🌱 Smart Soil & Crop Recommendation Platform

Complete Implementation Guide for Hackathon

Tech Stack:

- **Backend:** FastAPI (Python)
- **Frontend:** React
- **Database:** Supabase (PostgreSQL)
- **SMS:** Telerivet Platform
- **Deployment:** Railway (Backend) + Vercel (Frontend)

Timeline: 48 hours (Hackathon)

📋 Table of Contents

🚀 Quick Start: Get Your API Keys

Do this FIRST (takes 15 minutes):

1. Supabase (Database)

- Go to https://supabase.com
- Sign up → New Project
- Save: Database URL + Service Role Key

2. OpenWeather (Weather Data)

- Go to https://openweathermap.org/api
- Sign up → API Keys
- Copy your API key
- **Free: 1,000 calls/day** ✅

3. Google Gemini (AI Agronomist) - RECOMMENDED

- Go to https://makersuite.google.com/app/apikey
- Sign in with Google
- Create API Key
- **Free: 60 requests/min** ✅
- **No credit card needed!** ✅

4. Telerivet (SMS)

- Go to https://telerivet.com
- Sign up → Get API Key
- **Free tier: 50 messages/day** ✅

5. Railway (Backend Hosting)

- Go to https://railway.app
- Sign up with GitHub
- **Free: $5 credit/month** ✅

6. Vercel (Frontend Hosting)

- Go to https://vercel.com

- Sign up with GitHub
- **Free tier** ✅

Save all keys in a text file - you'll need them!

🏗️ System Architecture

```
┌─────────────────┐
│  IoT Sensor     │
│  (GSM Device)   │
└────────┬────────┘
         │ HTTP POST
         ▼
┌─────────────────────────────────────────┐
│      FastAPI Backend                    │
│  ┌──────────────────────────────────┐   │
│  │ /api/soil/upload                 │   │
│  │ /api/sms/receive (webhook)       │   │
│  │ /api/admin/*                     │   │
│  └──────────────────────────────────┘   │
│         │                    │          │
│         ▼                    │          │
│  ┌──────────────────┐        │          │
│  │ AI Engine        │        │          │
│  │ - Crop Suggest   │        │          │
│  │ - Fertilizer     │        │          │
│  └──────────────────┘        │          │
└───┬─────────────────────────┬───────────┘
    │                         │
    ▼                         ▼
┌──────────────┐   ┌──────────────────┐
│  Supabase    │   │  Telerivet       │
│  PostgreSQL  │   │  SMS Gateway     │
└──────────────┘   └────────┬─────────┘
                            │
                            ▼
                   ┌────────────────┐
                   │  Farmer Phone  │
                   │  (SMS only)    │
                   └────────────────┘
```

```
┌─────────────────────────────────────┐
│     React Admin Dashboard           │
│  - Farmer Management                │
│  - Device Registration              │
│  - Soil Test History                │
│  - SMS Logs                         │
└─────────────────────────────────────┘
```

✅ Prerequisites

Required Accounts (All FREE)

- [ ] GitHub account
- [ ] Supabase account
- [ ] Telerivet account
- [ ] Railway account (for backend)
- [ ] Vercel account (for frontend)

Required Software

- [ ] Python 3.9+
- [ ] Node.js 18+
- [ ] Git
- [ ] Code editor (VS Code recommended)

Optional (For Testing)

- [ ] Android phone with Telerivet app
- [ ] Ugandan SIM card with airtime

🚀 Phase 1: Setup & Configuration

1.1 Database Setup (Supabase)

Step 1: Create Supabase Project

1. Go to https://supabase.com
2. Click "New Project"
3. Name: `smart-soil-platform`

4. Database Password: (save this!)
5. Region: Choose closest to Uganda (EU or Singapore)

Step 2: Create Database Schema

Go to SQL Editor in Supabase and run:

```
sql
```

```sql
-- Farmers Table
CREATE TABLE farmers (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name VARCHAR(255) NOT NULL,
  phone_number VARCHAR(20) UNIQUE NOT NULL,
  region VARCHAR(100),
  district VARCHAR(100),
  pin VARCHAR(6) NOT NULL,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Devices Table
CREATE TABLE devices (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  device_id VARCHAR(100) UNIQUE NOT NULL,
  sim_number VARCHAR(20),
  farmer_id UUID REFERENCES farmers(id) ON DELETE CASCADE,
  api_token VARCHAR(255) UNIQUE NOT NULL,
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP DEFAULT NOW()
);

-- Soil Tests Table
CREATE TABLE soil_tests (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  device_id UUID REFERENCES devices(id),
  farmer_id UUID REFERENCES farmers(id),
  timestamp TIMESTAMP NOT NULL,
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8),
  ph DECIMAL(4, 2),
  moisture DECIMAL(5, 2),
  temperature DECIMAL(5, 2),
  ec DECIMAL(6, 2),
  nitrogen DECIMAL(6, 2),
  phosphorus DECIMAL(6, 2),
  potassium DECIMAL(6, 2),
  location_name VARCHAR(255),
  created_at TIMESTAMP DEFAULT NOW()
);

-- SMS Logs Table
```

```
CREATE TABLE sms_logs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  farmer_id UUID REFERENCES farmers(id),
  direction VARCHAR(10) NOT NULL, -- 'inbound' or 'outbound'
  phone_number VARCHAR(20) NOT NULL,
  message TEXT NOT NULL,
  status VARCHAR(50),
  telerivet_id VARCHAR(100),
  created_at TIMESTAMP DEFAULT NOW()
);

-- SMS Sessions Table (for conversation state)
CREATE TABLE sms_sessions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  farmer_id UUID REFERENCES farmers(id),
  soil_test_id UUID REFERENCES soil_tests(id),
  state VARCHAR(50) NOT NULL, -- 'awaiting_choice', 'awaiting_crop', 'completed'
  last_interaction TIMESTAMP DEFAULT NOW(),
  created_at TIMESTAMP DEFAULT NOW()
);

-- AI Recommendations Table
CREATE TABLE recommendations (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  soil_test_id UUID REFERENCES soil_tests(id),
  recommendation_type VARCHAR(50), -- 'crop_suggestion', 'crop_check', 'fertilizer'
  content TEXT NOT NULL,
  crops_suggested JSONB, -- Array of crops with scores
  created_at TIMESTAMP DEFAULT NOW()
);

-- Create indexes for performance
CREATE INDEX idx_farmers_phone ON farmers(phone_number);
CREATE INDEX idx_devices_device_id ON devices(device_id);
CREATE INDEX idx_soil_tests_farmer ON soil_tests(farmer_id);
CREATE INDEX idx_sms_logs_farmer ON sms_logs(farmer_id);
CREATE INDEX idx_sms_sessions_farmer ON sms_sessions(farmer_id);
```

