#include

#include // RFID MFRC255 Library

#include // NeoPixel Library

#define SS\_PIN 10 // RFID data pin

#define RST\_PIN 9 // Reset pin

#define LED\_PIN 4 // DataPin out for LED string

#define LED\_COUNT 46 // Number of LED's

#define BRIGHTNESS 100 // Brightness level 0 till 225

#define RFIDTAG\_NOR 80 // The number of RFID "normal use" tags #define RFIDTAG\_PRI 10 // The number of RFID "priority" tags #define RFIDTAG\_ONE 30 // The number of RFID "one time use" tags #define FLOWSENSOR 2 // Flow sensor pin

#define LITERS\_NOR 8 // Amount of water that can be used for one shower session #define LITERS\_PRI 10 // Amount of water that can be used for one shower session #define LITERS\_ONE 6 // Amount of water that can be used for one shower session #define TICKS\_TO\_LITERS 444 // Amount of senor ticks to indicate one liter

#define SHOWER\_PIN 8 // This relais activates the solenoid valve that lets water to the shower #define SHOWER\_RESET\_TIME 600000 // Time to reset the shower

—

#define RFID\_RESET\_TIME 900000 // Time to reset the RFID tag

#define MAX\_NORMAL\_TAG\_COUNT 100

#define MAX\_ONETIMEUSE\_TAG\_COUNT 30

#define MAX\_PRIORITY\_TAG\_COUNT 10

Adafruit\_NeoPixel strip (LED\_COUNT, LED\_PIN, NEO\_GRB + NEO\_KHZ800); // Declare our NeoPixel strip object MFRC522 mfrc522(SS\_PIN, RST\_PIN); // Create MFRC522 instance

=

static const uint32\_t green strip.Color( 0, 255, 0);

static const uint32\_t blue strip.Color( 0, 0, 255);

=

=

static const uint32\_t white strip.Color (225, 225, 225); static const uint32\_t red

// 80 normal tags

=

strip.Color(225, 0, 0);

static const uint32\_t UID\_NOR [RFIDTAG\_NOR]

=

{

0x042A92ED, 0x042A935F,0x042ABDC9,0x042AD89D, 0x042AE4B9,0x042B05A0,0x042B0CD6,0x042B0DD8,

0x042B1371, 0x042B1760, 0x042B19EA, 0x042B2D71, 0x042B308E, 0x042B34E0, 0x3D28C25C, 0x3D28DA5E, 0x3D28E455, 0x3D28F707, 0x3D28FA0F, 0x3D28FF4A, 0x3D2934C9, 0x3D293C59, 0x3D294300, 0x3D294E56,0x3D295B11, 0x3D29651B, 0x3D297DF7, Ox3D299649,0x3D29B25C, 0x3D29B5B3, 0x3D29B5CD, 0x3D29BA17, 0x3D29DDCF, 0x3D29E404, 0x3D29F0B5, 0x3D29FDFC, 0x3D2A0E1B, Ox3D2A147A, 0x3D2A1A0E, 0x3D2A1FBD, 0x3D2A5DC2, 0x3D2A5DF3,0x3D2AB52F,0x3D2ABF30, 0x3D2AC685,0x3D2AC896,0x3D2ACC65, Ox3D2AD35A, 0x3D2AD429,0x3D2AD69C, 0x3D2AD8DD, 0x3D2AE154,0x3D2AE476, 0x3D2AE704, 0x3D2AF36B, 0x3D2AF39F, 0x3D2AFDBO, 0X3D2AFE6E, 0x3D2B0134,0x3D2B016F, 0x3D2B0527, 0x3D2B0AC9, 0x3D2B0B47, 0x3D2B0B64, 0x3D2B0E29,0x3D2B0EAF, 0x3D2B14EF, 0x3D2B1517,0x3D2B159B,0x3D2B1770,0x3D2B1A7B, 0x3D2B20C4,0x3D2B2A0F, 0x3D2B2A63,0x3D2B36A5,0x3D2B36FB,0x3D2B3D47

};

// move to this array for priority tags

static const uint32\_t UID\_PRI[RFIDTAG\_PRI]

=

{

Ox3D2CEBA4, 0x042A8EB6, 0x042B2442,0x042AA6DF // the test tag without hole

};

// 22 one-time-use tags

static const uint32\_t UID\_ONE [RFIDTAG\_ONE]

=

{

0x042A8346, 0x042A942F, 0x042B0834, 0x3D28C230, 0x3D297878, 0x3D29C71D, 0x3D2A089B, 0x3D2A1CDE, 0x3D2AB45F, 0x3D2ACF66, 0x3D2AE1F5,0x3D2AEAAE, 0x3D2AEB57,0x3D2AEF4D, 0x3D2AF7E2, 0x3D2AFD70, 0x3D2B1204, 0x3D2B31B3, 0x3D2B3CE7,0x3D2B44F0, Ox3D297878,0x3D2A1CDE

