Section - D Table No. - 17

Group Members: Archit Atrey (220195), Manvi Verma (220631)

NFC-Based HealthCare Access System

Problem Description

Healthcare access for marginalized or underserved populations is often hampered by inefficient patient identification processes and medical record management. These populations face significant barriers including:

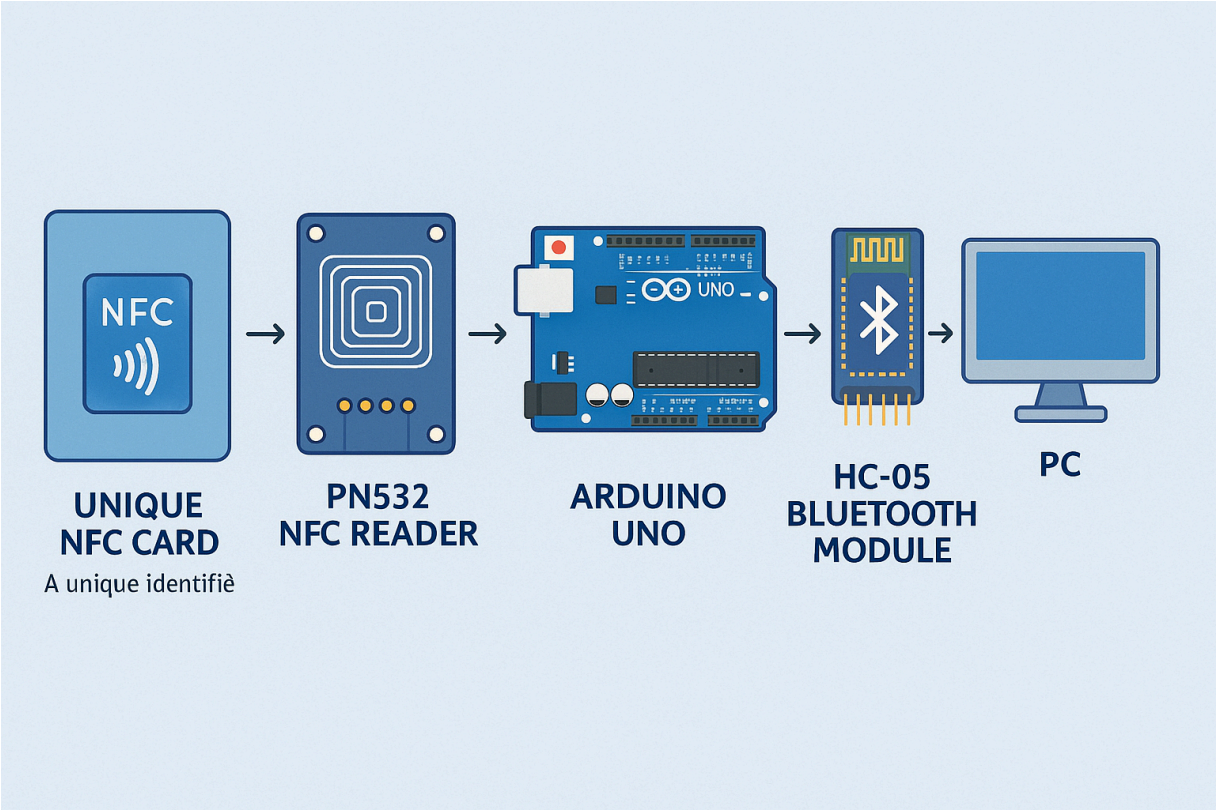
- Lack of proper identification documents, leading to difficulties in maintaining continuity of care.
- Inefficient or inaccessible medical record systems resulting in repeated tests and procedures.
- Medical errors due to unavailability of complete patient information.
- Delays in treatment caused by manual patient data retrieval.
- Limited healthcare access in remote or resource-constrained areas.

These challenges collectively lead to suboptimal healthcare delivery, misdiagnosis, delayed treatment, and reduced overall healthcare quality for vulnerable populations. The proposed system aims to address these issues through an innovative approach using widely available technology.

System Objectives

1. Create an affordable, efficient patient identification system using NFC technology
2. Enable quick and secure access to patient medical records
3. Eliminate dependence on physical documents
4. Reduce medical errors through accurate patient identification
5. Improve healthcare delivery for marginalized populations

System Block Diagram



Control Flow for Secure Patient Data Access via NFC & Bluetooth

1. **NFC Tag Scan (User Authentication Initiation)**
   - Patient brings an **NFC tag or phone** (with UID) near the **PN532 NFC Reader**, being used in I2C mode.
   - The **Arduino** reads the **UID** from the NFC tag.

2. **Send UID to Bluetooth Module (HC-05)**
   - Arduino sends the scanned **UID** securely to the **HC-05 Bluetooth Module**.

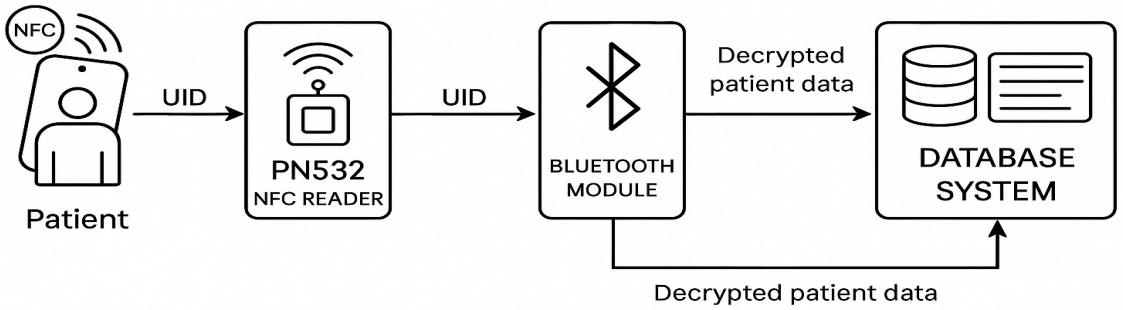
3. **Transmit UID to Secure Backend System**
   - The **Bluetooth module** (HC-05) sends this UID to a **paired secure system/server** (e.g., a local secure PC or Raspberry Pi acting as the database server).
   - This system has **exclusive Bluetooth access**—not connected to the internet or any external interface for security.