Step 3: Get Database Credentials

1. Go to Project Settings → Database
2. Copy and save:
    - Database URL

- API URL
- Anon/Public Key
- Service Role Key (for backend)

1.2 Telerivet Setup

Step 1: Create Account

1. Go to https://telerivet.com
2. Sign up (free account)
3. Verify email

Step 2: Set Up Android Gateway (Optional for Hackathon)

- If testing with real SMS, install Telerivet app on Android phone
- If demoing, you can use Telerivet's test mode

Step 3: Get API Key

1. Go to Account → API Keys
2. Create new API key
3. Save the key securely

Step 4: Get Project ID

1. Go to your project
2. Copy Project ID from URL or settings

1.3 OpenWeather API Setup

Step 1: Create Account

1. Go to https://openweathermap.org/api
2. Sign up for free account
3. Verify email

Step 2: Get API Key

1. Go to API keys section
2. Copy your default API key
3. Free tier includes:
   - 1,000 API calls/day
   - Current weather data
   - 5-day forecast
   - Perfect for hackathon!

1.4 AI API Setup (Choose ONE)

Option A: Google Gemini (RECOMMENDED for hackathon)

✅ **FREE tier is generous** ✅ **60 requests per minute** ✅ **Easy to use**

Steps:

1. Go to https://makersuite.google.com/app/apikey
2. Click "Get API Key"
3. Create new API key
4. Copy and save it

Option B: OpenAI ChatGPT

⚠️ **Requires credit card** (even for free trial) ⚠️ **$5 minimum credit** ✅ More established, very good quality

Steps:

1. Go to https://platform.openai.com/signup
2. Create account
3. Add payment method
4. Go to API Keys
5. Create new secret key
6. Copy and save it

For Hackathon: Use Gemini (free, no card needed!)

💻 Phase 2: Backend Development

2.1 Project Setup

```bash
# Create project directory
mkdir smart-soil-backend
cd smart-soil-backend

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Create project structure
mkdir -p app/{api,models,services,core}
touch app/__init__.py
touch app/main.py
touch app/api/__init__.py
touch app/models/__init__.py
touch app/services/__init__.py
touch app/core/__init__.py

# Create requirements.txt
```

requirements.txt:

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
supabase==2.0.3
pydantic==2.5.0
pydantic-settings==2.1.0
httpx==0.25.1
python-dotenv==1.0.0
google-generativeai==0.3.2
openai==1.6.1
```

```bash
```

```
# Install dependencies
pip install -r requirements.txt
```

2.2 Environment Configuration

Create `.env` file:

```env
# Supabase
SUPABASE_URL=your_supabase_url
SUPABASE_KEY=your_supabase_service_key

# Telerivet
TELERIVET_API_KEY=your_telerivet_api_key
TELERIVET_PROJECT_ID=your_project_id

# Weather API
OPENWEATHER_API_KEY=your_openweather_api_key

# AI Agronomist (choose one)
GOOGLE_GEMINI_API_KEY=your_gemini_api_key
# OR
OPENAI_API_KEY=your_openai_api_key

# App Settings
API_SECRET_KEY=your_random_secret_key_here
BACKEND_URL=http://localhost:8000
FRONTEND_URL=http://localhost:3000
```

2.3 Core Configuration

app/core/config.py:

```python
```

```
from pydantic_settings import BaseSettings
from typing import Optional

class Settings(BaseSettings):
    # Supabase
    supabase_url: str
    supabase_key: str

    # Telerivet
    telerivet_api_key: str
    telerivet_project_id: str

    # Weather
    openweather_api_key: str

    # AI (at least one required)
    google_gemini_api_key: Optional[str] = None
    openai_api_key: Optional[str] = None

    # App
    api_secret_key: str
    backend_url: str
    frontend_url: str

    class Config:
        env_file = ".env"

settings = Settings()
```

app/core/database.py:

```python
from supabase import create_client, Client
from app.core.config import settings

def get_supabase_client() -> Client:
    return create_client(settings.supabase_url, settings.supabase_key)

supabase: Client = get_supabase_client()
```

2.4 Models (Pydantic)

app/models/schemas.py:

```
python
```

```
from pydantic import BaseModel, Field
from typing import Optional, List
from datetime import datetime
from uuid import UUID

class SoilDataUpload(BaseModel):
    device_id: str
    farmer_id: str
    phone_number: str
    timestamp: datetime
    gps_latitude: float
    gps_longitude: float
    sample_number: int
    sample_depth_cm: int
    soil_temperature_c: float
    soil_moisture_percent: float
    soil_nitrogen_mgkg: float
    soil_phosphorus_mgkg: float
    soil_potassium_mgkg: float
    soil_ph: float

class DeviceAuth(BaseModel):
    api_token: str

class FarmerCreate(BaseModel):
    name: str
    phone_number: str
    region: str
    district: str

class DeviceCreate(BaseModel):
    device_id: str
    sim_number: str
    farmer_id: str

class SMSWebhook(BaseModel):
    id: str
    from_number: str
    content: str
    time_created: int

class CropRecommendation(BaseModel):
    crop_name: str
    suitability_score: float
    reason: str

class RecommendationResponse(BaseModel):
    recommendations: List[CropRecommendation]
    summary: str
```

2.5 Weather Service

app/services/weather_service.py:

```
python
```

```python
import httpx
from app.core.config import settings
from typing import Dict, Optional
from datetime import datetime, timedelta

class WeatherService:
    """OpenWeather API integration"""

    def __init__(self):
        self.api_key = settings.openweather_api_key
        self.base_url = "https://api.openweathermap.org/data/2.5"

    async def get_weather_data(self, latitude: float, longitude: float) -> Dict:
        """Get current weather and forecast for location"""

        try:
            async with httpx.AsyncClient() as client:
                # Current weather
                current_url = f"{self.base_url}/weather"
                current_params = {
                    "lat": latitude,
                    "lon": longitude,
                    "appid": self.api_key,
                    "units": "metric"
                }
                current_response = await client.get(current_url, params=current_params)
                current_data = current_response.json()