};

bool registered\_RFIDTAG\_NOR [RFIDTAG\_NOR]; // This array keeps track of the RFID tag that have been scanned long registered\_RFIDTAG\_TIME\_NOR [RFIDTAG\_NOR]; // This array keeps track of the RFID tag time

bool registered\_RFIDTAG\_PRI[RFIDTAG\_PRI]; // This array keeps track of the RFID tag that have been scanned long registered\_RFIDTAG\_TIME\_PRI[RFIDTAG\_PRI]; // This array keeps track of the RFID tag time

bool registered\_RFIDTAG\_ONE [RFIDTAG\_ONE]; // This array keeps track of the RFID tag that have been scanned //long registered\_RFIDTAG\_TIME\_ONE [RFIDTAG\_ONE]; // This array keeps track of the RFID tag time

const float totalTicks\_NOR

=

const float totalTicks PRI = const float totalTicks\_ONE

volatile int sensorTicks int ledIndex

=

=

=

\*

TICKS\_TO\_LITERS LITERS\_NOR;

\* TICKS\_TO\_LITERS LITERS\_PRI**;**

TICKS\_TO\_LITERS \* LITERS\_ONE;

0; //This is the value we intend to calculate.

0; // This index is a count of the LED's that need to be switched off

const float ticksToled\_NOR indicate flow

const float ticksToled\_PRI indicate flow

const float ticks Toled\_ONE indicate flow

=

=

=

totalTicks\_NOR / LED\_COUNT; // Amount of ticks needed for 1x LED to change to

totalTicks\_PRI **/** LED\_COUNT; // Amount of ticks needed for 1x LED to change to

totalTicks\_ONE / LED\_COUNT; // Amount of ticks needed for 1x LED to change to

bool shower\_Is\_Running

bool priority

bool oneTime

=

=

=

0; // Satusindication

0; // Satusindication

0; // Satusindication

long showerStartTime

=

0;

// resulting value is 4 bytes (32 bits), or 8 HEX characters uint32\_t toUint32(uint8\_t\* byteArray) {

uint32\_t value = 0x0;

for (int i = 0; i < 4; **++i)** {

==

value = (uint32\_t) (byteArray[i]) **<<** (i \* 8);

}

return value;

}

void printUint32AsHex (uint32\_t value) {

for (int i = 0; i < 4; ++**i**) {

Serial.print((uint8\_t)(value >> (i \* 8)), HEX);

Serial.print(" ");

Serial.println();

}

}

void printUint32(uint32\_t value) {

Serial.print("0x");

Serial.println(value, HEX);

}

void setup() {

Serial.begin(115200); // Initiate a serial communication

Serial.println();

Serial.println(F("Boot shower program"));

Serial.println();

SPI.begin(); // Initiate SPI bus

mfrc522.PCD\_Init(); // Initiate MFRC522

pinMode (SHOWER\_PIN, OUTPUT);

digitalWrite (SHOWER\_PIN, HIGH); // LOW is active HIGH is decativate. (inverted)

strip.begin();

strip.setBrightness (BRIGHTNESS); // Set LED BRIGHTNESS

strip.fill(blue); // Set LED strip to blue

strip.show();

pinMode(FLOWSENSOR, INPUT);

digitalWrite(FLOWSENSOR, HIGH);

attachInterrupt(digitalPinToInterrupt (2), flowSensorTicks, FALLING); //Configures interrupt 0 (pin 2 on the

Arduino Uno) to run the function "Flow"

}

void loop() {

/\*

Serial.print("Sensorticks: ");

Serial.print('\t');

Serial.print(sensorTicks);

Serial.print("Ticks "); Serial.println();

\*/

if (shower\_Is\_Running == **1)** {

uint32\_t tag = readTag();

if (tag) {

Serial.println(F("Already running"));

}

showerRunning();

} else {

showerReady();

}

for (int i = 0; i < RFIDTAG\_NOR; i++**)** {

if (registered\_RFIDTAG\_NOR[i]) {

if (millis() > registered\_RFIDTAG\_TIME\_NOR[i] + RFID\_RESET\_TIME) { Serial.println();

Serial.println(F("Timed out. Tag can be used again (if shower ended).")); registered\_RFIDTAG\_NOR [i] = 0;

}

}

}

for (int i

=

0; i < RFIDTAG\_PRI; i++) {

if (registered\_RFIDTAG\_PRI[i]) {

if (millis() > registered\_RFIDTAG\_TIME\_PRI[i] + RFID\_RESET\_TIME) { Serial.println();

Serial.println(F("Timed out. Tag can be used again (if shower ended).")); registered\_RFIDTAG\_PRI[i] = 0;

}

}

}

}

void flowSensorTicks() {

sensorTicks++;

}

void enableAccess (uint32\_t color) {

Serial.println("Authorized access");

