🤖 SOLO AI-DEVELOPER MASTER PROMPT: Precision Fertilizer Advisory System Prototype

🎯 ONE-SHOT DEVELOPMENT COMMAND

Create a complete, functional prototype of an AI-Enabled Precision Fertilizer Advisory System for Indian farmers. I have the dataset locally. Build a FastAPI backend + React frontend application with the following SPECIFIC requirements:

CONTEXT
I am a solo developer using AI coding assistants. I have the following dataset files locally:
- soil_data.csv (district-level soil parameters: pH, N, P, K, OC)
- crop_data.json (crop growth stages, ZREAC guidelines)
- farmer_samples.csv (sample farmer records)

TECH STACK REQUIREMENTS
- **Backend**: FastAPI (Python) with SQLite for prototype
- **Frontend**: React.js with Tailwind CSS
- **Authentication**: OTP-based (use fixed OTP: 123456 for dev)
- **Language**: Bilingual (English/Telugu)
- **Deployment**: Single docker-compose setup

COMPLETE SYSTEM ARCHITECTURE

1. FILE STRUCTURE
```
fertilizer-advisory-system/
├── backend/
│   ├── main.py (FastAPI app)
│   ├── database.py (SQLite models)
│   ├── rules_engine.py (fertilizer calculation logic)
│   ├── data_loader.py (load my CSV/JSON datasets)
│   ├── requirements.txt
│   └── Dockerfile
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   │   ├── Login.jsx
│   │   │   ├── Dashboard.jsx
│   │   │   ├── RecommendationForm.jsx
```

```
│ │ │ ├── Results.jsx
│ │ │ └── LanguageToggle.jsx
│ │ ├── App.jsx
│ │ ├── index.js
│ │ └── styles/
│ ├── package.json
│ └── Dockerfile
├── datasets/ (my local data)
│ ├── soil_data.csv
│ ├── crop_data.json
│ └── farmer_samples.csv
├── docker-compose.yml
└── README.md
```

2. BACKEND SPECIFICATIONS

```python
# FASTAPI ENDPOINTS REQUIRED:
# 1. POST /api/register - Register farmer with mobile number
# 2. POST /api/login - OTP verification (use 123456 always for dev)
# 3. POST /api/recommendation - Generate fertilizer advice
# 4. GET /api/history - Get past recommendations
# 5. GET /api/crops - List available crops
# 6. GET /api/districts - List districts from my soil data

# DATABASE MODELS:
# - Farmer: id, mobile, name, district, language
# - Field: id, farmer_id, location, crop_type, sowing_date
# - Recommendation: id, farmer_id, recommendation_json, created_at

# RULE ENGINE LOGIC (implement in rules_engine.py):
"""
Inputs:
- crop_type: string
- sowing_date: date
- district: string
- soil_data: from my CSV (pH, N, P, K, OC)
- current_date: for growth stage calculation
```

Processing:
1. Calculate crop stage:
   - Vegetative: 0-30 days after sowing
   - Flowering: 31-60 days
   - Ripening: 61+ days

2. Get soil parameters from my dataset based on district

3. Apply ZREAC rules (simplified for prototype):
   IF soil_N < threshold: recommend N fertilizer
   IF soil_P < threshold: recommend P fertilizer
   IF soil_K < threshold: recommend K fertilizer
   Adjust based on crop stage

4. Incorporate weather check (mock):
   IF heavy_rain_forecast: delay recommendation

Output JSON structure:
```
{
  "crop": "Rice",
  "stage": "Vegetative",
  "fertilizers": [
    {"type": "Urea", "amount_kg_per_acre": 50, "timing": "Immediate"},
    {"type": "DAP", "amount_kg_per_acre": 30, "timing": "After 15 days"}
  ],
  "total_cost": 2500,
  "expected_yield_increase": "10-15%",
  "notes": "Avoid application if rain > 20mm forecast"
}
"""
```