                # 5-day forecast (free tier)
                forecast_url = f"{self.base_url}/forecast"
                forecast_params = {
                    "lat": latitude,
                    "lon": longitude,
                    "appid": self.api_key,
                    "units": "metric"
                }
                forecast_response = await client.get(forecast_url, params=forecast_params)
                forecast_data = forecast_response.json()

                # Process forecast data
                forecast_summary = self._process_forecast(forecast_data)

                return {
                    "location": current_data.get("name", "Unknown"),
                    "current": {
                        "temperature": current_data["main"]["temp"],
                        "humidity": current_data["main"]["humidity"],
                        "pressure": current_data["main"]["pressure"],
                        "description": current_data["weather"][0]["description"],
                        "rainfall_1h": current_data.get("rain", {}).get("1h", 0)
                    },
                    "forecast": forecast_summary
                }
        except Exception as e:
            print(f"Weather API error: {e}")
            # Return default/fallback data
            return {
                "location": "Unknown",
                "current": {
                    "temperature": 25,
                    "humidity": 60,
                    "description": "Data unavailable",
                    "rainfall_1h": 0
                },
                "forecast": {
                    "avg_temperature": 25,
                    "total_rainfall_mm": 0,
                    "rainy_days": 0,
                    "summary": "Weather data unavailable"
                }
            }

    def _process_forecast(self, forecast_data: Dict) -> Dict:
        """Process 5-day forecast into useful summary"""

        if "list" not in forecast_data:
            return {
                "avg_temperature": 0,
                "total_rainfall_mm": 0,
                "rainy_days": 0,
                "summary": "No forecast available"
            }

        temps = []
        total_rain = 0
        rainy_days = set()
        for item in forecast_data["list"]:
            temps.append(item["main"]["temp"])

            # Check for rainfall
            rain = item.get("rain", {}).get("3h", 0)
            if rain > 0:
                total_rain += rain
                # Get date for this forecast
                date = datetime.fromtimestamp(item["dt"]).date()
                rainy_days.add(date)

        avg_temp = sum(temps) / len(temps) if temps else 0

        # Generate summary
        if total_rain > 100:
            rain_summary = "Heavy rainfall expected"
        elif total_rain > 50:
            rain_summary = "Moderate rainfall expected"
        elif total_rain > 10:
            rain_summary = "Light rainfall expected"
        else:
            rain_summary = "Little to no rainfall expected"

        return {
            "avg_temperature": round(avg_temp, 1),
            "total_rainfall_mm": round(total_rain, 1),
            "rainy_days": len(rainy_days),
            "summary": f"{rain_summary} over next 5 days. Avg temp: {avg_temp:.1f}°C"
        }

weather_service = WeatherService()
```

2.6 AI Agronomist Service

app/services/ai_agronomist.py:

```python
```

```python
from typing import Dict, List, Optional
from app.core.config import settings
import google.generativeai as genai
from openai import OpenAI

class AIAgronomist:
    """AI-powered agronomist using Gemini or ChatGPT"""

    def __init__(self):
        self.use_gemini = settings.google_gemini_api_key is not None

        if self.use_gemini:
            genai.configure(api_key=settings.google_gemini_api_key)
            self.model = genai.GenerativeModel('gemini-pro')
        elif settings.openai_api_key:
            self.client = OpenAI(api_key=settings.openai_api_key)
        else:
            raise ValueError("Either GOOGLE_GEMINI_API_KEY or OPENAI_API_KEY must be set")

    def _create_system_prompt(self) -> str:
        """Base system prompt for AI agronomist"""
        return """You are an expert agricultural advisor specializing in East African farming,
particularly Uganda. You provide practical, actionable advice to smallholder farmers based on
soil test data and weather conditions.

Your responses must be:
1. SHORT and SMS-friendly (max 160 characters per message)
2. Written in simple English that farmers understand
3. Practical and immediately actionable
4. Focused on crops suitable for Uganda
5. Consider local context (budget, resources, climate)

Common crops in Uganda: Maize, Beans, Coffee, Cassava, Bananas, Tomatoes, Sweet Potatoes,
Groundnuts, Sorghum, Millet.

When giving advice, consider:
- Soil pH, moisture, nutrients (N, P, K)
- Current weather and forecast
- Seasonal timing
- Local availability of inputs
- Budget constraints of smallholder farmers
"""
```

```
  async def get_crop_recommendations(
    self,
    soil_data: Dict,
    weather_data: Dict
  ) -> str:
    """Get top 3 crop recommendations with brief reasoning"""

    prompt = f"""Based on this data, recommend the TOP 3 crops for this farmer:

SOIL DATA:
- pH: {soil_data['ph']}
- Moisture: {soil_data['moisture']}%
- Temperature: {soil_data['temperature']}°C
- Nitrogen: {soil_data['nitrogen']} mg/kg
- Phosphorus: {soil_data['phosphorus']} mg/kg
- Potassium: {soil_data['potassium']} mg/kg

WEATHER (Next 5 days):
- {weather_data['forecast']['summary']}
- Average temp: {weather_data['forecast']['avg_temperature']}°C
- Total rainfall: {weather_data['forecast']['total_rainfall_mm']}mm
- Location: {weather_data['location']}

Respond in this EXACT format (max 160 chars total):
1. CROP1 (score/100): reason
2. CROP2 (score/100): reason
3. CROP3 (score/100): reason

Keep each line under 50 characters!"""

    return await self._generate(prompt)

  async def check_specific_crop(
    self,
    crop_name: str,
    soil_data: Dict,
    weather_data: Dict
  ) -> str:
    """Check if specific crop is suitable and give advice"""

    prompt = f"""A farmer wants to grow {crop_name.upper()}. Analyze if it's suitable:

SOIL DATA:
- pH: {soil_data['ph']}
- Moisture: {soil_data['moisture']}%
- Temperature: {soil_data['temperature']}°C
- Nitrogen: {soil_data['nitrogen']} mg/kg
- Phosphorus: {soil_data['phosphorus']} mg/kg
- Potassium: {soil_data['potassium']} mg/kg

WEATHER:
- {weather_data['forecast']['summary']}

Respond in this format (max 2 SMS = 320 chars total):
✓/✗ {crop_name} is SUITABLE/NOT SUITABLE (score/100)

ADVICE:
- Point 1
- Point 2
- Point 3

Be specific about fertilizers, lime, irrigation needs. Use simple language!"""

