Файл automaticMode.h:

001 #pragma once

002 #include "devices.h"

003 #include "tgCommunication.h"

004 #include "settings.h"

005 // Параметры для усреднения

006 #define MAX\_MEASUREMENTS (AVERAGE\_PERIOD / MEASUREMENT\_INTERVAL) // Количество измерений в периоде

007 // Структура для хранения измерений

008 struct SensorData

009 {

010 float lightLevel[MAX\_MEASUREMENTS];

011 float temperature[MAX\_MEASUREMENTS];

012 float soilTemperature[MAX\_MEASUREMENTS];

013 float humidity[MAX\_MEASUREMENTS];

014 float soilMoisture[MAX\_MEASUREMENTS];

015 int count;

016 unsigned long lastMeasurementTime;

017 };

018 // Глобальная структура для данных

019 SensorData sensorData = {{0}, {0}, {0}, {0}, {0}, 0, 0};

020 // Переменные для отслеживания времени последнего действия

021 unsigned long lastHeatingTime = 0;

022 unsigned long lastWateringTime = 0;

023 // Инициализация структуры

024 void initSensorData()

025 {

026 sensorData.count = 0;

027 sensorData.lastMeasurementTime = millis();

028 for (int i = 0; i < MAX\_MEASUREMENTS; i++)

029 {

030 sensorData.lightLevel[i] = 0;

031 sensorData.temperature[i] = 0;

032 sensorData.soilTemperature[i] = 0;

033 sensorData.humidity[i] = 0;

034 sensorData.soilMoisture[i] = 0;

035 }

036 }

037 // Сбор данных с датчиков

038 void collectSensorData()

039 {

040 if (millis() - sensorData.lastMeasurementTime >= MEASUREMENT\_INTERVAL)

041 {

042 int index = sensorData.count % MAX\_MEASUREMENTS;

043 if (isConnectedBh1750)

044 sensorData.lightLevel[index] = getLightLevel();

045 if (isConnectedBme280)

046 {

047 sensorData.temperature[index] = getTemperature();

048 sensorData.humidity[index] = getHumidity();

049 }

050 if (isConnectedDs18b20)

051 sensorData.soilTemperature[index] = getSoilTemperature();

052 if (isConnectedMoistureSensor)

053 sensorData.soilMoisture[index] = getSoilMoisturePercent();

054 sensorData.count++;

055 sensorData.lastMeasurementTime = millis();

056 }

057 }

058 // Вычисление средних значений

059 void calculateAverages(float &avgLight, float &avgTemp, float &avgSoilTemp, float &avgHum, float &avgSoil)

060 {

061 avgLight = 0;

062 avgTemp = 0;

063 avgSoilTemp = 0;

064 avgHum = 0;

065 avgSoil = 0;

066 int count = min(sensorData.count, MAX\_MEASUREMENTS);

067 if (count == 0)

068 return;

069 for (int i = 0; i < count; i++)

070 {

071 avgLight += sensorData.lightLevel[i];

072 avgTemp += sensorData.temperature[i];

073 avgSoilTemp += sensorData.soilTemperature[i];

074 avgHum += sensorData.humidity[i];

075 avgSoil += sensorData.soilMoisture[i];

076 }

077 avgLight /= count;

078 avgTemp /= count;

079 avgSoilTemp /= count;

080 avgHum /= count;

081 avgSoil /= count;

082 }

083 // Проверка освещенности (без ограничений по длительности)

084 void controlLightLevel(float avgLight)

085 {

086 static bool lightOn = false;

087 if (avgLight < minLightLevel - lightHysteresis && !lightOn)

088 {

089 lightingState = HIGH;

090 digitalWrite(LED\_STRIP\_PIN, lightingState);

091 lightOn = true;

092 bot.sendMessage(CHAT\_ID, "Average light level too low (" + String(avgLight) + " lux) - lighting ON", "");

093 }

094 else if (avgLight > minLightLevel + lightHysteresis && lightOn)

095 {

096 lightingState = LOW;

097 digitalWrite(LED\_STRIP\_PIN, lightingState);

