Fully Automatic Home Security System (Code)

Facial Recognition :

```
#include "esp_camera.h"
#include <WiFi.h>
#define CAMERA_MODEL_AI_THINKER
#define Relay 2
#define Red 13
#define Green 12
#include "camera_pins.h"
const char* ssid = "----------"; //Wifi Name SSID
const char* password = "----------"; //WIFI Password
void startCameraServer();
boolean matchFace = false;
boolean activateRelay = false;
long prevMillis=0;
int interval = 5000;
void setup() {
  pinMode(Relay,OUTPUT);
  pinMode(Red,OUTPUT);
  pinMode(Green,OUTPUT);
  digitalWrite(Relay,LOW);
  digitalWrite(Red,HIGH);
  digitalWrite(Green,LOW);
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
```

```
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  //init with high specs to pre-allocate larger buffers
  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
```

```
  pinMode(14, INPUT_PULLUP);
#endif
  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
  sensor_t * s = esp_camera_sensor_get();
  //initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);//flip it back
    s->set_brightness(s, 1);//up the blightness just a bit
    s->set_saturation(s, -2);//lower the saturation
  }
  //drop down frame size for higher initial frame rate
  s->set_framesize(s, FRAMESIZE_QVGA);
#if defined(CAMERA_MODEL_M5STACK_WIDE)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  startCameraServer();
  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
```

```
  Serial.println("' to connect");
}
void loop() {
  if(matchFace==true && activateRelay==false)
  {
    activateRelay=true;
    digitalWrite(Relay,HIGH);
    digitalWrite(Green,HIGH);
    digitalWrite(Red,LOW);
    prevMillis=millis();
    }
    if (activateRelay == true && millis()-prevMillis > interval)
    {
      activateRelay=false;
      matchFace=false;
      digitalWrite(Relay,LOW);
      digitalWrite(Green,LOW);
      digitalWrite(Red,HIGH);
      }
}
```

Biometric Recognition :

```
#include <Adafruit_Fingerprint.h>
#include <SoftwareSerial.h>
// Pin definitions
#define RELAY_PIN 8          // Relay pin to control solenoid lock
#define BUZZER_PIN 9         // Buzzer pin for unauthorized access
#define FINGERPRINT_RX 2     // Fingerprint sensor RX
#define FINGERPRINT_TX 3     // Fingerprint sensor TX
#define GSM_RX 10            // GSM module RX
#define GSM_TX 11            // GSM module TX
SoftwareSerial fingerprintSerial(FINGERPRINT_RX, FINGERPRINT_TX);
Adafruit_Fingerprint finger = Adafruit_Fingerprint(&fingerprintSerial);
SoftwareSerial gsmSerial(GSM_RX, GSM_TX);
void setup() {
  Serial.begin(9600);
  fingerprintSerial.begin(57600);
  gsmSerial.begin(9600);
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);  // Initially keep the lock closed
  digitalWrite(BUZZER_PIN, LOW); // Buzzer off initially
  if (finger.verifyPassword()) {
    Serial.println("Fingerprint sensor found.");
  } else {
    Serial.println("Fingerprint sensor not detected!");
    while (1);
  }
}
void loop() {
  Serial.println("Place finger on sensor...");
  int fingerprintID = getFingerprintID();
```

```
  if (fingerprintID == -1) {
    // Unauthorized access attempt
    Serial.println("Unauthorized access attempt!");
    digitalWrite(BUZZER_PIN, HIGH);  // Turn on buzzer
    delay(1000);                     // Buzzer sound for 1 second
    digitalWrite(BUZZER_PIN, LOW);    // Turn off buzzer
  } else {
    // Authorized access
    Serial.print("Access granted for ID: ");
    Serial.println(fingerprintID);
    digitalWrite(RELAY_PIN, HIGH); // Unlock the solenoid lock
    delay(5000);                   // Keep lock open for 5 seconds
    digitalWrite(RELAY_PIN, LOW);  // Lock again
    // Send SMS for authorized access
    String message = "Access granted for ID: " + String(fingerprintID);
    sendSMS(message);
  }
  delay(2000); // Short delay before next fingerprint scan
}
int getFingerprintID() {
  int result = finger.getImage();
  if (result != FINGERPRINT_OK) return -1;
  result = finger.image2Tz();
  if (result != FINGERPRINT_OK) return -1;
  result = finger.fingerFastSearch();
  if (result != FINGERPRINT_OK) return -1;
  return finger.fingerID;
}
void sendSMS(String message) {
  gsmSerial.println("AT+CMGF=1"); // Set GSM module to text mode
  delay(100);
  gsmSerial.println("AT+CMGS=\"+1234567890\""); // Replace with actual phone number
```

```
  delay(100);
  gsmSerial.println(message);
  delay(100);
  gsmSerial.write(26); // Send Ctrl+Z to send the message
  delay(1000);
}
```

Gas Leakage Detection & Wet Flour Detection (Integrated) :