        return await self._generate(prompt)

    async def get_fertilizer_advice(
        self,
        soil_data: Dict,
        target_crop: Optional[str] = None
    ) -> str:
        """Get specific fertilizer/soil treatment recommendations"""

        crop_context = f"for growing {target_crop}" if target_crop else "generally"

        prompt = f"""Give fertilizer recommendations {crop_context}:

SOIL DATA:
- pH: {soil_data['ph']}
- Nitrogen: {soil_data['nitrogen']} mg/kg
- Phosphorus: {soil_data['phosphorus']} mg/kg
- Potassium: {soil_data['potassium']} mg/kg

Respond in max 160 characters:
FERTILIZER NEEDED:
- [specific product or NPK ratio]
- [quantity per acre]
- [application method]
```

```
Be specific! Use locally available products in Uganda!"""

        return await self._generate(prompt)

    async def _generate(self, prompt: str) -> str:
        """Generate response using Gemini or ChatGPT"""

        try:
            if self.use_gemini:
                # Gemini
                full_prompt = self._create_system_prompt() + "\n\n" + prompt
                response = self.model.generate_content(full_prompt)
                return response.text.strip()
            else:
                # ChatGPT
                response = self.client.chat.completions.create(
                    model="gpt-3.5-turbo",  # Use gpt-4 if available
                    messages=[
                        {"role": "system", "content": self._create_system_prompt()},
                        {"role": "user", "content": prompt}
                    ],
                    max_tokens=200,
                    temperature=0.7
                )
                return response.choices[0].message.content.strip()
        except Exception as e:
            print(f"AI generation error: {e}")
            return "AI service temporarily unavailable. Please try again later."

ai_agronomist = AIAgronomist()
```

2.7 SMS Service

app/services/sms_service.py:

```
python
```

```python
import httpx
from app.core.config import settings
from app.core.database import supabase
from typing import Optional

class TelerivetSMSService:
    def __init__(self):
        self.api_key = settings.telerivet_api_key
        self.project_id = settings.telerivet_project_id
        self.base_url = "https://api.telerivet.com/v1"

    async def send_sms(self, phone_number: str, message: str, farmer_id: Optional[str] = None) -> dict:
        """Send SMS via Telerivet"""
        url = f"{self.base_url}/projects/{self.project_id}/messages/send"

        # Ensure phone number has country code
        if not phone_number.startswith("+"):
            phone_number = f"+256{phone_number.lstrip('0')}"  # Uganda code

        # Split long messages (SMS is 160 chars)
        messages = self._split_message(message)
        results = []

        for msg in messages:
            payload = {
                "content": msg,
                "to_number": phone_number
            }

            async with httpx.AsyncClient() as client:
                response = await client.post(
                    url,
                    json=payload,
                    auth=(self.api_key, "")
                )

            result = response.json()
            results.append(result)

            # Log to database
            supabase.table("sms_logs").insert({
                "farmer_id": farmer_id,
                "direction": "outbound",
                "phone_number": phone_number,
                "message": msg,
                "status": result.get("status"),
                "telerivet_id": result.get("id")
            }).execute()

        return results[0] if results else {}

    def _split_message(self, message: str, max_length: int = 160) -> list:
        """Split long messages into SMS-sized chunks"""
        if len(message) <= max_length:
            return [message]

        # Split by paragraphs first
        parts = message.split('\n\n')
        messages = []
        current = ""

        for part in parts:
            if len(current) + len(part) + 2 <= max_length:
                current += part + "\n\n"
            else:
                if current:
                    messages.append(current.strip())
                current = part + "\n\n"

        if current:
            messages.append(current.strip())

        return messages

    def generate_initial_sms(self, farmer_name: str, pin: str, location: str) -> str:
        """Generate initial SMS after soil test"""
        return f"""Hello {farmer_name}! Soil test for {location} is ready.

Reply:
1 - AI crop suggestions
2 - Check your crop
3 - Fertilizer advice

PIN: {pin}"""

sms_service = TelerivetSMSService()
```

2.8 API Endpoints

app/api/soil.py:

```
python
```

```python
from fastapi import APIRouter, HTTPException, Header
from app.models.schemas import SoilDataUpload
from app.core.database import supabase
from app.services.weather_service import weather_service
from app.services.ai_agronomist import ai_agronomist
from app.services.sms_service import sms_service

router = APIRouter()

@router.post("/upload")
async def upload_soil_data(
    data: SoilDataUpload,
    authorization: str = Header(None)
):
    """Receive soil data from IoT device"""

    # Verify device token
    if not authorization or not authorization.startswith("Bearer "):
        raise HTTPException(status_code=401, detail="Missing or invalid token")

    token = authorization.split(" ")[1]

    # Verify device exists
    device_result = supabase.table("devices")\
        .select("*, farmers(*)")\
        .eq("api_token", token)\
        .eq("is_active", True)\
        .execute()

    if not device_result.data:
        raise HTTPException(status_code=401, detail="Invalid device token")

    device = device_result.data[0]
    farmer = device.get("farmers")

    if not farmer:
        raise HTTPException(status_code=404, detail="Farmer not found for this device")

    # Get weather data for location
    weather_data = await weather_service.get_weather_data(
        data.gps_latitude,
        data.gps_longitude
    )
```

```python
# Insert soil test data
soil_test = supabase.table("soil_tests").insert({
    "device_id": device["id"],
    "farmer_id": farmer["id"],
    "timestamp": data.timestamp.isoformat(),
    "latitude": data.gps_latitude,
    "longitude": data.gps_longitude,
    "ph": data.soil_ph,
    "moisture": data.soil_moisture_percent,
    "temperature": data.soil_temperature_c,
    "ec": 0,  # Not in new data structure
    "nitrogen": data.soil_nitrogen_mgkg,
    "phosphorus": data.soil_phosphorus_mgkg,
    "potassium": data.soil_potassium_mgkg,
    "location_name": weather_data["location"]
}).execute()

soil_test_id = soil_test.data[0]["id"]

# Prepare data for AI
soil_data_dict = {
    "ph": data.soil_ph,
    "moisture": data.soil_moisture_percent,
    "temperature": data.soil_temperature_c,
    "nitrogen": data.soil_nitrogen_mgkg,
    "phosphorus": data.soil_phosphorus_mgkg,
    "potassium": data.soil_potassium_mgkg
}

# Generate AI recommendations (don't wait, do in background)
try:
    recommendations_text = await ai_agronomist.get_crop_recommendations(
        soil_data_dict,
        weather_data
    )

