Smart Banking Conversational AI
Proof of Concept - Design & Implementation

Akash Subramanian

July 27, 2025

1. System Architecture & Component Breakdown

Overview

The system is a modular Smart Banking Conversational AI consisting of:

- **User Interface (UI):** Streamlit web app with a chat-style interface.
- **Context Manager:** Uses Streamlit session state to maintain conversation context including authentication, intent, slots, and chat history.
- **Dialog Manager:** Orchestrates dialog flow—authentication, intent classification, slot filling, API handling, and session resets.
- **NLU Module:** Rule-based intent classification like apply loan, block card, get statement, knowledge queries.
- **API Handlers:** Mock APIs simulating backend banking services.
- **Retrieval-Augmented Generation (RAG) Engine:** Embeds knowledge base documents and retrieves relevant information to answer knowledge queries.

2. Conversational Flow Design

2.1 **Authentication:**

- User provides First Name or Customer ID.
- System verifies user against dataset and on success welcomes user.

2.2 **Intent Recognition:**

- NLU module classifies user intent such as applying for a loan, blocking card, retrieving account statement, or asking knowledge questions.

2.3 **Slot Filling & Multi-turn Dialogue:**

- For intents needing details, system asks for parameters (e.g., loan type, amount).
- User answers populate slots.
- After all required slots are filled, system calls appropriate API handler and returns result.

2.4 **Additional Features:**

- Account information queries.
- Logout command clears session.
- Fallback and clarification messages guide user when input is unclear or missing information.

3. Context / State Management Strategy

- Uses `Streamlit`'s `session_state` dict.
- Tracks keys:
  - `authenticated` (bool): user login status
  - `user_info` (dict): user details from dataset
  - `intent` (str): current active user intent
  - `slots` (dict): collected parameters for the intent
  - `chat` (list): chat history between user and bot
- Maintains multi-turn conversation context until logout or reset.

4. Decision-making & Routing Logic for Banking Tasks

At each user input:

- If not authenticated, process authentication input.
- Else if no active intent, run NLU to classify intent.
- If intent requires parameters, prompt for missing slots.
- Once all slots are ready, invoke mock API and return response.
- Reset slots and intent after task completion.
- Recognize logout commands and reset session accordingly.
- Provide fallback message for unknown inputs.

5. Working Proof of Concept (PoC)

5.1 Conversational Interface

- Streamlit web application.
- Chat UI for multi-turn conversations.
- Welcome prompt to enter Customer ID or First Name.

5.2 End-to-End Use Cases

- **Loan Application:** User requests a loan; system collects loan type and amount; confirms application with reference ID.
- **Block Card:** User commands to block a card; system asks last 4 digits; confirms card blocking.
- **Mini Statement:** User requests statement; provides account number; system returns last three transactions.

5.3 API Integration

- Mock APIs simulate backend banking functionality.
- Dialogue manager calls these APIs after slot fulfillment.
- Dataset is used for authentication and referencing user info.

6. Explanation of Key Functionalities

6.1 Task Delegation / External Tool Access

- API handler functions perform task delegation.
- Knowledge queries handled by RAG engine using semantic search over banking documents.
- External LLM service (e.g., Ollama) accessed via HTTP for enriched answers.

6.2 Tracking User Progress Across Multiple Steps

- Session state keeps track of current intent and collected slot values.
- System prompts for missing slots one at a time.
- On complete slot capture, performs API call and resets intent/slots.

6.3 Fallback and Clarification Logic

- Unrecognized inputs trigger fallback messages.
- Clarification questions are asked if slot inputs are invalid or missing.
- Logout command triggers session reset and user logout confirmation.

7. Limitations and Potential Enhancements

Current Limitations

- Rule-based NLU limits intent detection scope.
- Fixed slots and simple regex for parameter extraction.
- Mock APIs with static responses, no live backend.
- Dataset static, no concurrency handling or live updates.
- Minimal error handling, no sentiment analysis or anomaly detection.
- RAG knowledge base limited to local docs; external LLM dependency.

Potential Enhancements

- Upgrade to transformer-based NLU for better understanding.
- Integrate real banking APIs for dynamic data updates.
- Add natural language generation for personalized responses.
- Use database backed session for multi-user scalability.
- Improved fallback strategies and confirmation dialogs.
- Voice input, multi-lingual support, and OTP-based authentication.

8. README Summary

How to Setup and Run

```
git clone <repo_link>
cd SmartBankAI
pip install -r requirements.txt
streamlit run app.py
```

Supported Flows & Example Prompts

- Authenticate with First Name or Customer ID.
- Loan Application: ”I want a car loan of 250000” or respond to prompts.
- Block Card: ”Block my card” then provide card’s last 4 digits.
- Mini Statement: ”Show me last 3 transactions” and provide account number.
- Knowledge Queries: ”What is KYC?” or ”Explain EMI calculation”.
- Logout by typing ”logout”.

9. Suggested GitHub Repo Structure

```
SmartBankAI/
 app.py              # Streamlit UI entry point
 dialog_manager.py   # Dialog flow and API integration
 nlu.py              # Intent classification
 context_manager.py  # Session state management
 api_handler.py      # Mock banking APIs
 rag_engine.py       # Knowledge retrieval & LLM API integration
 data/
    kb/             # Knowledge base text documents
    Comprehensive_Banking_Database.csv
 requirements.txt
 README.md
 docs/
     Architecture_Design.md
```