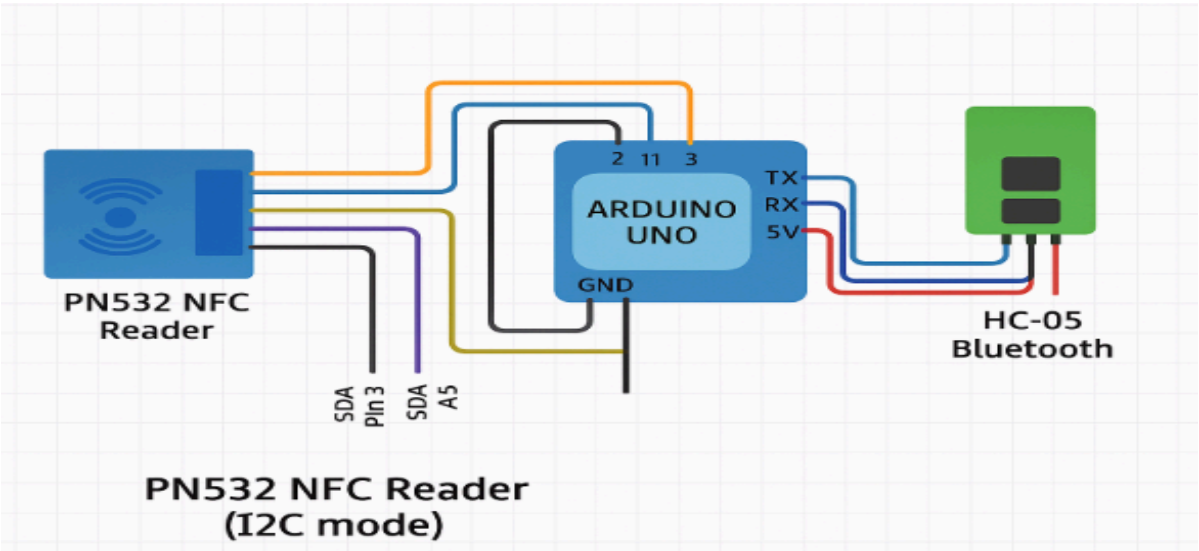
4. **Validation & Authentication**
   - The secure system receives the UID.
   - It **authenticates** the UID and checks if it is **valid and registered**.
   - If the UID is valid, it proceeds to **decrypt the corresponding encrypted**

patient file using stored keys and internal logic.

5. **Send Decrypted Data to Bluetooth Module**
   - Once decrypted, the patient's **relevant health data (file/summary)** is sent **back via Bluetooth** to the **HC-05 module**.

6. **Showcase Data on Main System (Arduino UI)**
   - Arduino receives the decrypted patient data via serial communication from the HC-05.
   - This data is then **displayed on a screen/interface** (e.g., LCD/OLED or serial monitor) connected to the Arduino.
   - This ensures **only authorized access via NFC & secure Bluetooth transmission**, **no direct access** to keys, UIDs, or encrypted files.





Software Implementation

1) Check if the NFC card is detected with its UID.
   Arduino Code for the same is given below:

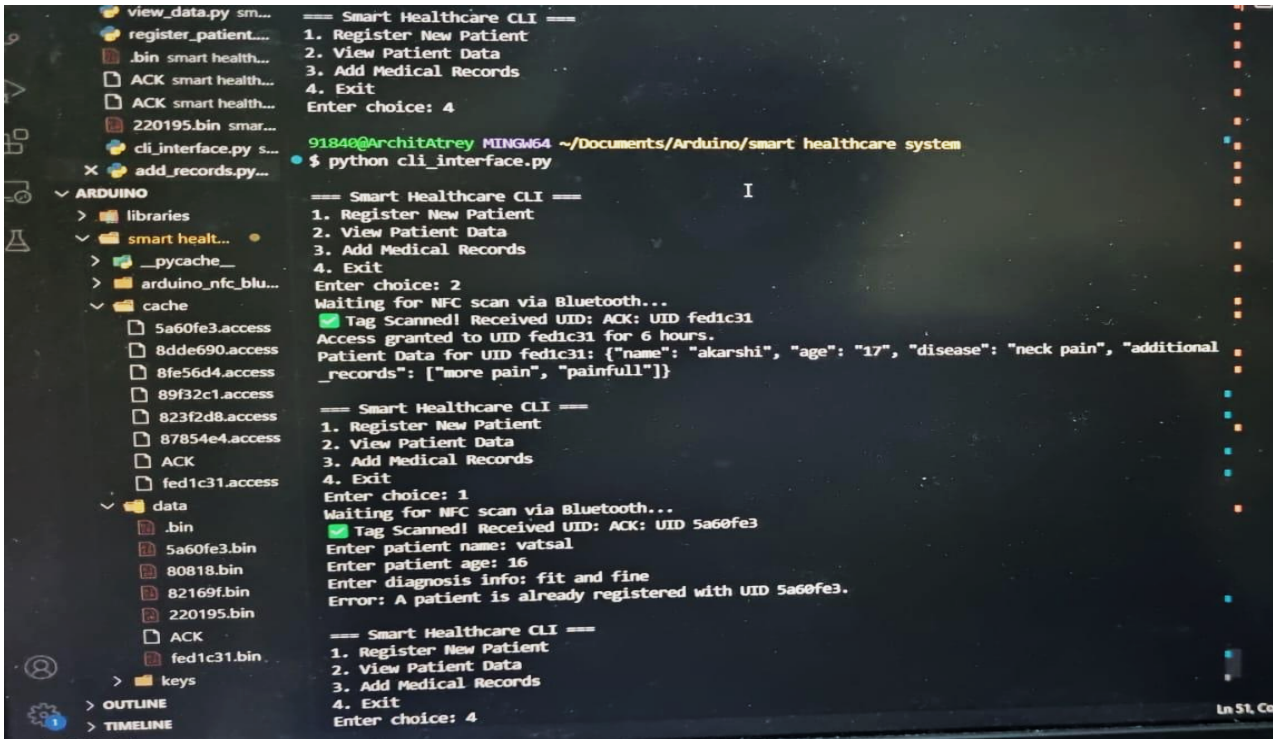
arduino_nfc_bluetooth.ino

```ino
#include <Wire.h>
#include <Adafruit_PN532.h>
#include <SoftwareSerial.h>

// Define PN532 pins for I2C mode
#define PN532_IRQ   2   // Connect PN532 IRQ to Arduino digital pin 2
#define PN532_RESET 3   // Connect PN532 RESET to Arduino digital pin 3

// Create an instance for PN532 using I2C with IRQ and RESET
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET, &Wire);

// Create a SoftwareSerial instance for the HC-05 Bluetooth
SoftwareSerial BT(10, 11); // RX, TX for HC-05

void setup(void) {
  Serial.begin(115200);
  BT.begin(9600); // HC-05 default baud rate

  Serial.println("Initializing PN532 (I2C mode)...");

  nfc.begin();

  uint32_t versiondata = nfc.getFirmwareVersion();
  if (!versiondata) {
    Serial.println("Didn't find PN53x board; check wiring!");
    while (1); // halt if the PN532 is not found
  }

  Serial.print("Found chip PN5");
  Serial.println((versiondata >> 24) & 0xFF, HEX);
  Serial.print("Firmware ver. ");
  Serial.print((versiondata >> 16) & 0xFF, DEC);
  Serial.print('.');
  Serial.println((versiondata >> 8) & 0xFF, DEC);
```

arduino_nfc_bluetooth.ino

```ino
  // Configure PN532 to read RFID tags
  nfc.SAMConfig();
  Serial.println("PN532 is ready. Waiting for an NFC tag...");
}

void loop(void) {
  uint8_t uid[7];    // Buffer to store the UID
  uint8_t uidLength; // Length of the UID

  // Read an ISO14443A type tag (NFC tag or phone) using PN532
  if (nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength)) {
    Serial.print("Tag detected! UID: ");
    String uidStr = "";
    for (uint8_t i = 0; i < uidLength; i++) {
      if (uid[i] < 0x10) Serial.print("0");
      Serial.print(uid[i], HEX);
      uidStr += String(uid[i], HEX);
      Serial.print(" ");
    }
    Serial.println();

    // Acknowledge successful scan via Bluetooth
    BT.print("ACK: UID ");
    BT.println(uidStr);

    delay(2000); // Delay to avoid reading the same tag multiple times rapidly
  }

  // (Optional) Echo data between Serial Monitor and HC-05
  if (BT.available()) {
    char c = BT.read();
    Serial.write(c);
  }

  if (Serial.available()) {
    char c = Serial.read();
    BT.write(c);
  }
```

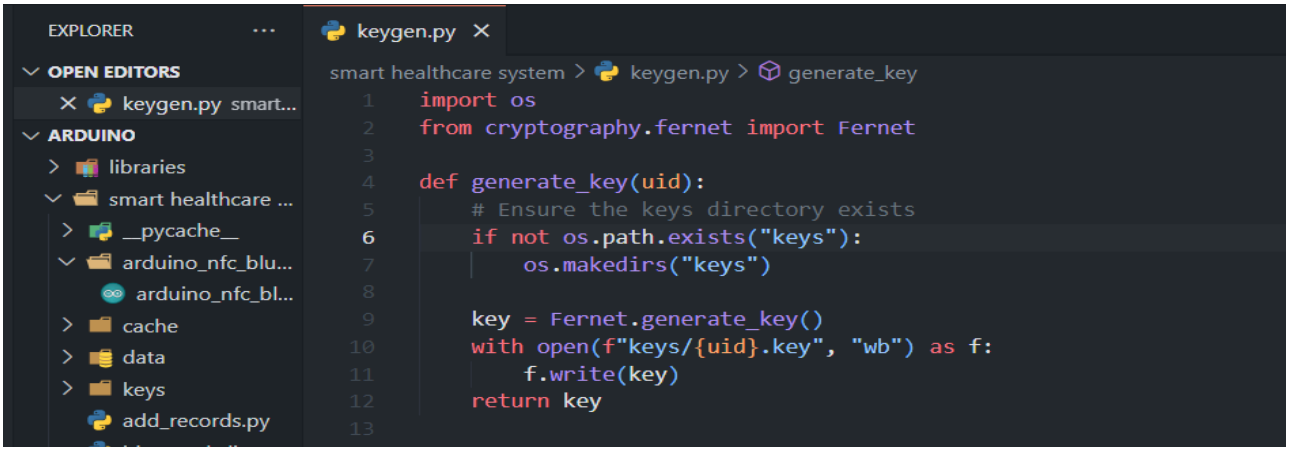
2) After detecting the tag, using python code we can register the new patient with its medical history details for that UID of that tag, then we can display the details of that patient using that tag.



