MINI SMART HOME SYSTEM

Team 2

MIND CLOUD

1. ABSTRACT

Project Summary

In smart homes, the systems developed and implemented are composed of a set of facilities, from security to other automation processes. It would now allow advanced technology to face recognition, password authentication, and residential automation.

- Salient Features:

Advanced capabilities of face recognition.

Password-based authentication using an Arduino 4x4 Keypad.

Temperature sensors Motion detectors Light level controlling sensors.

2. ARCHITECTURAL DESIGN

2.1 COMPUTER VISION DIVISION

- **Facial Verification:**

Once completed, the detected face is examined through the laptop's camera. The system was to match facial images with a database of known homeowners.

- **Visual Indicators:**

The green bounding boxes are overlaid over the detected names. Red box if there's no face.

- **Behaviors:**

It automatically detects the pre-enrolled faces and unlocks the door with connected door control.

Unrecognized face triggers a notice to householders with options to open the door or include the individual in the database.

- **Reserve Password Access:**

– In case of failure to recognize one's face, authentication by password keyed into the keypad may be allowed.

- Interface Management:

- The owner easily changes passwords or any database updates through control interface. Plus point: It uses GUI, too.

PYTHON CODE :

```
from functools import partial
import tkinter
import cv2
import face_recognition
import os
import glob
import numpy as np
from tkinter import *
from PIL import Image, ImageTk
import serial
connection = serial.Serial('COM4', 9600)
root = Tk()
pause_detection = False
background_img_path = "background.png"
background_img = Image.open(background_img_path)
background_img = background_img.resize((1720, 835))
background = ImageTk.PhotoImage(image=background_img)
canvas = tkinter.Canvas(root, width=1720, height=835)
canvas.pack(fill="both", expand=True)
canvas.create_image(0, 0, anchor="nw", image=background)
correct_pass = "1234"
root.title("Face Recognition Service")
root.geometry('1920x1080')
label = Label(root, text="Who are you?", font=("Helvetica", 32))
label.place(relx=0.5, rely=0.09, anchor=CENTER)
video = Label(root)
video.place(relx=0.5, rely=0.4, anchor=CENTER)

def clicked():
    global pause_detection
    pause_detection = False
    label.configure(text="Starting face recognition...")
    sfr = facerec()
    sfr.encodings_imgs("images")
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Camera not accessible.")
        return
    update_frame(cap, sfr)

button = Button(root, text="Face Recognition", fg="black", command=clicked,
font=16)
button.place(relx=0.5, rely=0.8, anchor=CENTER)
```

```
def change_pass():
    global pause_detection
    pause_detection = True
    for widget in root.winfo_children():
        if isinstance(widget, Entry) or isinstance(widget, Button) and widget !=
change_button:
            widget.destroy()

    change_button.destroy()
    label.configure(text="Enter Old password:", font=("Helvetica", 20))
    password = Entry(root, width=10, show='*')
    password.place(relx=0.6, rely=0.9, anchor=CENTER)

    def verify():
        entered_password = password.get()
        if entered_password == correct_pass:
            password.destroy()
            label.configure(text="Enter New password:")
            new_pass = Entry(root, width=10, show='*')
            new_pass.place(relx=0.6, rely=0.9, anchor=CENTER)

            def new_handle():
                new_password = new_pass.get()
                global correct_pass
                if len(new_password) == 4:
                    correct_pass = 'P' + new_password
                    new_pass.destroy()
                    connection.write(correct_pass.encode())
                    label.configure(text="Password Successfully Changed!")
                    root.after(1000, root.destroy)
                else:
                    error_label = Label(root, text="New Password Must Be 4
Characters Long", font=("Helvetica", 20))
                    error_label.place(relx=0.45, rely=0.5, anchor=CENTER)
                    root.after(1000, lambda: error_label.destroy())

