```
#include <Servo.h>
#include <DFPlayerMini_Fast.h>
#include <Wire.h>
#include "SSD1306Ascii.h"
#include "SSD1306AsciiWire.h"
#define I2C_ADDRESS 0x3C
DFPlayerMini_Fast myMP3;
SSD1306AsciiWire oled;
#include "SoftwareSerial.h"
SoftwareSerial mySerial(11, 10);  // RX, TX
Servo servo;
int pos = 0;  // Servoposition
String bluetoothMessage;
unsigned long resetElevatorStart = 0;
bool goingUp = false;    //Für die Pfeile in der App
bool goingDown = false;  //Für die Pfeile in der App
int lastfloor = 0;
bool doorchanged = false;  // Türstatus geändert oder nicht
unsigned long messages_send_timer = 0;

// Variable für den Fahrmodus oder Pausemodus (Pause=0, Fahren=1)
int F = 0;
int Etage, EtageIst, EtageSoll, AltesetageIst;

#define MAX_FLOORS 4
int requestedFloors[MAX_FLOORS];
int numRequestedFloors = 0;

//Pindefinierungen für die Ansteuerung des L293D Motortreibers
int OutputMspeed = 8, OutputMotorA = 6, OutputMotorB = 7;

//Pindefinierungen für den Lautsprecher
int Speaker = 9;

//Pindefinierungen für den die Lichtsschranken
int Sensor1 = 50, Sensor2 = 51, Sensor3 = 52, Sensor4 = 53;

//Pindefinierungen für die Ausgänge der Schalter, die gleichzeitig auch den Status der LEDs
darstellen.
int LEDSUntenDO = 35, LEDSUntenDC = 33, LEDSUntenF0 = 31, LEDSUntenF1 = 29,
LEDSUntenF2 = 27, LEDSUntenF3 = 25, LEDSUntenAlarm = 23;
int LEDSObenF0up = 38, LEDSObenF1dwn = 40, LEDSObenF1up = 42, LEDSObenF2dwn = 44,
LEDSObenF2up = 46, LEDSObenF3dwn = 48;

//Pindefinierungen für die aktivierung der Schalter
int SUntenDO = 34, SUntenDC = 32, SUntenF0 = 30, SUntenF1 = 28, SUntenF2 = 26, SUntenF3 =
24;
int SUntenAlarm = 22, SObenF0up = 39, SObenF1dwn = 41, SObenF1up = 43, SObenF2dwn = 45,
SObenF2up = 47, SObenF3dwn = 49;

// Variablen für den Status der jeweiligen Schalter festlegen
int StatSensor1, StatSensor2, StatSensor3, StatSensor4;
```

```
int StatUntenDO, StatUntenDC, StatUntenF0, StatUntenF1, StatUntenF2, StatUntenF3,
StatUntenAlarm;
int StatObenF0up, StatObenF1dwn, StatObenF1up, StatObenF2dwn, StatObenF2up,
StatObenF3dwn;

// Erstellen der Variablen per Array
int sensors[] = { Sensor1, Sensor2, Sensor3, Sensor4 };
int LEDS[] = { LEDSUntenDO, LEDSUntenDC, LEDSUntenF0, LEDSUntenF1, LEDSUntenF2,
LEDSUntenF3, LEDSUntenAlarm, LEDSObenF0up, LEDSObenF1dwn, LEDSObenF1up,
LEDSObenF2dwn, LEDSObenF2up, LEDSObenF3dwn };
int SWITCHES[] = { SUntenDO, SUntenDC, SUntenF0, SUntenF1, SUntenF2, SUntenF3,
SUntenAlarm, SObenF0up, SObenF1dwn, SObenF1up, SObenF2dwn, SObenF2up, SObenF3dwn
};

// Funktionen anlegen für wiederkehrende Abläufe
void MotorHoch();
void MotorRunter();
void MotorStop();
void TurAuf();
void TurZu();
void EtagenTon();

bool doorOpen = false;

void MP3uSpeaker_INIT() {
  mySerial.begin(9600);
  myMP3.begin(mySerial, true);
  delay(1000);
  myMP3.volume(10);  //Maximum=30
  myMP3.play(1);     // Erste Datei abspielen
}

// Funktion zum Zurücksetzen des Fahrstuhls auf Etage 0 vor dem Programmstart
void resetElevator() {
  resetElevatorStart = millis();        // Setze die Startzeit für das Zurücksetzen

