Loan Navigator Agent Suite

Abstract

In India's fast-paced fintech space, **BlueLoans4all** is empowering micro-entrepreneurs and small businesses by offering accessible, small-ticket loans. However, their support centers face a deluge of repetitive queries: "How many EMIs left?", "If I pre-pay ₹10,000, how will my EMI change?", "Am I eligible for a top-up?". These questions, while simple, are vital to financial planning for borrowers, and the answers must comply with strict regulatory guidelines and disclosure norms.

This Project introduces a **multi-agent AI system** built using **LangGraph or CrewAI**, that simulates a smart "Loan Navigator Agent" for BlueLoans4all. The solution combines **NLP-to-SQL capabilities**, **RAG-based policy lookup**, and a **What-if Simulation Engine** to accurately, securely, and promptly resolve customer queries. By automating repetitive service desk queries, the system not only enhances the borrower experience but also reduces the workload of call center staff and ensures regulatory compliance.

Background

- BlueLoans4all serves **thousands of citizens across India** via digital microlending.
- Customers commonly contact the support desk for simple queries related to loans—information which is readily available but often not easily interpreted.
- Support staff must give responses that are **compliant with RBI norms**, involve **loan database queries**, and require **policy document interpretation**.
- The current system has:
    - **Increased call volume**
    - **Delayed response times**
    - **High cost due to manual query resolution**
    - **Compliance risks from inconsistent answers**

Objectives & Success Metrics

Objectives

- Build a **Loan Navigator Agent Suite** that offers:
    - Accurate EMI tracking and top-up eligibility checks
    - What-if scenarios on prepayment impacts
    - Policy-compliant and audit-ready responses
- Automate support through **chat or agent interface**
- Minimize manual overhead and ensure 24×7 availability

Success Metrics

- Call center load reduction by **≥ 35%**
- Average resolution time reduced by **≥ 70%**
- Prepayment uptake improved by **≥ 20%**
- Regulatory compliance with **≥ 98% accuracy**
- User satisfaction (CSAT) above **90%**

Data Assets

Asset Type	Description
SQLite Loan Database	Contains structured loan records for ~1000 loans. Includes fields like loan_amount, tenure, interest_rate, amount_paid, status, and topup_eligible. Used by the SQL Analyst Agent for direct data queries.
Policy Documents (PDF)	A detailed corpus of internal operational policies, risk guidelines, regulatory compliance mandates, top-up eligibility, infrastructure governance, etc.
Chroma Vector DB (Pre-Fed)	Pre-populated vector database built using Chroma, storing dense vector representations of all policy document chunks. Enables instant semantic search and accurate citations in real time. No ingestion pipeline is required during runtime. Optionally similar structure can be replicated to any other vector DB storage.
Amortization Simulation Schedule	Amortization tables for different loan scenarios. Can be used for testing and validating the What-If Calculator Agent.
Amortization Explanation Document	Technical document that outlines amortization principles, EMI formulas, prepayment logic, and microservice design principles. Used by developers and testers.

Solution Design and Phases

The Loan Navigator solution is built using a modular, enterprise-grade, multi-agent architecture powered by Azure services. It enables automation of customer loan queries, simulation of financial scenarios, and regulation-compliant advisory using structured and unstructured data sources.

Phase 1: Data Foundation and Infrastructure Provisioning

- Deploy SQLite database containing loan information using Azure Files or Azure Blob Storage.
- Convert policy PDFs into clean, chunked text segments using an Azure Function (OCR + text extraction, if needed).
- Embed policy chunks using **Azure OpenAI Embedding Models** and store the vectors in **Chroma DB**.(already implemented)
- Pre-load Chroma vector store and host it using Azure Container Instance with persistent volume attached.
- Store sensitive keys, tokens, and DB URIs securely in **Azure Key Vault**.

Phase 2: Agent Orchestration Design

- Architect the multi-agent system with four specialized agents (use supervisor pattern):
    1. **Supervisor Agent**: Identifies intent, routes query, merges and audits responses.
    2. **SQL Analyst Agent**: Executes NLP-to-SQL transformations and interacts with SQLite DB.
    3. **Policy Guru Agent**: Performs semantic retrieval from vector DB using RAG pattern.
    4. **What-If Calculator Agent**: Simulates amortization and prepayment impact via Python logic.
- Define orchestration logic using **LangGraph or CrewAI**, running as a stateless application within **Azure Container Instance**.

Phase 3: Agent Implementation

The four-agent architecture is implemented with isolated responsibilities and orchestrated via LangGraph or CrewAI. Each agent operates independently but can interact via well-defined interfaces coordinated by the Supervisor Agent.

SQL Analyst Agent

- **Function**: Handles structured queries such as EMI tracking, tenure remaining, and top-up eligibility.
- **Tech Stack**:
    - LLM-powered natural language to SQL transformation using **Langchain/Langgraph** and **Azure OpenAI Codex**.
    - Connects to SQLite DB mounted in the runtime environment.
    - **Security**: Whitelisted SQL access only; Parameterized queries with schema enforcement.
- **Fallback Situation**:
    - **Trigger**: If the generated SQL is syntactically correct but returns zero results, or if the LLM-generated SQL fails validation.
    - **Resolution**: The agent falls back to the Supervisor Agent with a SQL_FALLBACK_FLAG. Supervisor then triggers a guided clarification prompt asking the user to rephrase or confirm parameters (e.g., "Could you confirm which loan ID or date range you're referring to?").