Python Backend Scripts

The following Python files live in /Arduino/smart healthcare system/ and together provide patient registration, secure storage, and record-viewing functionality over Bluetooth.

keygen.py

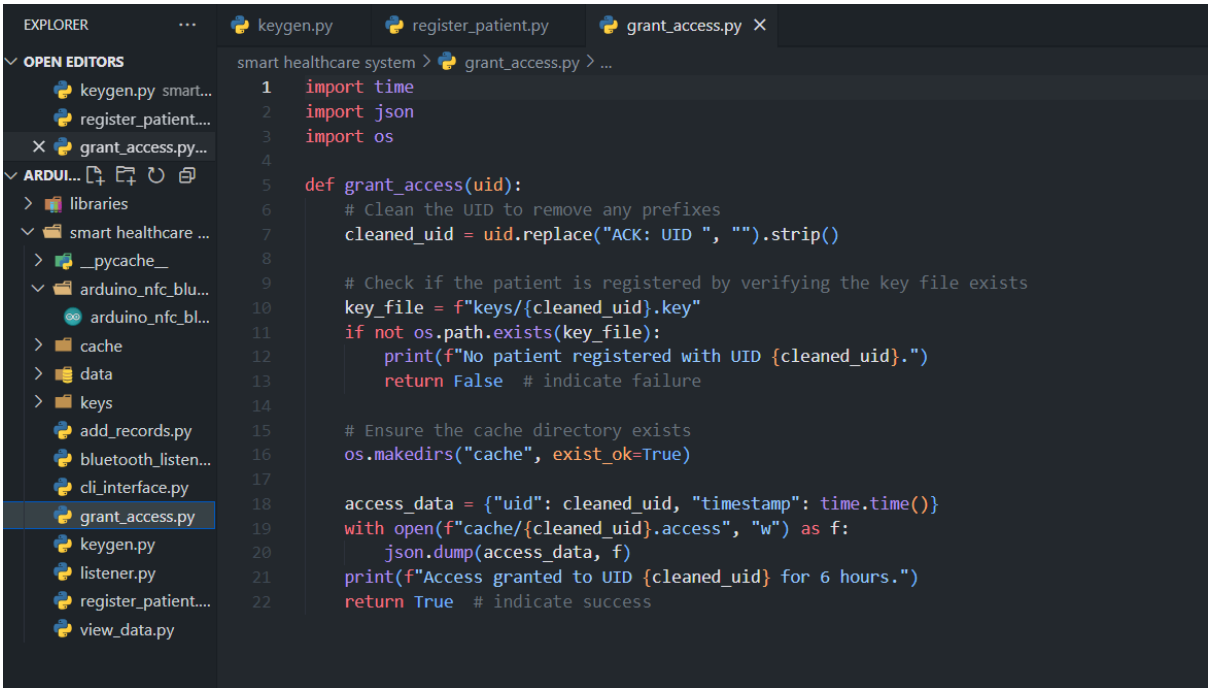


Purpose: Generates a new symmetric encryption key for a patient's UID, saving it under keys/{uid}.key. This key is used by other scripts to encrypt or decrypt that patient's data.

register_patient.py

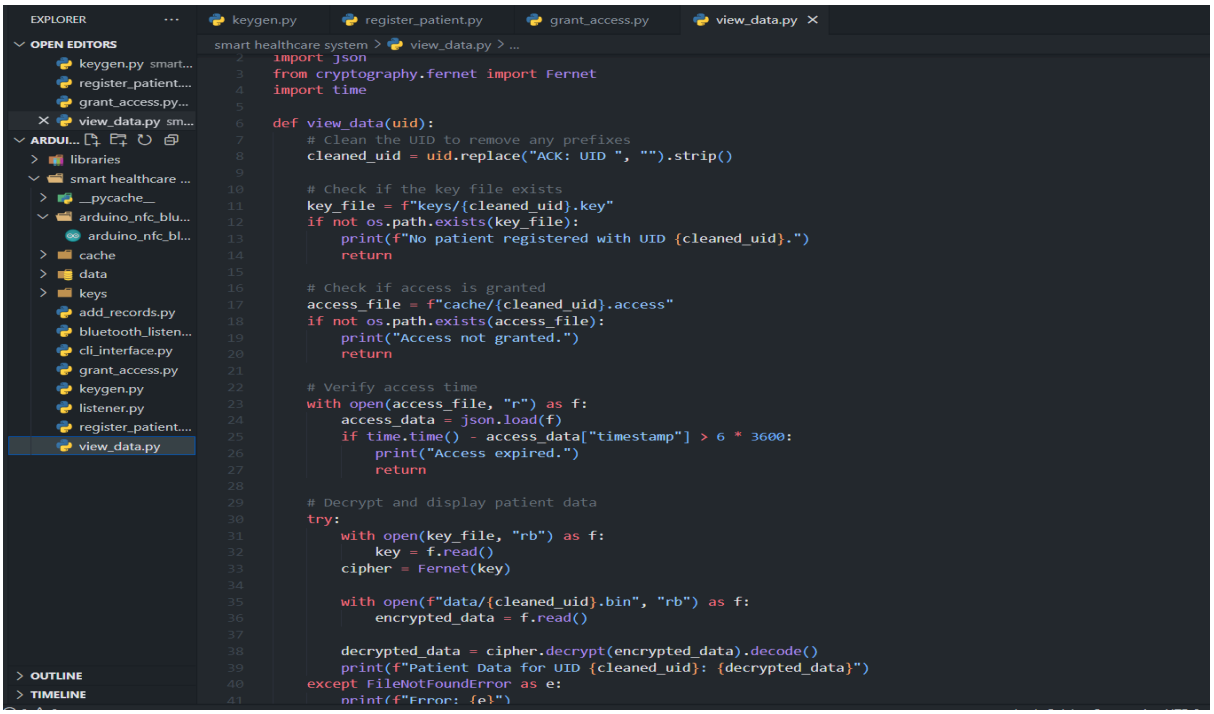
Purpose: Called when a new patient is scanned and their demographic / medical-history JSON is supplied. It generates a key (via keygen.py), encrypts the JSON, and writes it to data/{uid}.bin.