            send_button = Button(root, text="Submit New Password", fg="black",
command=new_handle, font=16)
            send_button.place(relx=0.5, rely=0.8, anchor=CENTER)
        else:
            label.configure(text="Incorrect password. Please try again.")

    ver = Button(root, text="Submit Password", fg="black", command=verify,
font=16)
    ver.place(relx=0.5, rely=0.8, anchor=CENTER)
```

```
change_button = Button(root, text="Change Password", fg="black",
command=change_pass, font=16)
change_button.place(relx=0.5, rely=0.9, anchor=CENTER)

def del_handle():
    global pause_detection
    pause_detection = True
    for widget in root.winfo_children():
        if isinstance(widget, Entry) or isinstance(widget, Button) and widget !=
del_button:
            widget.destroy()
    del_button.destroy()
    label4 = Label(root, text="Enter Password:", font=("Helvetica", 20))
    label4.place(relx=0.5, rely=0.9, anchor=CENTER)
    password = Entry(root, width=10, show='*')
    password.place(relx=0.6, rely=0.9, anchor=CENTER)
    def verify():
        entered_password = password.get()
        if entered_password == correct_pass:
            ver.destroy()
            password.destroy()
            label4.destroy()
            label.configure(text="Enter Name of User to Delete:",
font=("Helvetica", 20))
            name_entry = Entry(root, width=20)
            name_entry.place(relx=0.5, rely=0.9, anchor=CENTER)
            def delete_user():
                user_name = name_entry.get()
                if user_name:
                    user_image_path = f"images/{user_name}.jpg"
                    if os.path.exists(user_image_path):
                        os.remove(user_image_path)
                        label.configure(text=f"User '{user_name}' deleted!")
                    else:
                        label.configure(text=f"No user found with name
'{user_name}'.")
                else:
                    label.configure(text="Name cannot be empty.")

            delete_button = Button(root, text="Delete User", fg="black",
command=delete_user, font=16)
            delete_button.place(relx=0.5, rely=0.85, anchor=CENTER)
        else:
            label.configure(text="Incorrect password. Please try again.")
```

```
    ver = Button(root, text="Submit Password", fg="black", command=verify,
font=16)
    ver.place(relx=0.5, rely=0.8, anchor=CENTER)

del_button = Button(root, text="Delete User", fg="black", command=del_handle,
font=16)
del_button.place(relx=0.5, rely=0.85, anchor=CENTER)

class facerec:
    def __init__(self):
        self.known_face_encodings = []
        self.known_face_names = []
        self.frame_resize = 0.25
        self.current_face_location = None
        self.current_frame = None
        self.submit_button = None

    def encodings_imgs(self, images_path):
        images_path = glob.glob(os.path.join(images_path, "*.*"))
        for path in images_path:
            img = cv2.imread(path)
            rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            basename = os.path.basename(path)
            filename, ext = os.path.splitext(basename)
            encodings = face_recognition.face_encodings(rgb_img)
            if encodings:
                self.known_face_encodings.append(encodings[0])
                self.known_face_names.append(filename)
            else:
                print(f"No faces detected in image: {path}")

    def unknown_handling(self, password):
        global pause_detection
        pause_detection = True
        entered = password.get()
        if entered == correct_pass:
            password.destroy()
            if self.submit_button:
                self.submit_button.destroy()
            change_button.destroy()
            def open():
                connection.write(b'R')
                label.configure(text="Access granted!")
                add_button.destroy()
```

```
                open_button.destroy()

            def add():
                connection.write(b'R')
                add_button.destroy()
                open_button.destroy()
                self.ask_for_name()

            open_button = Button(root, text="Open", fg="black", command=open,
