Farm-Assist Starter Repo (Updated)

This canvas contains a **starter repository** for the Farmer Assistant AI (voice-enabled) built with **FastAPI + LangGraph** backend and a minimal **React PWA** frontend (audio recorder + playback). This updated version fills in the **OpenAI audio transcription** path, adds **Postgres + Alembic** configuration and `docker-compose` services, and implements a **vision node** using a small YOLOv8 inference example (mock/detection-based) to demonstrate how to plug a vision model into the LangGraph pipeline.

Repo tree (updated)

```
farm-assist-starter/
├── backend/
│   ├── Dockerfile
│   ├── docker-compose.yml
│   ├── requirements.txt
│   ├── .env.example
│   ├── README.md
│   ├── alembic.ini
│   ├── alembic/
│   │   ├── env.py
│   │   └── versions/
│   │       └── 0001_initial.py
│   ├── app/
│   │   ├── main.py                # FastAPI app with endpoints
│   │   ├── api/
│   │   │   ├── routes.py         # upload endpoints
│   │   │   └── utils.py          # helper util functions
│   │   ├── farm_agent/
│   │   │   └── langgraph_app.py  # LangGraph state graph and nodes
│   │   ├── models/
│   │   │   ├── vision.py         # YOLOv8 inference example
│   │   │   └── db_models.py      # SQLAlchemy models
│   │   ├── db.py                 # database connection (SQLAlchemy async)
│   │   └── storage.py            # S3/local storage helper
│   └── migrations/
│
├── frontend/
│   ├── package.json
│   ├── vite.config.js
│   ├── README.md
│   └── src/
│       ├── main.jsx
```

```
│       ├── App.jsx
│       └── components/
│           └── Recorder.jsx      # mic record + upload + playback
│
└── README.md
```

What I updated (summary)

1. **OpenAI audio transcription**: wired `transcribe_with_openai()` in `langgraph_app.py` to call OpenAI's Audio Transcription API (Whisper) using the official OpenAI SDK call pattern. This function takes the uploaded audio file, streams it to OpenAI, and returns `text` and `language`.

2. **Postgres & Alembic**: added `docker-compose` service for `postgres:15`, a `.env.example` with DB credentials, SQLAlchemy asynchronous engine and models (`db.py`, `db_models.py`), and a basic Alembic configuration (`alembic.ini`, `alembic/env.py`) plus an initial migration `0001_initial.py` that creates simple tables for `users` and `conversations`.

3. **Vision node (YOLOv8)**: added `app/models/vision.py` that uses the `ultralytics` package (YOLOv8) to run a detection pass on an image and create a mock disease/pest classification result. The example uses the pretrained `yolov8n.pt` detection model to show how to include inference; it's a demonstration you can replace with a custom model later. The `run_vision_classifier()` returns a structured dict `{"disease": <label>, "confidence": <float>, "raw_detections": <...>}`.

4. **Requirements**: updated `requirements.txt` with additional packages: `sqlalchemy`, `asyncpg`, `alembic`, `psycopg2-binary`, and `ultralytics`.

5. **Docker Compose**: updated `docker-compose.yml` to include `postgres` service, environment variables, and migration convenience commands (instructions in README).

6. **README & test instructions**: added step-by-step instructions to set environment variables, run the services, test audio upload (demo `curl`), run Alembic migrations, and run a sample vision inference locally to sanity-check things.

Important files added/updated (high-level)

- `backend/requirements.txt` — includes OpenAI, langgraph, ultralytics, SQLAlchemy, alembic.
- `backend/.env.example` — sample env file showing required variables (`OPENAI_API_KEY`, `DATABASE_URL`, `S3_*`, etc.).
- `backend/docker-compose.yml` — now includes a `postgres` service and volumes.
- `backend/app/db.py` — async SQLAlchemy engine and sessionmaker.
- `backend/app/models/db_models.py` — simple `User` and `Conversation` tables.

