BAKAME AI - Technical Specifications

📋 Executive Summary

This document provides detailed technical specifications for the BAKAME (Building African Knowledge through Accessible Mobile Education) AI system - a comprehensive voice and SMS-based learning platform designed for feature phones in underserved communities.

🎯 System Requirements

Functional Requirements

Core Learning Capabilities

- **FR-001:** Support 5 learning modules (English, Math, Comprehension, Debate, General)
- **FR-002:** Process voice calls with speech-to-text conversion
- **FR-003:** Handle SMS-based learning interactions
- **FR-004:** Maintain conversation context across sessions
- **FR-005:** Provide culturally-relevant Rwanda-specific content
- **FR-006:** Support both Kinyarwanda and English languages
- **FR-007:** Adapt difficulty based on user performance

AI Processing Requirements

- **FR-008:** Generate educational responses using GPT-4o-mini
- **FR-009:** Transcribe audio using ElevenLabs ConvAI or OpenAI Whisper
- **FR-010:** Provide fallback AI services (Llama, Deepgram)
- **FR-011:** Detect and respond to user emotions
- **FR-012:** Implement gamification with points and achievements
- **FR-013:** Support predictive learning analytics

Communication Requirements

- **FR-014:** Integrate with Twilio for voice and SMS
- **FR-015:** Generate TwiML responses for voice calls
- **FR-016:** Support concurrent user sessions
- **FR-017:** Maintain sub-3-second response times
- **FR-018:** Handle call interruptions gracefully

Non-Functional Requirements

Performance Requirements

- **NFR-001:** Support 1000+ concurrent users
- **NFR-002:** Achieve 99.9% uptime
- **NFR-003:** Response time < 3 seconds for AI generation
- **NFR-004:** Audio transcription < 2 seconds
- **NFR-005:** Database query response < 500ms

Scalability Requirements

- **NFR-006:** Horizontal scaling capability

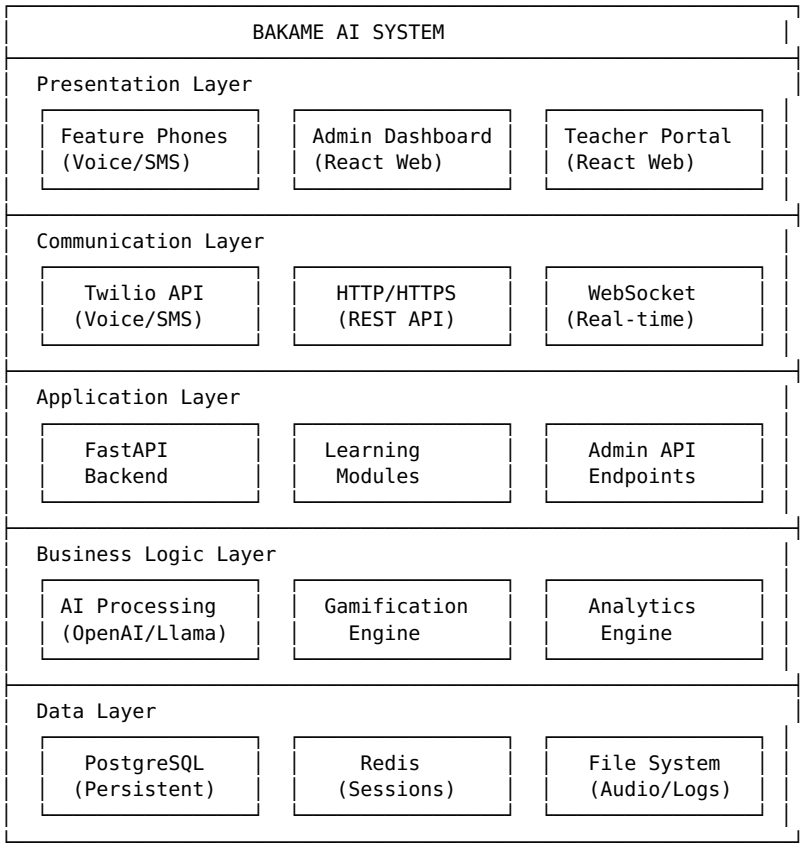
- **NFR-007:** Stateless application design
- **NFR-008:** Auto-scaling based on load
- **NFR-009:** Support for multiple regions

Security Requirements

- **NFR-010:** HTTPS/TLS encryption for all communications
- **NFR-011:** Secure API key management
- **NFR-012:** Role-based access control for admin features
- **NFR-013:** Data privacy compliance (GDPR-ready)
- **NFR-014:** Session data encryption