098 lightOn = false;

099 bot.sendMessage(CHAT\_ID, "Average light level restored (" + String(avgLight) + " lux) - lighting OFF", "");

100 }

101 }

102 // Проверка температуры воздуха

103 void controlTemperature(float avgTemp)

104 {

105 static bool coolerOn = false;

106 if (avgTemp > maxTemperature + tempHysteresis && !coolerOn)

107 {

108 coolerState = HIGH;

109 digitalWrite(COOLER\_PIN, coolerState);

110 coolerOn = true;

111 bot.sendMessage(CHAT\_ID, "Average air temperature too high (" + String(avgTemp) + " \*C) - cooler ON", "");

112 }

113 else if (avgTemp < maxTemperature - tempHysteresis && coolerOn)

114 {

115 coolerState = LOW;

116 digitalWrite(COOLER\_PIN, coolerState);

117 coolerOn = false;

118 bot.sendMessage(CHAT\_ID, "Average air temperature restored (" + String(avgTemp) + " \*C) - cooler OFF", "");

119 }

120 }

121 // Проверка температуры почвы (нагрев интервалами, не дольше 5 минут)

122 void controlSoilTemperature(float avgSoilTemp)

123 {

124 static bool heaterOn = false;

125 if (millis() - lastHeatingTime < heatingPause && heaterOn)

126 return; // Пауза между циклами нагрева

127 int heatingDuration = heatingFactor \* (minSoilTemperature - avgSoilTemp); // Длительность пропорциональна отклонению

128 heatingDuration = constrain(heatingDuration, 0, maxHeatingDuration);

129 if (avgSoilTemp < minSoilTemperature - soilTempHysteresis && !heaterOn)

130 {

131 heaterState = HIGH;

132 digitalWrite(HEATER\_PIN, heaterState);

133 highlightHeaterWork();

134 heaterOn = true;

135 lastHeatingTime = millis();

136 bot.sendMessage(CHAT\_ID, "Average soil temperature too low (" + String(avgSoilTemp) + " \*C) - heater ON for " + String(heatingDuration / 1000) + " sec", "");

137 delay(heatingDuration);

138 heaterState = LOW;

139 digitalWrite(HEATER\_PIN, heaterState);

140 highlightHeaterWork();

141 bot.sendMessage(CHAT\_ID, "Heater OFF after heating", "");

142 }

143 else if (avgSoilTemp > minSoilTemperature + soilTempHysteresis && heaterOn)

144 {

145 heaterState = LOW;

146 digitalWrite(HEATER\_PIN, heaterState);

147 highlightHeaterWork();

148 heaterOn = false;

149 bot.sendMessage(CHAT\_ID, "Average soil temperature restored (" + String(avgSoilTemp) + " \*C) - heater OFF", "");

150 }

151 }

152 // Проверка влажности воздуха

153 void controlHumidity(float avgHum)

154 {

155 static bool coolerOn = false;

156 if (avgHum > maxHumidity + humHysteresis && !coolerOn)

157 {

158 coolerState = HIGH;

159 digitalWrite(COOLER\_PIN, coolerState);

160 coolerOn = true;

161 bot.sendMessage(CHAT\_ID, "Average humidity too high (" + String(avgHum) + " %) - cooler ON", "");

162 }

163 else if (avgHum < maxHumidity - humHysteresis && coolerOn)

164 {

165 coolerState = LOW;

166 digitalWrite(COOLER\_PIN, coolerState);

167 coolerOn = false;

168 bot.sendMessage(CHAT\_ID, "Average humidity restored (" + String(avgHum) + " %) - cooler OFF", "");

169 }

170 }

171 // Проверка влажности почвы (импульсный полив)

172 void controlSoilMoisture(float avgSoil)

173 {

174 static bool watering = false;

175 if (millis() - lastWateringTime < wateringPause && watering)

176 return; // Пауза между циклами полива

177 int numPulses = wateringFactor \* (minSoilMoisture - avgSoil); // Количество импульсов пропорционально отклонению

178 numPulses = constrain(numPulses, 0, maxWateringPulses);