font=16)
            open_button.place(relx=0.45, rely=0.9, anchor=CENTER)
            add_button = Button(root, text="Add", fg="black", command=add,
font=16)
            add_button.place(relx=0.55, rely=0.9, anchor=CENTER)
        else:
            label.configure(text="Incorrect password. Please try again.")
    def ask_for_name(self):
        label_name = Label(root, text="Please enter a name for the new face:",
font=("Helvetica", 24))
        label_name.place(relx=0.5, rely=0.7, anchor=CENTER)
        name_entry = Entry(root, width=20)
        name_entry.place(relx=0.5, rely=0.8, anchor=CENTER)

        def save_face():
            user_name = name_entry.get()
            if user_name:
                self.save_face_image(user_name)
            else:
                error_label = Label(root, text="Name cannot be empty.",
font=("Helvetica", 18), fg="red")
                error_label.place(relx=0.5, rely=0.9, anchor=CENTER)

        submit_name_button = Button(root, text="Submit Name", fg="black",
command=save_face, font=16)
        submit_name_button.place(relx=0.5, rely=0.85, anchor=CENTER)

    def save_face_image(self, user_name):
        if self.current_face_location is not None:
            top, right, bottom, left = self.current_face_location[0]
            face_image = self.current_frame[top:bottom, left:right]
            face_image_rgb = cv2.cvtColor(face_image, cv2.COLOR_BGR2RGB)
            save_path = f"images/{user_name}.jpg"
            cv2.imwrite(save_path, face_image_rgb)
            label.configure(text=f"Face saved as {user_name}.jpg!")
            root.after(5000, root.destroy)
```

```
        else:
            label.configure(text="No face detected to save.")

    def detect(self, frame):
        self.current_frame = frame
        sm_frame = cv2.resize(frame, (0, 0), fx=self.frame_resize,
fy=self.frame_resize)
        rgb_frame = cv2.cvtColor(sm_frame, cv2.COLOR_BGR2RGB)
        locations = face_recognition.face_locations(rgb_frame)
        face_encodings = face_recognition.face_encodings(rgb_frame, locations)
        face_names = []
        name = "Unknown"
        for f_e in face_encodings:
            matches = face_recognition.compare_faces(self.known_face_encodings,
f_e)
            if True in matches:
                match_in = matches.index(True)
                name = self.known_face_names[match_in]
                break
        face_names.append(name)
        face_locations = np.array(locations)
        face_locations = face_locations / self.frame_resize
        self.current_face_location = face_locations.astype(int) if
face_locations.size > 0 else None
        return face_locations.astype(int), face_names

def update_frame(cap, sfr):
    global pause_detection
    ret, frame = cap.read()

    if not ret or pause_detection:
        return

    locs, face_name = sfr.detect(frame)
    recognised = False

    for (top, right, bottom, left), name in zip(locs, face_name):
        if name != "Unknown":
            cv2.rectangle(frame, (left, top), (right, bottom), (127, 127, 0), 3)
            cv2.putText(frame, name, (left, top - 10), cv2.FONT_HERSHEY_COMPLEX,
1, (127, 127, 127))
            recognised = True
        else:
            cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 3)
```

```
            cv2.putText(frame, name, (left, top - 10), cv2.FONT_HERSHEY_COMPLEX,
1, (127, 127, 127))
            pause_detection = True
            password = Entry(root, width=10, show='*')
            password.place(relx=0.6, rely=0.9, anchor=CENTER)
            submit = Button(root, text="Submit Password", fg="black",
command=partial(sfr.unknown_handling, password), font=16)
            submit.place(relx=0.6, rely=0.85, anchor=CENTER)

    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_img = Image.fromarray(rgb_frame)
    imgtkinter = ImageTk.PhotoImage(image=frame_img)
    video.imgtk = imgtkinter
    video.configure(image=imgtkinter)

    if recognised:
        connection.write(b'R')
        video.after(500, video.destroy)
        label.configure(text=f"Hello, {name}!")
    else:
        video.after(10, update_frame, cap, sfr)
root.mainloop()
```