🏗 System Architecture

High-Level Architecture



Component Specifications

Backend API (FastAPI)

Technology Stack: - **Framework:** FastAPI 0.116.1+ - **Python Version:** 3.12+ - **ASGI Server:** Uvicorn - **Dependency Management:** Poetry

Key Components:

```
# Main Application Structure
app/
├── main.py                 # FastAPI application entry
```

```
├── config.py              # Configuration management
├── routers/               # API route handlers
│   ├── webhooks.py        # Twilio webhook endpoints
│   ├── admin.py           # Admin dashboard APIs
│   ├── auth.py            # Authentication endpoints
│   └── content.py         # Content management APIs
├── services/              # Business logic services
│   ├── twilio_service.py  # Twilio integration
│   ├── openai_service.py  # AI text generation
│   ├── redis_service.py   # Session management
│   └── [15+ specialized services]
├── modules/               # Learning module logic
│   ├── english_module.py  # English learning
│   ├── math_module.py     # Mathematics
│   ├── comprehension_module.py  # Reading
│   ├── debate_module.py   # Critical thinking
│   └── general_module.py  # Entry point/routing
└── models/                # Data models
    ├── database.py        # SQLAlchemy models
    └── auth.py            # Authentication models
```

API Endpoints:

Endpoint	Method	Purpose	Authentication
`/webhook/call`	POST	Handle Twilio voice calls	Twilio signature
`/webhook/sms`	POST	Handle Twilio SMS	Twilio signature
`/admin/stats`	GET	Usage statistics	JWT required
`/admin/users`	GET	User management	Admin role
`/admin/export/csv`	GET	Data export	Admin role
`/auth/login`	POST	Admin authentication	None
`/auth/register`	POST	User registration	None
`/api/content`	GET	Content management	JWT required
`/healthz`	GET	Health check	None

Learning Modules

Module Interface:

```python
class LearningModule:
    def __init__(self):
        self.module_name: str

    async def process_input(self, user_input: str, user_context: Dict[str, Any]) -> str:
        """Process user input and return AI response"""
        pass

    def get_welcome_message(self) -> str:
        """Return module welcome message"""
        pass
```

English Module Specifications: - **Grammar Correction:** Identify and explain grammar mistakes - **Pronunciation Practice:** Provide feedback on spoken English - **Conversation Practice:** Engage in natural dialogue - **Vocabulary Building:** Contextual word learning - **Cultural Integration:** Rwanda-specific examples and phrases

Math Module Specifications: - **Mental Mathematics:** Arithmetic problem generation - **Adaptive Difficulty:** 5 levels (basic → complex) - **Rwanda Context:** RWF calculations, local measurements - **Dynamic Problems:** AI-generated contextual problems - **Progress Tracking:** Accuracy-based level progression

Comprehension Module Specifications: - **Story Generation:** AI-created Rwanda-themed stories - **Interactive Q&A:** Comprehension testing - **Critical Analysis:** Story interpretation skills - **Cultural Stories:** Traditional and modern Rwanda narratives

Debate Module Specifications: - **Topic Generation:** Rwanda-relevant debate topics - **Argument Structure:** Logical reasoning practice - **Counter-arguments:** AI-powered opposing viewpoints - **Respectful Dialogue:** Ubuntu philosophy integration

General Module Specifications: - **Entry Point:** Initial user interaction - **Module Routing:** Direct users to appropriate modules - **Help System:** Explain available features - **Context Management:** Maintain conversation state

AI Processing Services

ElevenLabs Service Specifications:

```
class ElevenLabsService:
    def __init__(self):
        self.agent_id = settings.elevenlabs_agent_id
        self.websocket_url = f"wss://api.elevenlabs.io/v1/convai/conversation?agent_id=
        {self.agent_id}"

    async def transcribe_audio(self, audio_data: bytes, user_context: Dict[str, Any]) ->
        str:
        """Transcribe audio using ElevenLabs ConvAI"""
        # WebSocket-based real-time transcription
        # Agent ID: agent_0301k3y6dwrve63sb37n6f4ffkrj
        # Public agent mode (no API key required)

    async def process_conversation(self, audio_data: bytes, user_context: Dict[str, Any])
        -> str:
        """Process complete conversation turn with ElevenLabs agent"""
        # Real-time voice processing with cultural context
        # Rwandan-specific prompts and responses
        # Natural conversation flow
```

