Vaani: The Voice-First Bank for Bharat

Round 2 Technical Submission

Team Name: Code Crusaders
Date: 23 November 2025

Executive Summary

India stands at a unique digital crossroads. While we boast over 900 million internet subscribers, a significant "usability divide" persists. Nearly half of our rural households struggle with complex, menu-driven banking interfaces designed for the digitally savvy urban population. **Vaani** is our answer to this challenge—a voice-first banking assistant designed specifically for the next 500 million users in Bharat.

This document details the technical architecture of Vaani. We have moved beyond simple chatbots to create a robust, agentic AI system capable of executing secure financial transactions, understanding "Hinglish" (Hindi-English code-switching), and providing hyper-personalized financial advice using Retrieval-Augmented Generation (RAG). Our solution is not just a wrapper around an LLM; it is a fully integrated banking ecosystem built on a microservices architecture, adhering strictly to RBI's security guidelines and the DPDP Act.

In this submission, we walk through our technology stack, the "Hybrid Supervisor" AI architecture, our secure data models, and the compliance-first security measures that make Vaani ready for real-world deployment.

1. Technology Stack

To build a solution that is both cutting-edge and accessible on low-end devices common in Tier-2 and Tier-3 cities, we selected a technology stack that balances performance, scalability, and developer velocity.

1.1 Frontend: Accessibility First

The user interface is built with **React 19** and **Vite**. We chose React for its component-based architecture, allowing us to build a responsive UI that works seamlessly across mobile browsers and desktops. Crucially, we leverage the **Web Speech API** for native Speech-to-Text (STT) and Text-to-Speech (TTS). This decision eliminates the need for heavy audio file uploads to the server for every utterance, significantly reducing latency and data usage—a critical factor for users with spotty internet connections in rural India. The frontend maintains a persistent WebSocket-like connection experience using efficient polling and state management via React Context, ensuring the conversation feels natural and real-time.

1.2 Backend API: High Performance & Async

Our core banking logic is powered by **FastAPI**. We chose FastAPI over Django or Flask because of its native support for asynchronous programming (async/await). Banking operations often involve waiting for database queries or external API calls (like UPI gateways). FastAPI handles these concurrent requests efficiently, ensuring that our server can handle thousands of simultaneous users without blocking. We use **SQLAlchemy 2.0** as our ORM, providing a robust abstraction layer over our database, allowing us to switch between SQLite (for development) and PostgreSQL (for production) with zero code changes. Pydantic v2 is used for rigorous data validation, ensuring that no malformed data ever reaches our core banking logic.

1.3 AI Engine: The Brain of Vaani

The intelligence layer is a separate microservice built with **LangGraph** and **FastAPI**. We moved away from simple linear chains to a graph-based agentic workflow. This allows for cyclic logic—essential for "conversational repair" (e.g., asking for a missing UPI PIN and then retrying the transaction).

For the LLM, we utilize **Ollama** to run models locally. We employ a dual-model strategy: 1. **Llama 3.2 3B**: A lightweight, ultra-fast model used for intent classification and simple routing. Its low latency ensures the user feels heard immediately. 2. **Qwen 2.5 7B**: A more powerful model used for complex reasoning, RAG synthesis, and handling multilingual "Hinglish" queries. Qwen demonstrates superior performance in understanding Indian contexts compared to standard Llama models.

ChromaDB serves as our vector store, enabling our RAG (Retrieval-Augmented Generation) pipeline to fetch accurate, up-to-date information about loan products and investment schemes without hallucinating.

2. System Architecture

Vaani follows a microservices architecture to ensure separation of concerns and independent scalability. The system is composed of three distinct layers: The Presentation Layer (Frontend), the Application Layer (Backend API), and the Intelligence Layer (AI Backend).

2.1 The Hybrid Supervisor Pattern

At the heart of our AI architecture lies the **Hybrid Supervisor Pattern**. Unlike a standard chatbot that tries to do everything with one prompt, our system uses a specialized orchestrator.

When a user speaks, the **Intent Router** first analyzes the utterance. Based on the intent, the request is routed to a specialized "Agent": **Banking Agent:** Has access to secure tools for balance checks, transaction history, and fund transfers. It cannot answer general questions, ensuring security. **UPI Agent:** A stateful agent designed specifically for the multi-step "Hello! UPI" flow (Verify Payee -> Enter PIN -> Confirm). **RAG Supervisor:** Routes information queries to domain experts (Loan Agent, Investment Agent, Support Agent). This modular design means we can upgrade the "Loan Agent" without risking the stability of the "Banking Agent". It is a robust, enterprise-grade approach to AI.

2.2 Data Flow & Latency Optimization

To minimize latency—a key KPI for voice interfaces—we optimized the data flow. Voice processing happens on the edge (browser). The text is sent to the AI Backend. The AI Backend communicates with the Core Banking Backend via internal high-speed APIs to fetch real-time data (e.g., account balance). The LLM then synthesizes the natural language response, which is sent back to the frontend for TTS synthesis. This "Text-in, Text-out" architecture over the network is 10x faster than streaming raw audio.