  while (StatSensor1 != 1) {            // Solange der Sensor für Etage 1 nicht erreicht ist
    StatSensor1 = digitalRead(Sensor1);  // Lese den Status des Sensors für Etage 1
    Serial.println(StatSensor1);
    if (millis() - resetElevatorStart > 500) {  // Wenn mehr als 500 Millisekunden vergangen sind
      MotorRunter();                    // Fahre den Aufzug nach unten
    } else {
      MotorHoch();  // Andernfalls fahre den Aufzug nach oben
    }
  }

  MotorStop();              // Stoppe den Aufzug
  delay(1000);              // Warte für 1 Sekunde
  EtageIst = 0;             // Setze die aktuelle Etage auf 0
  EtageSoll = 0;            // Setze die Ziel-Etage auf 0
  Serial3.println("stoped");  // Sende "stoped" über die Bluetooth-Schnittstelle
  delay(1000);              // Warte für 1 Sekunde
```

```
}


// random elevator floor selection for testing
void randomFloor() {

  int randomFloor = -1;
  while (randomFloor == -1) {
    randomFloor = random(0, 4);
    if (isFloorRequested(randomFloor)) {
      randomFloor = -1;
    }
  }
  // if there is already more then 1 floor requested, then we don't need to add another one
  if (numRequestedFloors > 1) {
    return;
  }
  addFloorToQueue(randomFloor);
  Serial.println("Calling from RANDOM");

  // Depending on which floor is selected, turn on the corresponding LED
  switch (randomFloor) {
    case 0:
      // set led to on
      // change to output from input
      pinMode(LEDSUntenF0, OUTPUT);
      pinMode(LEDSObenF0up, OUTPUT);
      digitalWrite(LEDSUntenF0, HIGH);
      digitalWrite(LEDSObenF0up, HIGH);
      delay(500);
      // // change back to input
      // pinMode(LEDSUntenF0, INPUT);
      // pinMode(LEDSObenF0up, INPUT);

      break;
    case 1:
      // set led to on
      pinMode(LEDSUntenF1, OUTPUT);
      pinMode(LEDSObenF1dwn, OUTPUT);
      pinMode(LEDSObenF1up, OUTPUT);

      digitalWrite(LEDSUntenF1, HIGH);
      digitalWrite(LEDSObenF1dwn, HIGH);
      digitalWrite(LEDSObenF1up, HIGH);
      delay(500);
      // pinMode(LEDSUntenF1, INPUT);
      // pinMode(LEDSObenF1dwn, INPUT);
      // pinMode(LEDSObenF1up, INPUT);
      break;
    case 2:
      pinMode(LEDSUntenF2, OUTPUT);
      pinMode(LEDSObenF2dwn, OUTPUT);
```

```
      pinMode(LEDSObenF2up, OUTPUT);
      // set led to on
      digitalWrite(LEDSUntenF2, HIGH);
      digitalWrite(LEDSObenF2dwn, HIGH);
      digitalWrite(LEDSObenF2up, HIGH);
      delay(500);
      // pinMode(LEDSUntenF2, INPUT);
      // pinMode(LEDSObenF2dwn, INPUT);

      break;
    case 3:
      pinMode(LEDSUntenF3, OUTPUT);
      pinMode(LEDSObenF3dwn, OUTPUT);
      // set led to on
      digitalWrite(LEDSUntenF3, HIGH);
      digitalWrite(LEDSObenF3dwn, HIGH);
      delay(500);

      // pinMode(LEDSUntenF3, INPUT);
      // pinMode(LEDSObenF3dwn, INPUT);


      break;
  }
}


unsigned long lastmotorUpdate = 0;

// Funktion zum Aufwärtsfahren des Motors
void MotorHoch() {
  Serial.println("hoch");               // Gib "hoch" auf der seriellen Schnittstelle aus