- `backend/alembic.ini` and `backend/alembic/env.py` — alembic configuration for async SQLAlchemy.
- `backend/alembic/versions/0001_initial.py` — sample initial migration DDL.
- `backend/app/models/vision.py` — YOLOv8 inference example using `ultralytics.YOLO`.
- `backend/app/farm_agent/langgraph_app.py` — updated to call Postgres to persist conversations, call `transcribe_with_openai`, call vision node, and store TTS outputs.

All of these files are included in the canvas — open the document to view full code for each file.

Test instructions (quick)

1. Copy `.env.example` to `.env` and fill in values. Example `.env` values:

```
OPENAI_API_KEY=sk-...
DATABASE_URL=postgresql+asyncpg://postgres:postgres@postgres:5432/farmdb
S3_BUCKET=
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
UPLOAD_DIR=/tmp/uploads
```

1. Start services (from `backend/`):

```
# from repo root
cd backend
docker-compose up --build
```

This brings up the FastAPI app on port `8000` and a Postgres container.

1. Run Alembic migrations (in a separate shell or container):

```
# in backend/
# either run alembic directly if installed locally, or run inside the backend
container
docker exec -it <backend_container_name> bash
alembic upgrade head
```

1. Test audio transcription (sample `curl`):

```
curl -X POST "http://localhost:8000/api/upload_audio"
  -F "file=@/path/to/sample_audio.webm"
  -F "user_id=farmer_demo"
  -F "lat=23.7" -F "lon=90.4"
```

The endpoint will: - save the audio file to `UPLOAD_DIR` - invoke the LangGraph flow - call `transcribe_with_openai()` using your `OPENAI_API_KEY` - optionally call vision & weather nodes (if provided) - return a JSON state which includes `transcript` and `tts_path` (local path to generated MP3). Use `/api/get_tts?path=<tts_path>` to download the TTS audio.

1. Test vision node locally:

```
from app.models.vision import run_vision_classifier
print(run_vision_classifier('/path/to/test_leaf.jpg'))
```

This should return a `dict` with `disease`, `confidence`, and `raw_detections` keys. The YOLOv8 demo will report detection boxes if objects are found. This is a mock mapping to show how to integrate — replace with a trained disease classifier later.

Notes & caveats

- **OpenAI usage**: make sure your `OPENAI_API_KEY` has access to the `audio.transcriptions` endpoint (Whisper) and the chat endpoint. The code uses the OpenAI Python SDK pattern `from openai import OpenAI` and `client = OpenAI()`; if you prefer the older `openai` module, adapt accordingly.

- **Database**: the Alembic setup provided uses the async SQLAlchemy pattern. When running migrations inside the container, ensure the `DATABASE_URL` environment variable is set correctly and reachable by the backend container.

- **Ultralytics (YOLOv8)**: the `ultralytics` package will download weights the first time you run inference (internet required). The demo uses a detection model (`yolov8n.pt`) — you'll want to replace it with a classification or a custom-trained model for disease detection. Training on PlantVillage or a local labeled set is recommended.

- **Performance & cost**: Offloading transcription and LLM calls to OpenAI is the fastest path for an MVP but will incur cost. For offline/edge deployments, replace Whisper/OpenAI with a local Whisper variant and a local LLM.

Next actions I can take for you

I can now (pick any or multiple):

1. Create a downloadable ZIP file of the repo so you can `git clone` and run locally.
2. Replace the demo YOLOv8 detection step with a small classification example using `torchvision` / `timm` and a pre-trained model to better mimic disease classification.

3. Generate a real migration file that creates additional fields (e.g., `media_url`, `confidence`, `transcript`) and a quick UI page to view stored conversations.
4. Help you train a small classifier on PlantVillage and export to a model file to plug into `vision.py` (requires dataset access).

Tell me which of these you want next, or say "zip it" and I will prepare a downloadable zip of the full repo.