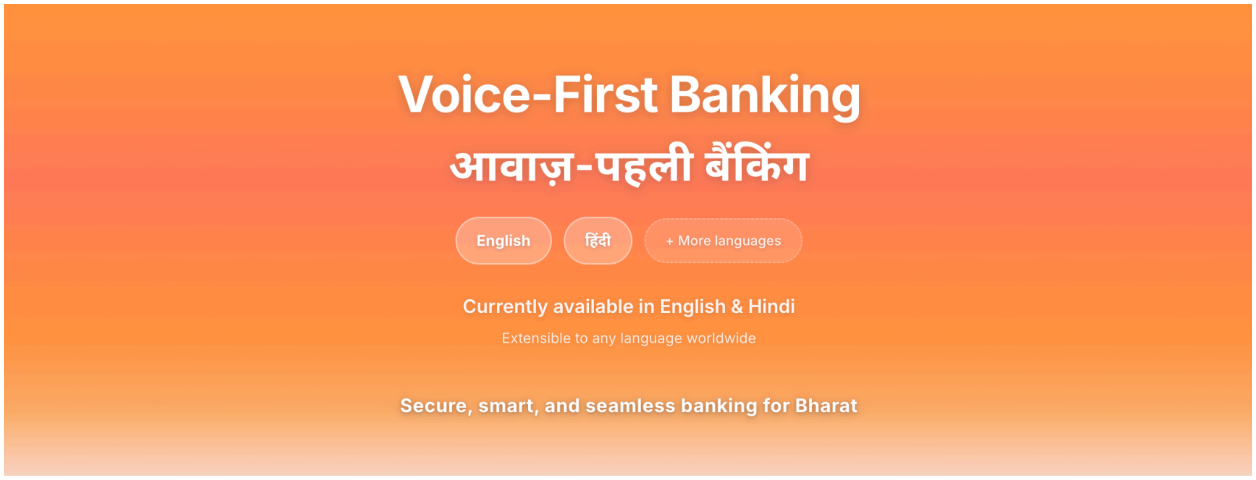


Vaani Banking Voice Assistant

Round 2 Submission Document

Project Theme: AI Voice Assistant for Financial Operations
Repository: Vaani Banking Voice Assistant : GitHub link
Demo Video: Demo Video link YouTube
Submission Date: 23 Nov 2025

Executive Summary

Vaani is an AI-powered voice banking assistant that enables customers to perform secure financial operations through natural conversational interactions. Built with a modern microservices architecture, Vaani supports multilingual operations (English & Hindi), implements comprehensive security guardrails, and provides seamless voice-first banking experiences compliant with RBI guidelines.

Key Achievements:

- Full-stack voice-enabled banking platform with biometric authentication
- Multi-agent AI system with LangGraph orchestration
- Bilingual RAG system (English & Hindi) for loan and investment information
- Hello UPI payment system with RBI complianceComprehensive security guardrails (content moderation, PII detection, prompt injection protection)
- Production-ready architecture with observability and scalability

1. Technology Stack

1.1 Frontend Layer

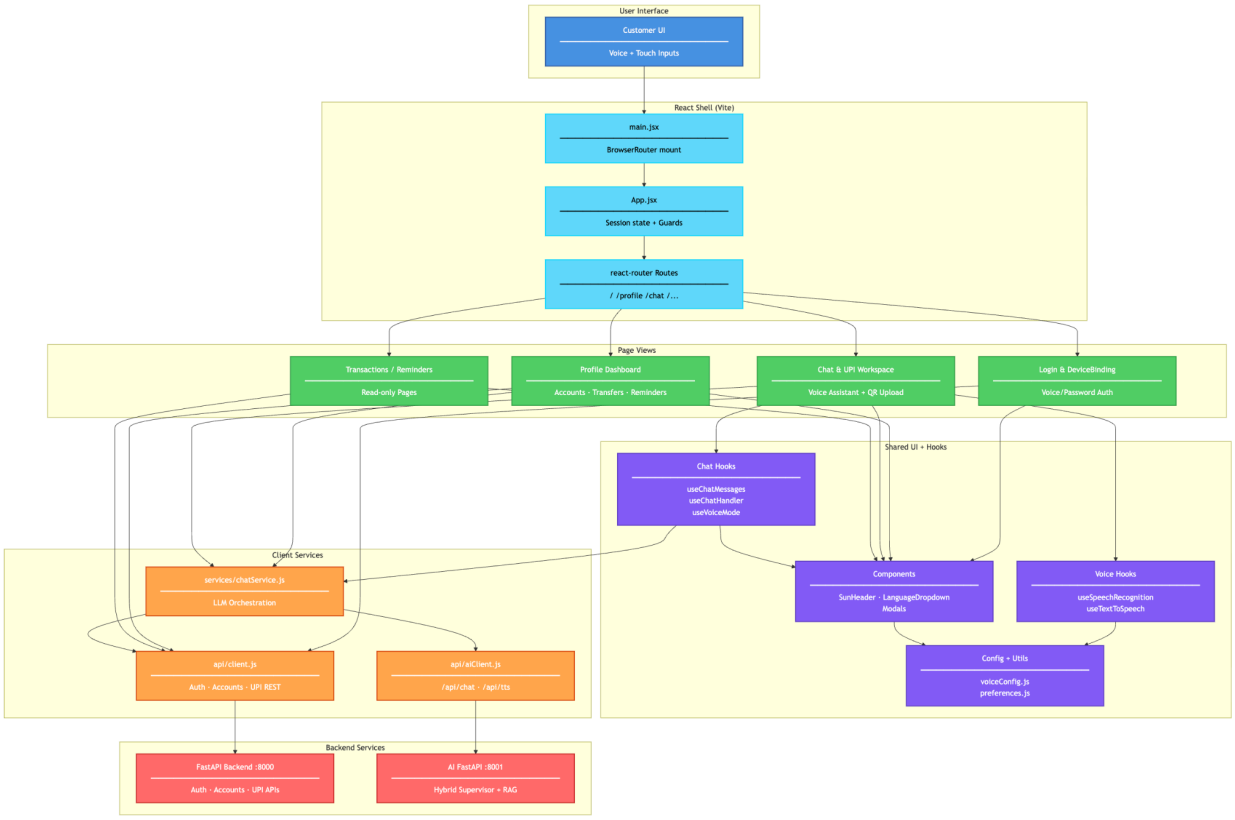
Framework & Build Tools:
- React 19 - Modern UI framework with hooks and context API

- Vite - Fast build tool and development server
- React Router v6 - Client-side routing
- Axios - HTTP client for API communication

Voice Technologies:
- Web Speech API - Browser-native Speech-to-Text (STT) and Text-to-Speech (TTS)
- Voice Recognition - Real-time voice input processing
- Voice Synthesis - Natural language audio output

State Management:
- Context API - Global state management
- Local Storage - Session persistence and user preferences

Frontend Architecture Diagram (Link to source file)



1.2 Backend API Layer

Framework & Runtime:
- FastAPI - High-performance async web framework
- Python 3.11+ - Modern Python with type hints

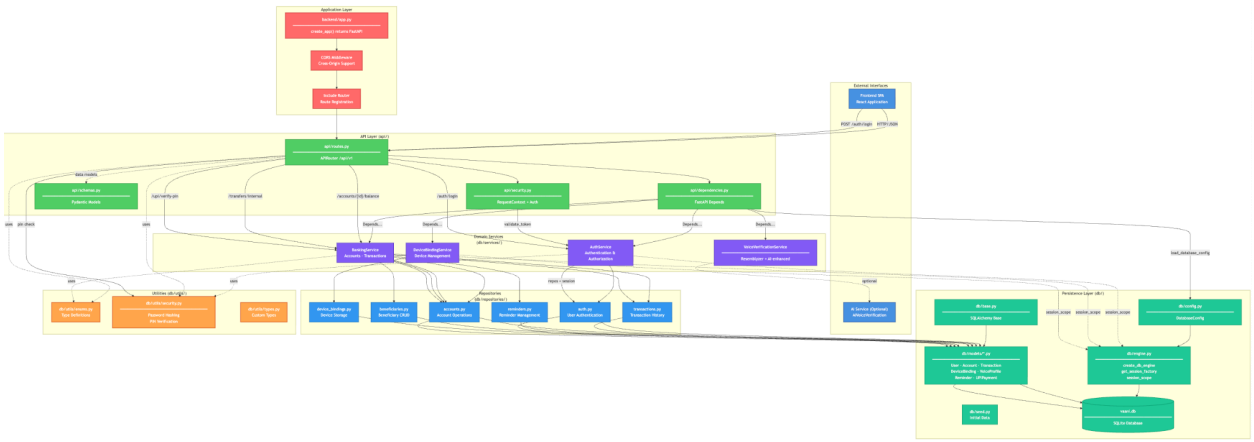
- Uvicorn - ASGI server for production deployment

Database & ORM:
- SQLAlchemy 2.0 - Modern ORM with async support
- SQLite (Development) - Lightweight database
- PostgreSQL Ready - Production database configuration

Authentication & Security:
- python-jose - JWT token generation and validation (for production)
- bcrypt - Password hashing (cost factor: 12)
- Resemblyzer - Voice biometric verification
- Pydantic v2 - Request/response validation

API Features:
- RESTful API design
- OpenAPI/Swagger documentation
- CORS configuration
- Request context tracking
- Structured error handling

Backend API Architecture (link to source file)