  analogWrite(OutputMspeed, 90);        // Setze die Geschwindigkeit des Motors auf Maximum
  digitalWrite(OutputMotorA, (int)1);    // Aktiviere Motor-A in eine Richtung
  digitalWrite(OutputMotorB, (int)0);    // Deaktiviere Motor-B in die andere Richtung
  goingUp = true;                       // Setze die Variable für Aufwärtsfahrt auf true
  goingDown = false;                     // Setze die Variable für Abwärtsfahrt auf false
  updateVariables();
  if (millis() - lastmotorUpdate > 1000)  // Wenn eine Sekunde vergangen ist
  {
    lastmotorUpdate = millis();   // Aktualisiere die Zeit des letzten Motor-Updates
    Serial3.println("going_up");  // Sende "going_up" über die Bluetooth-Schnittstelle
  }
}

// Funktion zum Abwärtsfahren des Motors
void MotorRunter() {
  Serial.println("runter");             // Gib "runter" auf der seriellen Schnittstelle aus
  analogWrite(OutputMspeed, 90);        // Setze die Geschwindigkeit des Motors auf Maximum
  digitalWrite(OutputMotorA, (int)0);    // Deaktiviere Motor-A in eine Richtung
  digitalWrite(OutputMotorB, (int)1);    // Aktiviere Motor-B in die andere Richtung
  goingDown = true;                     // Setze die Variable für Abwärtsfahrt auf true
```

```
  goingUp = false;                    // Setze die Variable für Aufwärtsfahrt auf false
  updateVariables();
  if (millis() - lastmotorUpdate > 1000)  // Wenn eine Sekunde vergangen ist
  {
    lastmotorUpdate = millis();     // Aktualisiere die Zeit des letzten Motor-Updates
    Serial3.println("going_down");  // Sende "going_down" über die Bluetooth-Schnittstelle
  }
}

// Funktion zum Stoppen des Motors
void MotorStop() {
  Serial.println("stop");             // Gib "stop" auf der seriellen Schnittstelle aus
  analogWrite(OutputMspeed, 255);      // Setze die Geschwindigkeit des Motors auf Maximum
  digitalWrite(OutputMotorA, (int)0);  // Deaktiviere Motor-A
  digitalWrite(OutputMotorB, (int)0);  // Deaktiviere Motor-B
  goingDown = false;                   // Setze die Variable für Abwärtsfahrt auf false
  goingUp = false;                    // Setze die Variable für Aufwärtsfahrt auf false
}

// Funktion zum Öffnen der Tür
void TurAuf() {
  // Aktuelle Position des Servomotors abrufen und auf 38 Grad von dort aus bewegen; aus der
Schleife aussteigen, wenn erreicht
    pos = servo.read();  // Aktuelle Position des Servomotors abrufen
  Serial.println(pos);
  while (pos != 2) {     // Solange die Position nicht 2 Grad erreicht hat
    if (pos < 2) {       // Wenn die Position kleiner als 2 Grad ist
      pos += 1;          // Erhöhe die Position um 1 Grad
      servo.write(pos);  // Setze die Servoposition
    } else {             // Andernfalls
      pos -= 1;          // Verringere die Position um 1 Grad
      servo.write(pos);  // Setze die Servoposition
    }
    delay(20);  // Warte für 20 Millisekunden
  }
  delay(1000);         // Warte für 1000 Millisekunden (1 Sekunde)
  doorOpen = true;     // Setze den Zustand der Tür auf geöffnet
