**1. face\_recognition.py**

This file contains the core face recognition functionality:

Key Components:

* **FaceAnalysis Initialization**: Uses InsightFace's buffalo\_l model with CUDA acceleration if available
* **Employee Database**: Loads employee images from a folder and extracts facial embeddings
* **Recognition Functions**:
  + get\_top\_predictions(): Returns top N matches with similarity scores
  + recognize(): Finds the best match for a face embedding
  + log\_recognition(): Logs successful recognitions to a JSON file
  + detect\_faces(): Detects faces in a frame using InsightFace

Process Flow:

1. On startup, loads all employee images from the specified folder
2. For each image, extracts facial embeddings and stores them in a dictionary
3. Provides functions to compare new face embeddings against the database

**2. camera.py**

This file handles video processing and real-time face recognition:

Key Components:

* **Video Capture**: Uses OpenCV with FFMPEG backend for RTSP streams
* **Frame Processing**:
  + Skips frames to reduce processing load
  + Detects faces in each frame
  + Finds the primary (largest) face
* **Recognition Display**:
  + Draws bounding boxes around detected faces
  + Displays recognition results
  + Shows a resized version of the recognized face in the corner
* **Performance Metrics**: Calculates and displays FPS

Process Flow:

1. Initializes video capture with optimized settings
2. Processes frames in a loop:
   * Detects faces
   * Gets recognition predictions
   * Updates shared predictions storage
   * Draws UI elements on the frame
   * Yields the processed frame as JPEG

**3. main.py**

This is the FastAPI web server that coordinates everything:

Key Components:

* **API Endpoints**:
  + /: Main page with camera selection
  + /video\_feed/{cam\_id}: Streams processed video
  + /predictions/{cam\_id}: Gets current recognition predictions
  + /log\_selection: Logs user feedback on recognition accuracy
* **Camera Management**: Defines RTSP URLs for each camera
* **Templates**: Uses Jinja2 to serve HTML pages

Process Flow:

1. Starts a FastAPI server with Uvicorn
2. Serves web pages with video feeds
3. Provides endpoints for:
   * Video streaming
   * Getting current predictions
   * Logging user feedback

**4. System Workflow**

1. **Initialization**:
   * Load employee database
   * Start FastAPI server
   * Initialize camera connections
2. **Real-time Processing**:
   * Each camera feed is processed in parallel
   * Frames are captured and analyzed
   * Faces are detected and recognized
   * Results are stored in shared memory
3. **User Interaction**:
   * Users view camera feeds via web interface
   * System displays recognition results
   * Users can provide feedback on accuracy
4. **Logging**:
   * Automatic logging of recognitions
   * Manual logging of user feedback