2.2 Microcontroller Unit

• House with automatic doors.

- **Turns the system on upon opening the door:**

It regulates the lighting system based on ambient light used.

Temperature Sensor: Room Temperature Measurement - using an RGB LED NTC Thermistor Module

Colored **Red**; T > 30

• **Green**: 20°C < T < 30°C

> **Blue**: T < 20°C

It modulates the speed of the fan according to the varied temperature reading using an L293D driver and a motor.

- **Security features:**

Password error: It will connect the buzzer for 500 milliseconds, thus warning at the control interface in case of a wrong password.

The PIR sensors detect movement inside the home, with a door locked; then it turns on the security features.

ARDUINO CODE :

```
#include <Keypad.h>
#include <Servo.h>
#include <EEPROM.h>

int RKey = 0;

const int lightPin = 10;
const int tempPin = A4;
const int pirPin = 9;
const int buzzerPin = A5;
const int redLedPin = A1;
const int greenLedPin = A2;
const int blueLedPin = A3;
const int motorPin1 = 12;
const int motorPin2 = 13;
const int enablePin = 11;
const int servoPin = 8;

const byte ROWS = 4;
const byte COLS = 3;

char keymap[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

byte rowPins[ROWS] = {7, 6, 5, 4};
byte colPins[COLS] = {3, 2, A0};

Keypad myKeypad = Keypad(makeKeymap(keymap), rowPins, colPins, ROWS, COLS);
Servo doorServo;

const char defaultPassword[] = "1234";
char enteredPassword[5] = ""; // 5 to fit 4 characters and the null terminator
char savedPassword[5] = "";

bool doorLocked = true;

void setup() {
  Serial.begin(9600);

  pinMode(tempPin, INPUT);
```

```
  pinMode(pirPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
  pinMode(redLedPin, OUTPUT);
  pinMode(greenLedPin, OUTPUT);
  pinMode(blueLedPin, OUTPUT);
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(enablePin, OUTPUT);
  pinMode(lightPin, INPUT);

  doorServo.attach(servoPin);
  doorServo.write(0);

  EEPROM.get(0, savedPassword);
  if (strlen(savedPassword) == 0) {
    strcpy(savedPassword, defaultPassword);  // If no password in EEPROM, use
default
  }

  Serial.println(F("System ready."));
}

void loop() {
  if (Serial.available()) {
    char command = Serial.read();

    if (command == 'R') {
      RKey = 1;
      Serial.println(F("Received RKey!"));
    } else if (command == 'P') {
      receiveNewPassword();
    }
  }

  if (RKey == 1) {
    Serial.println(F("Open! System functions started."));
    doorLocked = false;
    openDoor();
    runSystemFunctions();  // Continuously check for the stop condition inside
this function
  }

  char key = myKeypad.getKey();
  if (key) {
    Serial.print(F("Key pressed: "));
```

```
    Serial.println(key);

    if (key == '#') {
      if (strcmp(enteredPassword, savedPassword) == 0) {
        Serial.println(F("Correct password! System functions started."));
        doorLocked = false;
        openDoor();
        runSystemFunctions();  // Keep running system functions until stopped
      } else {
        Serial.println(F("Wrong password!"));
        handleWrongPassword();
      }
      clearPassword();
    } else if (key == '*') {
      clearPassword();
      Serial.println(F("Password cleared."));
    } else {
      appendToPassword(key);
    }
  }

  if (doorLocked && digitalRead(pirPin) == HIGH) {
    Serial.println(F("Motion detected inside while the door is locked!"));
  }

  delay(100);
}

void receiveNewPassword() {
  char newPassword[5];
  int index = 0;

  while (Serial.available()) {
    char c = Serial.read();
    if (index < 4 && isdigit(c)) {
      newPassword[index++] = c;
    }
  }
  newPassword[index] = '\0';

  updatePassword(newPassword);
  Serial.print(F("New password set: "));
  Serial.println(newPassword);
}
```

```
void openDoor() {
  doorServo.write(90);
  delay(1000);
  doorLocked = false;
}

void closeDoor() {
  doorServo.write(0);  // Rotate servo to close the door
  delay(1000);

  // Stop the fan
  analogWrite(enablePin, 0);  // Set motor speed to 0 to stop the fan
  digitalWrite(motorPin1, LOW);  // Ensure the motor is off
  digitalWrite(motorPin2, LOW);  // Ensure the motor is off
  digitalWrite(redLedPin, HIGH);
  digitalWrite(greenLedPin, LOW);
  digitalWrite(blueLedPin, LOW);

  doorLocked = true;
  Serial.println(F("System stopped, fan turned off, door closed."));
}

void runSystemFunctions() {
  while (true) {
    int light = digitalRead(lightPin);
    Serial.println(light ? F("Light: Off") : F("Light: ON"));

    int tempC = map(analogRead(tempPin), 0, 1023, -55, 125);
    if (tempC > 30) {
      digitalWrite(redLedPin, HIGH);
      digitalWrite(greenLedPin, LOW);
      digitalWrite(blueLedPin, LOW);
    } else if (tempC > 20) {
      digitalWrite(redLedPin, LOW);
      digitalWrite(greenLedPin, HIGH);
      digitalWrite(blueLedPin, LOW);
    } else {
      digitalWrite(redLedPin, LOW);
      digitalWrite(greenLedPin, LOW);
      digitalWrite(blueLedPin, HIGH);
    }

    int motorSpeed = map(tempC, -40, 125, 0, 255);
    analogWrite(enablePin, motorSpeed);
    digitalWrite(motorPin1, LOW);
```