Policy Guru Agent

- **Function**: Answers policy, compliance, and eligibility queries using semantic search.
- **Tech Stack**:
    - Agentic Implementation using LangGraph
    - Uses **Chroma Vector DB** with pre-embedded policy document chunks.

    - Retrieves top-k results and uses **Azure OpenAI GPT-4o Mini** for summarization.
- **Capabilities**:
    - Citation tagging for retrieved content.
    - Filters to ensure only relevant policy documents are referenced.
- **Fallback Situation**:
    - **Trigger**: If the confidence score from the vector similarity retrieval is below a threshold (e.g., cosine similarity < 0.75).
    - **Resolution**: The Supervisor Agent automatically appends additional query context (loan type, customer profile) and retries with a reformulated query. If retrieval still fails, a generic fallback message is provided: "We couldn't find an exact policy for this scenario, but here's what typically applies..."

What-If Calculator Agent

- **Function**: Performs amortization simulations for prepayments, tenure adjustments, and interest savings.
- **Tech Stack**:
    - Agentic implementation using Langgraph
    - Lightweight Python microservice with REST interface.
    - Hosted on **Azure Container Instance** with stateless design.
- **Inputs**: Loan amount, interest rate, tenure, optional prepayment.
- **Outputs**: Revised EMI or tenure, interest saved, schedule comparison.
- **Fallback Situation**:
    - **Trigger**: Input validation fails (e.g., negative tenure, prepayment more than outstanding balance).
    - **Resolution**: Returns error JSON with explanatory message. Supervisor Agent reformulates input or prompts user with clarification ("Your prepayment exceeds

the current balance. Would you like to simulate a complete foreclosure instead?").

Supervisor Agent

- **Function**: Central orchestration unit that receives user queries, detects intent, and routes them to the appropriate agent(s).
- **Tech Stack**:
    - Built using LangChain or CrewAI orchestration models.
    - Stateless FastAPI wrapper exposed via REST.
- **Responsibilities**:
    - Intent classification using prompt templates and keyword scoring.
    - Delegation of tasks to sub-agents.
    - Merge and tone-standardize responses.
    - Log interactions and capture feedback.
- **Fallback Situation**:
    - **Trigger**: If the query is ambiguous or involves overlapping agent domains (e.g., policy vs. numeric), or multiple agents fail.
    - **Resolution**: Triggers a clarification chain: "Your query may involve both policy terms and EMI computation. Would you like me to calculate your new EMI after a prepayment, or show you the policy on prepayment eligibility?"

Phase 4: API Interface and Deployment

- Wrap Supervisor Agent in a **FastAPI application**.
- Containerize using Docker and push to **Azure Container Registry**.
- Deploy via **Azure Container Instance** with autoscaling enabled.
- Authenticate requests using OAuth 2.0 flows via **Azure Active Directory**.

Phase 5: Monitoring, Logging, and Observability

- Integrate **Azure Monitor** to track:
    - Agent-level latency
    - Query volume and fallbacks
    - API response codes
- Use **Azure Application Insights** to:
    - Trace prompt-level performance
    - Log model invocations and token usage
    - Visualize trace flow across agents
- Enable **Langfuse** or **mlflow** for LLM-specific telemetry (fallback ratio, token usage, citation quality).

Phase 6: Security, Access Control, and Secret Management

- Store all credentials in **Azure Key Vault**.
- Secure all API endpoints via bearer token, JWT inspection, and IP whitelisting.
- Audit user access via **Azure AD logs**.
- Enable daily rotation of API keys for internal agents.
- Logs are archived to **Azure Storage** with retention policies.

Phase 7: Testing, UAT, and Release Governance

- Unit testing:
    - Test query types for each agent (e.g., numeric, policy, simulation).
- Integration testing:
    - Simulate end-to-end flows via Postman, CLI scripts, and frontend app (if available).
- User Acceptance Testing (UAT):
    - Business users and PFAs validate output tone, accuracy, and citations.
- Release Pipeline:
    - CI/CD via GitHub Actions or Azure DevOps
    - Container build → Push to ACR → Deploy to ACI

Phase 8: Feedback Loop and Continuous Learning

- Agent confidence scores, fallback frequency, and error categories fed into the analytics layer.
- Monthly retraining/optimization of SQL and semantic search prompts using user feedback.
- Internal feedback dashboard for PFAs and Supervisors to rate responses and annotate issues.
- Policy documents re-embedded every quarter or when updates occur, triggered via Azure Logic Apps.

Expected Deliverables

- **Deployment Artifacts**:
  Containerized services for all four agents (Supervisor, SQL Analyst, Policy Guru, What-If Calculator), deployed via Azure Container Registry and Azure Container Instances. FastAPI endpoints are secured and configured for production via Azure App Service or AKS.

- **Documentation**:
  Includes OpenAPI specification for all exposed APIs, detailed agent interaction flows, prompt templates, setup runbook, and an amortization explanation PDF for simulation logic clarity.

- **Monitoring and Observability**:
  Integrated with Azure Monitor, Application Insights, and Langfuse for full traceability. Tracks latency, fallback events, API errors, and token usage across all agents.