  doorchanged = true;  // Setze den Türstatus auf geändert
}

// Funktion zum Schließen der Tür
void TurZu() {
  // Aktuelle Position des Servomotors abrufen und auf 2 Grad von dort aus bewegen
    pos = servo.read();  // Aktuelle Position des Servomotors abrufen
  Serial.println(pos);
  while (pos != 180) {    // Solange die Position nicht 38 Grad erreicht hat
    if (pos < 180) {      // Wenn die Position kleiner als 38 Grad ist
      pos += 1;          // Erhöhe die Position um 1 Grad
      servo.write(pos);  // Setze die Servoposition
    } else {             // Andernfalls
      pos -= 1;          // Verringere die Position um 1 Grad
      servo.write(pos);  // Setze die Servoposition
```

```
    }
    delay(20);  // Warte für 20 Millisekunden
  }
  delay(1000);        // Warte für 1000 Millisekunden (1 Sekunde)
  doorOpen = false;   // Setze den Zustand der Tür auf geschlossen
  doorchanged = true;  // Setze den Türstatus auf geändert
}

// Funktion zum Generieren des Etagentons
void EtagenTon() {
  tone(Speaker, 450);  // Erzeuge einen Ton mit einer Frequenz von 450 Hertz
  delay(700);         // Warte für 700 Millisekunden (0,7 Sekunden)
  tone(Speaker, 350);  // Der zweite Ton mit einer Frequenz von 350 Hertz
  delay(300);         // Warte für 300 Millisekunden (0,3 Sekunden)
  noTone(Speaker);     // Schalte den Ton am Lautsprecher aus
  // An dieser Stelle endet die Funktion, und der Hauptteil des Sketches beginnt von vorn.
}

// Funktion zum Generieren des Alarmtons
void alarm() {
  for (int i = 0; i < 5; i++) {  // Wiederhole den folgenden Block 5-mal
    tone(Speaker, 450);         // Erzeuge einen Ton mit einer Frequenz von 450 Hertz
    delay(100);                 // Warte für 100 Millisekunden
    tone(Speaker, 350);         // Erzeuge einen Ton mit einer Frequenz von 350 Hertz
    delay(100);                 // Warte für 100 Millisekunden
    noTone(Speaker);            // Schalte den Ton am Lautsprecher aus
  }
}

void handleSensors()  // In dieser Funktion werden die Schalter zurückgesetzt (LED und
Spannungsausgabe deaktivieren), sobald eine Etage erreicht wurde
{
  // Wenn der Aufzug an Sensor 1 stopt, werden die Schalter zu Etage Null zurückgesetzt.
  if (StatSensor1 == 1) {
    // set led to input
    pinMode(LEDSUntenF0, INPUT);
    pinMode(LEDSObenF0up, INPUT);
    digitalWrite(SUntenF0, LOW);
    digitalWrite(SObenF0up, LOW);
    delay(500);
    digitalWrite(SUntenF0, HIGH);
    digitalWrite(SObenF0up, HIGH);
  }

  // Wenn der Aufzug an Sensor 2 stopt, werden die Schalter zu Etage1 zurückgesetzt.
  if (StatSensor2 == 1) {
    // set led to input
    pinMode(LEDSUntenF1, INPUT);
    pinMode(LEDSObenF1dwn, INPUT);
    pinMode(LEDSObenF1up, INPUT);

    digitalWrite(SUntenF1, LOW);
```

```
    digitalWrite(SObenF1dwn, LOW);
    digitalWrite(SObenF1up, LOW);
    delay(500);
    digitalWrite(SUntenF1, HIGH);
    digitalWrite(SObenF1dwn, HIGH);
    digitalWrite(SObenF1up, HIGH);
  }
  // Wenn der Aufzug an Sensor 3 stopt, werden die Schalter zu Etage2 zurückgesetzt.