1.3 AI Backend Layer

AI Framework & Orchestration:
- LangGraph - Multi-agent orchestration and state management
- LangChain - LLM integration and tool calling
- LangSmith - Observability and tracing (optional)

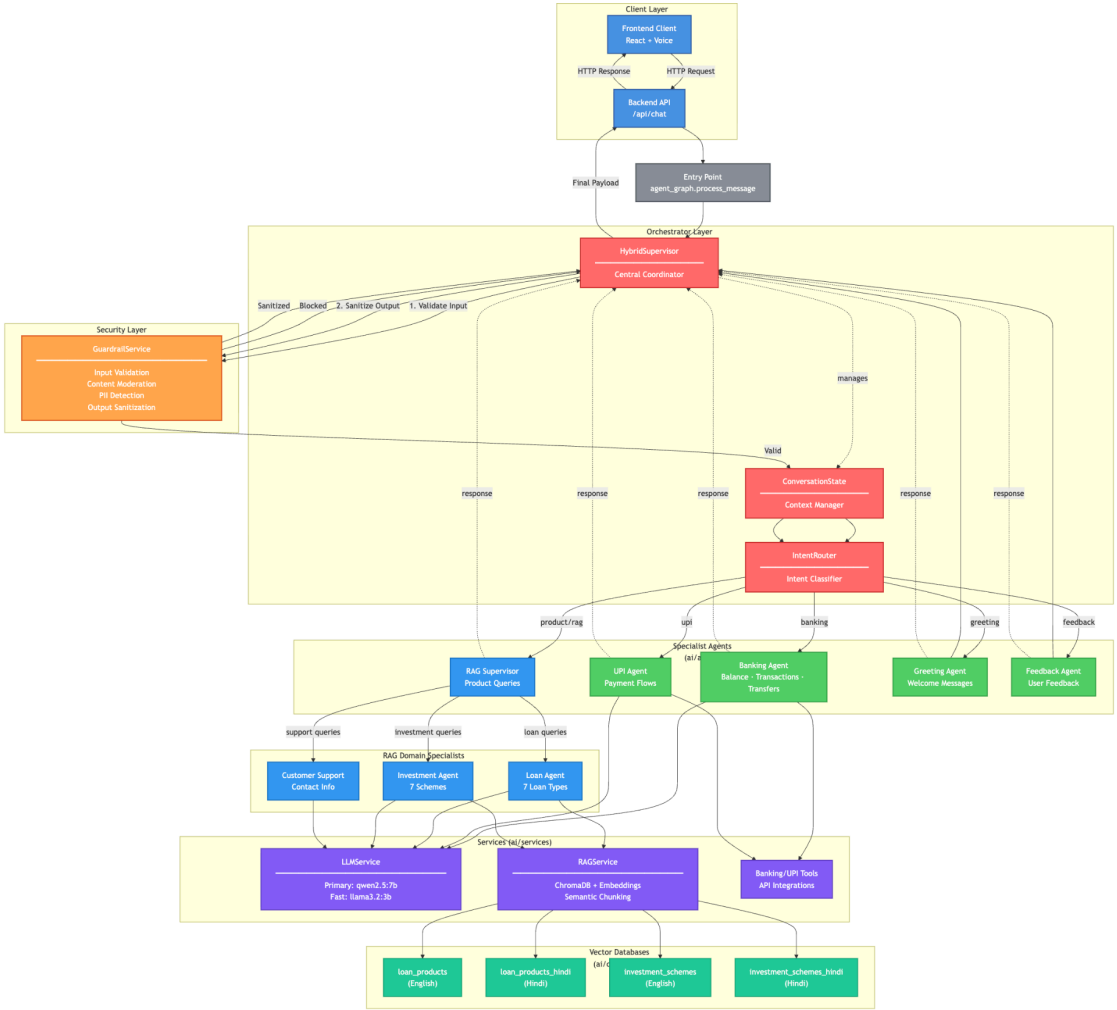
LLM Providers:
- Ollama (Primary) - Local LLM inference

  - Qwen 2.5 7B - Comprehensive responses, multilingual support
  - Llama 3.2 3B - Fast classification and voice mode
- OpenAI (Alternative) - Cloud-based LLM (configurable)

Vector Database & RAG:
- ChromaDB - Vector database for document embeddings
- sentence-transformers/all-MiniLM-L6-v2 - Multilingual embeddings (384 dimensions)
- Semantic Chunking - Intelligent document segmentation

AI Services:
- LLMService - Unified LLM provider abstraction
- RAGService - Retrieval-Augmented Generation with caching
- GuardrailService - Input/output safety checks
- Azure TTS (Optional) - Cloud-based text-to-speech

AI Architecture Diagram (Link to source file)



1.4 Development & Deployment Tools

Package Management:
- uv - Fast Python package manager
- npm - Node.js package manager

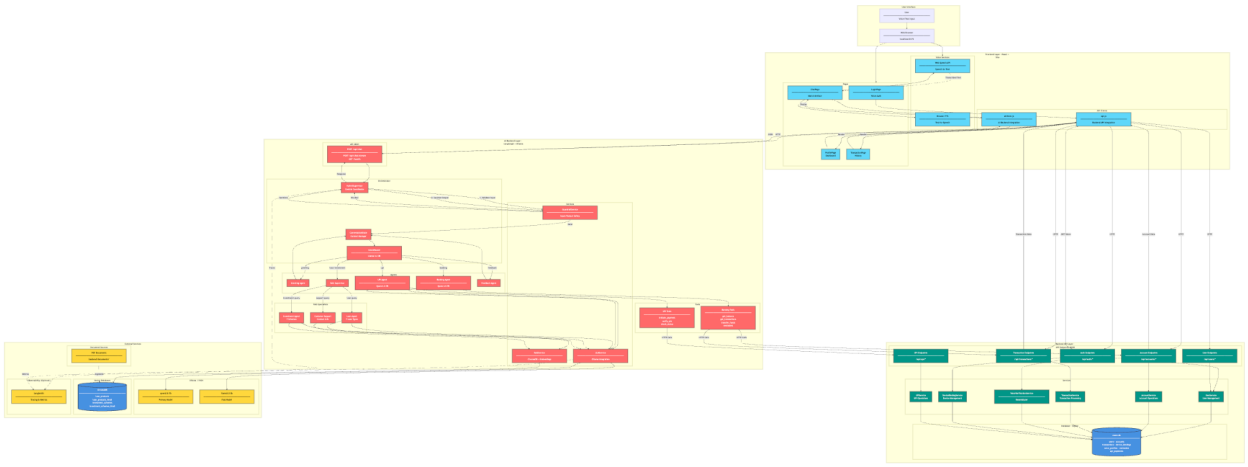
Version Control:
- Git - Source code versioning
- GitHub - Repository hosting

Deployment (for production):
- Vercel - Frontend and backend deployment
- Docker (Ready) - Containerization support
- Kubernetes (Ready) - Orchestration support

Monitoring & Observability:
- structlog - Structured logging
- LangSmith - AI tracing and monitoring
- Python logging - Application logging