grant_access.py



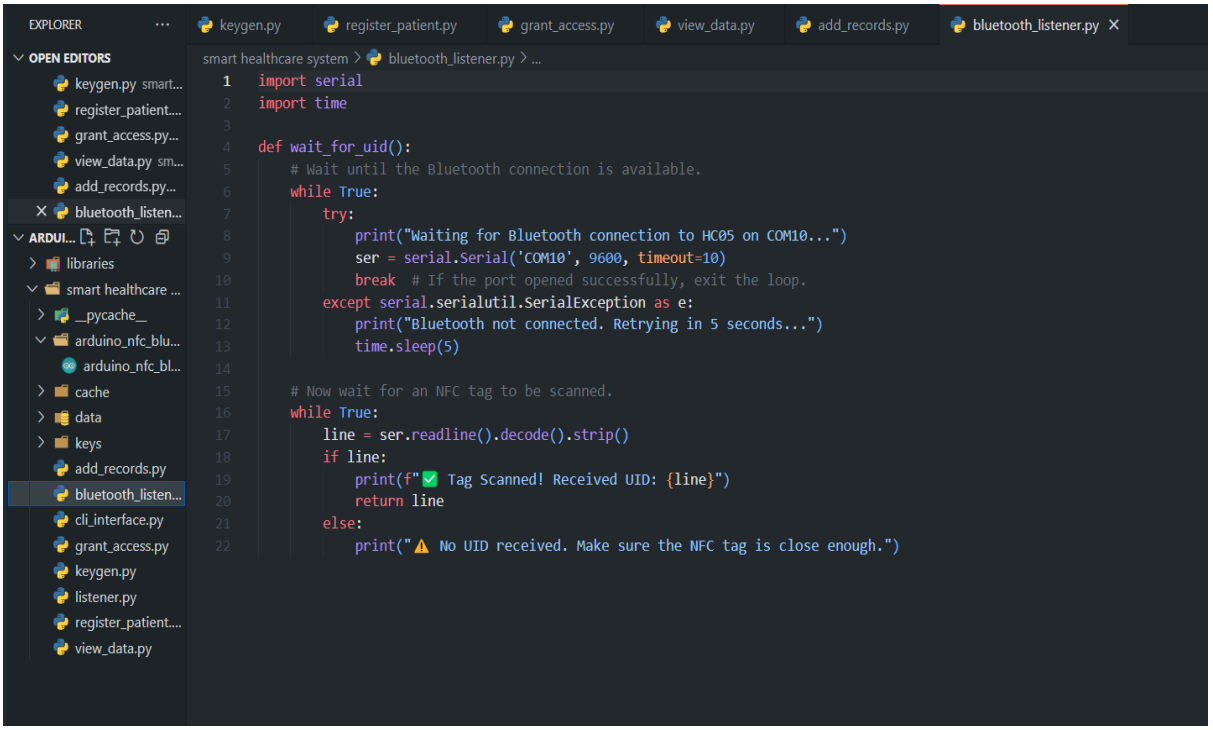
Purpose: Verifies that a scanned UID is registered, then writes a timestamped “access granted” file to cache/. Other scripts check cache/ to ensure only recently-authorized requests succeed.