179 if (avgSoil < minSoilMoisture - soilHysteresis && !watering)

180 {

181 watering = true;

182 lastWateringTime = millis();

183 bot.sendMessage(CHAT\_ID, "Average soil Moisture too low (" + String(avgSoil) + " %) - starting " + String(numPulses) + " watering pulses", "");

184 for (int i = 0; i < numPulses; i++)

185 {

186 pumpState = HIGH;

187 digitalWrite(PUMP\_PIN, pumpState);

188 bot.sendMessage(CHAT\_ID, "Pump ON for pulse " + String(i + 1), "");

189 delay(pulseWateringDuration);

190 pumpState = LOW;

191 digitalWrite(PUMP\_PIN, pumpState);

192 bot.sendMessage(CHAT\_ID, "Pump OFF after pulse " + String(i + 1), "");

193 if (i < numPulses - 1)

194 delay(wateringPause); // Пауза между импульсами

195 }

196 bot.sendMessage(CHAT\_ID, "Finished watering", "");

197 }

198 else if (avgSoil > minSoilMoisture + soilHysteresis && watering)

199 {

200 watering = false;

201 pumpState = LOW;

202 digitalWrite(PUMP\_PIN, pumpState);

203 bot.sendMessage(CHAT\_ID, "Average soil Moisture restored (" + String(avgSoil) + " %) - watering OFF", "");

204 }

205 static bool coolerOn = false;

206 if (avgSoil > maxSoilMoisture + soilHysteresis && !coolerOn)

207 {

208 coolerState = HIGH;

209 digitalWrite(COOLER\_PIN, coolerState);

210 coolerOn = true;

211 bot.sendMessage(CHAT\_ID, "Average soil Moisture too high (" + String(avgSoil) + " %) - cooler ON", "");

212 }

213 else if (avgSoil < maxSoilMoisture - soilHysteresis && coolerOn)

214 {

215 coolerState = LOW;

216 digitalWrite(COOLER\_PIN, coolerState);

217 coolerOn = false;

218 bot.sendMessage(CHAT\_ID, "Average soil Moisture restored (" + String(avgSoil) + " %) - cooler OFF", "");

219 }

220 }

221 // Основная функция автоматического режима

222 void automaticMode()

223 {

224 if (!automaticModeState)

225 return;

226 collectSensorData();

227 if (millis() - sensorData.lastMeasurementTime >= AVERAGE\_PERIOD)

228 {

229 float avgLight, avgTemp, avgSoilTemp, avgHum, avgSoil;

230 calculateAverages(avgLight, avgTemp, avgSoilTemp, avgHum, avgSoil);

231 if (isConnectedBh1750)

232 controlLightLevel(avgLight);

233 if (isConnectedBme280)

234 {

235 controlTemperature(avgTemp);

236 controlHumidity(avgHum);

237 }

238 if (isConnectedDs18b20)

239 controlSoilTemperature(avgSoilTemp);

240 if (isConnectedMoistureSensor)

241 controlSoilMoisture(avgSoil);

242 // Сброс счетчика после обработки

243 sensorData.count = 0;

244 }

245 }

Файл config.h:

001 #pragma once

002 #define MOISTURE\_SENSOR\_PIN 17

003 #define SOIL\_TEMP\_SENSOR\_PIN 15

004 #define LED\_STRIP\_PIN 35

005 #define PUMP\_PIN 36

006 #define COOLER\_PIN 37

007 #define HEATER\_PIN 38

008 #define YELLOW\_LED\_PIN 1

009 #define RED\_LED\_PIN 20

010 #define GREEN\_LED\_PIN 21

011 // GPIO 8 (SDA)

012 // GPIO 9 (SCL)

013 char \*ssid = "iPhone Ann";

014 char \*password = "12345678910";

015 #define BOT\_TOKEN "8127877503:AAGpV7Wk6JFMBG-SFpz3vonvdET0n0ZKUTk"

016 #define CHAT\_ID "922443025"

017 bool isConnectedMoistureSensor = false;

018 bool isConnectedBme280 = false;

019 bool isConnectedBh1750 = false;