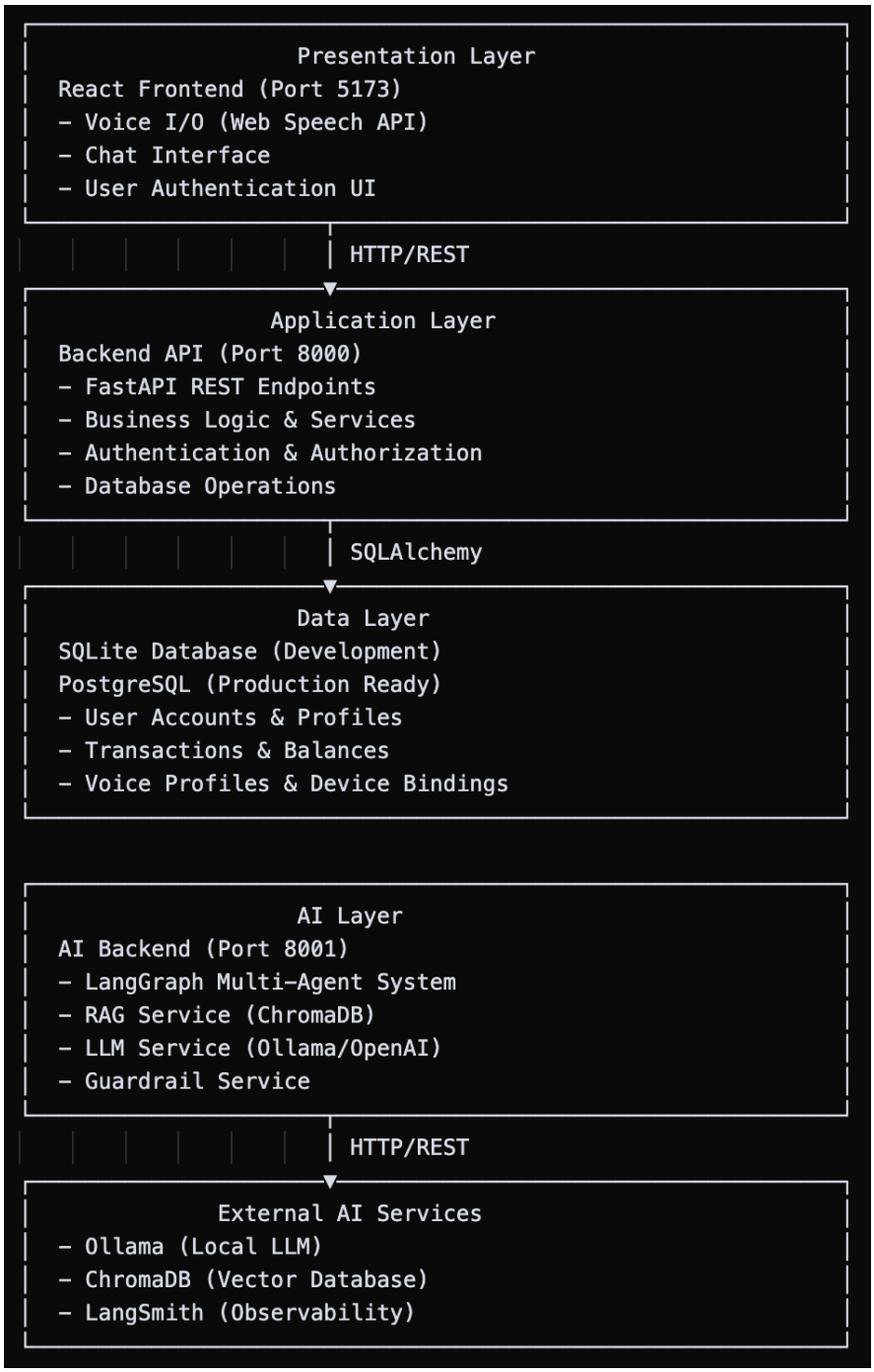
2. System Architecture

Technology Stack Overview (Link to source file)



2.1 Three-Tier Microservices Architecture

Vaani follows a modern microservices architecture with clear separation of concerns:

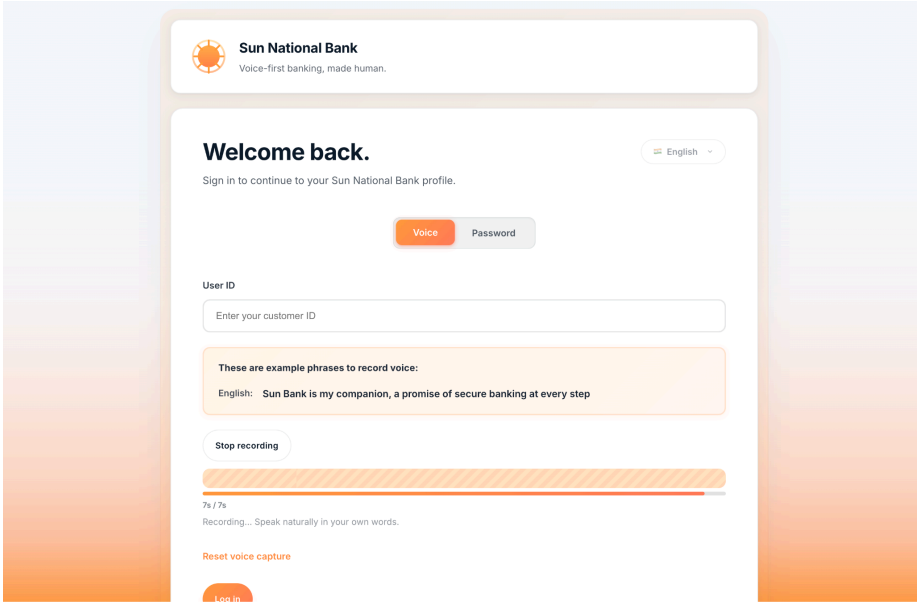


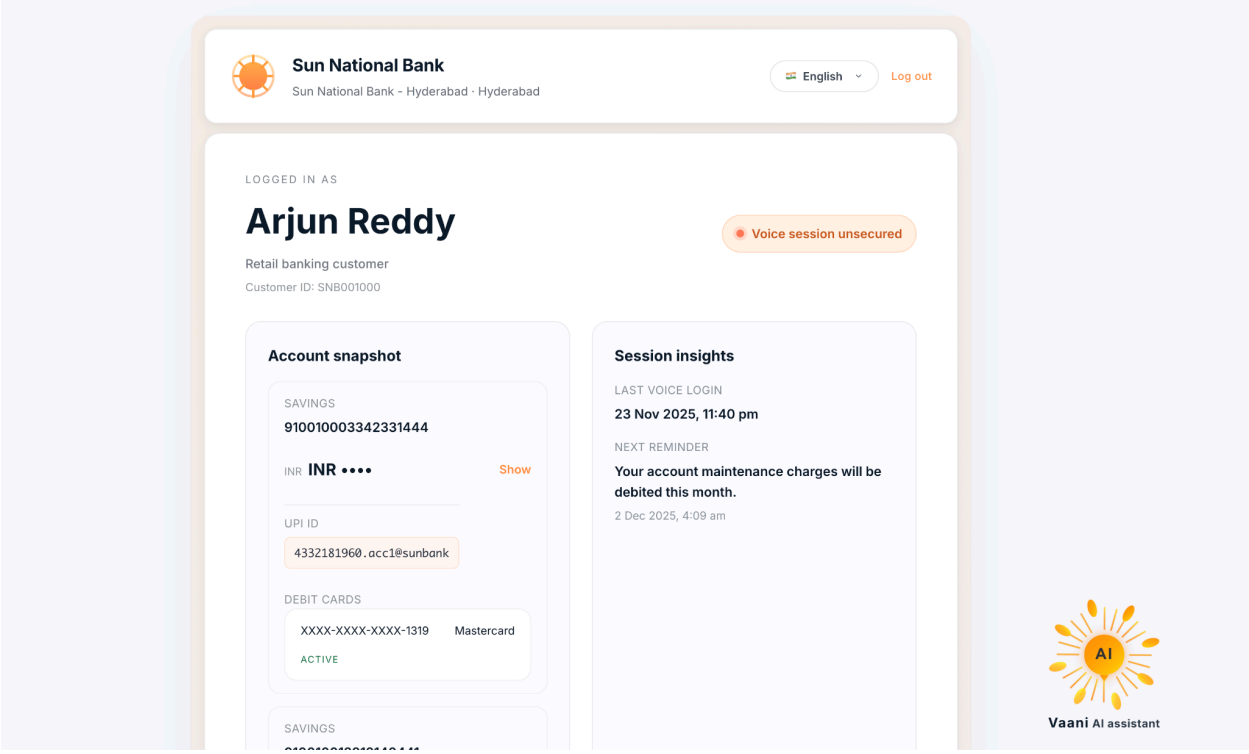
2.2 Request Flow Architecture

2.2.1 Voice Login Flow

```
User speaks enrollment phrase
   ↓
Frontend: Web Speech API transcribes
   ↓
POST /api/auth/voice-login
   {
     username: "user123",
     voice_sample: <audio_blob>,
     device_info: {...}
   }
   ↓
Backend: Resemblyzer generates voice embedding
   ↓
Backend: Compares with stored voice profile (cosine similarity)
   ↓
Backend: Verifies device binding
   ↓
Backend: Returns JWT token (24h expiry)
   ↓
Frontend: Stores token, redirects to chat
```

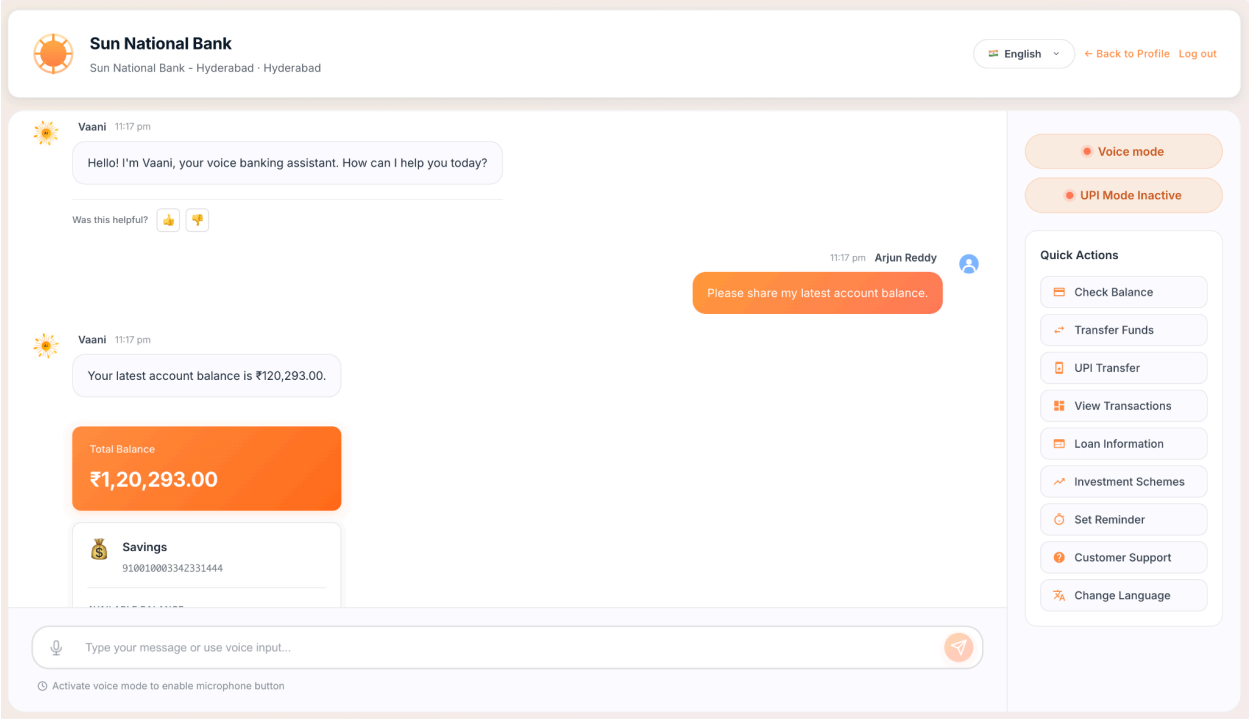
Voice Login Flow: Frontend login page screenshot