  if (StatSensor3 == 1) {
    // set led to input
    pinMode(LEDSUntenF2, INPUT);
    pinMode(LEDSObenF2dwn, INPUT);
    pinMode(LEDSObenF2up, INPUT);

    digitalWrite(SUntenF2, LOW);
    digitalWrite(SObenF2dwn, LOW);
    digitalWrite(SObenF2up, LOW);
    delay(500);
    digitalWrite(SUntenF2, HIGH);
    digitalWrite(SObenF2dwn, HIGH);
    digitalWrite(SObenF2up, HIGH);
  }
  // Wenn der Aufzug an Sensor 4 stopt, werden die Schalter zu Etage3 zurückgesetzt.
  if (StatSensor4 == 1) {
    // set led to input
    pinMode(LEDSUntenF3, INPUT);
    pinMode(LEDSObenF3dwn, INPUT);

    digitalWrite(SUntenF3, LOW);
    digitalWrite(SObenF3dwn, LOW);
    delay(500);
    digitalWrite(SUntenF3, HIGH);
    digitalWrite(SObenF3dwn, HIGH);
  }
}

void updateVariables()  // Funktion zur Aktualisierung der Variablen
{
  // Alarmstatus aktualisieren
  StatUntenDO = digitalRead(LEDSUntenDO);
  StatUntenDC = digitalRead(LEDSUntenDC);

 // Untere Etage - Anforderungen prüfen und ggf. zur Warteschlange hinzufügen
  StatUntenF0 = digitalRead(LEDSUntenF0);
  if (StatUntenF0 == 1 && !isFloorRequested(0)) {
    Serial.println("Calling from x00");
    addFloorToQueue(0);
  }

  StatUntenF1 = digitalRead(LEDSUntenF1);
  if (StatUntenF1 == 1 && !isFloorRequested(1)) {
    Serial.println("Calling from x10");
```

```
  addFloorToQueue(1);
}

StatUntenF2 = digitalRead(LEDSUntenF2);
if (StatUntenF2 == 1 && !isFloorRequested(2)) {
  Serial.println("Calling from x20");
  addFloorToQueue(2);
}

StatUntenF3 = digitalRead(LEDSUntenF3);
if (StatUntenF3 == 1 && !isFloorRequested(3)) {
  Serial.println("Calling from x30");
  addFloorToQueue(3);
}

StatUntenAlarm = digitalRead(LEDSUntenAlarm);

// Obere Etage - Anforderungen prüfen und ggf. zur Warteschlange hinzufügen
StatObenF0up = digitalRead(LEDSObenF0up);
if (StatObenF0up == 1 && !isFloorRequested(0)) {
  Serial.println("Calling from x01");
  addFloorToQueue(0);
}

StatObenF1dwn = digitalRead(LEDSObenF1dwn);
if (StatObenF1dwn == 1 && !isFloorRequested(1)) {
  Serial.println("Calling from x11");
  addFloorToQueue(1);
}

StatObenF1up = digitalRead(LEDSObenF1up);
if (StatObenF1up == 1 && !isFloorRequested(1)) {
  Serial.println("Calling from x12");
  addFloorToQueue(1);
}

StatObenF2dwn = digitalRead(LEDSObenF2dwn);
if (StatObenF2dwn == 1 && !isFloorRequested(2)) {
  Serial.println("Calling from x21");
  addFloorToQueue(2);
}

StatObenF2up = digitalRead(LEDSObenF2up);
if (StatObenF2up == 1 && !isFloorRequested(2)) {
  Serial.println("Calling from x22");
  addFloorToQueue(2);
}

StatObenF3dwn = digitalRead(LEDSObenF3dwn);
if (StatObenF3dwn == 1 && !isFloorRequested(3)) {
  Serial.println("Calling from x31");
  addFloorToQueue(3);
```

```
  }

  // Die aktuelle Position des Aufzugsensors auslesen und speichern
  StatSensor1 = digitalRead(Sensor1);
  if (StatSensor1 == 1)
    EtageIst = 0;

  StatSensor2 = digitalRead(Sensor2);
  if (StatSensor2 == 1)
    EtageIst = 1;

  StatSensor3 = digitalRead(Sensor3);
  if (StatSensor3 == 1)
    EtageIst = 2;

  StatSensor4 = digitalRead(Sensor4);
  if (StatSensor4 == 1)
    EtageIst = 3;
}

#pragma region Elevator Queue  //in neueren ArduinoIDE-Versionen kann mit dem pragma-Befehl ein Sketchbereich eingeklappt werden um die Übersichtlichkeit zu verbessern.