020 bool isConnectedDs18b20 = false;

021 bool heaterState = LOW;

022 bool coolerState = LOW;

023 bool lightingState = LOW;

024 bool pumpState = LOW;

025 bool automaticModeState = false;

026 bool redLedState = LOW;

027 bool greenLedState = LOW;

028 bool yellowLedState = LOW;

029 const int AirValue = 590;

030 const int WaterValue = 360;

Файл connectWiFi.h:

001 #pragma once

002 #include <WiFi.h>

003 #include "config.h"

004 #include "devices.h"

005 void printWifiStatus()

006 {

007 Serial.print("SSID: ");

008 Serial.println(WiFi.SSID());

009 IPAddress ip = WiFi.localIP();

010 Serial.print("IP Address: ");

011 Serial.println(ip);

012 long rssi = WiFi.RSSI();

013 Serial.print("signal strength (RSSI):");

014 Serial.print(rssi);

015 Serial.println(" dBm");

016 }

017 void initWiFi()

018 {

019 redLedState = HIGH;

020 digitalWrite(RED\_LED\_PIN, redLedState);

021 WiFi.mode(WIFI\_STA);

022 WiFi.begin(ssid, password);

023 Serial.print("Connecting to WiFi ");

024 Serial.println(ssid);

025 while (WiFi.status() != WL\_CONNECTED)

026 {

027 flash\_led(RED\_LED\_PIN, 1, 300);

028 Serial.print("Status: ");

029 Serial.println(WiFi.status());

030 redLedState = HIGH;

031 digitalWrite(RED\_LED\_PIN, redLedState);

032 }

033 redLedState = LOW;

034 digitalWrite(RED\_LED\_PIN, redLedState);

035 greenLedState = HIGH;

036 digitalWrite(GREEN\_LED\_PIN, greenLedState);

037 Serial.println("Successful connection to WiFi!");

038 printWifiStatus();

039 }

040 bool scanWiFi(char \*ssid)

041 {

042 bool isFoundSsid = false;

043 Serial.println("scan start");

044 int n = WiFi.scanNetworks();

045 Serial.println("scan done");

046 if (n == 0)

047 {

048 Serial.println("no networks found");

049 }

050 else

051 {

052 Serial.print(n);

053 Serial.println("networks found");

054 for (int i = 0; i < n; ++i)

055 {

056 Serial.print(i + 1);

057 Serial.print(": ");

058 Serial.print(WiFi.SSID(i));

059 Serial.print(" (");

060 Serial.print(WiFi.RSSI(i));

061 Serial.print(")");

062 Serial.println((WiFi.encryptionType(i) == WIFI\_AUTH\_OPEN) ? " " : "\*");

063 delay(10);

064 if (!isFoundSsid)

065 isFoundSsid = ((WiFi.SSID(i)) == ssid);

066 }

067 }

068 Serial.println("");

069 return isFoundSsid;

070 }

071 void setupWiFi()

072 {

073 WiFi.mode(WIFI\_STA);

074 WiFi.disconnect();

075 while (!scanWiFi(ssid))

076 flash\_led(RED\_LED\_PIN, 5, 100);

077 initWiFi();

078 }

079 bool isConnectWiFi()

080 {

081 if (WiFi.status() != WL\_CONNECTED)

082 {

083 redLedState = HIGH;

084 digitalWrite(RED\_LED\_PIN, redLedState);

085 greenLedState = LOW;

086 digitalWrite(GREEN\_LED\_PIN, greenLedState);

087 }

088 return WiFi.status() == WL\_CONNECTED;

089 }

Файл devices.h:

001 #pragma once

002 #include <Wire.h>

003 #include <BH1750.h>

004 #include <Adafruit\_Sensor.h>

005 #include <Adafruit\_BME280.h>

006 #include <OneWire.h>

007 #include <DallasTemperature.h>

008 #include "config.h"

009 #define SEALEVELPRESSURE\_HPA (1013.25)

010 Adafruit\_BME280 bme;

011 BH1750 lightMeter;

012 OneWire oneWire(SOIL\_TEMP\_SENSOR\_PIN);

013 DallasTemperature sensors(&oneWire);

014 void setupLeds()