    # Store recommendations
    supabase.table("recommendations").insert({
        "soil_test_id": soil_test_id,
        "recommendation_type": "crop_suggestion",
        "content": recommendations_text,
        "crops_suggested": {"ai_response": recommendations_text}
    }).execute()
    except Exception as e:
        print(f"AI recommendation error: {e}")
        recommendations_text = "AI recommendations temporarily unavailable"

    # Create SMS session
    supabase.table("sms_sessions").insert({
        "farmer_id": farmer["id"],
        "soil_test_id": soil_test_id,
        "state": "awaiting_choice"
    }).execute()

    # Send initial SMS
    sms_message = sms_service.generate_initial_sms(
        farmer["name"],
        farmer["pin"],
        weather_data["location"]
    )

    await sms_service.send_sms(
        data.phone_number,  # Use phone from device data
        sms_message,
        farmer["id"]
    )

    return {
        "status": "success",
        "soil_test_id": soil_test_id,
        "location": weather_data["location"],
        "weather_summary": weather_data["forecast"]["summary"],
        "message": "Data received, weather fetched, AI analyzed, SMS sent to farmer"
    }
```

app/api/sms.py:

```python
```

```
from fastapi import APIRouter, HTTPException, Request
from app.core.database import supabase
from app.services.weather_service import weather_service
from app.services.ai_agronomist import ai_agronomist
from app.services.sms_service import sms_service

router = APIRouter()

@router.post("/receive")
async def receive_sms(request: Request):
    """Webhook to receive SMS from Telerivet"""

    payload = await request.json()

    from_number = payload.get("from_number", "")
    content = payload.get("content", "").strip()

    # Find farmer by phone
    farmer_result = supabase.table("farmers")\
        .select("*")\
        .eq("phone_number", from_number)\
        .execute()

    if not farmer_result.data:
        # Unknown number
        await sms_service.send_sms(
            from_number,
            "Phone number not registered. Contact B&J Agrotech support."
        )
        return {"status": "error", "message": "Unknown number"}

    farmer = farmer_result.data[0]

    # Log incoming SMS
    supabase.table("sms_logs").insert({
        "farmer_id": farmer["id"],
        "direction": "inbound",
        "phone_number": from_number,
        "message": content,
        "status": "received"
    }).execute()

    # Get active session
```

```
session_result = supabase.table("sms_sessions")\
    .select("*, soil_tests(*)")\
    .eq("farmer_id", farmer["id"])\
    .order("created_at", desc=True)\
    .limit(1)\
    .execute()

if not session_result.data:
    await sms_service.send_sms(
        from_number,
        "No active session. Please run a soil test first."
    )
    return {"status": "no_session"}

session = session_result.data[0]
soil_test = session["soil_tests"]

# Prepare soil data
soil_data = {
    "ph": soil_test["ph"],
    "moisture": soil_test["moisture"],
    "temperature": soil_test["temperature"],
    "nitrogen": soil_test["nitrogen"],
    "phosphorus": soil_test["phosphorus"],
    "potassium": soil_test["potassium"]
}

# Get weather data
weather_data = await weather_service.get_weather_data(
    soil_test["latitude"],
    soil_test["longitude"]
)

# Handle user response
response_message = ""
content_upper = content.upper()

if content_upper in ["1", "ONE"]:
    # AI crop suggestions
    response_message = await ai_agronomist.get_crop_recommendations(
        soil_data,
        weather_data
    )
```

```
        # Update session
        supabase.table("sms_sessions")\
            .update({"state": "completed"})\
            .eq("id", session["id"])\
            .execute()

    elif content_upper in ["2", "TWO"]:
        # Ask for crop name
        response_message = "Which crop? Reply: MAIZE, BEANS, COFFEE, CASSAVA, BANANAS, TOMATOES, etc."
        supabase.table("sms_sessions")\
            .update({"state": "awaiting_crop"})\
            .eq("id", session["id"])\
            .execute()

    elif content_upper in ["3", "THREE"]:
        # Fertilizer advice
        response_message = await ai_agronomist.get_fertilizer_advice(soil_data)

        supabase.table("sms_sessions")\
            .update({"state": "completed"})\
            .eq("id", session["id"])\
            .execute()

    elif session["state"] == "awaiting_crop":
        # User sent crop name - check it
        response_message = await ai_agronomist.check_specific_crop(
            content_upper,
            soil_data,
            weather_data
        )

        supabase.table("sms_sessions")\
            .update({"state": "completed"})\
            .eq("id", session["id"])\
            .execute()

    else:
        response_message = "Invalid option. Reply:\n1-Crop suggestions\n2-Check your crop\n3-Fertilizer advice"

    # Send response
    if response_message:
        await sms_service.send_sms(from_number, response_message, farmer["id"])
    return {"status": "success", "action": "processed"}
```

app/api/admin.py:

```
python
```

```python
from fastapi import APIRouter, HTTPException
from app.models.schemas import FarmerCreate, DeviceCreate
from app.core.database import supabase
import secrets

router = APIRouter()

@router.post("/farmers")
async def create_farmer(farmer: FarmerCreate):
    """Create new farmer account"""

    # Generate random PIN
    pin = str(secrets.randbelow(900000) + 100000)  # 6 digits

    result = supabase.table("farmers").insert({
        "name": farmer.name,
        "phone_number": farmer.phone_number,
        "region": farmer.region,
        "district": farmer.district,
        "pin": pin
    }).execute()

    return {"status": "success", "farmer": result.data[0]}

@router.get("/farmers")
async def list_farmers():
    """List all farmers"""
    result = supabase.table("farmers").select("*").execute()
    return {"farmers": result.data}

@router.post("/devices")
async def register_device(device: DeviceCreate):
    """Register new device"""

    # Generate API token
    api_token = secrets.token_urlsafe(32)

    result = supabase.table("devices").insert({
        "device_id": device.device_id,
        "sim_number": device.sim_number,
        "farmer_id": device.farmer_id,
        "api_token": api_token
    }).execute()
    return {"status": "success", "device": result.data[0], "api_token": api_token}

