**University Project: Fingerprint-Based Attendance System**

**Team Members:**

* Edon Berisha – Backend Developer (C# ASP.NET Core, SQL, SignalR)
* Alban Rrahmani – Hardware-Engineer & Full-Stack Developer
* Petrit Rexha – Backend Developer and Documentation
* Leutrim Istrefi – Overseer and Documentation

## Sprint 1 Documentation – Arduino Fingerprint Logic and Initial Integration

**Sprint Duration:** 1 week  
**Main Goal:** Establish and test the core fingerprint functionality using an Arduino microcontroller with a fingerprint sensor module.

### 1. Objectives

* Connect fingerprint sensor to Arduino.
* Enable enrollment of new fingerprints.
* Verify fingerprints using the stored fingerprint templates on the sensor.
* Establish basic serial communication between Arduino and the C# backend.
* Confirm Arduino readiness by sending a known message over serial ("ArduinoFingerPrintSensorReady").

### 2. Arduino Development

#include <Adafruit\_Fingerprint.h>

#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3); // RX, TX

Adafruit\_Fingerprint finger = Adafruit\_Fingerprint(&mySerial);

bool enrolling = false;

bool verifying = false;

bool sessionActive = false;

int userIDs[] = {1, 2, 3}; // List of user IDs to verify

int currentUserIndex = 0; // Index to track which user we're verifying

void setup() {

Serial.begin(9600);

while (!Serial); // Wait for serial connection

delay(100);

finger.begin(57600);

if (finger.verifyPassword()) {

Serial.println("Found fingerprint sensor!");

} else {

Serial.println("Did not find fingerprint sensor :(");

while (1) { delay(1); }

}

}

void loop() {

// Handle serial commands

if (Serial.available()) {

String command = Serial.readStringUntil('\n');

command.trim();

if (command == "ENROLL") {

enrolling = true;

Serial.println("Starting enrollment...");

enrollFingerprint();

} else if (command == "VERIFY") {

verifying = true;

sessionActive = true;

currentUserIndex = 0; // Start from the first user

Serial.println("Verification session started.");

} else if (command == "ENDSESSION") {

sessionActive = false;

verifying = false;

Serial.println("Session ended.");

} else {

Serial.println("Unknown command");

}

}

// Perform fingerprint verification if in session

if (verifying && sessionActive) {

verifyFingerprint();

}

}

void enrollFingerprint() {

int id = 1; // You can make this dynamic

Serial.print("Enrolling ID #"); Serial.println(id);

int p = -1;

Serial.println("Place finger on sensor...");

while (p != FINGERPRINT\_OK) {

p = finger.getImage();

if (p == FINGERPRINT\_NOFINGER) {

delay(100);

} else if (p != FINGERPRINT\_OK) {

Serial.println("Error capturing image. Try again.");

}

}

p = finger.image2Tz(1);

if (p != FINGERPRINT\_OK) {

Serial.println("Image conversion failed.");

return;

}

Serial.println("Remove finger...");

delay(2000);

while (finger.getImage() != FINGERPRINT\_NOFINGER);

Serial.println("Place same finger again...");

p = -1;

while (p != FINGERPRINT\_OK) {

p = finger.getImage();

if (p == FINGERPRINT\_NOFINGER) {

delay(100);

} else if (p != FINGERPRINT\_OK) {

Serial.println("Error capturing image. Try again.");

}

}

p = finger.image2Tz(2);

if (p != FINGERPRINT\_OK) {

Serial.println("Image conversion failed (2nd try).");

return;

}

p = finger.createModel();

if (p != FINGERPRINT\_OK) {

Serial.println("Failed to create model.");

return;

}

p = finger.storeModel(id);

if (p == FINGERPRINT\_OK) {

Serial.println("Fingerprint enrolled successfully.");

} else {

Serial.println("Failed to store fingerprint.");

}

enrolling = false;

}

// Non-blocking fingerprint check

void verifyFingerprint() {

static bool prompted = false;

if (!prompted) {

Serial.println("Waiting for a valid finger to verify...");

prompted = true;

}

uint8\_t p = finger.getImage();

if (p == FINGERPRINT\_NOFINGER) {

delay(200); // No finger yet

return;

} else if (p != FINGERPRINT\_OK) {

Serial.println("Image capture error");

return;

}

p = finger.image2Tz();

if (p != FINGERPRINT\_OK) {

Serial.println("Image conversion failed");

return;

}

p = finger.fingerSearch();

if (p == FINGERPRINT\_OK) {

Serial.print("Match found! ID: ");

Serial.print(finger.fingerID);

Serial.print(" with confidence ");

Serial.println(finger.confidence);

if (finger.fingerID == userIDs[currentUserIndex]) {

Serial.println("User verified successfully.");

currentUserIndex++; // Move to next user

// If all users are verified, end the session

if (currentUserIndex >= sizeof(userIDs) / sizeof(userIDs[0])) {

Serial.println("All users verified. Ending session.");

verifying = false;

sessionActive = false;

}

} else {

Serial.println("Verification failed.");

}

} else if (p == FINGERPRINT\_NOTFOUND) {

Serial.println("No match found.");

} else {

Serial.println("Search error.");

}

}

### 3. Database Planning

We created an SQL schema designed to work with ASP.NET Identity. Student fingerprint data was linked via a FingerprintID field in the Students table.

