CapServ

Assignment for Role : (AI Engineer)

Overview

Build a minimal **voice-style bot** service that:

1. accepts a text transcript (simulate STT output),
2. classifies the **intent** and extracts key **entities**, and
3. calls our **CRM REST APIs** (mock provided below) to perform:
    - **Lead onboarding**
    - **Visit scheduling**
    - **Lead status update**

To understand the user's intent, we expect you to process natural language using NLP fundamentals and LLMs. It is up to the candidate to decide how complicated of a user's query they are willing to handle.

Return a structured response. Handle errors gracefully. You do **not** need a UI or a true STT engine.

- **Timebox:** 3–4 hours (hard cap)
- **Primary language:** Python (3.10+)
- **What we're testing:** API integration, clean code, NLP basics, reasoning, reliability.

Deliverables

1. README.md with setup & run steps, design notes, and any assumptions.
2. Your bot service code (Python).
3. Unit tests (pytest preferred).
4. Postman collection or curl examples to demo happy paths + error paths.

Functional Requirements

A. Bot HTTP API (your service)

Expose a simple REST API:

POST /bot/handle

```
# Request body (JSON):
{
 "transcript": "text that user spoke",
 "metadata": {"user_id": "optional-uuid"}
}

# Response body (JSON, on success):

{
 "intent": "LEAD_CREATE | VISIT_SCHEDULE | LEAD_UPDATE | UNKNOWN",
 "entities": { "name": "...", "phone": "...", "city": "...", "lead_id": "...",
"visit_time": "...", "status": "..." },
 "crm_call": { "endpoint": "/...", "method": "POST", "status_code": 200 },
 "result": { "message": "..." }
}

# Response body (on error):

{
 "intent": "...",
 "error": { "type": "VALIDATION_ERROR | CRM_ERROR | PARSING_ERROR", "details":
"..." }
}
```

B. Intents & Minimal Entity Extraction

You must support **three** intents:

1. **LEAD_CREATE**
    - Required entities: name, phone (Indian formats acceptable), city (string).
    - Optional: source (e.g., “Instagram”, “Referral”).
2. **VISIT_SCHEDULE**
    - Required: lead_id (UUID or numeric), visit_time (ISO 8601 datetime).

- Optional: notes.

3. **LEAD_UPDATE**
    - Required: lead_id, status (one of: NEW, IN_PROGRESS, FOLLOW_UP, WON, LOST).
    - Optional: notes.

C. CRM Integration (Mock)

Call the provided **Mock CRM** (run locally—code below). Use **real HTTP** calls from your bot.

Endpoints (base URL http://localhost:8001 by default):

```
# 1. POST /crm/leads → create lead

# 1.1 Request:

{ "name": "string", "phone": "string", "city": "string", "source": "string?" }

# 1.2 Response:

{ "lead_id": "uuid", "status": "NEW" }


# 2. POST /crm/visits → schedule a visit

# 2.1 Request:

{ "lead_id": "uuid", "visit_time": "2025-10-02T17:00:00+05:30", "notes": "string?"
}

# 2.2 Response:

{ "visit_id": "uuid", "status": "SCHEDULED" }


# 3. POST /crm/leads/{lead_id}/status → update status

#3.1 Request:

{ "status": "NEW|IN_PROGRESS|FOLLOW_UP|WON|LOST", "notes": "string?" }

# 3.2 Response:

{ "lead_id": "uuid", "status": "..." }
```

D. Reliability & Edge Cases

- Validate required entities; return VALIDATION_ERROR with helpful messages.
- If CRM returns non-2xx, return CRM_ERROR and keep a human-readable message.
- Log the raw transcript, parsed intent/entities, and CRM response.
- Basic **rate limiting** or input size guard (e.g., transcripts ≤ 1,000 chars).

Input Examples (use in tests)

These are just a few basic examples. We expect the system to be able to handle multiple and more complex queries at the same time.

LEAD_CREATE

- “Add a new lead: Rohan Sharma from Gurgaon, phone 98 765 43210, source Instagram.”
- “Create lead name Priya Nair, city Mumbai, contact 91234-56789.”

VISIT_SCHEDULE

- “Schedule a visit for lead 7b1b8f54 at 3 pm tomorrow.”
- “Fix a site visit for lead 8f2a… on 2 Oct 2025 at 5:00 pm IST.”

LEAD_UPDATE

- “Update lead 7b1b8f54 to in progress.”
- “Mark lead 7b1b8f54 as won. Notes: booked unit A2.”

Ambiguous/Unknown

- “Can you help me?” → UNKNOWN

Tip: Parse casual date phrases like “tomorrow 3 pm” if you can; if not, return a clear VALIDATION_ERROR suggesting ISO format.

