Top-Level Repository Structure

```
smart-door-system/
├── README.md
├── .gitignore
├── .gitlab-ci.yml
├── docs/
│ ├── ArchitectureDiagram.png
│ ├── System_Architecture_Overview.pdf
│ ├── Functional_Requirements.md
│ ├── NonFunctional_Requirements.md
│ ├── FaceRecognition_Guide.md
│ ├── Setup_Guide.md
│ └── PPT_Slides/
│ └── FinalPresentation.pptx
├── api/
│ ├── src/
│ │ ├── main.py
│ │ ├── controllers/
│ │ │ ├── auth_controller.py
│ │ │ ├── roles_controller.py
│ │ │ └── door_controller.py
│ │ ├── services/
│ │ │ ├── role_service.py
│ │ │ └── video_management_service.py
│ │ ├── models/
│ │ │ ├── user_model.py
│ │ │ ├── role_model.py
│ │ │ └── event_log_model.py
│ │ ├── db/
│ │ │ ├── database_config.py
│ │ │ └── migrations/
│ │ ├── mqtt_integration.py
│ │ └── config.py
│ ├── requirements.txt
│ ├── tests/
│ │ ├── test_auth.py
│ │ ├── test_roles.py
│ │ └── test_integration.py
│ └── README.md
├── mobile_app/
```

```
│ ├── android/
│ │ ├── app/
│ │ │ ├── src/main/java/com/smartdoor/
│ │ │ │ ├── ui/
│ │ │ │ │ ├── MainActivity.java
│ │ │ │ │ ├── DoorControlFragment.java
│ │ │ │ │ └── RolesFragment.java
│ │ │ │ ├── network/
│ │ │ │ │ ├── MqttClientHelper.java
│ │ │ │ │ └── ApiService.java
│ │ │ │ ├── data/
│ │ │ │ │ ├── models/
│ │ │ │ │ └── repositories/
│ │ │ │ └── utils/
│ │ │ ├── res/
│ │ │ │ ├── layout/
│ │ │ │ ├── values/
│ │ │ │ └── drawable/
│ │ │ └── build.gradle
│ ├── ios/ (if applicable)
│ └── README.md
├── edge_device/
│ ├── raspberry_pi/
│ │ ├── main_pi.py
│ │ ├── camera_stream.py
│ │ ├── face_recognition.py
│ │ ├── motion_detection.py
│ │ ├── role_filter.py
│ │ ├── event_storage.py
│ │ ├── mqtt_client.py
│ │ └── requirements.txt
│ ├── arduino/
│ │ └── servo_control.ino
│ └── README.md
├── database/
│ ├── schema.sql
│ ├── migrations/
│ │ └── 001_create_tables.sql
│ ├── seeds/
│ │ └── default_roles.sql
│ └── README.md
```

```
└── extras/
├── speech_recognition/
│ ├── speech_test.py
│ └── README.md
├── advanced_ml_human_nonhuman/
│ ├── custom_model.py
│ └── data/
└── README.md
```

Folder-by-Folder Breakdown & Rationale

1. README.md & .gitignore

README.md: Contains a high-level project description, setup instructions, required environment variables, and quickstart commands.

.gitignore: Lists files and directories to ignore (e.g., virtual environments, compiled binaries, secret files).

2. docs/

This folder centralizes all project documentation:

ArchitectureDiagram.png & System_Architecture_Overview.pdf: Visual representations of how the mobile app, API, and edge devices (Raspberry Pi/Arduino) interact.

Functional_Requirements.md & NonFunctional_Requirements.md: Summaries (or direct excerpts) of your system's requirements.

FaceRecognition_Guide.md: Integration notes and setup instructions (e.g., from your Pi Camera guide).

Setup_Guide.md: Step-by-step instructions for new developers.

PPT_Slides/: Contains presentation materials for stakeholder briefings.

3. api/ (Python-based backend)

This folder holds the server-side logic.

src/main.py: The entry point (e.g., a Flask/FastAPI app) that registers endpoints.

Example (main.py):

```
from flask import Flask
from controllers.auth_controller import auth_bp
from controllers.roles_controller import roles_bp
from controllers.door_controller import door_bp
```

app = Flask(**name**)
app.register_blueprint(auth_bp, url_prefix='/auth')
app.register_blueprint(roles_bp, url_prefix='/roles')
app.register_blueprint(door_bp, url_prefix='/door')

if **name** == '**main**':
app.run(host='0.0.0.0', port=5000)

controllers/: Contains HTTP route handlers (e.g., login, role creation, door commands).

services/: Business logic functions (e.g., verifying roles, managing video clips).

models/: Database entity definitions (using an ORM such as SQLAlchemy).

db/: Database configuration and migration scripts.

mqtt_integration.py: If the API also needs to publish/subscribe via MQTT, that logic is here.

config.py: Central configuration for environment-specific settings.

tests/: Unit and integration tests to ensure the API's endpoints and logic work as intended.