// Überprüfung, ob der Aufzug gerade in Bewegung ist. Gibt true zurück, wenn der Aufzug fährt.
bool isCurrentlyRunning() {
  return goingUp || goingDown;
}

// Prüft, ob ein bestimmter Flur bereits angefragt wurde.
// Verbesserte Überprüfung und Abfrage, ob die maximale Anzahl von Etagen überschritten wird.
bool isFloorRequested(int floor) {
  if (floor < 0 || floor >= MAX_FLOORS) {
    return false;  // Unerlaubte Flur-Anfrage, daher false zurückgeben.
  }

  // Überprüfen, ob der Aufzug sich bereits auf dem angefragten Flur befindet.
  if (EtageIst == floor) {
    return true;
  }

  // Durchsuchen der Liste der angefragten Etagen.
  for (int i = 0; i < numRequestedFloors; i++) {
    if (requestedFloors[i] == floor) {
      return true;
    }
  }

  return false;  // Der angegebene Flur wurde nicht angefragt.
}

// Funktion zum Hinzufügen eines Flurs zur Warteschlange.
void addFloorToQueue(int floor) {
```

```
  Serial.print("addFloorToQueue: ");
  Serial.println(floor);

  // Überprüfen, ob der Flur bereits angefragt wurde und die maximale Anzahl nicht überschritten
wurde.
  if (!isFloorRequested(floor) && numRequestedFloors < MAX_FLOORS) {
    requestedFloors[numRequestedFloors] = floor;
    numRequestedFloors++;
  }
}

// Funktion zum Entfernen eines Flurs aus der Warteschlange.
void removeFloorFromQueue(int index) {
  Serial.print("removeFloorFromQueue: ");
  Serial.println(index);

  // Verschieben der nachfolgenden Etagen in der Liste, um den angegebenen Flur zu entfernen.
  for (int i = index; i < numRequestedFloors - 1; i++) {
    requestedFloors[i] = requestedFloors[i + 1];
  }

  numRequestedFloors--;  // Reduzieren der Anzahl angefragter Etagen.
}

// Abrufen des ältesten angefragten Flurs und sicherstellen, dass der Aufzug nicht bereits in
Bewegung ist.
int getOldestEntry() {
  if (numRequestedFloors > 0 && !isCurrentlyRunning()) {
    return requestedFloors[0];
  }

  return -1;  // Kein ältester Flur verfügbar oder der Aufzug ist bereits in Bewegung.
}

#pragma endregion

// Initialisiert das Display, verwendet die Wire-Bibliothek für die Kommunikation über I2C.
void initDisplay() {
  Wire.begin();                        // Initialisiert die Wire-Bibliothek für die I2C-Kommunikation.
  oled.setFont(Cooper21);              // Setzt die Schriftart des Displays.
  oled.begin(&Adafruit128x64, I2C_ADDRESS);  // Initialisiert das Display mit der angegebenen
Adresse.
}

// Aktualisiert den Inhalt des Displays.
void updateDisplay() {
  // oled.set2X();
  oled.clear();  // Löscht den aktuellen Inhalt des Displays.

  // Zeigt die aktuelle Etage (Ist) auf dem Display an.
  oled.print("Ist: ");
  oled.println(EtageIst);
```

```
  // Zeigt die gewünschte Etage (Soll) auf dem Display an.
  oled.print("Soll: ");
  oled.println(EtageSoll);

}


void setup() {
  // Legt den Speaker-Pin als Output fest.
  pinMode(Speaker, OUTPUT);

  // Initialisiert die serielle Kommunikation für Bluetooth und Debugging.
