Receptionist Robot App: Backend Guide

Hello! This document explains how the Receptionist Robot's "brain" (our FastAPI backend) works. It handles all the smart parts: seeing faces, recognizing people, managing conversations, and talking to officials.

1. Backend's Role in the System (The Brain)

Think of the backend as the central control room for our robot.

- **It processes what the robot "sees":** Your mobile app (frontend) sends camera pictures to the backend.
- **It understands who's who:** It uses advanced AI to recognize faces.
- **It manages the conversation:** It knows what to say next based on the visitor's actions and manages the entire interaction flow.
- **It talks to other systems:** It can send notifications to company officials.
- **It "speaks":** It uses text-to-speech to deliver messages.

The backend is built using **FastAPI**, a modern Python web framework, which makes it fast and easy to build APIs.

2. How the Backend is Organized (The Files)

The backend code is split into three main files for better organization:

2.1. utils.py (The Toolbox)

This file is like our shared toolbox. It contains all the helpful functions that many parts of the robot might need.

- **Model Initialization:** Sets up our AI models (like the ResNet for face recognition and MediaPipe for finding faces in pictures) when the backend starts. This happens only once.
- **Face Embedding:** Takes a picture of a face and turns it into a unique number code (an "embedding") that the robot can use to compare faces.
- **Face Detection:** Finds where faces are in a camera picture.
- **Known Faces Management:** Loads and saves our list of known people (their name and their face's number code) to special files (.pkl files).
- **FAISS Search:** Uses a super-fast library called FAISS to quickly search through all known face codes to find matches. This is much faster than checking every single face one by one.
- **Text-to-Speech (TTS) Manager:** This is new! It uses pyttsx3 to turn text messages into spoken words. It runs in a special "background thread" so the

robot can talk without slowing down other operations.

2.2. robot_brain.py (The Robot's Mind)

This is the core intelligence of our robot. It's a Python "class" called RobotBrain that holds everything the robot needs to remember and decide.

- **Robot States:** It keeps track of where the robot is in its interaction (e.g., "IDLE" - just waiting, "CONFIRMING_IDENTITY" - asking "Is that you?", "WAITING_FOR_OFFICIAL_RESPONSE" - waiting for a reply from an official). These states strictly follow the workflow diagram you provided.
- **Visitor Information:** It remembers details about the current visitor, like their name, the picture taken of them, and if they are a known person.
- **Official Contact Logic:** It manages sending messages to officials, retrying if they don't respond, and offering options to the visitor if the official is unreachable.
- **Decision Maker:** For every camera frame or user input, the RobotBrain decides what the robot should do or say next, based on its current state and the workflow rules.
- **Speaking Messages:** Whenever the RobotBrain decides to send a message to the frontend, it also tells the TTSManager (from utils.py) to speak that message aloud.

2.3. main.py (The Server & API)

This is the FastAPI application itself. It's the "front door" for your mobile app to talk to the RobotBrain.

- **FastAPI Setup:** Initializes the web server.
- **RobotBrain Instance:** It creates one RobotBrain object when the server starts. This single RobotBrain handles all interactions for the robot.
- **API Endpoints:** It defines the specific URLs (like /process_frame or /submit_input) that your mobile app will send requests to. Each endpoint calls a specific function within the RobotBrain to handle the request.
- **Data Formats:** It uses Pydantic models to make sure that data sent to and from the API is in the correct format (e.g., expecting an image file or a JSON with input_text).
- **Background Tasks:** It runs a special background task that periodically checks if an official has timed out on their response, without blocking the main server.

3. How the Backend Implements the Workflow

The backend follows the workflow diagram very closely, step-by-step:

- **Starting Up:** When the backend starts, the RobotBrain is IDLE.

- **Seeing a Visitor (/process_frame):** Your app sends continuous camera frames.
    - The backend detects if a face is present.
    - It times how long the face is visible (3 seconds).
    - If the face stays for 3 seconds, it takes a "snapshot" and tries to recognize the person.
- **Recognizing / Greeting (/process_frame):**
    - **If Known:** It finds matching faces in its FAISS index. It then asks your app to display "Are you [Name]?" with "Yes/No" buttons.
    - **If Unknown:** It asks your app to display "Please tell me your name after the beep" with a text input box.
- **Handling User Input (/submit_input):** Your app sends back what the user types or chooses.
    - **Name Input:** If the robot asked for a name (unknown visitor), the backend takes that name, adds it to its known faces (with the captured photo), and asks your app to confirm the name and photo.
    - **Confirming Identity/Photo:** If the user says "Yes," the robot moves to the next step. If "No," it might try other guesses or reset.
    - **Whom to Meet:** If the robot asked "Whom would you like to meet?", the backend takes the name, finds matching officials, and asks your app to display a list of choices.
- **Contacting Officials:** Once the official is confirmed, the backend "sends" a notification (simulated for now) and waits.
- **Official Response / Timeout (/official_response & Background Task):**
    - The backend waits for an external "official response" (from a testing tool or a real official app).
    - If no response comes in 3 minutes, a background task in main.py notices this. It retries contacting the official up to 3 times.
    - If still no response after 3 tries, the backend tells your app to display "Official unreachable. Wait, leave message, or leave?" with buttons.
- **Visitor's Choice (Wait/Message/Leave) (/submit_input):** Your app sends the user's choice back.
    - **Wait:** The backend resets the timer and keeps waiting.
    - **Leave Message:** The backend asks your app for the message, then "sends" it to the official.
    - **Leave:** The backend says goodbye and resets.
- **Visitor Leaves:** The backend continuously checks if the recognized visitor is still in front of the camera. If they leave for too long, the backend resets to IDLE.

4. How to Run and Test the Backend

1. **Get the Code:** Make sure you have utils.py, robot_brain.py, and main.py in the same folder.
2. **Get AI Models:** You'll also need the AI model files (Resnet50_3.pth, faces_data.pkl, names.pkl) placed in the BASE_PROJECT_FOLDER that's set in main.py and robot_brain.py. **Make sure this path is correct on the machine running the backend!**
3. **Install Tools:** Open your computer's command line (terminal) and run:
   pip install uvicorn fastapi opencv-python mediapipe Pillow torch torchvision scipy faiss-cpu pyttsx3

4. **Start the Backend:** In your terminal, navigate to the folder where you saved the files and run:
   uvicorn main:app --host 0.0.0.0 --port 8000 --reload

   - --host 0.0.0.0 means it's accessible from other devices on your network (like your phone).
   - --port 8000 means it runs on port 8000.
   - --reload means it automatically restarts if you change the code (useful for development).
5. **Test with Interactive Docs:** Once running, open your web browser and go to http://127.0.0.1:8000/docs. This page lets you see all the API endpoints and even send test requests directly to your backend!

This backend is ready to be connected to your frontend application. Let us know if you have any questions!