Technical Assignment – Farmer Parcel Assistant (Chat-Based Bot)

Duration: 5 days

Difficulty: Intermediate

Goal: Build a small backend service that simulates a WhatsApp-like assistant that helps farmers get information about their parcels.

---

1. Overview

You will build a backend service that behaves like a simple chat bot.

A farmer can send natural-language messages (simulated, not actual WhatsApp), and your service should respond with information about:

- Their parcels
- Parcel status
- Vegetation and soil indices
- Periodic reports
- Optional: natural-language summaries

The goal is to evaluate:

- Data modeling
- Ability to handle user input
- Ability to transform raw data into meaningful responses
- Reasoning & explanation during the final walkthrough call
- Code structure & clarity

Since we use Python with FastAPI (for backend), and React with Typescript (for frontend), it would be preferred you also use the same, but other tech stacks are accepted as well.

You do **NOT** need to integrate real WhatsApp or run real cron jobs. Simulated versions are enough.

---

2. Features & Requirements

First, make a private github repository and invite this username as a collaborator:

`Dywane12`

2.1 Chat Endpoint (Required)

Implement an endpoint:

```
POST /message
```

Request payload:

```
{
  "from": "+40123456789",
  "text": "Message sent by the farmer"
}
```

Response payload:

```
{
  "reply": "Your response message as text"
}
```

The `from` value identifies the farmer.

Your service should parse the message and produce an appropriate reply.

2.2 Account Linking Requirement (Additional Feature)

Before a farmer can use the assistant, they must **link their farmer account to their phone number**.

How it should work (Simulation)

When a user first “opens the app” (simulated by sending a chat message via `POST /message`), the system should check whether their phone number is already

linked to an existing farmer account.

If not, the bot should request their **username**.

Example flow:

User:

`Hi`

Bot:

`Welcome! Please type your username to link your account.`

User:

`my_username123`

Bot:

`Great, your account has been linked to +40123456789. You can now ask about your parcels.`

Technical details:

- You will receive usernames inside the provided `farmers.json`.
- Linking = storing `phone → farmer_id`
- If the user's phone number is already linked, the bot should skip the onboarding and respond normally.
- If the given username does not exist, respond with an error and ask again.

Minimum requirement:

- Implement a simple flow that:
    1. Detects unlinked users
    2. Requests username
    3. Validates the username
    4. Links the user's phone number
    5. Allows normal bot usage afterward

This feature simulates the real-world onboarding flow where a farmer identifies themselves the first time they message the assistant.

---

3. Supported Commands (Minimum Requirements)

Your bot must support at least the following user requests:

3.1 List all parcels

Example messages:

- “Show me my parcels”
- “List parcels”
- “What parcels do I have?”

Expected output:

A list of the farmer’s parcels (IDs, names, area).

3.2 Parcel details

Example messages:

- “Show details for parcel P1”
- “Tell me about parcel P1”
- “Information on parcel P1”

Expected output:

Basic details about the specified parcel:

- Name
- Area
- Crop
- Latest available indices mean values (vegetation and soil)

3.3 Parcel status summary (rule-based)

Example messages:

- “How is parcel P1?”
- “What’s the status of P1?”
- “Give me a summary for P1”

Your service must generate a **plain-language summary** based on index values.

You should implement **simple rule-based logic**, for example:

- If **NDVI > 0.6** → vegetation is healthy
- If **nitrogen < certain threshold** → low nitrogen
- If **pH < 6 → soil is acidic**
- Etc.

This shows how well you can turn numerical data into meaningful explanations.

(LLM-based summaries are a *bonus*, not required.)

3.4 Set report frequency

Allow the farmer to set how often they want parcel summary reports.

Example messages:

- “Set my report frequency to daily”
- “Set reports every 2 days”
- “Make reports weekly”

You should store the chosen frequency per farmer (in memory or simple storage is enough).

---

4. Report Generation (Required)

Implement one of the following:

Option A (simplest): Fake scheduled endpoint

Create an endpoint:

```
POST /generate-reports
```

When called, it should:

- Check report frequency for all farmers
- For those who should receive a report “today”, generate a summary
- Return an array like:

```
[
 {
  "to": "+40123456789",
  "message": "Your weekly parcel report: Parcel P1 is stable..."
 }
]
```

You do NOT need a real cron job.