3. FRONTEND COMPONENTS

```
// REQUIRED COMPONENTS:

// 1. Login.jsx - OTP-based login
// Features: Mobile input, OTP entry, language toggle (EN/TE)
```

```
// 2. Dashboard.jsx - Farmer dashboard
// Shows: Welcome message, recent recommendations, field list

// 3. RecommendationForm.jsx - Input form
// Fields: Crop dropdown, sowing date picker, district selector
// Button: "Get Recommendation"

// 4. Results.jsx - Display results
// Shows: Fertilizer table, cost analysis, print/share button
// Bilingual display

// 5. LanguageToggle.jsx - Switch between English/Telugu
// Store preference in localStorage
```

4. DATA INTEGRATION

Since I have the dataset:

1. Load soil_data.csv into SQLite on startup
2. Load crop_data.json for ZREAC rules
3. Use farmer_samples.csv for testing
4. Mock external APIs:
    - Weather: Return mock data
    - NDVI: Return random 0.3-0.8 value
    - e-Panta: Use farmer_samples.csv

5. DOCKER CONFIGURATION

```
# docker-compose.yml
version: '3.8'
services:
  backend:
    build: ./backend
    ports:
      - "8000:8000"
    volumes:
```

```
    - ./datasets:/app/datasets
    - ./backend_data:/app/data
  environment:
    - DATABASE_URL=sqlite:///./data/advisory.db

 frontend:
  build: ./frontend
  ports:
    - "3000:3000"
  depends_on:
    - backend
```

IMPLEMENTATION PRIORITY ORDER

PHASE 1: BACKEND FOUNDATION (Day 1)
1. Set up FastAPI with SQLite
2. Create database models
3. Load my CSV/JSON datasets
4. Implement OTP auth (hardcoded 123456)

PHASE 2: RULE ENGINE (Day 1-2)
1. Create rules_engine.py with fertilizer logic
2. Integrate my soil data
3. Implement crop stage calculation
4. Create recommendation endpoint

PHASE 3: FRONTEND BASICS (Day 2)
1. Create React app with Tailwind
2. Build Login component
3. Build RecommendationForm
4. Connect to backend API

PHASE 4: COMPLETE FEATURES (Day 3)
1. Add bilingual support
2. Create Results display
3. Add Dashboard
4. Implement localStorage for preferences

PHASE 5: DOCKERIZE & TEST (Day 4)
1. Create Dockerfiles
2. docker-compose setup
3. Test complete flow
4. Fix bugs

DEPLOYMENT READINESS CHECKLIST

✓ Backend runs on http://localhost:8000
✓ Frontend runs on http://localhost:3000
✓ Database initialized with my datasets
✓ OTP auth works (123456)
✓ Fertilizer recommendations generate
✓ Bilingual toggle works
✓ Mobile-responsive design
✓ Docker containers build successfully
✓ API documentation at /docs (FastAPI auto-generate)

TESTING SCENARIOS

Test 1: Farmer registration and login
Test 2: Rice crop recommendation (Vegetative stage)
Test 3: Wheat crop recommendation (Flowering stage)
Test 4: Language switch (English ↔ Telugu)
Test 5: Print/share recommendation
Test 6: Docker compose up/down

MOCK DATA FOR QUICK TESTING

Use these sample values for testing:

```
SAMPLE_FARMER = {
    "mobile": "9876543210",
    "name": "Ravi Kumar",
    "district": "Guntur",
    "language": "te"
```

```
}

SAMPLE_RECOMMENDATION_REQUEST = {
    "crop": "Rice",
    "sowing_date": "2024-01-01",
    "district": "Guntur",
    "field_size": 2.5
}
```

ONE-LINE STARTUP COMMANDS

```
# Option 1: Quick start with Docker
git clone [repository] && cd fertilizer-advisory-system && docker-compose up --build

# Option 2: Manual start
# Backend: cd backend && pip install -r requirements.txt && python main.py
# Frontend: cd frontend && npm install && npm start
```

EXPECTED FINAL OUTPUT

A fully functional web application accessible at `http://localhost:3000` with:

1. Login Page: Mobile + OTP (use 123456)
2. Dashboard: Welcome with farmer details
3. Recommendation Form: Crop, sowing date, district selection
4. Results Page: Detailed fertilizer advice with costs
5. Language Toggle: Switch between English/Telugu
6. Print/Share: Export recommendation as PDF
7. Responsive Design: Works on mobile/desktop