OpenAI Service Specifications:

```
class OpenAIService:
    def __init__(self):
        self.client = openai.OpenAI(api_key=settings.openai_api_key)

    async def transcribe_audio(self, audio_data: bytes) -> str:
        """Convert speech to text using Whisper"""
        # Model: whisper-1
        # Format: WAV/MP3 support
        # Response: Plain text

    async def generate_response(self, messages: List[Dict], module_name: str) -> str:
        """Generate educational response using GPT-4o-mini"""
        # Model: gpt-4o-mini
        # Max tokens: 300
        # Temperature: 0.8
        # Cultural context integration
```

Cultural Context Prompts: - **English:** “You’re a friendly, encouraging English conversation partner who understands Rwandan culture deeply…” - **Math:** “You’re an enthusiastic math mentor who makes numbers fun using Rwandan contexts. Use examples with Rwandan francs (RWF)…” - **Comprehension:** “You’re an engaging storyteller who loves Rwandan culture and traditions…” - **Debate:** “You’re a thoughtful discussion partner who understands Rwandan society and values deeply…” - **General:** “You’re BAKAME, a warm and intelligent AI learning companion who understands Rwandan culture deeply…”

Session Management (Redis)

Session Data Structure:

```
{
  "user_context:{phone_number}": {
    "current_module": "math",
    "conversation_history": [
      {
        "user": "I want to practice math",
        "ai": "Muraho! Let's practice math with RWF...",
        "timestamp": "2024-01-01T12:00:00Z"
      }
    ],
    "user_state": {
      "math_level": "medium",
      "math_problems_attempted": 15,
      "math_problems_correct": 12,
      "current_math_problem": {
        "question": "150 + 75",
        "answer": 225,
        "operation": "+"
      }
    },
    "user_name": "Jean",
    "phone_number": "+250781234567",
    "session_start": "2024-01-01T11:45:00Z"
  }
}
```

Session Management Features: - **TTL Management:** 1-hour default, 24-hour for active sessions - **Memory Fallback:** In-memory storage when Redis unavailable - **Conversation Summarization:** Automatic history compression - **Context Optimization:** AI-ready conversation formatting

Database Schema (PostgreSQL)

Core Tables:

```
-- User Management
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    phone_number VARCHAR UNIQUE NOT NULL,
    user_type VARCHAR DEFAULT 'student',
    name VARCHAR,
    region VARCHAR,
    school VARCHAR,
    grade_level VARCHAR,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW(),
    last_active TIMESTAMP DEFAULT NOW(),
    total_points INTEGER DEFAULT 0,
    current_level VARCHAR DEFAULT 'beginner'
);

-- Session Tracking
CREATE TABLE user_sessions (
    id SERIAL PRIMARY KEY,
    phone_number VARCHAR NOT NULL,
    session_id VARCHAR NOT NULL,
    module_name VARCHAR,
    interaction_type VARCHAR,
    user_input TEXT,
    ai_response TEXT,
    timestamp TIMESTAMP DEFAULT NOW(),
    session_duration FLOAT
```

```
);

-- Module Analytics
CREATE TABLE module_usage (
    id SERIAL PRIMARY KEY,
    phone_number VARCHAR NOT NULL,
    module_name VARCHAR,
    usage_count INTEGER DEFAULT 1,
    last_used TIMESTAMP DEFAULT NOW(),
    total_duration FLOAT DEFAULT 0.0
);

-- Community Features
CREATE TABLE learning_groups (
    id SERIAL PRIMARY KEY,
    name VARCHAR NOT NULL,
    description TEXT,
    group_type VARCHAR,
    region VARCHAR,
    school VARCHAR,
    grade_level VARCHAR,
    subject VARCHAR,
    teacher_phone VARCHAR,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW(),
    max_members INTEGER DEFAULT 50
);

-- Authentication
CREATE TABLE web_users (
    id SERIAL PRIMARY KEY,
    email VARCHAR UNIQUE NOT NULL,
    full_name VARCHAR,
    hashed_password VARCHAR,
    role VARCHAR DEFAULT 'user',
    organization VARCHAR,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW()
);
```

Indexing Strategy:

```
-- Performance indexes
CREATE INDEX idx_users_phone ON users(phone_number);
CREATE INDEX idx_sessions_phone ON user_sessions(phone_number);
CREATE INDEX idx_sessions_timestamp ON user_sessions(timestamp);
CREATE INDEX idx_module_usage_phone ON module_usage(phone_number);
CREATE INDEX idx_groups_region ON learning_groups(region);
```

Twilio Integration

Voice Call Processing:

```
@router.post("/webhook/call")
async def handle_voice_call(
    From: str = Form(...),
    To: str = Form(...),
    CallSid: str = Form(...),
    SpeechResult: Optional[str] = Form(None),
    RecordingUrl: Optional[str] = Form(None)
):
    # 1. Extract user context from Redis
    # 2. Process speech input (if voice)
    # 3. Route to appropriate learning module
    # 4. Generate AI response
    # 5. Create TwiML voice response
    # 6. Log interaction to database
```

