic documents. When a user submits a question, relevant document chunks are retrieved from the vector store to form a knowledge base. This context is then passed to the language model (LLM) to generate a final answer — as illustrated in the following figure from the rag\_app.

streamlit run rag\_app.pyFor the Arabic I am using small model which can not handle Arabic with limited resources.

It could perform much better when larger model and other embedding technology such as OpenAI is used