view_data.py



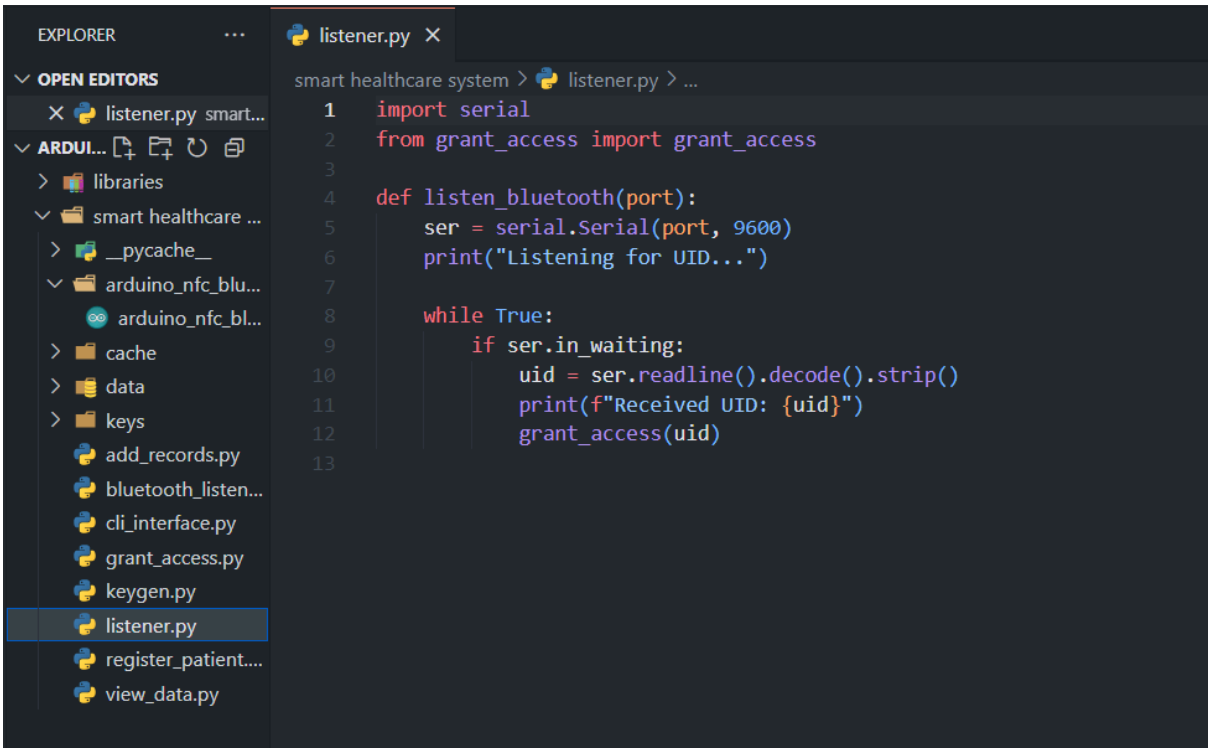
Purpose: After access has been granted, this script decrypts the patient’s stored JSON data and prints it to the console.

bluetooth_listener.py



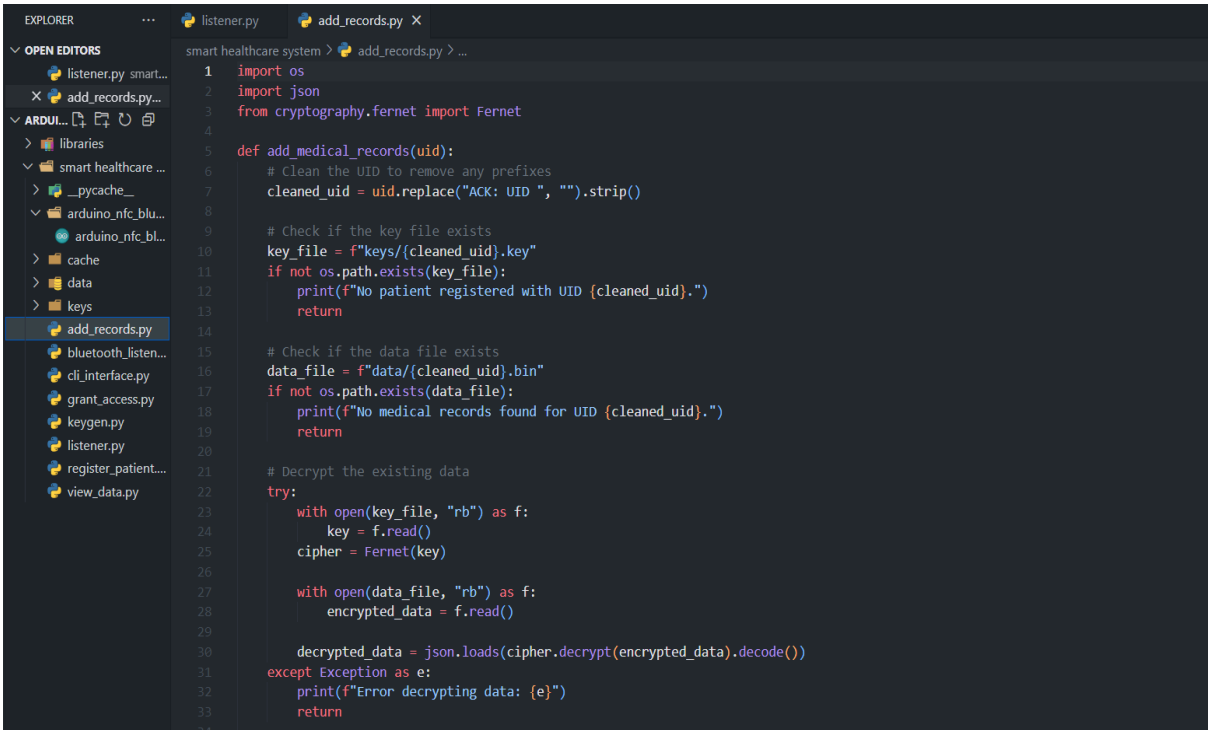
Purpose: Blocks until the HC-05 Bluetooth link is up, then reads and returns the next NFC-UID string sent by the Arduino.