2.2.2 Banking Query Flow (Balance Check)

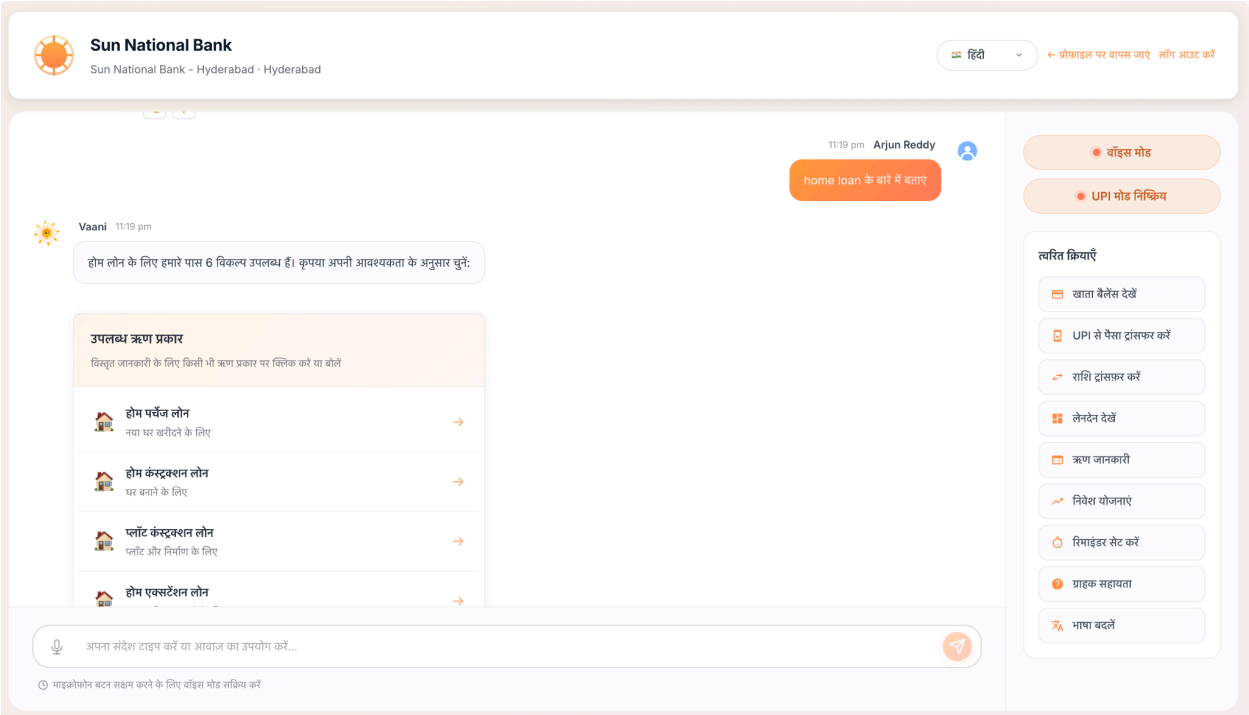
```
User: "What's my account balance?"
   ↓
Frontend: aiClient.sendChatMessage()
   → POST http://localhost:8001/api/chat
   ↓
AI Backend: HybridSupervisor receives message
   ↓
HybridSupervisor:
  1. Input Guardrails (content moderation, PII detection)
  2. IntentRouter classifies as "banking_operation"
  3. Routes to Banking Agent
   ↓
Banking Agent (Qwen 2.5 7B):
  1. Decides to use get_account_balance tool
  2. Tool calls Backend API:
     GET http://localhost:8000/api/accounts/{id}
   ↓
Backend API:
  1. Validates JWT token
  2. Queries SQLite database
```

3. Returns: { balance: 50000.00, currency: "INR" }
↓
Banking Agent:
1. Formats response: "Your account balance is ₹50,000.00"
2. Output Guardrails (language consistency, safety check)
↓
AI Backend returns response
↓
Frontend displays message + speaks via TTS
```

**Chat Interface with Balance Query : Frontend chat page screenshot**


```

2.2.3 RAG Query Flow (Loan Information)

```
User: "Tell me about home loans" / "होम लोन के बारे में बताओ"
    ↓
AI Backend: Intent classified as "loan_inquiry"
    ↓
RAG Supervisor: Routes to Loan Agent
    ↓
Loan Agent:
  1. Detects language (English or Hindi)
  2. Queries appropriate vector DB:
     - loan_products (English)
     - loan_products_hindi (Hindi)
  3. Retrieves relevant PDF chunks (semantic search)
  4. Sends context + query to LLM (Qwen 2.5 7B)
    ↓
LLM generates structured response with loan card
    ↓
Frontend displays loan information card
```

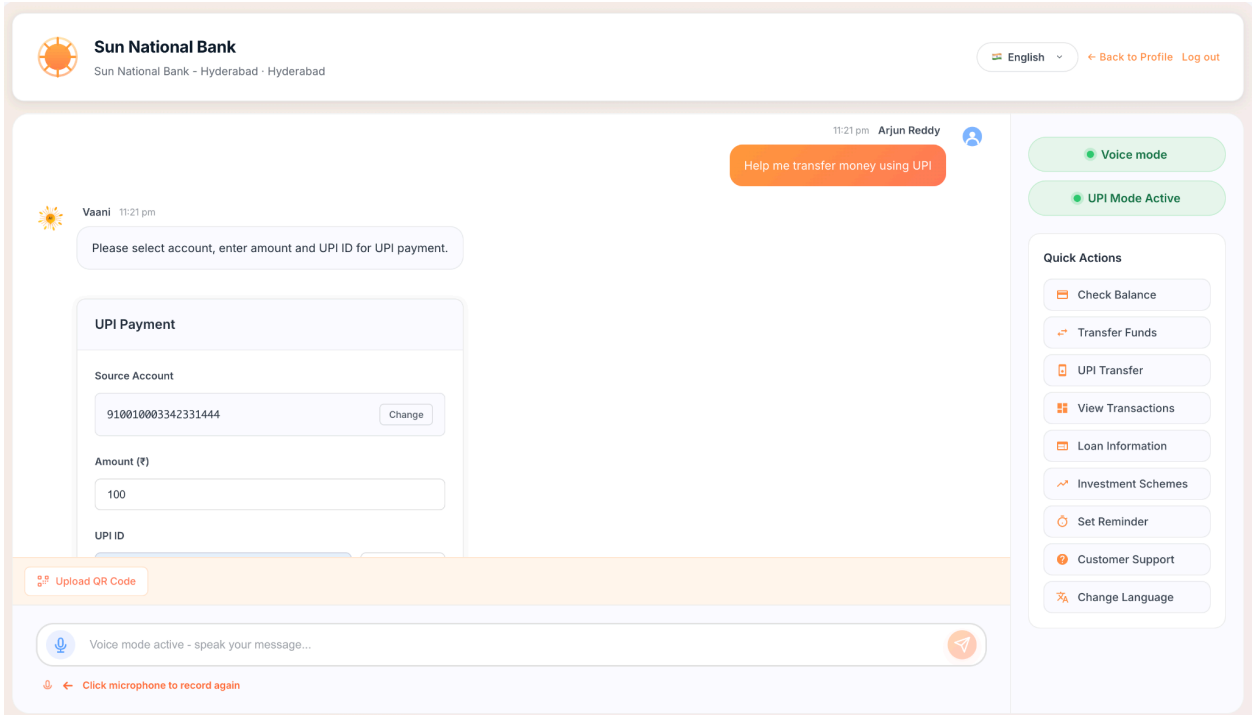
Loan Information Card Display: Frontend chat page showing loan card

