# Fingerprint Server for R502-A Fingerprint Sensor

................................................................................................................

This Python server, built with Flask, enables the enrollment and detection of fingerprints from the R502-A fingerprint sensor. It communicates with the sensor via a microcontroller, processes the fingerprint data, and stores or matches templates against a local SQLite database.

## Features

Enrollment Mode: Adds new fingerprint templates to the database with a username.

Detection Mode: Matches incoming fingerprint templates against stored ones and identifies the closest match.

Database Management: Uses an SQLite database to store and retrieve fingerprint data.

Code Components Overview

**1. Dependencies**

The server uses:

Flask: To handle HTTP requests and responses.

SQLite: For database management.

Struct: To handle binary fingerprint data from the sensor.

Install dependencies using:

*pip install flask*

**2. Database Initialization**

The database is initialized with a table named fingerprints:

**sql**

*CREATE TABLE IF NOT EXISTS fingerprints (*

*id INTEGER PRIMARY KEY AUTOINCREMENT,*

*username TEXT NOT NULL,*

*template BLOB NOT NULL*

*);*

This table stores:

id: A unique identifier for each fingerprint.

username: The name associated with the fingerprint.

template: The binary data of the fingerprint template.

**3. Server Modes**

The server operates in two modes:

**Enroll:** For adding new fingerprints to the database.

**Detect:** For matching incoming fingerprints with stored templates.

When the server starts, it prompts the user to select a mode:

Select server mode (enroll/detect):

**4. Fingerprint Data Processing**

Extraction

The function extract\_fingerprint\_data processes raw fingerprint data received from the microcontroller. It:

Identifies packets using the R502-A header (*0xEF01FFFFFF*).

Extracts the payload, removes unnecessary padding, and combines the data into a single fingerprint template.

Similarity Calculation

The function calculate\_similarity computes the byte-by-byte similarity percentage between two fingerprint templates:

python

*(matches / len(shorter)) \* 100*

This percentage helps determine if the fingerprint matches a stored template.

**5. API Endpoints**

**5.1. GET /get\_mode**

Returns the current server mode:

**json**

*{*

*"mode": "enroll" // or "detect"*

*}*

**5.2. POST /upload**

Handles fingerprint enrollment:

Extracts the fingerprint template from the request.

Saves the template to the database with a username.

Example response:

**json**

*{*

*"status": "success",*

*"message": "Fingerprint saved for John."*

*}*

**5.3. POST /detect**

Handles fingerprint detection:

Extracts the fingerprint template from the request.

Compares it with stored templates.

Returns the best match if similarity exceeds 80%.

Example response:

**json**

*{*

*"status": "success",*

*"message": "Match found for John (Similarity: 92.35%)"*

*}*

**6. How to Run the Server**

Start the server:

**bash**

*python ByteByByte\_Matching\_complete.py*

Select a mode (enroll or detect).

Use a microcontroller (e.g., ESP32) to send fingerprint data to the server via HTTP.

**7. Usage Flow**

Enrollment

Select "enroll" mode.

Use the microcontroller to send a POST request with fingerprint data to /upload.

Enter a username when prompted.

Detection

Select "detect" mode.

Send a POST request with fingerprint data to /detect.

View the response to see if a match is found.

**8. Important Notes**

Threshold: The similarity threshold for a match is 80%. Adjust this in the /detect route as needed.

Data Size: Ensure that the microcontroller sends fingerprint data in the expected format with a valid header.

Debugging: Logs print packet details and similarity percentages to the console for debugging.