Serial.println();

strip.begin();

strip.setBrightness (BRIGHTNESS);

strip.fill(color);

strip.show();

// Millis() is being called for the shower reset in the showerRunning function showerStartTime **=**

shower Running();

}

millis();

uint32\_t readTag() {

if (!mfrc522.PICC\_IsNewCardPresent()) {

return 0;

}

if (!mfrc522.PICC\_ReadCardSerial()) {

return 0; }

uint32\_t tagId = toUint32 (mfrc522.uid.uidByte); return tagId;

}

void showerReady() {

strip.begin();

strip.setBrightness (BRIGHTNESS);

strip.fill(blue);

strip.show();

uint32\_t tagId = readTag();

if (!tagId) {

return;

}

Serial.print("UID tag ");

printUint32AsHex(tagId);

for (int i = 0; i < RFIDTAG\_NOR; i++){ if (tagId

==

UID\_NOR[i] && registered\_RFIDTAG\_NOR[i] registered\_RFIDTAG\_NOR[i] = **1;**

==

0){

registered\_RFIDTAG\_TIME\_NOR[i] = millis(); enableAccess (green);

}

}

for (int i

=

0; i < RFIDTAG\_PRI**; i**++){

**if** (tagId == UID\_PRI[i] && registered\_RFIDTAG\_PRI[i]

registered\_RFIDTAG\_PRI**[i**]

registered\_RFIDTAG\_TIME\_PRI[i]

priority = **1;**

Serial.println("Priority mode");

enableAccess(white);

==

0) {

= **1;**

=

millis();

}

}

for (int i = 0; i < RFIDTAG\_ONE; i++){

if (tagId == UID\_ONE [i] && registered\_RFIDTAG\_ONE [i] registered\_RFIDTAG\_ONE [i] = 1;

//registered\_RFIDTAG\_TIME\_ONE[i] = millis();

oneTime

=

**1;**

Serial.println("One-time use mode"); enableAccess(red);

}

}

}

void showProgress() {

static int progress\_dot

progress\_dot++**;**

=

0;

if (progress\_dot % 1000 == 0) {

static int progress\_dot\_line = 0; progress\_dot\_line++**;**

Serial.print(".");

if (progress\_dot\_line % 100 == 0) {

Serial.println();

}

}

}

void showerRunning() {

showProgress();

//Serial.println("Shower Running");

shower\_Is\_Running **= 1;**

digitalWrite(SHOWER\_PIN, LOW);

int ledIndex\_NOR = sensorTicks / ticksToled\_NOR;

==

0){

int ledIndex\_PRI

=

sensorTicks / ticksToled\_PRI;

int ledIndex\_ONE = sensorTicksticks Toled\_ONE;

// Requires a small rewrite if you want to do for example a short RED flash when a tag is nearby for the // second or nth time.

// This is because you want to store and restore a pattern.

// Hence, you will need an array for LED\_COUNT pixels

if (ledIndex\_NOR > 0) {

strip.setPixelColor (LED\_COUNT-led Index\_NOR, strip.Color(0,0,0)); // This turns off leds by water usage strip.show();

} else if (ledIndex\_PRI > 0) {

strip.setPixelColor (LED\_COUNT-led Index\_PRI**,** strip.Color(0,0,0)); // This turns off leds by water usage strip.show();

} else if (ledIndex\_ONE > 0) {

strip.setPixelColor (LED\_COUNT-led Index\_ONE, strip.Color(0,0,0)); // This turns off leds by water usage strip.show();

}

if (priority == **1)** {

if (sensorTicks > totalTicks\_PRI) {

Serial.println();

Serial.println(F("Reset shower. All water is used"));

reset();

}

else if (millis() > showerStartTime + SHOWER\_RESET\_TIME) { Serial.println();

Serial.println(F("Reset shower. No water is used, but time is up."));

reset();

}

} else if (oneTime

==

**1)** {

if (sensorTicks totalTicks\_ONE) {

Serial.println();

Serial.println(F("Reset shower. All water is used"));

reset();

}

else if (millis() > showerStartTime + SHOWER\_RESET\_TIME) { Serial.println();

Serial.println(F("Reset shower. No water is used, but time is up."));

reset();

}

} else {

if (sensorTicks > totalTicks\_NOR) {

Serial.println();

Serial.println(F("Normal mode. Reset shower. All water is used"));

reset();

}

else if (millis() > showerStartTime + SHOWER\_RESET\_TIME) { Serial.println();

Serial.println(F("Reset shower. No water is used, but time is up."));

reset();

}

}

}

void reset() {

if (priority) {

Serial.println(F("End prio mode"));

} else if (oneTime) {

Serial.println(F("End one-time mode"));

} else {

Serial.println(F("End normal mode"));

}

Serial.println(F("Shower reset")); shower\_Is\_Running = 0;

priority = 0;

oneTime

=

0;

sensorTicks

=

0;

digitalWrite(SHOWER\_PIN, HIGH);

}