When we test your project, we will manually call this endpoint.

---

5. Provided Data (You Will Receive)

You will receive a `data/` folder containing:

5.1 `farmers.json`

Example:

```
[
 {
  "id": "F1",
  "username": "john.popescu",
  "phone": "+40123456789"
 },
 {
   "id": "F2",
   "name": "michael.doe",
   "phone": null
 }
]
```

5.2 `parcels.json`

Example:

```
[
 {
  "id": "P1",
```

```
    "farmer_id": "F1",
    "name": "North Field",
    "area_ha": 10.5,
    "crop": "Wheat"
  }
]
```

5.3 parcel_indices.json

Example:

```
{
  "P1": [
    {
      "date": "2025-05-01",
      "ndvi": 0.55,
      "ndmi": 0.30,
      "ndwi": 0.20,
      "soc": 1.8,
      "nitrogen": 0.9,
      "phosphorus": 0.4,
      "potassium": 0.7,
      "ph": 6.5
    }
  ]
}
```

> In our real system, these values come from TIFF satellite and soil maps. For this assignment, you only use the provided JSON files.

---

6. Technical Expectations

6.1 Code Quality

- Clear structure
- Modular design
- No “god files” (not everything in one file)

- Readable variable and function names
- Good separation of concerns (parsing messages vs data fetching vs analysis)

6.2 Documentation (Required)

Your project must include a **README** containing:

- Setup instructions
- How to run the project
- How to test the endpoints
- Short explanation of your architecture
- Description of your rule-based analysis
- How you would extend it (LLM, real WhatsApp, real cron, databases)

---

7. Bonuses (Optional)

These are **NOT required**, but they will positively influence the evaluation.

7.1 LLM Integration

Use an external LLM (e.g., OpenAI, Anthropic, etc.) to:

- interpret user messages
- produce more natural summaries

This must be optional (via env variable) so your project still runs without costs.

7.2 Basic Automated Tests

Unit tests or integration tests for some components.

7.3 Persistence

Use a real database or simple local storage instead of in-memory structures.

7.4 Basic trend analysis

Interpret time-series data:

- NDVI increasing → improving vegetation
- Nitrogen decreasing over 3 weeks → potential nutrient depletion

7.5 Clean API design

Use DTOs, interfaces, or schemas for message formats.

7.6 Build a frontend to test the app

(Optional, only if you want a challenge after you build everything else)

Create a simple chat frontend that integrates your backend. Styling and design are of your choosing.

8. What We Will Evaluate

Your submission will be judged on:

Criteria	Description
Correctness	Required features work as specified
Code quality	Readable, structured, maintainable
Data handling	Farmers → parcels → indices mapping
Reasoning	Logical summaries based on index values
Communication	README clarity + final walkthrough
Architecture	Good separation of responsibilities
Bonus features	(Optional) LLM, trends, tests, etc.

9. Final Walkthrough Call

After submission, you will join a call with us to:

- Run the project
- Explain your code structure
- Walk us through message flow
- Explain your rule-based analysis
- Answer questions about design decisions

- Describe how you would extend this to:
  - Real WhatsApp
  - Real scheduling
  - Real TIFF ingestion
  - Integrating a live LLM

This is an important part of the evaluation.

---

10. Submission Instructions

You must submit:

1. **GitHub repository link** (public or private invite)
2. With:
   - Source code
   - `data/` folder
   - README
3. Optional:
   - Tests
   - Demo video
   - Postman collection

Please ensure the project runs using **only free resources**.

---

11. FAQ

Can I use any programming language?

Yes.

Do I need a real WhatsApp integration?

No. You only simulate WhatsApp using a `POST /message` endpoint.

Do I need a real cron job?

No. Use the `POST /generate-reports` endpoint.

Do I need LLMs?

No. LLM integration is optional and treated as a bonus.

Can I use my own free-tier AI API key?

Yes, but the project must still run without it.

How long should this take?

The assignment is designed to take **1–2 days of focused work** for an experienced developer.

You have **5 days**, allowing time for research, coding, documenting, and preparation.

Good luck!

We're excited to see your solution and discuss it with you.

If anything is unclear, you may ask for clarification during the assignment period.