015 {

016 pinMode(YELLOW\_LED\_PIN, OUTPUT);

017 digitalWrite(YELLOW\_LED\_PIN, LOW);

018 pinMode(RED\_LED\_PIN, OUTPUT);

019 digitalWrite(RED\_LED\_PIN, LOW);

020 pinMode(GREEN\_LED\_PIN, OUTPUT);

021 digitalWrite(GREEN\_LED\_PIN, LOW);

022 }

023 void flash\_led(int LED\_PIN, int flash\_number, int duration)

024 {

025 for (int i = 0; i < flash\_number; i++)

026 {

027 digitalWrite(LED\_PIN, HIGH);

028 delay(duration);

029 digitalWrite(LED\_PIN, LOW);

030 delay(duration);

031 }

032 }

033 // GY-302 or BH1750

034 void setupLightmeter()

035 {

036 bool status = lightMeter.begin();

037 if (!status)

038 {

039 Serial.println("Could not find a valid BH1750 sensor");

040 isConnectedBh1750 = false;

041 }

042 else

043 isConnectedBh1750 = true;

044 }

045 uint16\_t getLightLevel()

046 {

047 uint16\_t lux = lightMeter.readLightLevel();

048 return lux;

049 }

050 void printLightLevel()

051 {

052 uint16\_t lux = getLightLevel();

053 Serial.print("Light: ");

054 Serial.print(lux);

055 Serial.println(" lx");

056 }

057 // BME280

058 void setupBme()

059 {

060 bool status;

061 status = bme.begin(0x76);

062 if (!status)

063 {

064 Serial.println("Could not find a valid BME280 sensor");

065 isConnectedBme280 = false;

066 }

067 else

068 isConnectedBme280 = true;

069 }

070 float getTemperature()

071 {

072 return bme.readTemperature();

073 }

074 int getPressure()

075 {

076 return bme.readPressure() / 100.0F;

077 }

078 int getHumidity()

079 {

080 return bme.readHumidity();

081 }

082 void printBmeValues()

083 {

084 Serial.print("Temperature = ");

085 Serial.print(getTemperature());

086 Serial.println(" \*C");

087 Serial.print("Pressure = ");

088 Serial.print(getPressure());

089 Serial.println(" hPa");

090 Serial.print("Humidity = ");

091 Serial.print(getHumidity());

092 Serial.println(" %");

093 }

094 // Soil Moisture Sensor v1.2

095 void setupSoilMoistureSensor()

096 {

097 Serial.println(analogRead(MOISTURE\_SENSOR\_PIN));

098 if (analogRead(MOISTURE\_SENSOR\_PIN) > 100)

099 isConnectedMoistureSensor = true;

100 else

101 Serial.println("Could not find a valid Soil Moisture Sensor");

102 }

103 int getSoilMoisturePercent()

104 {

105 float soilMoistureValue = analogRead(MOISTURE\_SENSOR\_PIN);

106 float soilMoisturepercent = 100 - (float)(soilMoistureValue - WaterValue) / (AirValue - WaterValue) \* 100;

107 return soilMoisturepercent;

108 }

109 void printSoilMoisturePercent()

110 {

111 Serial.print("Soil moisture value = ");

112 float soilMoisturepercent = getSoilMoisturePercent();

113 if (soilMoisturepercent >= 100)

114 {

115 Serial.println("100 %");

116 }

117 else if (soilMoisturepercent <= 0)

118 {

119 Serial.println("0 %");

120 }

121 else if (soilMoisturepercent > 0 && soilMoisturepercent < 100)

122 {

123 Serial.print(soilMoisturepercent);

124 Serial.println("%");

125 }

126 }

127 // DS18B20

128 void setupDs18b20()

129 {

130 sensors.begin();

131 sensors.requestTemperatures();

132 if (sensors.getTempCByIndex(0) == -127)

133 {

134 Serial.println("Could not find a valid DS18B20 sensor");

135 isConnectedDs18b20 = false;