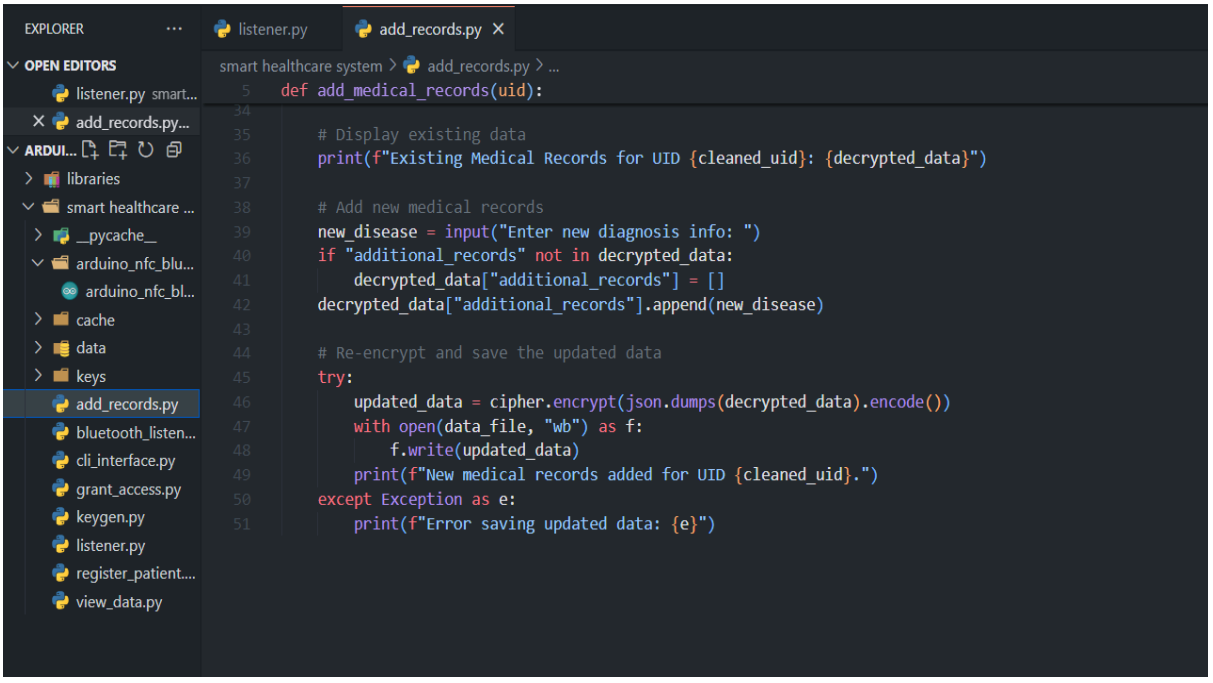
listener.py



Purpose: A simple daemon that listens on a serial port for incoming UIDs and immediately calls grant_access(uid) to authorize them.

add_records.py

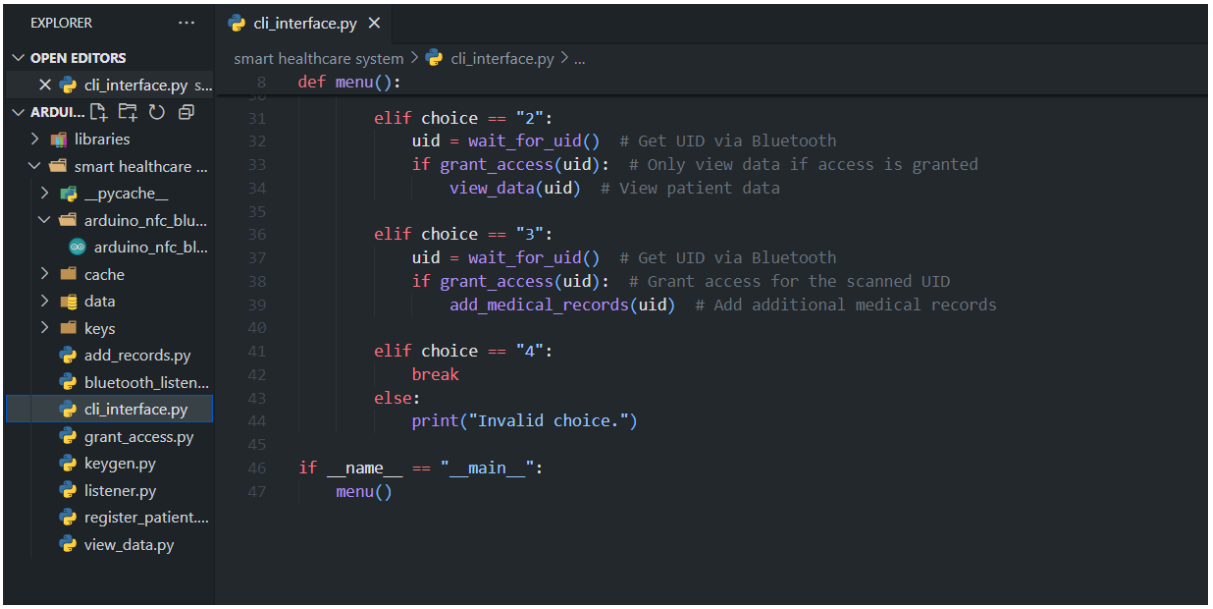




Purpose: Loads and decrypts existing patient data, prompts the user to enter new medical-history items, appends them, then re-encrypts and saves back to data/{uid}.bin.