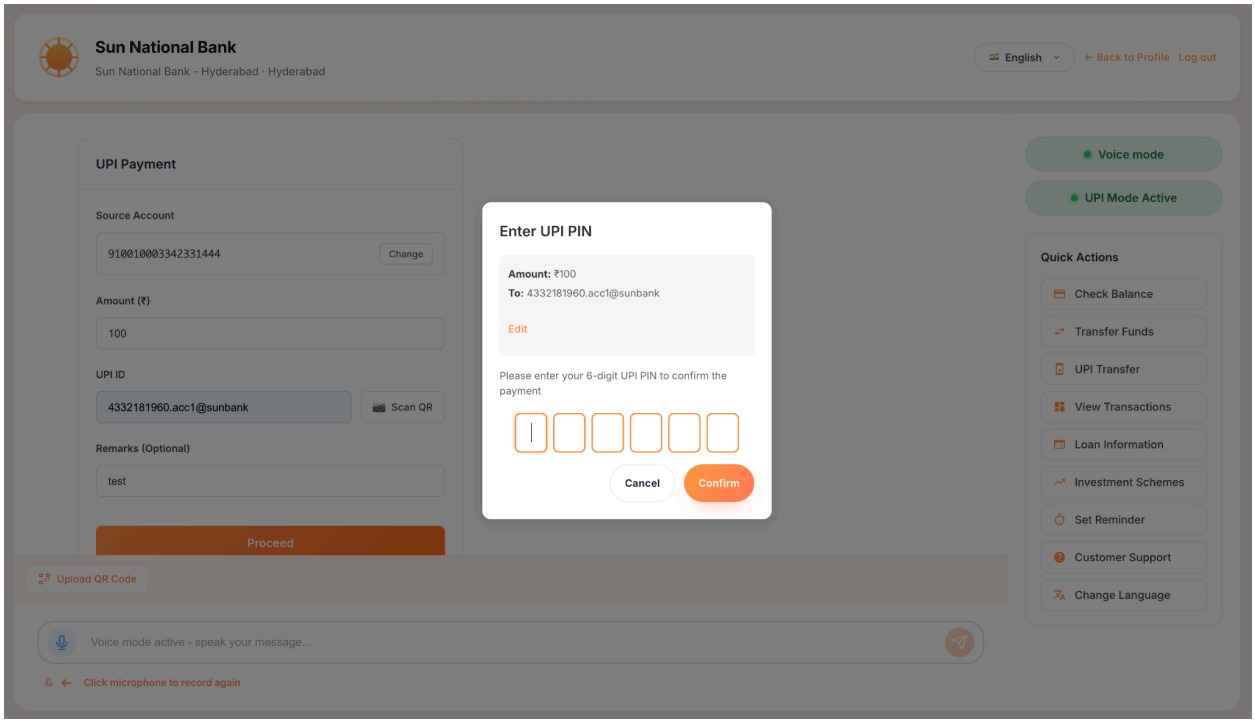


2.2.4 UPI Payment Flow

```
User: "Hello Vaani, send ₹500 to John via UPI"
   ↓
AI Backend: Intent classified as "upi_payment"
   ↓
UPI Agent (Multi-step flow):
  Step 1: Confirm recipient and amount
    → "Confirming: Send ₹500 to John?"

  User: "Yes"

  Step 2: Request UPI PIN
    → "Please enter your UPI PIN"
    → structured_data: { action: "collect_upi_pin" }

  Frontend: Shows PIN input modal
  User enters PIN

  Step 3: Verify PIN
    → Tool calls: POST /api/upi/verify-pin

  Step 4: Process payment
    → Tool calls: POST /api/upi/send-payment

  Backend:
    1. Verifies PIN
    2. Creates debit/credit transactions
    3. Updates balances
    4. Returns confirmation

  Step 5: Confirmation
    → "Payment successful! ₹500 sent to John."
    → structured_data: { payment_id: "TXN123", status: "success" }
```

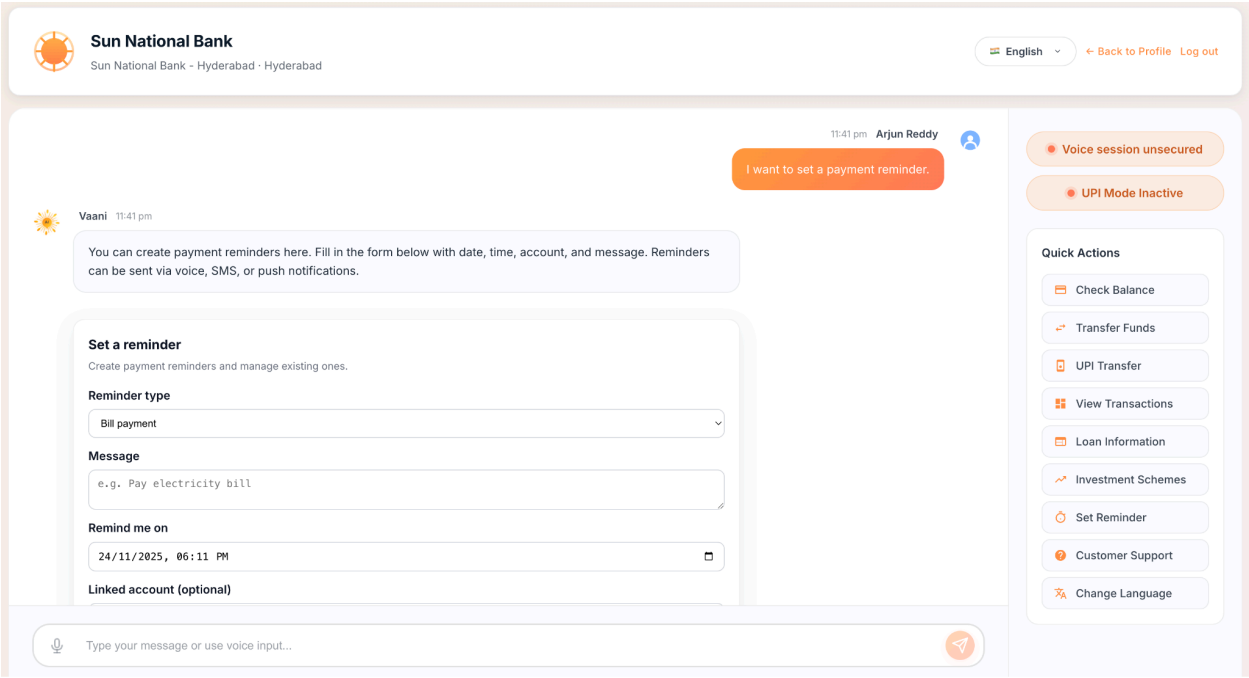
UPI Payment Flow : Frontend showing UPI payment modal, PIN entry, success message



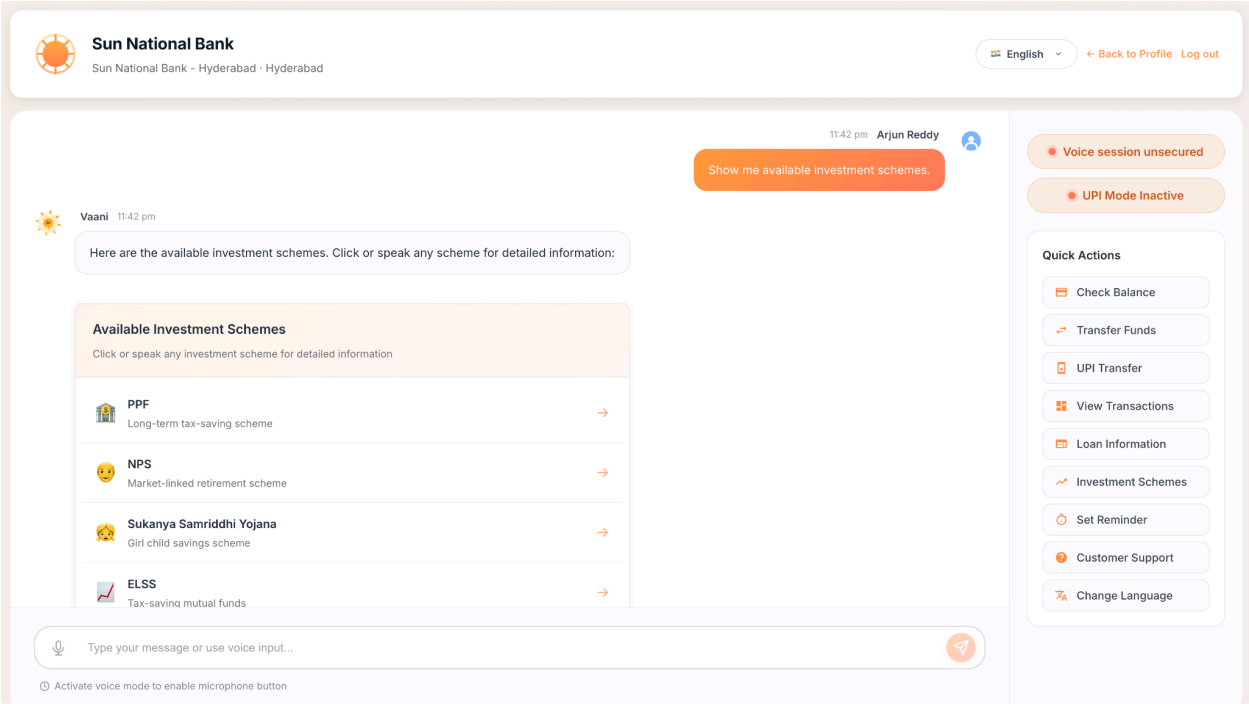


Other banking operations:

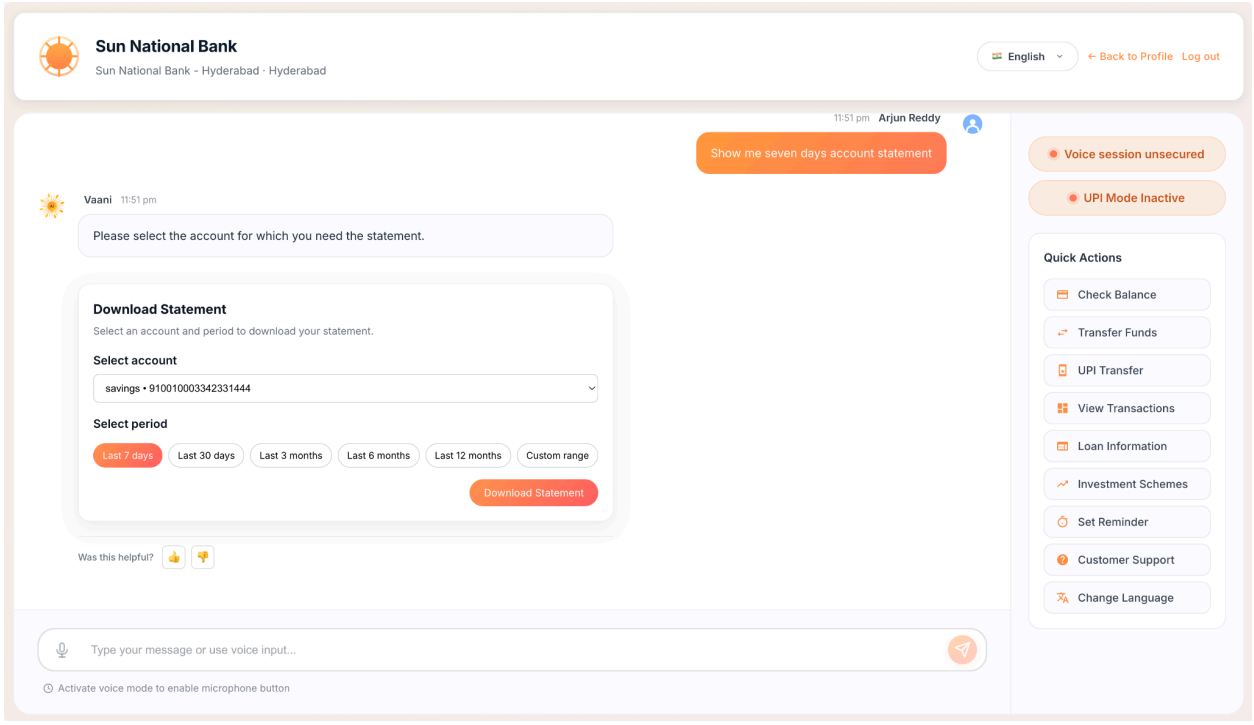
Set reminder

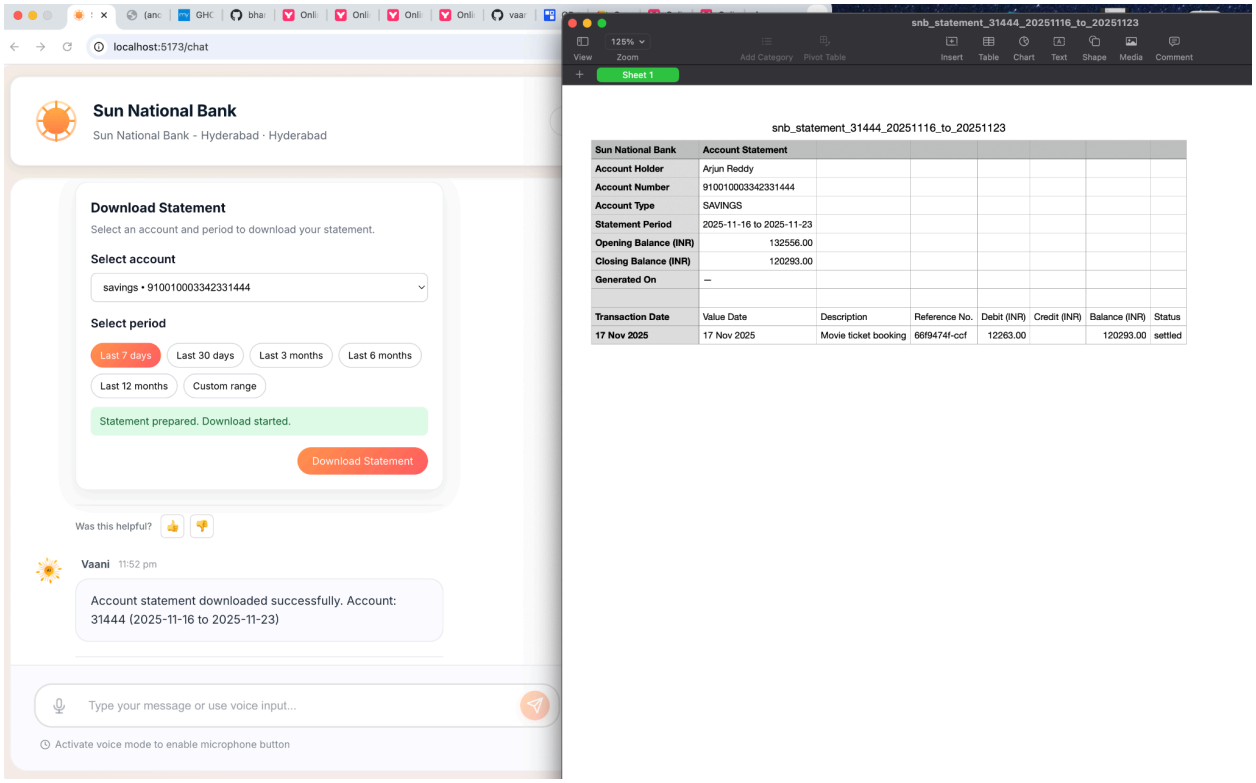


Get Investment details



Get Account Statement:



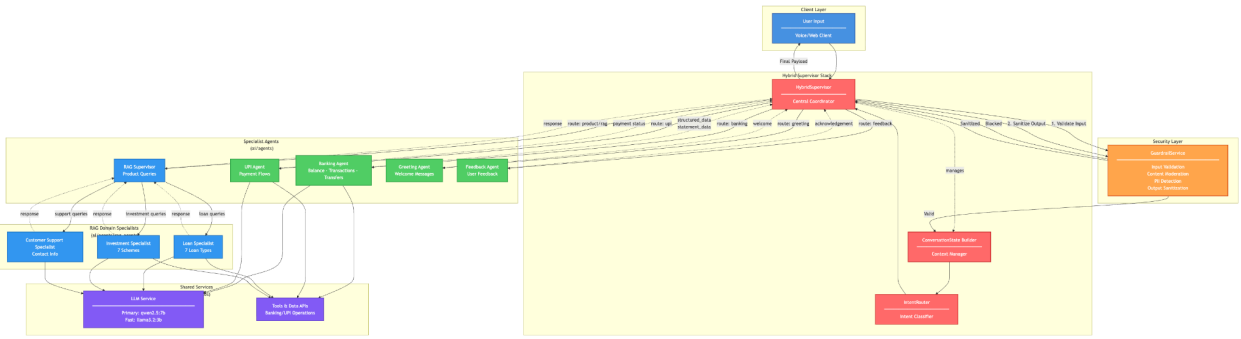


2.3 Multi-Agent Architecture

Vaani uses a Hybrid Supervisor Pattern with specialized agents:

```
HybridSupervisor
  ├── IntentRouter (classifies user intent)
  ├── ConversationState (maintains context)
  └── Agent Dispatcher (routes to appropriate agent)
      │
      ├── Greeting Agent (deterministic greetings)
      ├── Banking Agent (balances, transactions, transfers, reminders)
      ├── UPI Agent (Hello UPI payments with multi-step flow)
      ├── RAG Supervisor (routes to specialized RAG agents)
      │   ├── Loan Agent (7 loan types: Home, Personal, Car, etc.)
      │   ├── Investment Agent (7 schemes: PPF, NPS, SSY, etc.)
      │   └── Customer Support Agent (contact information)
      └── Feedback Agent (collects ratings/complaints)
```

Agent Architecture Diagram:Agent hierarchy,routing logic,tool calling (Link to source file)



---

3. Data Model & Storage

3.1 Database Schema

Primary Database: SQLite (Development) / PostgreSQL (Production Ready)

Core Tables

Users Table

```sql
CREATE TABLE users (
    id VARCHAR PRIMARY KEY,
    username VARCHAR UNIQUE NOT NULL,
    password_hash VARCHAR NOT NULL,
    first_name VARCHAR,
    last_name VARCHAR,
    email VARCHAR,
    phone_number VARCHAR,
    upi_id VARCHAR UNIQUE,
    created_at TIMESTAMP,
    updated_at TIMESTAMP
);
```

Accounts Table

```sql
CREATE TABLE accounts (
    id VARCHAR PRIMARY KEY,
    user_id VARCHAR REFERENCES users(id),
    account_number VARCHAR UNIQUE NOT NULL,
    account_type VARCHAR,  -- SAVINGS, CHECKING, etc.
    balance DECIMAL(15, 2) DEFAULT 0.00,
    currency VARCHAR DEFAULT 'INR',
    created_at TIMESTAMP,
    updated_at TIMESTAMP
);
```

Transactions Table

```sql
CREATE TABLE transactions (
    id VARCHAR PRIMARY KEY,
    account_id VARCHAR REFERENCES accounts(id),
    transaction_type VARCHAR,  -- DEBIT, CREDIT
    amount DECIMAL(15, 2) NOT NULL,
    channel VARCHAR,  -- UPI, TRANSFER, ATM, etc.
    reference_id VARCHAR,
    description TEXT,
    created_at TIMESTAMP
);
```

Voice Profiles Table

```sql
CREATE TABLE voice_profiles (
    id VARCHAR PRIMARY KEY,
    user_id VARCHAR REFERENCES users(id),
    voice_embedding BLOB,  -- Resemblyzer embedding (256 floats)
    enrollment_phrase TEXT,
    created_at TIMESTAMP,
    updated_at TIMESTAMP
);
```