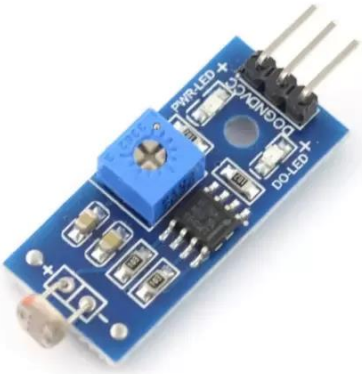
```
    digitalWrite(motorPin2, HIGH);

    Serial.print(F("Temperature: "));
    Serial.print(tempC);
    Serial.print(F(" °C, Fan speed: "));
    Serial.println(motorSpeed);

    char key = myKeypad.getKey();
    if (key == '*') {
        Serial.println(F("Stopping system and closing door."));
        closeDoor();
        break;  // Exit the system function loop
    }
  }
}

void handleWrongPassword() {
  tone(buzzerPin, 1000, 500);
  Serial.println(F("Warning: Wrong password entered."));
  doorLocked = true;
}

void clearPassword() {
  memset(enteredPassword, 0, sizeof(enteredPassword));
}

void appendToPassword(char key) {
  size_t len = strlen(enteredPassword);
  if (len < sizeof(enteredPassword) - 1) {
    enteredPassword[len] = key;
    enteredPassword[len + 1] = '\0'; // Null-terminate the string
  }
}

void updatePassword(const char* newPassword) {
  strcpy(savedPassword, newPassword);
  EEPROM.put(0, savedPassword);
  Serial.println(F("Password updated and saved to EEPROM."));
}
```

3. PARTS

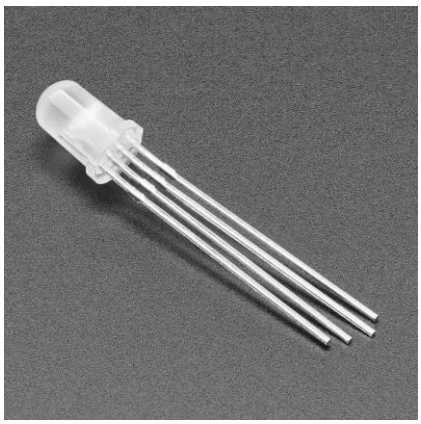
The main microcontroller is the ATMega8A, which interfaces sensors and the system logics.

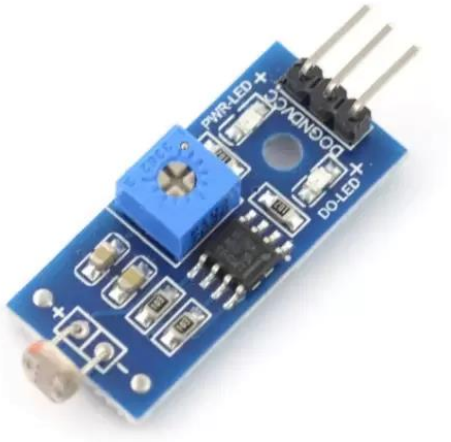
- **Keypad**: a 4x4 matrix through which one enters passwords.

- **Buzzer**: Beep to alert an invalid password entered.
- **Temperature Sensor NTC Thermistor**: Measures the ambient temperature.



- **PIR Sensor**: A motioned triggered relay used in the house.
- **LDR Sensor**: Adjusting the intensity of the illuminations for the residents.
- **RGB LED**: Shiny, colored lights that change according to temperature.