cli_interface.py

```python
import os
from register_patient import register_patient
from view_data import view_data
from grant_access import grant_access
from bluetooth_listener import wait_for_uid
from add_records import add_medical_records

def menu():
    while True:
        print("\n=== Smart Healthcare CLI ===")
        print("1. Register New Patient")
        print("2. View Patient Data")
        print("3. Add Medical Records")
        print("4. Exit")

        choice = input("Enter choice: ")

        if choice == "1":
            uid = wait_for_uid()
            cleaned_uid = uid.replace("ACK: UID ", "").strip()
            # Check if the patient is already registered by checking for the key file
            if os.path.exists(f"keys/{cleaned_uid}.key"):
                print(f"Patient with UID {cleaned_uid} is already registered.")
            else:
                name = input("Enter patient name: ")
                age = input("Enter patient age: ")
                disease = input("Enter diagnosis info: ")
                data = {"name": name, "age": age, "disease": disease}
                register_patient(uid, data)
```



```python
def menu():
        elif choice == "2":
            uid = wait_for_uid()  # Get UID via Bluetooth
            if grant_access(uid):  # Only view data if access is granted
                view_data(uid)  # View patient data

        elif choice == "3":
            uid = wait_for_uid()  # Get UID via Bluetooth
            if grant_access(uid):  # Grant access for the scanned UID
                add_medical_records(uid)  # Add additional medical records

        elif choice == "4":
            break
        else:
            print("Invalid choice.")

if __name__ == "__main__":
    menu()
```

Purpose: Provides a single-entry command-line menu that ties together all functionality: scanning a UID over Bluetooth, then registering, viewing, or updating a patient's encrypted record.

How It All Fits Together:

1. **Arduino → HC-05** sends raw UID over Bluetooth when an NFC tag/phone is tapped.
2. **bluetooth_listener.py** picks up that UID and returns it.
3. **cli_interface.py** drives the workflow: on "Register" it calls register_patient.py; on "View"/"Add" it first calls grant_access.py then either view_data.py or add_records.py.
4. **keygen.py** underpins secure storage by generating per-patient keys.

5. All JSON patient data lives encrypted in data/; encryption keys in keys/; temporary access flags in cache/.

Encryption Mechanism & File-Storage Formats

Encryption Scheme

All patient files are encrypted using the **Fernet** implementation from the Python `cryptography` library. Fernet provides:

- **Symmetric AES-128 in CBC mode** under the hood
- **Per-message IV:** a fresh 16-byte random IV on every encryption
- **HMAC-SHA256** for integrity/authenticity over the entire payload
- **Timestamping:** embeds an 8-byte UNIX timestamp so you can later enforce expiration if desired
- **Single URL-safe Base64 token** that concatenates:
    1. Version byte
    2. Timestamp
    3. IV
    4. Ciphertext
    5. HMAC signature

Because the final token is base64-encoded, it's easy to store as text or binary on disk.

Key Files

- Location:

```
keys/{UID}.key
```

Content: the 32-byte Fernet key, URL-safe Base64-encoded (44 ASCII chars + newline).
Handling: created by `keygen.py`, tested for existence by every reader/writer.

Encrypted Data Files

- Location:

```
data/{UID}.bin
```

Content: the raw bytes of `Fernet.encrypt(...)`—i.e. a URL-safe Base64 string, ~300–400 bytes for a small JSON.
Format: binary mode write, but file contents are ASCII Base64. Decrypted back to the original JSON with `Fernet.decrypt()`.

Cache Files (Access Tokens)

- Location:

```
cache/{UID}.access
```

Content: plain-text JSON, e.g.

```
{"uid":"5a60fe3","timestamp":1711109732.12345}
```

Purpose: track when `grant_access()` was last invoked so you can enforce a time-window (e.g. 6 hours) on subsequent reads or record additions.

Raw Patient Data (Pre-Encryption)

Always a JSON-serializable Python `dict`, for example:

```
{
  "name": "Archit Atrey",
  "age": 23,
  "history": ["hypertension", "allergy: penicillin"],
  "additional_records": []
}
```

This is dumped via `json.dumps(...)` before being encrypted, and re-loaded after decrypting.

System Architecture

The proposed healthcare access system consists of four main components:

1. Patient Identification Module:
    - Uses NFC-enabled smartphones instead of traditional RFID tags
    - Stores a unique patient identifier in the phone's NFC chip
    - Eliminates need for physical cards or wristbands
2. Data Reading Module:
    - PN532 NFC Reader connected to Arduino UNO
    - Reads patient identification data from NFC-enabled phones
    - Processes the information for transmission
3. Data Transmission Module:
    - HC-05 Bluetooth module for wireless communication
    - Transmits patient identification data to healthcare provider's device
    - Enables local operation without requiring internet connectivity
4. Database Management Module:
    - Stores comprehensive patient health records
    - Links records to unique patient identifiers
    - Provides secure access to authorized healthcare providers

Key Results

The implementation of this system is expected to deliver the following key outcomes:

1. Improved Patient Identification: Elimination of patient misidentification errors through unique NFC identifiers.
2. Enhanced Efficiency: Reduction in patient registration and record retrieval time from minutes to seconds.
3. Reduced Barriers to Care: Removal of dependency on physical identification documents, particularly benefiting marginalized populations.
4. Better Continuity of Care: Complete medical history available to healthcare providers at point of care.
5. Cost-Effective Solution: Utilization of widely available technology (smartphones, Arduino) reduces implementation costs compared to proprietary systems.

Conclusion

The RFID/NFC-Based Healthcare Access System offers a practical, affordable solution to address healthcare access challenges faced by marginalized populations. By utilizing widely available technology (smartphones, Arduino components), the system overcomes barriers to healthcare while ensuring security and privacy of patient information.

The proof-of-concept demonstrates that NFC-enabled mobile phones can effectively replace dedicated RFID tags, making the system more accessible and user-friendly. Bluetooth-based communication enables operation in areas with limited connectivity, ensuring that healthcare facilities in underserved areas can benefit from this technology.

This project directly addresses the needs of marginalized populations by simplifying patient identification, streamlining healthcare delivery, and reducing dependence on physical documentation. The system's implementation could significantly improve healthcare outcomes for vulnerable groups by ensuring continuous, error-free access to medical care.