TwiML Response Generation:

```python
def create_voice_response(self, message: str, gather_input: bool = True) -> str:
    response = VoiceResponse()

    if gather_input:
        gather = response.gather(
            input='speech',
            timeout=10,
            speech_timeout='auto',
            action='/webhook/voice/process',
            method='POST'
        )
        gather.say(message, voice='man', language='en-US')
        response.say("I didn't hear anything. Please try again.")
        response.redirect('/webhook/voice/process')
    else:
        response.say(message, voice='man', language='en-US')
        response.hangup()

    return str(response)
```

SMS Processing:

```python
@router.post("/webhook/sms")
async def handle_sms(
    From: str = Form(...),
    To: str = Form(...),
    Body: str = Form(...),
    MessageSid: str = Form(...)
):
    # 1. Extract text from SMS body
    # 2. Get user context from Redis
    # 3. Process through learning modules
    # 4. Generate AI response
    # 5. Create TwiML SMS response
    # 6. Log interaction and update context
```

🔧 Configuration Management

Environment Variables

```python
class Settings(BaseSettings):
    # Twilio Configuration
    twilio_account_sid: str
    twilio_auth_token: str
    twilio_phone_number: str

    # AI Service Configuration
    openai_api_key: str
    llama_api_key: str
    use_llama: bool = True
    deepgram_api_key: str
    newsapi_key: str

    # Infrastructure Configuration
    redis_url: str = "redis://localhost:6379/0"
    database_url: str

    # Application Configuration
    app_env: str = "development"
    debug: bool = True

    class Config:
        env_file = ".env"
```

Deployment Configuration

Production Settings:

```
# Fly.io deployment configuration
app: "bakame-mvp"
primary_region: "fra"

[build]
  builder = "paketobuildpacks/builder:base"

[env]
  PORT = "8000"
  PYTHONPATH = "/workspace"

[http_service]
  internal_port = 8000
  force_https = true
  auto_stop_machines = false
  auto_start_machines = true
  min_machines_running = 1

[[services]]
  protocol = "tcp"
  internal_port = 8000

  [[services.ports]]
    port = 80
    handlers = ["http"]

  [[services.ports]]
    port = 443
    handlers = ["tls", "http"]
```

📊 Performance Specifications

Response Time Requirements

Operation	Target Time	Maximum Time
Voice call initiation	< 1 second	2 seconds
Speech transcription	< 2 seconds	3 seconds
AI response generation	< 2 seconds	4 seconds
SMS response	< 1 second	2 seconds
Database queries	< 500ms	1 second
Redis operations	< 100ms	200ms

Throughput Requirements

Metric	Target	Peak Capacity
Concurrent voice calls	100	500
SMS messages/minute	1000	5000
API requests/second	100	500
Database connections	50	200
Redis connections	100	500

Resource Utilization

Resource	Normal Load	Peak Load
CPU utilization	< 50%	< 80%

Memory usage	< 1GB	< 2GB
Database connections	< 20	< 50
Network bandwidth	< 10 Mbps	< 50 Mbps

🔒 Security Specifications

Authentication & Authorization

Phone-Based Authentication: - No registration required for learners - Phone number as primary identifier - Session-based context management - Automatic session cleanup

Web Authentication:

```
# JWT-based authentication for admin users
class WebUser(Base):
    id: int
    email: str
    full_name: str
    hashed_password: str
    role: str  # 'user', 'admin', 'super_admin'
    organization: str
    is_active: bool
    created_at: datetime

# Role-based access control
@router.get("/admin/users")
async def get_users(current_user: WebUser = Depends(get_current_user)):
    if current_user.role not in ["admin", "super_admin"]:
        raise HTTPException(status_code=403, detail="Insufficient permissions")
```

Data Protection

Encryption Standards: - TLS 1.3 for all HTTP communications - AES-256 encryption for sensitive database fields - Secure API key storage using environment variables - Session data encryption in Redis

Privacy Compliance: - Data minimization principles - User consent mechanisms - Right to data export/deletion - Transparent data processing policies

API Security

Rate Limiting:

```
# Implement rate limiting for API endpoints
@app.middleware("http")
async def rate_limit_middleware(request: Request, call_next):
    # Implement sliding window rate limiting
    # 100 requests per minute per IP
    # 1000 requests per hour per user
```