136 }

137 else

138 isConnectedDs18b20 = true;

139 }

140 float getSoilTemperature()

141 {

142 sensors.requestTemperatures();

143 float temp = sensors.getTempCByIndex(0);

144 return temp;

145 }

146 void printSoilTemperature()

147 {

148 sensors.requestTemperatures();

149 float temperatureC = sensors.getTempCByIndex(0);

150 float temperatureF = sensors.getTempFByIndex(0);

151 Serial.print(temperatureC);

152 Serial.println("ºC");

153 Serial.print(temperatureF);

154 Serial.println("ºF");

155 }

156 void printSensorsValues()

157 {

158 printLightLevel();

159 printBmeValues();

160 printSoilMoisturePercent();

161 printSoilTemperature();

162 }

163 void checkConnectsSensors()

164 {

165 if (analogRead(MOISTURE\_SENSOR\_PIN) < 100 || getSoilMoisturePercent() < 0)

166 {

167 isConnectedMoistureSensor = false;

168 Serial.println("Could not find a valid Soil Moisture Sensor");

169 }

170 else

171 isConnectedMoistureSensor = true;

172 if (!bme.begin(0x76))

173 {

174 Serial.println("Could not find a valid BME280 sensor");

175 isConnectedBme280 = false;

176 }

177 else

178 isConnectedBme280 = true;

179 if (!lightMeter.begin())

180 {

181 Serial.println("Could not find a valid BH1750 sensor");

182 isConnectedBh1750 = false;

183 }

184 else

185 isConnectedBh1750 = true;

186 sensors.requestTemperatures();

187 if (sensors.getTempCByIndex(0) == -127)

188 {

189 Serial.println("Could not find a valid DS18B20 sensor");

190 isConnectedDs18b20 = false;

191 }

192 else

193 isConnectedDs18b20 = true;

194 }

195 void highlightHeaterWork()

196 {

197 if (heaterState == LOW)

198 {

199 yellowLedState = LOW;

200 digitalWrite(YELLOW\_LED\_PIN, yellowLedState);

201 }

202 else

203 {

204 yellowLedState = HIGH;

205 digitalWrite(YELLOW\_LED\_PIN, yellowLedState);

206 }

207 }

208 // setup output pins

209 void setupDevices()

210 {

211 pinMode(LED\_STRIP\_PIN, OUTPUT);

212 digitalWrite(LED\_STRIP\_PIN, LOW);

213 pinMode(PUMP\_PIN, OUTPUT);

214 digitalWrite(PUMP\_PIN, LOW);

215 pinMode(HEATER\_PIN, OUTPUT);

216 digitalWrite(HEATER\_PIN, LOW);

217 pinMode(COOLER\_PIN, OUTPUT);

218 digitalWrite(COOLER\_PIN, LOW);

219 }

Файл settings.h:

001 #pragma once

002 // Пороговые значения

003 const int minLightLevel = 50; // Минимальный уровень освещенности (люкс)

004 const int maxLightLevel = 1500; // Максимальный уровень освещенности (люкс)

005 const int minTemperature = 20; // Минимальная температура воздуха (°C)

006 const int maxTemperature = 35; // Максимальная температура воздуха (°C)

007 const int minSoilTemperature = 18; // Минимальная температура почвы (°C)

008 const int maxSoilTemperature = 30; // Максимальная температура почвы (°C)

009 const int minHumidity = 30; // Минимальная влажность воздуха (%)

010 const int maxHumidity = 80; // Максимальная влажность воздуха (%)

011 const int minSoilMoisture = 20; // Минимальная влажность почвы (%)

012 const int maxSoilMoisture = 80; // Максимальная влажность почвы (%)

013 // Гистерезис для предотвращения скачков

014 const int lightHysteresis = 10; // Гистерезис для освещенности (люкс)

015 const int tempHysteresis = 3; // Гистерезис для температуры воздуха (°C)

016 const int soilTempHysteresis = 3; // Гистерезис для температуры почвы (°C)

017 const int humHysteresis = 5; // Гистерезис для влажности воздуха (%)