Device Bindings Table

```sql
CREATE TABLE device_bindings (
    id VARCHAR PRIMARY KEY,
    user_id VARCHAR REFERENCES users(id),
    device_identifier VARCHAR NOT NULL,
    device_fingerprint VARCHAR,
    device_label VARCHAR,
    platform VARCHAR,
    is_trusted BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP,
    last_used_at TIMESTAMP
);
```

Reminders Table

```sql
CREATE TABLE reminders (
    id VARCHAR PRIMARY KEY,
    user_id VARCHAR REFERENCES users(id),
    title VARCHAR NOT NULL,
    description TEXT,
    reminder_type VARCHAR,  -- PAYMENT, BILL, etc.
    due_date DATE,
    status VARCHAR,  -- PENDING, COMPLETED, CANCELLED
    created_at TIMESTAMP
);
```

UPI Payments Table

```sql
CREATE TABLE upi_payments (
```

```
    id VARCHAR PRIMARY KEY,
    user_id VARCHAR REFERENCES users(id),
    source_account_id VARCHAR REFERENCES accounts(id),
    destination_account_id VARCHAR REFERENCES accounts(id),
    amount DECIMAL(15, 2) NOT NULL,
    upi_reference_id VARCHAR UNIQUE,
    status VARCHAR,  -- PENDING, SUCCESS, FAILED
    created_at TIMESTAMP
);
```

3.2 Vector Database (RAG System)

Vector Database: ChromaDB

Collections

1. Loan Products (English)
- Collection: `loan_products`
- Documents: 7 PDF files (Home, Personal, Car, Education, Business, Gold, Agriculture loans)
- Embeddings: `sentence-transformers/all-MiniLM-L6-v2` (384 dimensions)
- Chunks: ~350 semantic chunks with metadata

Note: Similarly for other documents on investments and loans

3.3 Data Storage Strategy

Structured Data:
- SQLite (Development) - File-based, zero configuration
- PostgreSQL (Production) - Managed service (AWS RDS, Supabase)

Vector Data:
- ChromaDB - Local persistence with `persist_directory`
- Future: Managed ChromaDB or Pinecone/Weaviate for production

Voice Data:
- Embeddings Only - Stored as BLOB (256 floats per embedding)
- Raw Audio - Not stored (privacy-first approach)
- Future: Encrypted storage for production

File Storage:
- PDF Documents - Local filesystem (`backend/documents/`)
- Future: S3/blob storage for production

4. AI / ML / Automation Components

4.1 Multi-Agent System (LangGraph)

Vaani implements a sophisticated multi-agent system using LangGraph for orchestration:

4.1.1 Intent Classification

Model: Llama 3.2 3B (Fast Model)
Purpose: Quick intent classification for routing
Accuracy: ~90%+ intent detection

4.1.2 Banking Agent
Model: Qwen 2.5 7B

4.1.3 UPI Agent
Model: Qwen 2.5 7B

4.1.4 RAG System (Retrieval-Augmented Generation)

Architecture:
- Supervisor Pattern: RAG Supervisor routes to specialized agents
- Loan Agent: Handles 7 loan types with bilingual support
- Investment Agent: Handles 7 investment schemes with bilingual support
- Customer Support Agent: Provides contact information

RAG Process:
1. Query Processing: User query in English or Hindi
2. Language Detection: Automatic language identification
3. Vector Search: Semantic similarity search in appropriate database
4. Context Retrieval: Top-k relevant chunks (k=3-5)
5. LLM Generation: Qwen 2.5 7B generates response with context
6. Structured Output: Returns loan/investment card with structured data

Performance:
- Retrieval Latency: 50-200ms
- LLM Generation: 500-1000ms
- Total RAG Query: < 3 seconds
- Cache Hit Rate: ~40% (120s TTL)

4.2 Voice Biometrics

Technology: Resemblyzer
Purpose: Voice-based authentication
Process:
1. Enrollment: User speaks enrollment phrase, embedding generated (256 floats)
2. Storage: Embedding stored in database (not raw audio)
3. Verification: New voice sample compared using cosine similarity
4. Threshold: 0.6 similarity score for authentication

Security Features:
- Device binding required for voice login
- Single trusted device at a time
- AI-enhanced verification (optional augmentation)

4.3 Guardrails & Safety

Implementation: Open-source only, no external API dependencies

4.3.1 Input Guardrails

Content Moderation:
- Toxic content detection (English & Hindi keywords)
- Profanity, threats, hate speech, harassment detection

PII Detection:
- Aadhaar numbers (12 digits)
- PAN cards (5 letters + 4 digits + 1 letter)
- Account numbers (9-18 digits)
- CVV/PIN (3-6 digits with keywords)
- Phone numbers, card numbers, IFSC codes

Prompt Injection Protection:
- Detects attempts to override system instructions
- Blocks commands like "ignore previous instructions"
- Supports English and Hindi injection patterns

Rate Limiting:
- Per-user limits: 30 requests/minute, 500 requests/hour
- In-memory tracking (Redis-ready for distributed)

4.3.2 Output Guardrails

Language Consistency:
- Verifies response matches requested language
- Detects language mixing (max 30% mixing allowed)
- Ensures Hindi responses use Devanagari script

Response Safety:
- Checks AI output for toxic content
- Prevents PII leakage in responses
- Validates structured data format

4.4 Multilingual Support

Languages Supported:
- English (en-IN) - Full support
- Hindi (hi-IN) - Full support with Devanagari script

Implementation:
- Bilingual RAG Databases: Separate vector databases for English and Hindi
- Language Detection: Automatic detection from user query
- LLM Multilingual: Qwen 2.5 7B supports 100+ languages
- Embeddings: Multilingual model (all-MiniLM-L6-v2) works for both languages

Hindi Features:
- Native Hindi PDF documents with proper fonts
- Hindi loan and investment information
- Hindi conversational responses
- Cultural appropriateness (female gender for Vaani)

4.5 Observability & Monitoring

LangSmith Integration:
- Tracing: All LLM interactions traced
- Metrics: Token usage, latency, error rates
- Visualization: Trace waterfall, conversation flows
- Debugging: Agent decisions, tool calls, errors

Structured Logging:
- Backend: Python logging with request/response tracking
- AI Backend: structlog for structured events
- Events: LLM calls, tool executions, agent decisions, errors

Performance Metrics:
- Response time tracking

- Token usage monitoring
- Cache hit rates
- Error rate tracking

5. Security & Compliance

5.1 Authentication & Authorization

5.1.1 Multi-Factor Authentication

Methods Supported:
1. Username/Password - Traditional authentication
2. Voice Biometric - Resemblyzer-based voice verification
3. Device Binding - Device fingerprint verification
4. OTP Validation - One-time password (future enhancement)

Voice Authentication Flow:
- User speaks enrollment phrase
- Voice embedding generated (256-dimensional vector)
- Embedding stored securely (not raw audio)
- Future logins: voice sample compared using cosine similarity
- Threshold: 0.6 similarity score
- Device binding required for security

5.1.2 JWT Token Security

Implementation:
- Algorithm: HS256
- Expiry: 24 hours (configurable)
- Storage: Local storage (future: HTTP-only cookies)
- Validation: All protected routes validate JWT

Token Structure:
```json
{
  "user_id": "uuid",
  "username": "user123",
  "exp": 1234567890,
  "iat": 1234567890
}
```

5.2 Data Security

5.2.1 Password Security

- Hashing: bcrypt with cost factor 12
- Storage: Hashed passwords only (never plaintext)
- Validation: Secure comparison using bcrypt

5.2.2 Voice Data Security

- Storage: Embeddings only (256 floats), not raw audio
- Privacy: Raw voice samples never stored
- Encryption: Future: encrypted embeddings at rest

5.2.3 Database Security

- Development: SQLite with file permissions
- Production: PostgreSQL with encryption at rest
- Connection: TLS for database connections (production)

5.2.4 API Security

- CORS: Configured origins (development: permissive, production: strict)
- Input Validation: Pydantic schemas for all requests
- Rate Limiting: Guardrail-based rate limiting (30/min, 500/hour)
- HTTPS: TLS for all API calls (production)

5.3 RBI Compliance (UPI Payments)

5.3.1 Hello UPI Compliance

PIN Security:
- PIN must be entered manually (keyboard/screen)
- PIN never spoken aloud
- PIN masked during entry
- PIN cleared immediately after use
- PIN not stored anywhere

User Consent:
- Explicit consent before first UPI payment
- Terms and conditions displayed
- Consent recorded and stored
- Can be revoked

Transaction Verification:
- Amount, recipient, and account displayed before PIN
- User confirms before PIN entry
- Can cancel at any time
- Confirmation after success

Audit Trail:
- Every transaction has unique reference ID (UPI-YYYYMMDD-HHMMSS)
- UPI channel marked in transactions
- Complete transaction log
- Timestamps in IST

5.4 Guardrails & Content Safety

Comprehensive Guardrails:
- Input Guardrails: Content moderation, PII detection, prompt injection protection, rate limiting
- Output Guardrails: Language consistency, response safety, PII leakage prevention
- Open-Source Only: No external API dependencies
- Bilingual Support: English and Hindi guardrails

5.5 Privacy & Data Protection

Privacy-First Design:
- Voice samples not stored (embeddings only)
- PII detection prevents accidental sharing
- No tracking or analytics without consent
- User data encrypted at rest (production)