@router.get("/soil-tests/{farmer_id}")
async def get_farmer_tests(farmer_id: str):
    """Get all soil tests for a farmer"""
    result = supabase.table("soil_tests")\
        .select("*, recommendations(*)")\
        .eq("farmer_id", farmer_id)\
        .order("created_at", desc=True)\
        .execute()

    return {"tests": result.data}

@router.get("/sms-logs/{farmer_id}")
async def get_sms_logs(farmer_id: str):
    """Get SMS conversation history"""
    result = supabase.table("sms_logs")\
        .select("*")\
        .eq("farmer_id", farmer_id)\
        .order("created_at", desc=True)\
        .execute()

    return {"logs": result.data}
```

2.8 Main Application

app/main.py:

```
python
```

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.api import soil, sms, admin

app = FastAPI(title="Smart Soil Platform API")

# CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # In production, specify frontend URL
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include routers
app.include_router(soil.router, prefix="/api/soil", tags=["soil"])
app.include_router(sms.router, prefix="/api/sms", tags=["sms"])
app.include_router(admin.router, prefix="/api/admin", tags=["admin"])

@app.get("/")
async def root():
    return {"message": "Smart Soil Platform API", "status": "running"}

@app.get("/health")
async def health():
    return {"status": "healthy"}
```

🎨 Phase 3: Frontend Development

3.1 Setup React Project

bash

```
# Create React app
npx create-react-app smart-soil-dashboard
cd smart-soil-dashboard

# Install dependencies
npm install axios @supabase/supabase-js react-router-dom recharts
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

Configure Tailwind (tailwind.config.js):

```javascript
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

3.2 Environment Setup

Create `.env` file:

```
REACT_APP_API_URL=http://localhost:8000
REACT_APP_SUPABASE_URL=your_supabase_url
REACT_APP_SUPABASE_ANON_KEY=your_supabase_anon_key
```

3.3 Key Components (Simplified)

src/App.js:

```javascript
```

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const API_URL = process.env.REACT_APP_API_URL;

function App() {
  const [farmers, setFarmers] = useState([]);
  const [selectedFarmer, setSelectedFarmer] = useState(null);
  const [tests, setTests] = useState([]);
  const [smsLogs, setSmsLogs] = useState([]);

  useEffect(() => {
    fetchFarmers();
  }, []);

  const fetchFarmers = async () => {
    const res = await axios.get(`${API_URL}/api/admin/farmers`);
    setFarmers(res.data.farmers);
  };

  const fetchFarmerData = async (farmerId) => {
    const [testsRes, smsRes] = await Promise.all([
      axios.get(`${API_URL}/api/admin/soil-tests/${farmerId}`),
      axios.get(`${API_URL}/api/admin/sms-logs/${farmerId}`)
    ]);
    setTests(testsRes.data.tests);
    setSmsLogs(smsRes.data.logs);
  };

  return (
    <div className="min-h-screen bg-gray-100 p-8">
      <h1 className="text-3xl font-bold mb-8">🌱 Smart Soil Admin Dashboard</h1>

      <div className="grid grid-cols-3 gap-6">
        {/* Farmers List */}
        <div className="bg-white p-6 rounded-lg shadow">
          <h2 className="text-xl font-bold mb-4">Farmers</h2>
          {farmers.map(farmer => (
            <div
              key={farmer.id}
              className="p-3 mb-2 border rounded cursor-pointer hover:bg-blue-50"
              onClick={() => {
                setSelectedFarmer(farmer);
                fetchFarmerData(farmer.id);
              }}
            >
              <div className="font-semibold">{farmer.name}</div>
              <div className="text-sm text-gray-600">{farmer.phone_number}</div>
            </div>
          ))}
        </div>

        {/* Soil Tests */}
        <div className="bg-white p-6 rounded-lg shadow">
          <h2 className="text-xl font-bold mb-4">Soil Tests</h2>
          {selectedFarmer && tests.map(test => (
            <div key={test.id} className="p-3 mb-2 border rounded">
              <div className="text-sm text-gray-500">
                {new Date(test.created_at).toLocaleDateString()}
              </div>
              <div className="grid grid-cols-2 gap-2 mt-2 text-sm">
                <div>pH: {test.ph}</div>
                <div>Moisture: {test.moisture}%</div>
                <div>Temp: {test.temperature}°C</div>
                <div>N: {test.nitrogen}</div>
              </div>
            </div>
          ))}
        </div>

        {/* SMS Logs */}
        <div className="bg-white p-6 rounded-lg shadow">
          <h2 className="text-xl font-bold mb-4">SMS Conversation</h2>
          {selectedFarmer && smsLogs.map(log => (
            <div
              key={log.id}
              className={`p-3 mb-2 rounded ${
                log.direction === 'outbound'
                  ? 'bg-blue-100 ml-4'
                  : 'bg-gray-100 mr-4'
              }`}
            >
              <div className="text-xs text-gray-500 mb-1">
                {log.direction === 'outbound' ? 'Sent' : 'Received'}
              </div>
              <div className="text-sm">{log.message}</div>
            </div>
```

```
        ))}
      </div>
    </div>
  </div>
 );
}

export default App;
```

📱 Phase 4: SMS Integration

4.1 Configure Telerivet Webhook

1. Log into Telerivet dashboard
2. Go to your project
3. Navigate to **Services** → **Webhooks**
4. Create new webhook:
   - **URL:** `https://your-backend-url.com/api/sms/receive`
   - **Event:** Message Received
   - **Method:** POST

🚀 Phase 5: Deployment

5.1 Backend Deployment (Railway)

Why Railway over Vercel:

- Vercel is serverless (cold starts, not ideal for FastAPI)
- Railway supports long-running Python apps
- Free tier: $5 credit/month
- Easier webhook handling

Steps:

1. **Push to GitHub:**

```bash
git init
git add .
git commit -m "Initial commit"
git remote add origin your-repo-url
git push -u origin main
```

2. **Deploy on Railway:**
   - Go to https://railway.app
   - Sign up with GitHub
   - Click "New Project" → "Deploy from GitHub repo"
   - Select your repository
   - Railway auto-detects Python
   - Add environment variables in Settings
   - Deploy!
3. **Get URL:**
   - Railway provides: `https://your-app.railway.app`
   - Update Telerivet webhook URL

5.2 Frontend Deployment (Vercel)

Steps:

1. **Update API URL in `.env`:**

```
REACT_APP_API_URL=https://your-app.railway.app
```

2. **Deploy to Vercel:**

```bash
npm install -g vercel
vercel
```

3. **Follow prompts:**
    - Link to existing project or create new
    - Vercel auto-deploys from Git

🧪 Testing Guide

Test 1: Create Farmer

bash