018 const int soilHysteresis = 10; // Гистерезис для влажности почвы (%)

019 // Параметры для нагрева

020 const int maxHeatingDuration = 300000; // Максимальная длительность одного цикла нагрева (5 минут = 300000 мс)

021 const int heatingPause = 60000; // Пауза между циклами нагрева (1 минута = 60000 мс)

022 const float heatingFactor = 60000; // Коэффициент для нагрева (мс на °C отклонения)

023 // Параметры для полива

024 const int pulseWateringDuration = 4000; // Длительность одного импульса полива (4 секунды = 4000 мс)

025 const int wateringPause = 10000; // Пауза между импульсами полива (10 секунд = 10000 мс)

026 const int maxWateringPulses = 5; // Максимальное количество импульсов за один цикл полива

027 const float wateringFactor = 0.5; // Коэффициент для количества импульсов (импульсов на % отклонения)

028 // Параметры для усреднения данных

029 const int MEASUREMENT\_INTERVAL = 10000; // Интервал между измерениями (10 секунд)

030 const int AVERAGE\_PERIOD = 60000; // Период усреднения (1 минута)

Файл SmartGrow.ino:

001 #include "devices.h"

002 #include "tgCommunication.h"

003 #include "automaticMode.h"

004 #include "connectWiFi.h"

005 void setup()

006 {

007 Serial.begin(115200);

008 Wire.begin();

009 setupLightmeter();

010 setupBme();

011 setupSoilMoistureSensor();

012 setupDs18b20();

013 setupLeds();

014 setupDevices();

015 setupWiFi();

016 setupTelegram();

017 initSensorData();

018 }

019 void loop()

020 {

021 checkConnectsSensors();

022 printSensorsValues();

023 if (automaticModeState)

024 automaticMode();

025 if (!isConnectWiFi())

026 {

027 setupWiFi();

028 }

029 loopTelegram();

030 delay(500);

031 }

Файл tgCommunication.h:

001 #pragma once

002 #include <WiFiClientSecure.h>

003 #include <ArduinoJson.h>

004 #include <UniversalTelegramBot.h>

005 #include "config.h"

006 #include "devices.h"

007 #include <WiFi.h>

008 WiFiClientSecure client;

009 UniversalTelegramBot bot(BOT\_TOKEN, client);

010 int botRequestDelay = 500;

011 unsigned long lastTimeBotRan;

012 void sendStats(String chat\_id)

013 {

014 String stats = "Stats:\n";

015 stats += isConnectedBh1750 ? ("Light: " + String(getLightLevel()) + " lux \n") : "Could not find a valid BH1750 sensor\n";

016 stats += isConnectedBme280 ? ("Temperature: " + String(getTemperature()) + " \*C\nPressure: " + String(getPressure()) + " hPa\nHumidity: " + String(getHumidity()) + " %\n") : "Could not find a valid BME280 sensor\n";

017 stats += isConnectedMoistureSensor ? ("Soil moisture: " + String(getSoilMoisturePercent()) + " %\n") : "Could not find a valid Soil Moisture Sensor\n";

018 stats += isConnectedDs18b20 ? ("Soil temperature: " + String(getSoilTemperature()) + " \*C\n") : "Could not find a valid DS18B20 Sensor\n";

019 bot.sendMessage(chat\_id, stats, "");

020 }

021 void handleNewMessages(int numNewMessages)

022 {

023 Serial.println("handle New Messages ");

024 Serial.println(String(numNewMessages));

025 for (int i = 0; i < numNewMessages; i++)

026 {

027 String chat\_id = String(bot.messages[i].chat\_id);

028 if (chat\_id != CHAT\_ID)

029 {

030 bot.sendMessage(chat\_id, "Unauthorized user", "");

031 continue;

032 }

033 String text = bot.messages[i].text;

034 Serial.println(text);

035 String from\_name = bot.messages[i].from\_name;

036 if (text == "/automatic\_mode\_on")

037 {

038 bot.sendMessage(chat\_id, "automatic mode set to ON ", "");

039 automaticModeState = true;

040 }

041 if (text == "/automatic\_mode\_off")

