**AI-Powered Social Support Eligibility Screening System**

**1. Our Approach**

The goal of this solution is to build a **semi-automated eligibility screening system** for social support programs. The system simulates how government or NGO caseworkers can streamline applicant intake by combining **document processing, data validation, machine learning eligibility models, and explainable AI**.

**Workflow**

1. **Document Upload**  
   Applicants upload required documents such as Bank Statements, Emirates ID, Resumes, Credit Reports, and Assets/Liabilities sheets.
2. **Automated Data Extraction**  
   The backend leverages PDF parsers, OCR, and Excel readers to automatically extract key information such as applicant name, date of birth, family size, income, assets, liabilities, and credit score.
3. **Semi-Automated Validation**  
   Extracted fields are pre-populated into a form on the Streamlit frontend. Applicants review and confirm the information or make corrections before submission.
4. **Eligibility Assessment**  
   The cleaned and validated data is passed into a **scikit-learn pipeline**. The model predicts eligibility outcomes (Approve, Soft Decline, Reject) based on features such as income, family size, employment status, and credit score.
5. **Decision Explanation**  
   The outcome is enhanced with natural language explanations from **Gemini 2.0 Flash**. Explanations help applicants understand why their application was accepted or rejected.
6. **Persistent Storage**  
   Each decision is saved as a JSON file in data/processed/ so applicants can later query their results using their application ID.
7. **Chatbot Interface**  
   Applicants can ask follow-up questions via the chatbot. The chatbot retrieves context from saved decisions and generates answers using Gemini 2.0 Flash.

**2. Tool Choices**

**Streamlit (Frontend)**

* **Why**: Rapid prototyping of interactive web apps with minimal code.
* **Benefit**: Allows us to build applicant-facing quickly.
* **Alternatives considered**: React.js or Django templates. Chosen Streamlit because it requires no frontend expertise and integrates seamlessly with Python ML stack.

**FastAPI (Backend)**

* **Why**: Modern, fast, asynchronous Python web framework.
* **Benefit**: Provides robust REST endpoints (/extract, /predict, /explain) with auto-generated docs.
* **Alternatives considered**: Flask and Django REST Framework. FastAPI chosen for performance, async support, and built-in Pydantic validation.

**Scikit-Learn (Eligibility Model)**

* **Why**: Reliable ML library for structured data tasks like classification.
* **Benefit**: Easy to train, export, and serve models (joblib).
* **Alternatives considered**: XGBoost or TensorFlow. Scikit-learn chosen due to interpretability and simplicity.

**Gemini 2.0 Flash (Explanations & Chatbot)**

* **Why**: Provides state-of-the-art natural language generation.
* **Benefit**: Human-friendly explanations, conversational Q&A, and multi-turn context.
* **Alternatives considered**: GPT-4 or Llama 3. Chosen Gemini for its integration with Google ecosystem and optimized inference.

**Data Storage (JSON & CSV)**

* **Why**: Lightweight, portable, and human-readable.
* **Benefit**: For a case study prototype, avoids the overhead of a full database.
* **Alternatives considered**: PostgreSQL or MongoDB. Could be used in production, but JSON/CSV chosen for simplicity in demos.

**3. AI Solution**

Applicants need to trust that eligibility decisions are made fairly and transparently.

1. **Structured Feature Contributions**
   * Features such as income, credit score, assets, liabilities, and family size are extracted and passed to the ML model.
   * Business rule cutoffs (e.g., age < 18 → reject) can be explicitly added.
2. **Model Transparency**
   * The scikit-learn pipeline is interpretable, enabling easy inspection of feature importance.
   * Validation reports highlight inconsistencies (e.g., mismatch between declared income and bank statement).
3. **Natural Language Explanations**
   * Gemini 2.0 Flash transforms structured output into plain English explanations.
   * Example:  
     *“Your application was rejected because your reported income exceeds the eligibility threshold. You may reapply if your circumstances change.”*
4. **Chatbot Q&A**
   * Applicants can ask *“Why was I rejected?”* or *“What can I do to improve my chances?”*.
   * Chatbot retrieves stored decision context and provides personalized, transparent answers.

This combination of **rule-based validation + ML interpretability + LLM explanations** ensures decisions are not black boxes.

**4. Suggestions for Future Improvements**

While the current prototype demonstrates feasibility, several enhancements can make the system more production-ready and scalable:

**Data Handling**

* Replace JSON/CSV persistence with a **relational database** (PostgreSQL) for large-scale deployments.
* Add **audit logging** for all predictions and chatbot responses for compliance.

**Model Enhancements**

* Use **XGBoost or CatBoost** for improved predictive performance.
* Train with **real-world anonymized datasets** to capture more nuanced eligibility patterns.
* Incorporate **feature attribution methods** (e.g., SHAP) for richer interpretability.

**Document Processing**

* Replace simple OCR with **advanced document AI** (Google Document AI, Azure Form Recognizer).
* Add support for **multilingual OCR** and structured field extraction.

**Embeddings & Semantic Search**

* Store application decisions in **ChromaDB or FAISS** with embeddings.
* Enable semantic search across all applications, so caseworkers can analyze trends (e.g., “find all applicants rejected for high liabilities”).

**Frontend Enhancements**

* Expand Admin Dashboard with full list of applications and filter/search functionality.
* Add **visual analytics** (charts on income distribution, approval rates).
* Multi-user support with authentication for applicants vs. admins.

**Deployment**

* Containerize with **Docker** and deploy to **Google Cloud Run** or **AWS ECS**.
* Enable **CI/CD pipelines** for continuous delivery.
* Add monitoring with **Prometheus + Grafana** for system health.

**Conclusion**

This AI-powered case study successfully demonstrates how **document understanding, machine learning, and large language models** can transform the eligibility screening process.

By combining **semi-automated validation, ML-driven decisioning, and LLM-powered explainability**, the system reduces manual workload, increases fairness, and improves applicant trust.

With future improvements in **data pipelines, embeddings, and deployment**, this prototype can evolve into a robust real-world solution for government or NGO social support programs.