4. mobile_app/

Contains the mobile application source code.

android/: Standard Android project structure.

ui/: Activities and fragments (e.g., DoorControlFragment.java for sending MQTT commands to open/close the door).

Example (DoorControlFragment.java):

/**

- DoorControlFragment.java
- Allows the user to send door commands.
  */
  public class DoorControlFragment extends Fragment {
  private void sendOpenDoorCommand() {
  // Publish MQTT message or call the API to open the door
  }
  private void sendCloseDoorCommand() {
  // Publish MQTT message or call the API to close the door
  }
  }

network/: Handles MQTT (via helper classes) and HTTP clients for API communication.

data/ & utils/: For managing data models and utility functions.

ios/ (if applicable): iOS counterpart if you decide to support it.

5. edge_device/

Houses code running on your hardware (Raspberry Pi and Arduino).

raspberry_pi/

main_pi.py: The entry point on the Pi. Initializes sensors, camera, and MQTT client.

Example (main_pi.py):

""""

main_pi.py

- Initializes sensors (PIR, ultrasonic)
- Starts camera streaming and face recognition
- Listens for door commands via MQTT and sends events accordingly
  """"
  def init_sensors():

  ## Setup GPIO, calibrate sensors

  pass

def init_camera():
# Configure and start the Pi Camera stream
pass

def run_main_loop():
# Detect motion, capture frames, run face recognition, and publish MQTT messages
pass

if **name** == "**main**":
init_sensors()
init_camera()
run_main_loop()

camera_stream.py: Manages the live video feed (e.g., MJPEG or RTSP stream).

face_recognition.py: Contains functions to load face encodings and recognize faces from the camera feed.

motion_detection.py: Reads sensor data and detects motion events.

role_filter.py: Checks if a recognized face has "auto-open" permissions (based on roles).

event_storage.py: Saves event-based footage and manages deletion of recordings older than 72 hours.

mqtt_client.py: Subscribes to door control topics and publishes events (e.g., "motion/detected", "face/recognized").

requirements.txt: Python dependencies for the edge device code.

arduino/servo_control.ino: Code for controlling the servo motor.

Example (servo_control.ino):

```
#include <Servo.h>
Servo doorServo;

void setup() {
doorServo.attach(9);
Serial.begin(9600);
}

void loop() {
if (Serial.available()) {
char cmd = Serial.read();
if (cmd == 'o') {
doorServo.write(0); // Open door
} else if (cmd == 'c') {
doorServo.write(90); // Close door
}
}
}
```

6. database/

Contains the database schema and migration scripts.

schema.sql: Core table definitions (users, roles, events, video logs).

migrations/: Versioned changes (e.g., creating new tables).

seeds/: Initial data (default roles such as "homeowner", "admin", "guest").

7. extras/

A sandbox for experimental code.

speech_recognition/: Prototype code to convert voice commands into actions.

advanced_ml_human_nonhuman/: Additional models to improve human/non-human differentiation.

Each subfolder includes its own README for context.

How the Pieces Connect

1. Edge Device & Hardware:
   The Raspberry Pi (in edge_device/raspberry_pi/) gathers sensor data (motion, video) and runs local AI (face recognition).

It sends MQTT messages (e.g., "motion/detected", "face/recognized") and, based on role filtering (in role_filter.py), may trigger the door to open by sending a serial command (to the Arduino in arduino/servo_control.ino).

2. API (Python):
   The API (in api/) manages user authentication, role-based permissions, and logs events from the Pi. It can also be queried by the Pi for detailed role information (e.g., auto-open privileges) and by the mobile app for administrative tasks.

3. Mobile App:
   The mobile app (in mobile_app/) receives real-time notifications (via MQTT and push services) and allows users (homeowner, admin, guests) to interact with the door remotely. It also lets authorized users manage roles and permissions.

4. Database:
   All user data, role hierarchies, event logs, and face embedding references are stored and managed via the scripts in database/. The API uses this data to enforce business logic.

5. Extras:
   New features (such as speech recognition) are developed in extras/ and can later be merged into the main codebase as they stabilize.

Summary

Clear Separation: Hardware code is in edge_device/, server logic in api/, and the mobile interface in mobile_app/.

Scalability & Collaboration: Each team (hardware, backend, mobile, and R&D) works in isolated subfolders with clear integration points (via MQTT, HTTP, and database calls).

Role-Based Access & Event Management: The API and database manage roles/permissions, while the edge device uses local role filtering to decide actions (like auto-opening the door).

Flexible Video Storage: The event-based storage script on the Pi ensures that only important events are recorded, with older clips auto-deleted after 72 hours.