Input Validation:

```
# Pydantic models for request validation
class UserSessionCreate(BaseModel):
    phone_number: str = Field(..., regex=r'^\+\d{10,15}$')
    module_name: str = Field(..., max_length=50)
    user_input: str = Field(..., max_length=1000)
```

📈 Monitoring & Analytics

Application Monitoring

Health Checks:

```
@app.get("/healthz")
async def health_check():
    return {
        "status": "healthy",
        "timestamp": datetime.utcnow(),
        "version": "1.0.0",
        "services": {
            "database": await check_database_health(),
            "redis": await check_redis_health(),
            "openai": await check_openai_health(),
            "twilio": await check_twilio_health()
        }
    }
```

Metrics Collection: - Request/response times - Error rates and types - User engagement metrics - Learning outcome tracking - Resource utilization

Logging Strategy:

```
# Structured logging with correlation IDs
import structlog

logger = structlog.get_logger()

async def log_interaction(
    phone_number: str,
    session_id: str,
    module_name: str,
    interaction_type: str,
    user_input: str,
    ai_response: str
):
    logger.info(
        "user_interaction",
        phone_number=phone_number,
        session_id=session_id,
        module_name=module_name,
        interaction_type=interaction_type,
        input_length=len(user_input),
        response_length=len(ai_response),
        timestamp=datetime.utcnow()
    )
```

Analytics Dashboard

Key Metrics: - Total users and sessions - Module usage statistics - Learning progress tracking - Geographic usage patterns - Engagement and retention rates

Real-time Analytics: - Active user count - Current session distribution - Response time monitoring - Error rate tracking

🚀 Deployment Specifications

Infrastructure Requirements

Production Environment: - **Platform:** Fly.io - **Runtime:** Python 3.12 + Poetry - **Database:** PostgreSQL 14+ - **Cache:** Redis 6+ - **CDN:** Fly.io edge locations

Scaling Configuration:

```
# fly.toml
[http_service]
  internal_port = 8000
  force_https = true
  auto_stop_machines = false
  auto_start_machines = true
  min_machines_running = 1
  max_machines_running = 10

[scaling]
  min_machines_running = 1
  max_machines_running = 10

[[services.concurrency]]
  type = "requests"
  soft_limit = 100
  hard_limit = 200
```

CI/CD Pipeline

Deployment Process: 1. Code commit to main branch 2. Automated testing (unit + integration) 3. Security scanning 4. Build Docker image 5. Deploy to staging environment 6. Run end-to-end tests 7. Deploy to production 8. Health check verification 9. Rollback capability

Testing Strategy:

```python
# Unit tests for learning modules
def test_english_module_grammar_correction():
    module = EnglishModule()
    result = await module.process_input(
        "I are going to school",
        {"phone_number": "+250781234567"}
    )
    assert "am" in result.lower()

# Integration tests for API endpoints
def test_voice_webhook_processing():
    response = client.post("/webhook/call", data={
        "From": "+250781234567",
        "To": "+250700000000",
        "CallSid": "test_call_sid",
        "SpeechResult": "I want to practice math"
    })
    assert response.status_code == 200
    assert "math" in response.text.lower()
```

🛍 Quality Assurance

Testing Requirements

Test Coverage Targets: - Unit tests: > 80% code coverage - Integration tests: All API endpoints - End-to-end tests: Critical user journeys - Performance tests: Load and stress testing

Test Categories: 1. **Unit Tests:** Individual component testing 2. **Integration Tests:** Service interaction testing 3. **API Tests:** Endpoint functionality testing 4. **Performance Tests:** Load and stress testing 5. **Security Tests:** Vulnerability scanning 6. **Accessibility Tests:** Voice interface testing

Code Quality Standards

Code Style: - PEP 8 compliance for Python code - Type hints for all functions -

Docstring documentation - Automated linting with flake8/black

Code Review Process: - All changes require peer review - Automated testing must pass - Security review for sensitive changes - Performance impact assessment

🔄 Maintenance & Support

Backup & Recovery

Data Backup Strategy: - Daily automated database backups - Redis snapshot backups - Configuration backup to version control - 30-day retention policy

Disaster Recovery: - RTO (Recovery Time Objective): 4 hours - RPO (Recovery Point Objective): 1 hour - Multi-region deployment capability - Automated failover procedures

Maintenance Procedures

Regular Maintenance: - Weekly security updates - Monthly dependency updates - Quarterly performance reviews - Annual architecture reviews

Monitoring & Alerting: - 24/7 system monitoring - Automated alert notifications - Escalation procedures - Performance threshold monitoring

---

Document Version: 1.0
Last Updated: September 6, 2025
Status: Production Ready Specifications
Next Review: December 6, 2025