Data Minimization:
- Only necessary data collected
- Voice embeddings minimal (256 floats)
- Transaction data minimal (amount, parties, timestamp)

---

6. Scalability & Performance

6.1 Architecture Scalability

6.1.1 Horizontal Scaling

Frontend:
- Static Files: Infinitely scalable via CDN
- No State: Can serve unlimited concurrent users

- Deployment: Vercel, Netlify, S3 + CloudFront

Backend API:
- Stateless Design: JWT tokens enable horizontal scaling
- Load Balancer: Multiple instances behind load balancer
- Database Pooling: Connection pooling for database access
- Deployment: Docker containers, Kubernetes orchestration

AI Backend:
- Stateless Agents: Agent execution is stateless
- Multiple Instances: Can scale with multiple instances
- Shared ChromaDB: Managed service for vector database
- LLM Scaling: GPU instances for Ollama or cloud LLM APIs

6.1.2 Caching Strategy

Current Implementation:
- RAG Context Cache: 120-second TTL for retrieved context
- In-Memory Caching: Per-instance caching
- Cache Hit Rate: ~40% for frequent queries

Future Enhancements:
- Redis: Distributed caching for multiple instances
- Query Caching: Cache frequent queries (balance checks)
- Session State Caching: User session state caching
- Vector Search Caching: Cache vector search results

6.2 Performance Metrics

6.2.1 Response Times

Target Performance:
- Voice Mode (Fast Model): < 1 second
- Standard Chat: < 2 seconds
- RAG Queries: < 3 seconds
- Complex Banking Ops: < 5 seconds

Current Performance (Apple M4 Max, 128GB RAM):
- Intent Classification (Llama 3.2 3B): 100-200ms
- Balance Query (Qwen 2.5 7B): 500-800ms
- General FAQ (Qwen 2.5 7B): 1-2s
- RAG Retrieval: 50-200ms
- LLM Generation: 500-1000ms
- UPI Payment: 1-1.5s

6.2.2 Optimization Strategies

Model Selection:
- Fast Model (Llama 3.2 3B): Used for voice mode and intent classification
- Comprehensive Model (Qwen 2.5 7B): Used for complex queries and RAG

RAG Optimization:
- Semantic Chunking: Intelligent document segmentation
- Metadata Filtering: Language and document type filtering
- Context Caching: 120-second TTL reduces redundant retrievals

Future Optimizations:
- Streaming Responses: First token in 200-400ms
- Parallel Tool Calls: Execute multiple tools concurrently
- Embedding Caching: Cache document embeddings
- Warm Start: Pre-load models for faster first request

6.3 Error Handling & Resilience

6.3.1 Retry Strategy

Implementation:
- Automatic retry (3 attempts)
- Exponential backoff (1s, 2s, 4s)
- Graceful degradation on failure

6.3.2 Circuit Breaker Pattern

Future Implementation:
- Monitor Backend API health
- Open circuit after 5 failures in 1 minute
- Use cached data or fallback
- Half-open after 30 seconds for testing

6.3.3 Graceful Degradation

Fallback Chain:
1. Ollama unavailable → Fallback to OpenAI API (if configured)
2. OpenAI unavailable → Use rule-based responses
3. No LLM available → Return helpful error message

6.4 Database Performance

Current Setup:
- SQLite: File-based, suitable for development
- Connection Pooling: SQLAlchemy connection pooling
- Query Optimization: Indexed queries, efficient joins

Production Ready:
- PostgreSQL: Managed service (AWS RDS, Supabase)
- Read Replicas: For read-heavy workloads
- Connection Pooling: PgBouncer or SQLAlchemy pooling
- Query Optimization: Indexes, query analysis, optimization

7. Innovation Highlights

7.1 Novel Features

7.1.1 Hybrid Supervisor Pattern
- Innovation: Multi-level agent orchestration with supervisor routing
- Benefit: Specialized agents for different domains, better accuracy
- Differentiator: Not just a single LLM, but a coordinated multi-agent system

7.1.2 Bilingual RAG System
- Innovation: Separate vector databases for English and Hindi with same embedding model
- Benefit: Native language support without performance penalty
- Differentiator: True multilingual support, not just translation

7.1.3 Voice-First Banking
- Innovation: Complete voice-enabled banking operations
- Benefit: Hands-free banking, accessibility for all users
- Differentiator: Not just voice commands, but natural conversation

7.1.4 Comprehensive Guardrails
- Innovation: Open-source guardrails with no external API dependencies
- Benefit: Privacy-preserving, cost-effective, customizable
- Differentiator: Production-ready safety without vendor lock-in

7.1.5 Hello UPI Implementation
- Innovation: Voice-assisted UPI payments with RBI compliance
- Benefit: Secure, compliant, user-friendly payment experience
- Differentiator: Full UPI flow with voice, not just balance checks

7.2 Technical Differentiators

1. Multi-Agent Architecture: Not a single LLM, but coordinated agents
2. Bilingual Native Support: True Hindi support, not translation
3. Production-Ready Guardrails: Open-source, no vendor lock-in
4. Voice Biometrics: Secure voice authentication
5. RBI-Compliant UPI: Full payment flow with compliance
6. Observability: LangSmith integration for debugging
7. Scalable Architecture: Microservices, stateless design, horizontal scaling

8. Business Impact & Outcomes

8.1 Core Banking Operations Coverage

Fully Implemented:
- Account balance checking
- Transaction history viewing
- Fund transfers between accounts
- Payment reminders (create, list, update)
- Loan information (7 types)
- Investment scheme information (7 schemes)
- UPI payments (Hello UPI)
- Customer support queries

8.2 Accessibility & Inclusion

Voice-First Design:
- Hands-free banking for all users
- Natural language understanding
- No complex menu navigation
- Multilingual support (English & Hindi)

User Benefits:
- Elderly users: Simple voice commands
- Visually impaired: Voice-only interface
- Non-tech-savvy users: Natural conversation
- Regional language users: Hindi support

8.3 Business Metrics & ROI

Expected Outcomes:
- Reduced Call Center Load: Voice assistant handles common queries
- 24/7 Availability: Always-on banking assistant
- Faster Transactions: Voice commands faster than app navigation
- Higher Engagement: Natural conversation increases usage
- Cost Reduction: Automated support reduces operational costs

Pilot Plan:
1. Phase 1 (Months 1-2): Internal testing, bug fixes, performance optimization
2. Phase 2 (Months 3-4): Beta testing with 100-500 users
3. Phase 3 (Months 5-6): Gradual rollout to 10% of user base
4. Phase 4 (Months 7-12): Full rollout with monitoring and improvements

ROI Projection:
- Year 1: 30% reduction in call center queries
- Year 2: 50% reduction in call center queries
- Year 3: 70% reduction in call center queries
- Cost Savings: $500K - $2M annually (depending on scale)

8.4 Compliance & Privacy

Regulatory Compliance:
- RBI guidelines for UPI payments
- PIN security requirements
- User consent mechanisms
- Audit trail and logging
- Data privacy (embeddings only, not raw audio)

Privacy Features:
- Voice samples not stored
- PII detection and prevention
- Encrypted data storage (production)
- User consent for data usage

9. Repository Structure

9.1 Code Organization

```
vaani-banking-voice-assistant/
├── frontend/              React frontend (Port 5173)
│   ├── src/
│   │   ├── pages/         Page components (Login, Chat, Profile, Transactions)
│   │   ├── hooks/         Custom hooks (useVoiceRecognition, useTextToSpeech)
│   │   ├── services/      API clients (aiClient, api)
│   │   └── components/    Reusable components
│   └── package.json
│
├── backend/               FastAPI backend (Port 8000)
│   ├── api/               API routes, schemas, security
│   ├── db/                Database models, repositories, services
│   │   ├── models/        SQLAlchemy models
│   │   ├── repositories/  Data access layer
│   │   └── services/      Business logic
│   └── app.py             FastAPI application
│
├── ai/                    AI backend (Port 8001)
```

```
│   ├── agents/            LangGraph agents
│   │   ├── banking_agent.py
│   │   ├── upi_agent.py
│   │   └── rag_agents/    Loan, Investment, Support agents
│   ├── orchestrator/      HybridSupervisor, IntentRouter
│   ├── services/          LLM, RAG, Guardrail services
│   ├── tools/             Banking and UPI tools
│   ├── chroma_db/         Vector databases
│   └── main.py            FastAPI server
│
├── documentation/         Comprehensive documentation
│   ├── system_architecture.md
│   ├── ai_architecture.md
│   ├── backend_architecture.md
│   ├── guardrails_implementation.md
│   ├── hindi_support.md
│   └── upi_payment_flow.md
│
└── requirements.txt       Python dependencies
```

11. Conclusion

Vaani Banking Voice Assistant represents a comprehensive, production-ready solution for voice-enabled financial operations. With its innovative multi-agent architecture, bilingual RAG system, comprehensive security guardrails, and RBI-compliant UPI implementation, Vaani demonstrates:

1. Deep Technical Understanding: Modern microservices architecture, LangGraph orchestration, vector databases
2. Security & Compliance: Voice biometrics, guardrails, RBI compliance, privacy-first design
3. Innovation: Hybrid supervisor pattern, bilingual RAG, voice-first banking
4. Business Impact: Complete banking operations, accessibility, scalability, ROI potential
5. Production Readiness: Observability, error handling, scalability, comprehensive documentation

Document Version: 1.0
Last Updated: Nov 2025
Prepared By: Vaani Development Team