  Serial3.begin(9600);
  Serial.begin(9600);

  // Initialisiert das Display.
  initDisplay();

  // Initialisiert den MP3-Player und den Speaker.
  MP3uSpeaker_INIT();

  // Initialisiert Schalter, LEDs und Sensoren.
  for (int i = 0; i < 13; i++) {
    pinMode(SWITCHES[i], OUTPUT);
    digitalWrite(SWITCHES[i], HIGH);
    pinMode(LEDS[i], INPUT);
  }

  // Initialisiert Sensoren.
  for (int i = 0; i < 4; i++) {
    pinMode(sensors[i], INPUT);
  }

  // Initialisiert den Servomotor für die Tür.
  servo.attach(37);
  for (pos = 2; pos >= 0; pos -= 1) {
    servo.write(pos);
    delay(15);
  }

  // Öffnet die Tür.
  TurAuf();

  // Schließt die Tür.
  TurZu();

  // Setzt den Aufzug zurück.
  resetElevator();

  // Debug-Ausgaben für den Setup-Status.
  Serial.println("Setup beendet");
```

```
  Serial3.println("STARTED");
}

int lastrequest = -1;

#pragma region Bluetooth
// listen function for bluetooth remove tx= and make sure to send only one message
void listen() {
  // Überprüfen, ob Daten über die Bluetooth-Schnittstelle verfügbar sind.
  if (Serial3.available()) {
    // Einzelnes Zeichen lesen.
    char c = Serial3.read();

    // Überprüfen, ob das Zeichen ein Zeilenumbruch ist.
    if (c == '\n') {
      // Entfernen des Präfixes "tx=" aus der Bluetooth-Nachricht.
      bluetoothMessage.remove(0, 3);

      // Behandlung der Bluetooth-Anfrage.
      handleBLErequest();

      // Zurücksetzen des Nachrichtenpuffers.
      bluetoothMessage = "";
    } else {
      // Zeichen zum Nachrichtenpuffer hinzufügen.
      bluetoothMessage += c;
    }
  }
}

void handleBLErequest() {
  // Überprüfen, ob die Bluetooth-Nachricht leer oder nur aus Leerzeichen besteht.
  if (bluetoothMessage == "" || bluetoothMessage == " " || bluetoothMessage == "\n") {
    return;
  }

  // Überprüfen, ob die Nachricht das Gleichheitszeichen enthält, und in diesem Fall die Funktion
verlassen.
  if (bluetoothMessage.indexOf("=") != -1) {
    return;
  }

  // Ausgabe für Debugging-Zwecke.
  Serial.print("handleBLErequest: ");
  Serial.println(bluetoothMessage);

  // Handhabung verschiedener Bluetooth-Anfragen.
  if (bluetoothMessage == "close_door") {
    TurZu();
  } else if (bluetoothMessage == "open_door") {
    TurAuf();
  } else if (bluetoothMessage == "request_floor_0") {
```

```
    Serial.println("Calling from x02");
    addFloorToQueue(0);
  } else if (bluetoothMessage == "request_floor_1") {
    Serial.println("Calling from x13");
    addFloorToQueue(1);
  } else if (bluetoothMessage == "request_floor_2") {
    Serial.println("Calling from x23");
    addFloorToQueue(2);
  } else if (bluetoothMessage == "request_floor_3") {
    Serial.println("Calling from x32");
    addFloorToQueue(3);
  } else if (bluetoothMessage == "alarm") {
    alarm();
  }
}

// Bluetooth-Nachricht senden.
void sendbluetooth() {
  // Überprüfen, ob sich die Etage geändert hat, und in diesem Fall die neue Etage senden.
  if (lastfloor != EtageIst) {
    Serial3.println("floor_" + String(EtageIst));
    lastfloor = EtageIst;

    // Display aktualisieren.