**Relevant Tables:**

* Students (StudentID, Name, FingerprintID,Attendances)
* Attendances (AttendanceID, StudentID, SubjectID, LessonDate, Present)
* SessionState (SubjectID, IsActive, StartDate,Subject)
* Subject (SubjectID, SubjectName, ProfessorID)

### 

### 

### 4. Challenges

* **False Positives**: When comparing fingerprint hashes in C#, the system occasionally marked the wrong student as present.
* **Sensor Reliability**: Fingerprint recognition sometimes failed due to dirty or poorly placed fingers.

### 5. Resolution

* The fingerprint matching logic was moved entirely to the Arduino.
* Arduino reports only a success/failure message via serial, reducing C# logic errors.

## Sprint 2 Documentation – Backend Integration and SignalR Communication

**Sprint Duration:** 1 week  
**Main Goal:** Connect Arduino with C# backend, integrate SQL Server, and develop live real-time attendance logging.

### 1. C# Backend

**Framework:** ASP.NET Core Web Application  
**Structure:**

* /Services/ArduinoService.cs – Manages dynamic COM port detection and serial communication.
* /Hubs/ArduinoHub.cs – SignalR hub for real-time updates to frontend.
* /Data/AppDbContext.cs – Handles EF Core integration with the database.

**Key Features:**

* Auto-detects Arduino by comparing system COM ports and verifying via a special message.
* Broadcasts fingerprint scan results to the front-end via SignalR (ReceiveSerialLog).
* Uses a command protocol (e.g., SendCommand("e")) to instruct Arduino to enroll or verify.

### 

### 2. Code Example – ArduinoService.cs (simplified overview)

public class ArduinoService {

private SerialPort \_serialPort;

private readonly IHubContext<ArduinoHub> \_hubContext;

public ArduinoService(IHubContext<ArduinoHub> hubContext) {

\_hubContext = hubContext;

Task.Run(() => InitializeSerialPort());

}

private void InitializeSerialPort() {

// Dynamically detect and verify Arduino

// Subscribe to SerialDataReceived

}

private void SerialDataReceived(object sender, SerialDataReceivedEventArgs e) {

string data = \_serialPort.ReadLine();

\_hubContext.Clients.All.SendAsync("ReceiveSerialLog", data);

}

}

### 3. Connection String

"ConnectionStrings": {

"DefaultConnectionString": "Server=.;Database=AttendanceDB;Trusted\_Connection=True;"

}

### 

### 

### 4. Role Initialization

* Users are assigned roles (Student, Professor, Admin) via ASP.NET Identity.
* Students link to their AspNetUsers.Id with UserId foreign key.

### 5. Testing

* Team tested using personal fingerprints for enrollment and repeated verification.
* After moving logic to Arduino, accuracy increased significantly.
* Manual comparisons via C# led to false positives—Arduino-only comparison resolved this.

### 6. Result

* Arduino successfully handles all fingerprint logic.
* C# backend receives verified attendance events.
* Events are written to SQL Server with associated student and subject.

### 7. Project Highlights

* Real-time integration via SignalR.
* Secure user handling through ASP.NET Identity.
* Error recovery and logging on serial connection failures.

### 8. Remaining Tasks

* Final UI polish.
* Professor dashboard improvements.
* Automated testing coverage (basic scripts completed).

**Conclusion:** Sprints 1 and 2 successfully completed the foundation of the fingerprint attendance system. All hardware/software integration is working as intended. Moving fingerprint matching to the Arduino solved key recognition issues. The backend now robustly supports student enrollment, attendance tracking, and real-time updates using SignalR.

The team is ahead of schedule and preparing documentation and polish tasks for Sprint 3.

### 

### 

### 9.Repository

* All code has been uploaded to a GitHub repository, including:
  + Arduino code for fingerprint sensor.
  + C# backend with ASP.NET Core and SignalR integration.
  + SQL schema and initial data.

[GIthub](https://github.com/edonberishaa/AttendanceSystem)

### 10.User Instructions

**For Students:**

* Place finger on sensor.
* If match is found, attendance is marked.

**For Professors:**

* Login to dashboard.
* View attendance by subject and date.

**For Admins:**

* Manage users, assign roles.
* Monitor system status and logs.

https://github.com/edonberishaa/AttendanceSystem