```
int watersense = A0;    // Analog sensor input pin 0 (waterdrop module output)
int gassense = A1;      // Analog sensor input pin 1 (gas sensor module output)
int buzzerout = 10;     // Digital output pin 10 - buzzer output
int redLed = 11;        // Digital output pin 11 - Red LED for leak/gas detected
int greenLed = 12;      // Digital output pin 12 - Green LED for no leak/gas
unsigned long previousMillis = 0;   // Store the last time the buzzer state was toggled
const long interval = 500;          // Blinking interval for buzzer (500 milliseconds)
bool buzzerState = false;           // Track the on/off state for buzzer blinking
void setup() {
  Serial.begin(9600);
pinMode(buzzerout, OUTPUT);   // Set buzzer pin as output
  pinMode(redLed, OUTPUT);      // Set red LED pin as output
  pinMode(greenLed, OUTPUT);    // Set green LED pin as output
  pinMode(watersense, INPUT);   // Set waterdrop module pin as input
  pinMode(gassense, INPUT);     // Set gas sensor module pin as input
}
void loop() {
  int leakSenseReading = analogRead(watersense);  // Read the waterdrop module
  int gasSenseReading = analogRead(gassense);     // Read the gas sensor module
  Serial.print("Water Sensor: ");
  Serial.println(leakSenseReading);               // Print water sensor value
  Serial.print("Gas Sensor: ");
  Serial.println(gasSenseReading);                // Print gas sensor value
  // Check if water leak or gas is detected
  if (leakSenseReading < 500 || gasSenseReading > 55) {
    // Leak or gas detected: turn on red LED and blink the buzzer
    digitalWrite(redLed, HIGH);   // Turn on the red LED
    digitalWrite(greenLed, LOW);  // Turn off the green LED
    unsigned long currentMillis = millis();     // Get the current time
    if (currentMillis - previousMillis >= interval) {  // Check if it's time to toggle the buzzer
```

```
      previousMillis = currentMillis;  // Save the last time the state was toggled
      // Toggle the buzzer state
      buzzerState = !buzzerState;
      // Set the buzzer based on the buzzerState
      digitalWrite(buzzerout, buzzerState ? HIGH : LOW);
    }
     Serial.println("Leak or gas detected! Red LED ON, Buzzer blinking.");
  } else {
    // No leak or gas: turn on green LED and turn off the red LED and buzzer
    digitalWrite(redLed, LOW);    // Turn off the red LED
    digitalWrite(greenLed, HIGH); // Turn on the green LED
    digitalWrite(buzzerout, LOW); // Turn off the buzzer
    Serial.println("No leak or gas. Green LED ON, Buzzer OFF.");
  }
}
```

Remote control appliance management system :

```
// Uncomment the following line to enable serial debug output
//#define ENABLE_DEBUG
#ifdef ENABLE_DEBUG
      #define DEBUG_ESP_PORT Serial
      #define NODEBUG_WEBSOCKETS
      #define NDEBUG
#endif
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include "SinricPro.h"
#include "SinricProSwitch.h"
#include <map>
#define WIFI_SSID         "Kichee's Mobile"
#define WIFI_PASS         "Asta2703"
// Should look like "de0bxxxx-1x3x-4x3x-ax2x-5dabxxxxxxxx"
#define APP_KEY           "c364b121-59cb-487a-a422-c34ff2a865a9"
// Should look like "5f36xxxx-x3x7-4x3x-xexe-e86724a9xxxx-4c4axxxx-3x3x-x5xe-x9x3-333d65xxxxxx"
#define APP_SECRET        "40394bd6-42ca-4db0-81af-9f3781227166-f1f52e3b-efde-4bfa-a436-efb6f0e9066a"
//Enter the device IDs here
#define device_ID_1   "66f2ed6b46a89b75c274d97b"
#define device_ID_2   "66f9168c0d73840bd1ca36bb"
#define device_ID_3   "66f2ed6b46a89b75c274d97b"
#define device_ID_4   "66f2ed6b46a89b75c274d97b"
// define the GPIO connected with Relays and switches
#define RelayPin1 5  //D1
#define RelayPin2 4  //D2
#define RelayPin3 14 //D5
#define RelayPin4 12 //D6
```

```
#define SwitchPin1 10  //SD3
#define SwitchPin2 0   //D3
#define SwitchPin3 13  //D7
#define SwitchPin4 3   //RX
#define wifiLed   16   //D0
// comment the following line if you use a toggle switches instead of tactile buttons
#define TACTILE_BUTTON 1
#define BAUD_RATE   9600
#define DEBOUNCE_TIME 250
typedef struct {      // struct for the std::map below
  int relayPIN;
  int flipSwitchPIN;
} deviceConfig_t;
// this is the main configuration
// please put in your deviceId, the PIN for Relay and PIN for flipSwitch
// this can be up to N devices...depending on how much pin's available on your device ;)
// right now we have 4 devicesIds going to 4 relays and 4 flip switches to switch the relay manually
std::map<String, deviceConfig_t> devices = {
    //{deviceId, {relayPIN,  flipSwitchPIN}}