    updateDisplay();
  }
}
#pragma endregion

void loop() {
  // Bluetooth-Nachrichten abhören und verarbeiten.
  listen();

  // Bluetooth-Nachricht senden und Display aktualisieren.
  sendbluetooth();

  // Aufzugsvariablen aktualisieren.
  updateVariables();

  // Türlogik für Untergeschoss
  if (StatUntenDO == 1) {
    // Tür-LED ausschalten und nach kurzer Verzögerung wieder einschalten.
    digitalWrite(SUntenDO, LOW);
    delay(500);
    digitalWrite(SUntenDO, HIGH);

    // Tür öffnen
    TurAuf();
  }

  // Türlogik für Untergeschoss
  if (StatUntenDC == 1) {
```

```
  // Tür-LED ausschalten und nach kurzer Verzögerung wieder einschalten.
  digitalWrite(SUntenDC, LOW);
  delay(500);
  digitalWrite(SUntenDC, HIGH);

  // Tür schließen
  TurZu();
}

// Alarmlogik für Untergeschoss
if (StatUntenAlarm == 1) {
  // Alarm-LED ausschalten und nach kurzer Verzögerung wieder einschalten.
  digitalWrite(SUntenAlarm, LOW);
  delay(500);
  digitalWrite(SUntenAlarm, HIGH);

  // Alarm auslösen
  alarm();
}
// make sure there is a request available otherwise dont move the
if (numRequestedFloors > 0) {
  if(numRequestedFloors > 1){
    // continue

  }else{
    randomFloor();
  }
  EtageSoll = requestedFloors[0];
}else{
  return;
}

  if (getOldestEntry() != -1) {
  if (!isCurrentlyRunning()) {
    updateDisplay();
  }
}

// Fahrmodus
// Hochfahren
if (EtageSoll > EtageIst) {
  if (F == 0) {
    // Tür schließen, wenn noch nicht geschehen.
    TurZu();
  }
  F = 1;
  // Motor für Aufwärtsbewegung aktivieren.
  MotorHoch();
}

// Runterfahren
else if (EtageSoll < EtageIst) {
```

```
    if (F == 0) {
      // Tür schließen, wenn noch nicht geschehen.
      TurZu();
    }
    F = 1;
    // Motor für Abwärtsbewegung aktivieren.
    MotorRunter();
  }

  // Stoppen, wenn die Ziel-Etage erreicht wurde und der Aufzug bereits in Bewegung ist.
  if (EtageSoll == EtageIst && isCurrentlyRunning()) {
    // Aufzug hat die Ziel-Etage erreicht.
    // Motor stoppen, Ton wiedergeben, Bluetooth aktualisieren und Status ausgeben.
    MotorStop();
    EtagenTon();
    sendbluetooth();
    Serial.println("STOPPED");
    Serial3.println("stoped");
    goingUp = false;
    goingDown = false;

    // MP3-Datei basierend auf der erreichten Etage abspielen.
    switch (EtageIst) {
      case 0:
        // Erdgeschoss wurde erreicht.
        myMP3.play(1);
        break;
      case 1:
        // Erster Stock wurde erreicht.
        myMP3.play(2);
        break;
      case 2:
        // Zweiter Stock wurde erreicht.
        myMP3.play(3);
        break;
      case 3:
        // Dritter Stock wurde erreicht.
        myMP3.play(4);
        break;
      default:
        // Unerwartete Etage erreicht.
        Serial.println("Unerwartete Etage erreicht.");
        // Fügen Sie hier ggf. weitere Aktionen für unerwartete Etage hinzu.
        break;
    }

    // Tür öffnen.
    TurAuf();

    // Ältesten Eintrag aus der Warteschlange entfernen und zusätzliche Aktionen basierend auf der
erreichten Etage durchführen.
    removeFloorFromQueue(0);
```

```
    // if queue is empty randomFloor otherwise dont
    if (numRequestedFloors == 0) {
      randomFloor();
    }

    F = 0;
    handleSensors();
  }
}
```