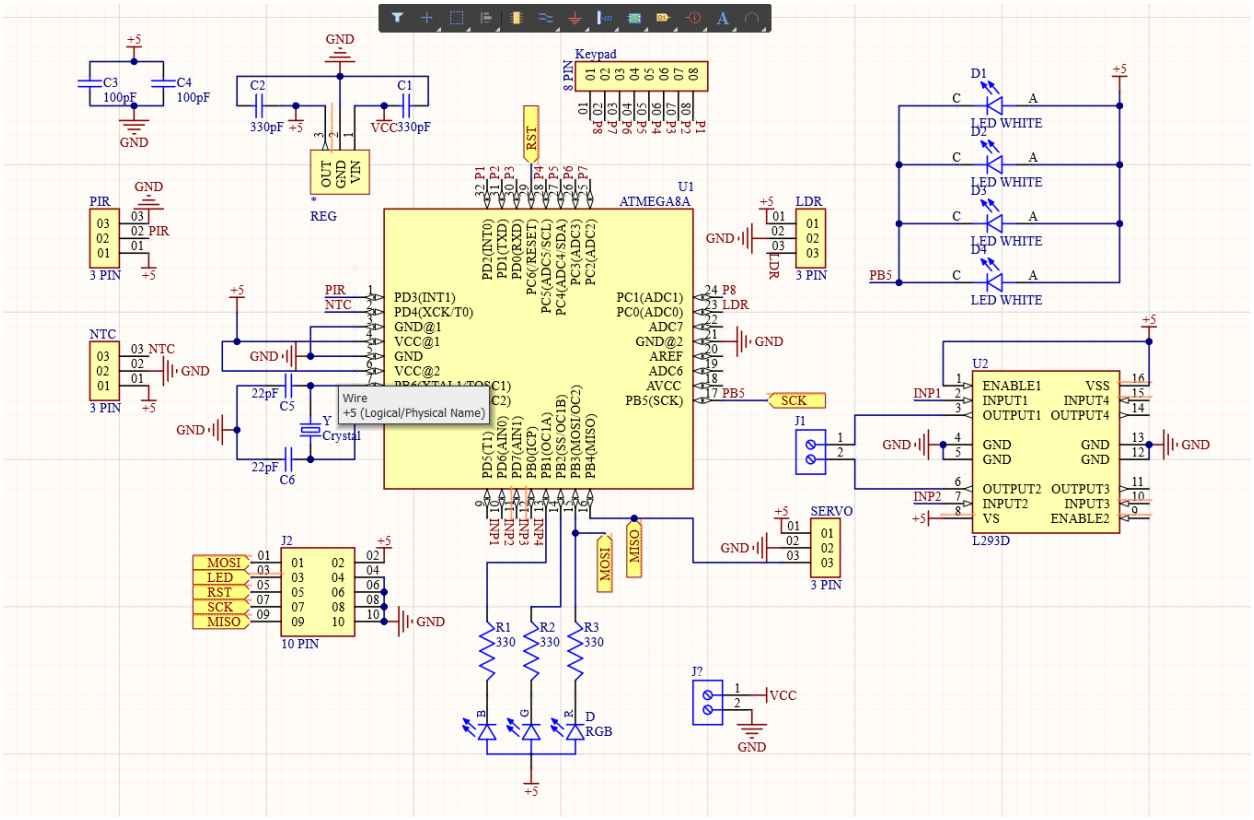


- **Motor + Fan Blades**: Interfaced with L293D to control the speed of the fan in any variation of temperature.
- **L293D Driver**: This chip drives the motor controlling the fan.
- **LED Lighting System**: Controlled as per LDR Value.
- **Servo Motor**: To automatically open the door.

4. Schematic Design

A schematic utilizing Altium Designer, encompassing comprehensive details regarding the connectivity of each component—such as the keypad, sensors, an LED, and a servo motor—will be developed. The components will be systematically created in the 'Component Creation' section, accompanied by their respective parameters and

designations.



Design Principles and Criteria: All SMD components used . Proper labeling, bounds, and arrangements.