Non-Functional Requirements

- Clean, readable code; PEP8; docstrings on public functions.
- Clear separation: parsing → intent & entities → CRM client → response.
- Minimal configuration via env vars: CRM_BASE_URL, LOG_LEVEL.
- Unit tests covering:
    - Happy path for each intent
    - Missing required entity
    - CRM 500/timeout (can simulate with a flag or test double)

Starter Mock CRM (FastAPI)

Put this in mock_crm.py and run with:

uvicorn mock_crm:app --host 0.0.0.0 --port 8001 --reload

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from uuid import uuid4
from typing import Optional
from datetime import datetime

app = FastAPI(title="Mock CRM")

class LeadCreate(BaseModel):
   name: str
   phone: str
   city: str
   source: Optional[str] = None

class VisitCreate(BaseModel):
   lead_id: str
   visit_time: datetime
   notes: Optional[str] = None

class LeadStatusUpdate(BaseModel):
   status: str = Field(pattern="^(NEW|IN_PROGRESS|FOLLOW_UP|WON|LOST)$")
   notes: Optional[str] = None
```

```python
# In-memory stores
LEADS = {}
VISITS = {}

@app.post("/crm/leads")
def create_lead(payload: LeadCreate):
   lead_id = str(uuid4())
   LEADS[lead_id] = {**payload.dict(), "lead_id": lead_id, "status": "NEW"}
   return {"lead_id": lead_id, "status": "NEW"}

@app.post("/crm/visits")
def create_visit(payload: VisitCreate):
   if payload.lead_id not in LEADS:
       raise HTTPException(status_code=404, detail="Lead not found")
   visit_id = str(uuid4())
   VISITS[visit_id] = {**payload.dict(), "visit_id": visit_id, "status":
"SCHEDULED"}
   return {"visit_id": visit_id, "status": "SCHEDULED"}

@app.post("/crm/leads/{lead_id}/status")
def update_lead_status(lead_id: str, payload: LeadStatusUpdate):
   if lead_id not in LEADS:
       raise HTTPException(status_code=404, detail="Lead not found")
   LEADS[lead_id]["status"] = payload.status
   return {"lead_id": lead_id, "status": payload.status}
```

Minimal Bot Skeleton (suggested structure)

```
/bot
 ├─ app.py               # FastAPI/Flask app exposing /bot/handle
 ├─ nlu.py               # intent + entity extraction
 ├─ crm_client.py        # HTTP client for mock CRM
 ├─ models.py            # Pydantic request/response models
 ├─ settings.py          # env config
 └─ tests/
     ├─ test_lead_create.py
     ├─ test_visit_schedule.py
     └─ test_lead_update.py
```

Design hints:

- nlu.py: start rule-based (regex for phone, UUID; keyword sets for intents; simple city list fallback).

- crm_client.py: wrap requests with timeouts + retries (e.g., 2 retries, 1s backoff).
- app.py: orchestrates → validate → NLU → call CRM → build response.

Example

curl

Scripts (candidate should include equivalents)

Create lead

```
curl -X POST http://localhost:8000/bot/handle \
 -H "Content-Type: application/json" \
 -d '{"transcript":"Add a new lead Rohan Sharma from Gurgaon phone 9876543210 source Instagram."}'
```

Schedule visit

```
 curl -X POST http://localhost:8000/bot/handle \
 -H "Content-Type: application/json" \
 -d '{"transcript":"Schedule a visit for lead 7b1b8f54-aaaa-bbbb-cccc-1234567890ab at 2025-10-02T17:00:00+05:30"}'
```

Update status

```
curl -X POST http://localhost:8000/bot/handle \
 -H "Content-Type: application/json" \
 -d '{"transcript":"Update lead 7b1b8f54-aaaa-bbbb-cccc-1234567890ab to WON notes booked unit A2"}'
```

Acceptance Criteria (pass/fail)

- Runs locally with mock CRM and your bot in separate processes.
- All three intents work end-to-end, returning structured JSON.

- Clear errors for missing entities and CRM failures.
- At least **6 unit tests** (≥2 per intent) passing locally.
- README is sufficient for a teammate to run in <10 minutes.

Evaluation Rubric (100 pts)

- **API Integration & Error Handling (25)** – correct endpoints, timeouts, retries, meaningful errors.
- **NLU Quality (10) + Use of LLM (15)** – intent classification accuracy on provided samples + a few of your own; entity extraction robustness + the ability to handle longer more complex inputs.
- **Code Quality (20)** – structure, readability, typing, docstrings, tests.
- **Reliability & Logging (15)** – sensible logs, input validation, edge-case handling.
- **DevEx (15)** – good README, simple setup, usable curl/Postman, env config.

Bonus (up to +20):

- **Ability to handle multiple requests in the same input.**
- Simple **confidence score** & fallback prompts.
- Minimal **analytics log** (JSONL) with timestamp, intent, entities, success/failure.
- Casual datetime parsing ("tomorrow 3 pm IST").

Flutter/Mobile Integration Note (context)

Assume your bot will be called from a Flutter app. Keep the bot's /bot/handle request/response **stable and documented** so mobile developers can consume it without changes.

What to Submit

- GitHub repo link (public or private invite).
- Short note on what you'd improve with more time.