3. Data Model and Storage

Data integrity is paramount in banking. We employ a polyglot persistence strategy, using Relational Databases for transactional data and Vector Databases for unstructured knowledge.

3.1 Relational Schema (SQL)

Our primary database (SQLite for dev, Postgres for prod) manages the core banking entities. Key tables include: **Users:** Stores KYC details, hashed passwords, and voice profile references. **Accounts:** Manages account numbers, types (Savings/Current), and real-time balances. **Transactions:** An immutable ledger of all credits and debits, linked to accounts. **DeviceBindings:** Implements our security layer, mapping specific device fingerprints to user accounts. **VoiceProfiles:** Stores the mathematical embeddings of user voice prints (not raw audio) for biometric verification. We use strict foreign key constraints and ACID transactions to ensure that money is never "lost" during a transfer.

3.2 Vector Storage (ChromaDB)

For our RAG system, we ingest PDF documents (Loan policies, Investment schemes) into **ChromaDB**. We maintain separate collections for English and Hindi documents: loan_products & loan_products_hindi investment_schemes & investment_schemes_hindi We use **HuggingFace embeddings** (all-MiniLM-L6-v2) to convert text into vectors. This separation ensures that when a user asks in Hindi, we search the Hindi vector space, providing culturally and linguistically accurate results rather than machine-translated approximations.

4. AI/ML & Automation Components

Vaani is not just a wrapper; it is a sophisticated composition of multiple AI disciplines.

4.1 Retrieval-Augmented Generation (RAG)

Hallucinations are unacceptable in banking. If a user asks about the interest rate for a Home Loan, the AI cannot guess. Our RAG pipeline intercepts these queries. It retrieves the exact paragraph from the bank's official policy PDF stored in ChromaDB and inserts it into the LLM's context window. The system prompt strictly instructs the LLM: "Answer ONLY based on the provided context." This ensures 100% factual accuracy for product queries.

4.2 Voice Biometrics (Resemblyzer)

Security is our differentiator. We integrated **Resemblyzer**, a deep learning model for voice verification. During onboarding, the user speaks a passphrase. We generate a d-vector (voice embedding) and store it. For sensitive transactions, we capture the user's voice again, generate a new embedding, and calculate the cosine similarity. If the score exceeds our threshold (0.75), the transaction is authorized. This provides a seamless, password-less authentication experience that is hard to spoof.

4.3 Multilingual 'Hinglish' Support

Our target demographic often speaks "Hinglish" (e.g., "Mera account balance kya hai?"). Standard English models fail here. We utilize **Qwen 2.5**, which has shown remarkable ability to understand code-switched Indic languages. Furthermore, our RAG system is language-aware. The IntentRouter detects the language of the query and instructs the downstream agents to respond in the same language, maintaining a natural conversational flow.

5. Security and Compliance

Building for Bharat means building with trust. Our architecture is "Secure by Design" and aligns with RBI's Master Directions on Digital Payment Security.

5.1 Zero Trust & Device Binding

We implement a **Zero Trust** model. Merely having a login credential is not enough. We enforce **Device Binding**. When a user logs in, we capture the device fingerprint. Subsequent requests must originate from this trusted device. If a login attempt comes from a new device, we trigger a step-up authentication flow. This prevents remote attacks even if credentials are compromised.

5.2 Data Privacy (DPDP Act 2023)

In compliance with India's Digital Personal Data Protection (DPDP) Act: **Data Minimization:** We only fetch the data needed for the specific query. The AI context window is cleared after the session ends. **Purpose Limitation:** Voice samples are used strictly for authentication and are stored as irreversible mathematical embeddings, not raw audio files. **Local Processing:** By using Ollama locally, customer PII (Personally Identifiable Information) never leaves the bank's secure infrastructure to go to a public cloud LLM provider like OpenAI. This is a critical compliance feature for banking data sovereignty.

6. Scalability and Performance

To serve millions of users, the system must be elastic and resilient.

6.1 Stateless Architecture

Both our Backend API and AI Backend are designed to be stateless. Session state is managed via JWT tokens and external databases (Redis/SQL). This allows us to horizontally scale our application servers. We can spin up 100 instances of the AI Backend behind a load balancer to handle traffic spikes during demonetization-like events or festival sales.

6.2 Caching Strategy

Database hits are expensive. We implement a multi-layer caching strategy. 1. **RAG Cache:** Frequently asked questions (e.g., "What is the interest rate for FD?") are cached. If the same query comes in, we serve the answer from the cache, bypassing the vector search and LLM generation entirely. 2. **Session Cache:** User profiles and account metadata are cached in memory (with Redis support planned) to reduce SQL queries during an active conversation session.

Conclusion

Vaani is not just a hackathon prototype; it is a blueprint for the future of inclusive banking in India. By combining state-of-the-art Generative AI with rigorous banking security standards, we have created a system that is intuitive enough for a farmer in a remote village and robust enough for a national bank. We are ready to bridge the digital divide, one voice command at a time.