```bash
curl -X POST https://your-backend.railway.app/api/admin/farmers \
  -H "Content-Type: application/json" \
  -d '{
    "name": "John Farmer",
    "phone_number": "+256700123456",
    "region": "Central",
    "district": "Kampala"
  }'
```

Test 2: Register Device

bash

```bash
curl -X POST https://your-backend.railway.app/api/admin/devices \
  -H "Content-Type: application/json" \
  -d '{
    "device_id": "SENSOR001",
    "sim_number": "+256700111222",
    "farmer_id": "farmer-uuid-here"
  }'
```

Test 3: Simulate Soil Data Upload

bash

```
curl -X POST https://your-backend.railway.app/api/soil/upload \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer your-device-token" \
  -d '{
    "device_id": "BJ-DEVICE-002",
    "farmer_id": "FARMER-014",
    "phone_number": "+256712345678",
    "timestamp": "2026-01-10T11:18:30Z",
    "gps_latitude": 0.3476,
    "gps_longitude": 32.5825,
    "sample_number": 1,
    "sample_depth_cm": 15,
    "soil_temperature_c": 26.4,
    "soil_moisture_percent": 23.1,
    "soil_nitrogen_mgkg": 145,
    "soil_phosphorus_mgkg": 12,
    "soil_potassium_mgkg": 210,
    "soil_ph": 5.8
  }'
```

Expected Response:

```json
{
  "status": "success",
  "soil_test_id": "uuid-here",
  "location": "Kampala",
  "weather_summary": "Moderate rainfall expected over next 5 days. Avg temp: 25.3°C",
  "message": "Data received, weather fetched, AI analyzed, SMS sent to farmer"
}
```

What happens:

1. ✅ Data saved to database
2. ✅ Weather fetched from OpenWeather
3. ✅ AI generates crop recommendations
4. ✅ SMS sent to farmer's phone
5. ✅ Farmer can now reply with 1, 2, or 3

Test 4: Test AI Recommendations Manually

You can test the AI without the full flow:

Create test script `test_ai.py`:

```python
import asyncio
from app.services.ai_agronomist import ai_agronomist
from app.services.weather_service import weather_service

async def test():
    # Sample soil data
    soil_data = {
        "ph": 5.8,
        "moisture": 23.1,
        "temperature": 26.4,
        "nitrogen": 145,
        "phosphorus": 12,
        "potassium": 210
    }

    # Get weather for Kampala
    weather = await weather_service.get_weather_data(0.3476, 32.5825)

    # Test crop recommendations
    print("=== CROP RECOMMENDATIONS ===")
    recommendations = await ai_agronomist.get_crop_recommendations(soil_data, weather)
    print(recommendations)

    # Test specific crop check
    print("\n=== CHECK MAIZE ===")
    maize_check = await ai_agronomist.check_specific_crop("MAIZE", soil_data, weather)
    print(maize_check)

    # Test fertilizer advice
    print("\n=== FERTILIZER ADVICE ===")
    fertilizer = await ai_agronomist.get_fertilizer_advice(soil_data)
    print(fertilizer)

asyncio.run(test())
```

Run it:

```bash
python test_ai.py
```

🐛 Troubleshooting

Issue: SMS not sending

- Check Telerivet API key in .env
- Verify phone number format (+256...)
- Check Telerivet dashboard for errors
- Verify you have airtime/credits

Issue: Database connection failed

- Verify Supabase URL and key in .env
- Check if IP is whitelisted (Supabase allows all by default)
- Run the SQL schema again

Issue: Webhook not receiving SMS

- Ensure backend URL is publicly accessible (use Railway URL)
- Check Telerivet webhook configuration
- Verify webhook URL is HTTPS (Railway provides this)
- Check Railway logs for errors

Issue: Weather API failing

- Check OpenWeather API key is valid
- Verify you haven't exceeded 1,000 calls/day
- Check GPS coordinates are valid (latitude/longitude)
- OpenWeather free tier has slight delay - be patient

Issue: AI not responding

- **Gemini**: Check API key, verify 60 req/min limit not exceeded
- **ChatGPT**: Check you have credits, API key is valid
- Check Railway logs for specific error messages
- Test AI service locally first with `test_ai.py`
- Gemini sometimes has rate limits - wait 1 minute and retry

Issue: "AI responses too long for SMS"

- AI is instructed to keep responses under 160 chars
- If responses are still long, they'll be split automatically
- Check `sms_service._split_message()` is working
- You can adjust max_tokens in AI calls

Issue: Railway deployment failing

- Check all environment variables are set in Railway dashboard
- Verify requirements.txt has all dependencies
- Check Railway logs for specific error
- Make sure you're using Python 3.9+

Issue: Sensor data not matching format

- Ensure JSON exactly matches the schema in `SoilDataUpload`
- All fields must be present
- Phone number must include country code
- Timestamp must be ISO format

💰 Cost Estimation

For Hackathon (48 hours):

- Total cost: **$0** (all free tiers!)

For MVP (1 month, 100 farmers, 10 tests each = 1,000 tests):

Service	Usage	Cost
Supabase	1,000 rows	**$0** (free tier)
OpenWeather	1,000 API calls	**$0** (free tier)
Gemini AI	~3,000 requests	**$0** (free tier)
Telerivet	1,000 SMS	~$15-30 (airtime only)
Railway	Backend hosting	**$0-5** (free tier)
Vercel	Frontend hosting	**$0** (free tier)
TOTAL		**$15-35/month**

Scaling (1,000 farmers, 100,000 SMS/month):

- Supabase: $25/month (Pro plan)
- OpenWeather: $40/month (Startup plan)
- Gemini: Still free (very generous limits)
- Telerivet: ~$1,500-3,000 (airtime)
- Railway: $20/month
- Vercel: Free
- **TOTAL: ~$1,600-3,100/month**

Most cost is SMS airtime - this is unavoidable for any SMS solution!

⏱️ Hackathon Timeline (48 Hours)

Hour 0-4: Setup & API Keys

- Create accounts (Supabase, Telerivet, Railway, Vercel)
- **Get OpenWeather API key** (5 min)
- **Get Gemini API key** (5 min)
- Set up database schema

- Initialize projects
- Test API keys work

Hour 4-14: Backend Core

- Build FastAPI endpoints
- Implement Weather Service
- **Implement AI Agronomist Service** (most important!)
- Test AI responses locally
- Test API locally

Hour 14-22: SMS Integration

- Set up Telerivet
- Implement SMS send/receive
- **Test full flow: Sensor → Weather → AI → SMS**
- Implement conversation logic

Hour 22-32: Frontend

- Build React dashboard
- Connect to API
- Display soil tests, weather, AI responses
- Test admin features

Hour 32-42: Deployment

- Deploy backend to Railway
- Add all environment variables (including API keys!)
- Deploy frontend to Vercel
- Configure Telerivet webhooks
- End-to-end testing with real SMS

Hour 42-48: Demo Prep & Testing

- Create demo script

- Test with real device data
- Prepare presentation slides
- **Test AI responses are good quality**
- Buffer for bugs!
- Practice demo flow

🎯 MVP Features Checklist

- [ ] Farmer registration
- [ ] Device registration
- [ ] Receive soil data from device (new JSON format)
- [ ] **Fetch weather data from OpenWeather**
- [ ] **AI crop recommendation via Gemini/ChatGPT**
- [ ] **AI specific crop analysis**
- [ ] **AI fertilizer advice**
- [ ] Send SMS to farmer
- [ ] Receive SMS from farmer
- [ ] Two-way SMS conversation with AI responses
- [ ] Admin dashboard (view farmers, tests, SMS, weather)
- [ ] Backend deployed
- [ ] Frontend deployed
- [ ] **End-to-end flow: Sensor → Weather → AI → SMS works!**

📚 Additional Resources

- FastAPI Docs: https://fastapi.tiangolo.com
- Supabase Docs: https://supabase.com/docs
- Telerivet API: https://telerivet.com/api
- Railway Docs: https://docs.railway.app
- OpenWeather API: https://openweathermap.org/api
- Google Gemini: https://ai.google.dev/docs
- OpenAI API: https://platform.openai.com/docs

🎨 Pro Tips: Improving AI Responses

The AI agronomist is the **star of your demo**. Here's how to make it amazing:

1. Prompt Engineering Tips

Current prompts work well, but you can improve them:

For better crop recommendations:

```python
prompt = f"""You're advising a Ugandan smallholder farmer with LIMITED budget.

SOIL: pH {ph}, Moisture {moisture}%, N {nitrogen} mg/kg
WEATHER: {weather_summary}

Give 3 crops ranked by:
1. Profitability in Uganda
2. Suitability to this soil
3. Market demand

Format (max 160 chars):
1. CROP (85/100): grows well, high market price
2. CROP (75/100): reason
3. CROP (65/100): reason"""
```

For better fertilizer advice:

```python
prompt = f"""Farmer wants to improve soil for {crop_name}.

Current: pH {ph}, N {nitrogen}, P {phosphorus}, K {potassium}

Recommend SPECIFIC products available in Uganda:
- DAP, Urea, NPK 17-17-17, or organic options
- Exact quantity per acre
- Where/when to apply

Max 160 chars, be SPECIFIC!"""
```

2. Make AI Responses More Actionable

Add these to your prompts:

- "Include prices in UGX if possible"
- "Mention local supplier names (Yara, Amsons, etc.)"
- "Include planting calendar dates (e.g., 'Plant in March-April')"
- "Warn about common pests/diseases for this crop"

3. Handle Edge Cases

Add validation before calling AI:

```python
# In ai_agronomist.py
async def get_crop_recommendations(self, soil_data, weather_data):
    # Validate inputs
    if soil_data['ph'] < 3 or soil_data['ph'] > 10:
        return "⚠️ pH reading seems unusual. Please recalibrate sensor and retry test."

    if soil_data['nitrogen'] == 0:
        return "⚠️ Nitrogen reading is zero. Check sensor connection."

    # ... continue with AI call
```

4. Cache Common Queries (Advanced)

If same crop+soil combo asked frequently:

```python
# Add simple caching
cache = {}
cache_key = f"{crop_name}_{soil_data['ph']}_{soil_data['nitrogen']}"

if cache_key in cache:
    return cache[cache_key]

response = await self._generate(prompt)
cache[cache_key] = response
return response
```

5. Test AI Responses Quality

Create test cases:

```python
# test_ai_quality.py
test_scenarios = [
    {
        "name": "Acidic soil",
        "soil": {"ph": 4.5, "nitrogen": 20, ...},
        "expected_mention": ["lime", "pH", "acid"]
    },
    {
        "name": "Low nitrogen",
        "soil": {"ph": 6.5, "nitrogen": 10, ...},
        "expected_mention": ["fertilizer", "nitrogen", "NPK"]
    }
]

# Run tests, check if AI mentions expected terms
```

6. Multilingual Support (Future)

Gemini supports multiple languages:

```python
# Add to system prompt:
"Respond in {language} (English, Luganda, or Swahili)"
```

For hackathon, stick to English - but mention this capability to judges!

🏆 Demo Script

Show judges (10-minute demo):

1. **Admin Dashboard** (2 min)
    - Show farmer list
    - Show registered devices
    - Explain the concept

2. **Simulate Sensor Data** (3 min)
   - Use Postman/curl to send the exact JSON from your sensor
   - Show it appearing in database
   - **Show weather data fetched for location**
   - Explain: "Device sends data → Backend gets weather → AI analyzes"
3. **The Magic: AI + SMS** (3 min)
   - Show SMS arrives on your phone (or demo phone)
   - Read the message: "Reply 1, 2, or 3"
   - Reply with "1" for AI crop suggestions
   - **Show AI-generated crop recommendations**
   - Explain: "This is ChatGPT/Gemini analyzing soil + weather in real-time"
   - Reply "2" then "MAIZE"
   - **Show personalized advice from AI**
4. **Show Full Conversation** (2 min)
   - Go back to admin dashboard
   - Show SMS conversation history
   - Show soil test data + weather + AI responses stored
   - Show how admin can view all farmer interactions

Key Points to Emphasize:

- ✅ **No internet needed for farmers** (SMS only!)
- ✅ **Real AI** (Gemini/ChatGPT acting as agronomist)
- ✅ **Real weather data** (OpenWeather forecasts)
- ✅ **Scalable** (works for 10 or 10,000 farmers)
- ✅ **Low cost** (Telerivet uses normal SMS rates)
- ✅ **Works NOW** (everything is functional!)

Backup Plan: If SMS doesn't work during demo:

- Show the curl request → backend response
- Show AI responses in database
- Explain "SMS would send this to farmer"

Boom! Working AI-powered agricultural platform! 🚀🌱

Good luck with your hackathon! 🌱