042 {

043 bot.sendMessage(chat\_id, "automatic mode set to OFF ", "");

044 automaticModeState = false;

045 }

046 if (text == "/sensors\_stats")

047 {

048 sendStats(chat\_id);

049 }

050 if (text == "/devices\_status")

051 {

052 String status = "Devices status :\n";

053 status += "Heater: " + String(heaterState == LOW ? "OFF" : "ON") + "\n";

054 status += "Cooler: " + String(coolerState == LOW ? "OFF" : "ON") + "\n";

055 status += "Lighting: " + String(lightingState == LOW ? "OFF" : "ON") + "\n";

056 status += "Pump: " + String(pumpState == LOW ? "OFF" : "ON") + "\n";

057 bot.sendMessage(chat\_id, status, "");

058 }

059 if (text == "/heater\_on")

060 {

061 bot.sendMessage(chat\_id, "heater state set to ON", "");

062 heaterState = HIGH;

063 digitalWrite(HEATER\_PIN, heaterState);

064 highlightHeaterWork();

065 }

066 if (text == "/heater\_off")

067 {

068 bot.sendMessage(chat\_id, "heater state set to OFF", "");

069 heaterState = LOW;

070 digitalWrite(HEATER\_PIN, heaterState);

071 highlightHeaterWork();

072 }

073 if (text == "/cooler\_on")

074 {

075 bot.sendMessage(chat\_id, "cooler state set to ON", "");

076 coolerState = HIGH;

077 digitalWrite(COOLER\_PIN, coolerState);

078 }

079 if (text == "/cooler\_off")

080 {

081 bot.sendMessage(chat\_id, "cooler state set to OFF", "");

082 coolerState = LOW;

083 digitalWrite(COOLER\_PIN, coolerState);

084 }

085 if (text == "/lighting\_on")

086 {

087 bot.sendMessage(chat\_id, "lighting state set to ON", "");

088 lightingState = HIGH;

089 digitalWrite(LED\_STRIP\_PIN, lightingState);

090 }

091 if (text == "/lighting\_off")

092 {

093 bot.sendMessage(chat\_id, "lighting state set to OFF", "");

094 lightingState = LOW;

095 digitalWrite(LED\_STRIP\_PIN, lightingState);

096 }

097 if (text == "/pump\_on")

098 {

099 bot.sendMessage(chat\_id, "pump state set to ON", "");

100 pumpState = HIGH;

101 digitalWrite(PUMP\_PIN, pumpState);

102 }

103 if (text == "/pump\_off")

104 {

105 bot.sendMessage(chat\_id, "pump state set to OFF", "");

106 pumpState = LOW;

107 digitalWrite(PUMP\_PIN, pumpState);

108 }

109 if (text == "/help")

110 {

111 String help = "Hello, " + from\_name + "!\n";

112 help += "Use the following commands to control your greenhouse:\n\n";

113 help += "/automatic\_mode\_on and /automatic\_mode\_off - to use the automatic mode \n";

114 help += "/sensors\_stats - to request sensors values \n";

115 help += "/devices\_status - to request devices status\n";

116 help += "/heater\_on and /heater\_off - to control heating\n";

117 help += "/cooler\_on and /cooler\_off - to control ventilation\n";

118 help += "/lighting\_on and /lighting\_off - to control lighting\n";

119 help += "/pump\_on and /pump\_off - to control watering\n";

120 bot.sendMessage(chat\_id, help, "");

121 }

122 }

123 }

124 void setupTelegram()

125 {

126 client.setCACert(TELEGRAM\_CERTIFICATE\_ROOT);

127 }

128 void loopTelegram()

129 {

130 if (millis() > lastTimeBotRan + botRequestDelay)

131 {

132 int numNewMessages = bot.getUpdates(bot.last\_message\_received + 1);

133 while (numNewMessages)

134 {

135 Serial.println("got response");

136 handleNewMessages(numNewMessages);

137 numNewMessages = bot.getUpdates(bot.last\_message\_received + 1);

138 }

139 lastTimeBotRan = millis();

140 }

141 }