Pepper Medical Assistance Robot – Pipeline

Think of your system as **three connected layers**:

1. **Pepper Layer (robot/)**
    - Runs in **Python 2.7** using the **NAOqi SDK**.
    - `bridge.py` connects Pepper to your backend via a WebSocket.
    - It receives commands like:
        - **TTS** → Pepper speaks (`ALTextToSpeech`).
        - **Tablet** → Pepper displays your web UI (`ALTabletService`).
        - **Alert** → Pepper calls for nurse assistance.
    - Essentially, Pepper is the **output/interaction device** (voice, tablet, gestures).

---

2. **Backend Layer (server/)**
    - Runs in **Python 3.x** using **FastAPI**.
    - Files inside `server/app/` handle different jobs:
        - `main.py` → Entry point, defines APIs (`/api/speak`, `/api/faq/search`, `/api/triage/submit`).
        - `rag.py` → FAQ retrieval (patients can ask “What are visiting hours?” → retrieves stored answers).
        - `triage.py` → Simple rule-based health questionnaire scoring (not diagnostic).

        - `bridge_bus.py` → Publishes messages to Pepper (via the WebSocket bridge).
    - Connects patients/staff to Pepper, manages all logic, and ensures requests → robot actions.

---

3. **User Interface Layer (pepper_ui/)**
    - A **web app** displayed on Pepper's tablet.
    - Files:
        - `index.html` → Tabs for **Check-in, FAQ, Triage**.
        - `style.css` → Large fonts, big buttons (elder-friendly).
        - `script.js` → Sends API requests to the backend (`fetch()` → `/api/faq/search`, `/api/triage/submit`).
    - Patients interact via **touch** or **voice**, Pepper responds through **speech** and **tablet display**.

🔹 How Data Flows (Step-by-Step)

1. **Patient interacts**
    - Example: Types a question on the tablet or says "What are the visiting hours?"
2. **UI → Backend**
    - The web app (`script.js`) sends the input to FastAPI (`/api/faq/search`).
3. **Backend → Logic**
    - `rag.py` searches the FAQ database and finds an answer.

- `main.py` then calls `publish({"type":"tts", "text": answer})`.

4. **Backend → Robot Bridge**
   - The message is pushed through WebSocket (`bridge_bus.py`).
5. **Robot Bridge → Pepper**
   - `bridge.py` receives it, calls `ALTextToSpeech`, and Pepper speaks the answer.

🔷 Example Pipelines in Action

- **Reception / Check-in**
  - Tablet → enter ID → `/api/speak` → Pepper says: “Welcome Mr. Ahmed, your appointment is confirmed.”
- **FAQ**
  - Patient asks → Backend retrieves from `rag.py` → Pepper speaks + shows text.
- **Triage**
  - Patient answers yes/no on symptoms → `/api/triage/submit` → `triage.py` scores.
  - If urgent → `publish({"type":"alert"})` → Pepper says: *“Urgent case, please call a nurse.”*

🔷 Big Picture

- **Pepper = Face + Voice + Tablet** (interaction device).
- **Backend = Brain** (logic, FAQ, triage, nurse alerts).
- **UI = Hands** (how patients actually input info).

Everything runs locally (no cloud required), ensuring **privacy, low latency, and reliability**.