    {device_ID_1, {  RelayPin1, SwitchPin1 }},
    {device_ID_2, {  RelayPin2, SwitchPin2 }},
    {device_ID_3, {  RelayPin3, SwitchPin3 }},
    {device_ID_4, {  RelayPin4, SwitchPin4 }}
};
typedef struct {      // struct for the std::map below
  String deviceId;
  bool lastFlipSwitchState;
  unsigned long lastFlipSwitchChange;
} flipSwitchConfig_t;
// this map is used to map flipSwitch PINs to deviceId and handling debounce and last flipSwitch state checks
```

```
std::map<int, flipSwitchConfig_t> flipSwitches;
// it will be setup in "setupFlipSwitches" function, using informations from devices map
void setupRelays() {
  for (auto &device : devices) {          // for each device (relay, flipSwitch combination)
    int relayPIN = device.second.relayPIN; // get the relay pin
    pinMode(relayPIN, OUTPUT);            // set relay pin to OUTPUT
    digitalWrite(relayPIN, HIGH);
  }
}
void setupFlipSwitches() {
  for (auto &device : devices)  {  // for each device (relay / flipSwitch combination)
    flipSwitchConfig_t flipSwitchConfig;      // create a new flipSwitch configuration
    flipSwitchConfig.deviceId = device.first;         // set the deviceId
    flipSwitchConfig.lastFlipSwitchChange = 0;        // set debounce time
    flipSwitchConfig.lastFlipSwitchState = true;     // set lastFlipSwitchState to false (LOW)--
    int flipSwitchPIN = device.second.flipSwitchPIN;  // get the flipSwitchPIN
// save the flipSwitch config to flipSwitches map
    flipSwitches[flipSwitchPIN] = flipSwitchConfig;
    pinMode(flipSwitchPIN, INPUT_PULLUP);                 // set the flipSwitch pin to INPUT
  }
}
bool onPowerState(String deviceId, bool &state)
{
  Serial.printf("%s: %s\r\n", deviceId.c_str(), state ? "on" : "off");
  int relayPIN = devices[deviceId].relayPIN; // get the relay pin for corresponding device
  digitalWrite(relayPIN, !state);            // set the new relay state
  return true;
}
void handleFlipSwitches() {
  unsigned long actualMillis = millis();       // get actual millis
  for (auto &flipSwitch : flipSwitches) {     // for each flipSwitch in flipSwitches map
// get the timestamp when flipSwitch was pressed last time (used to debounce / limit events)
```

```
    unsigned long lastFlipSwitchChange = flipSwitch.second.lastFlipSwitchChange;
{  // if time is > debounce time...
    if (actualMillis - lastFlipSwitchChange > DEBOUNCE_TIME)
      int flipSwitchPIN = flipSwitch.first;    // get the flipSwitch pin from configuration
// get the lastFlipSwitchState
      bool lastFlipSwitchState = flipSwitch.second.lastFlipSwitchState;
      bool flipSwitchState = digitalRead(flipSwitchPIN);   // read the current flipSwitch state
      if (flipSwitchState != lastFlipSwitchState) {  // if the flipSwitchState has changed...
#ifdef TACTILE_BUTTON
        if (flipSwitchState) {                                  // if the tactile button is pressed
#endif
// update lastFlipSwitchChange time
          flipSwitch.second.lastFlipSwitchChange = actualMillis;
          String deviceId = flipSwitch.second.deviceId;   // get the deviceId from config
          int relayPIN = devices[deviceId].relayPIN;          // get the relayPIN from config
          bool newRelayState = !digitalRead(relayPIN);           // set the new relay State
          digitalWrite(relayPIN, newRelayState);              // set the trelay to the new state
// get Switch device from SinricPro
          SinricProSwitch &mySwitch = SinricPro[deviceId];
          mySwitch.sendPowerStateEvent(!newRelayState);                    // send the event
#ifdef TACTILE_BUTTON
        }
#endif
// update lastFlipSwitchState
          flipSwitch.second.lastFlipSwitchState = flipSwitchState;
      }
    }
  }
}
void setupWiFi()
{
  Serial.printf("\r\n[Wifi]: Connecting");
```

```
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.printf(".");
    delay(250);
  }
  digitalWrite(wifiLed, LOW);
  Serial.printf("connected!\r\n[WiFi]: IP-Address is %s\r\n", WiFi.localIP().toString().c_str());
}
void setupSinricPro()
{
  for (auto &device : devices)  {
    const char *deviceId = device.first.c_str();
    SinricProSwitch &mySwitch = SinricPro[deviceId];
    mySwitch.onPowerState(onPowerState);
  }
  SinricPro.begin(APP_KEY, APP_SECRET);
  SinricPro.restoreDeviceStates(true);
}
void setup(){
  Serial.begin(BAUD_RATE);
  pinMode(wifiLed, OUTPUT);
  digitalWrite(wifiLed, HIGH);
  setupRelays();
  setupFlipSwitches();
  setupWiFi();
  setupSinricPro();
}
void loop(){
  SinricPro.handle();
  handleFlipSwitches();
}
```