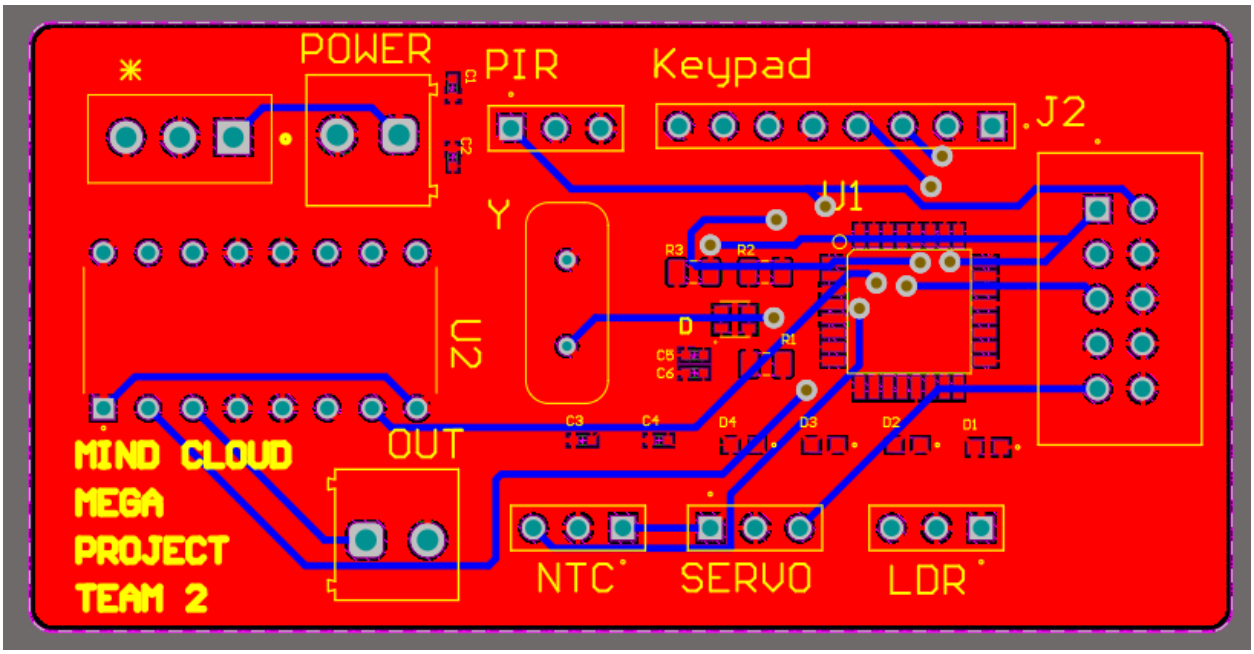
5. PCB Design

Design Description:

The PCB design will be planned in Altium Designer, considering the created rules and specifications by JLCPCB.

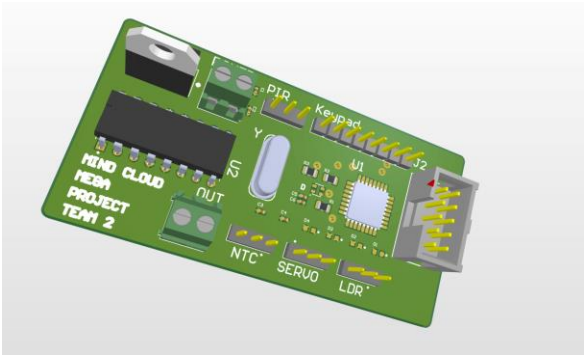
- Features:

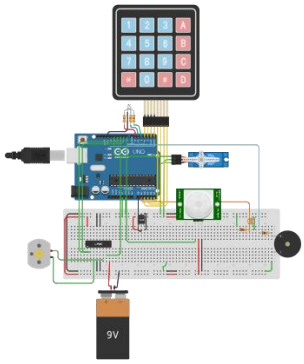
Proper application of available parameters of suppliers. Layer application Layer 29 was given the designation of a layer.



6. Operationalization

Circuit Realization: This physical implementation can be done using a breadboard, fabricated PCB, or any other proper way. The logic of the system lies in programming the Atmega8A microcontroller.





7. Conclusion

This report has laid its key emphasis on a smart home integrated system having a facial recognition system with security and sensors, automation to have the proper working of a house. Annexes of this report will concern everything related to your project after design and implementation phases are done