Folyosóvilágítás

A feladat az otthoni folyosó világításának automatizálása volt. A jelenlegi rendszerben vagy nyomógombokkal lehetett kapcsolni a világítást egy Fibaro FGS-211 Switch-en keresztül valamint egy elemes mozgásérzékelővel automatikusan. A mozgásérzékelést idáig egy IKEA-s mozgásérzékelővel oldottam meg mely több problémát is előhozott.

Ez és a hasonló mozgásérzékelők elemes működésűek, ahol állandó probléma az elemek gyors lemerülése. A másik jellemzően felmerülő gond, az érzékelők érzékelési idejének (változás esetén a következő érzékelés) nagy időintervalluma. Ez például az IKEA-s érzékelő esetén minimum 3 perc, de más típusoknál sem találni általában 1 percen belül újra érzékelő típusokat. Ez általában az energiafogyasztás csökkentése, a tápellátást biztosító áramforrás gyors lemerülésének elkerülése miatt van így. Mivel általában az ember csak végigmegy a folyosón nincs szükség hosszabb időtartamú megvilágításra, viszonylag hamar le lehet kapcsolni a lámpákat. Azonban mivel hosszú az ismételt érzékelés időpontja, ha valami miatt hamarabb érkezik valaki a folyosóra, a világítás nem kapcsol fel. A kézi felkapcsolás természetesen működik, de a kényelem miatt valamilyen másik megoldást kellett keresni.

A házban egy régebbi típusú riasztórendszer működik, melynek minden helységben vannak érzékelői és a központi egységben minden érzékelőnek van egy normál Off kimenete. A riasztórendszer központi áramforrásról működik, nincs szükség az egyes mozgásérzékelőkben elemcserére, emiatt a mozgásérzékelés is folyamatosan történik. Felmerült, hogy gazdaságosan, minél kisebb ráfordítással valamilyen módon a riasztórendszert felhasználva működtessük a folyosóvilágítást.

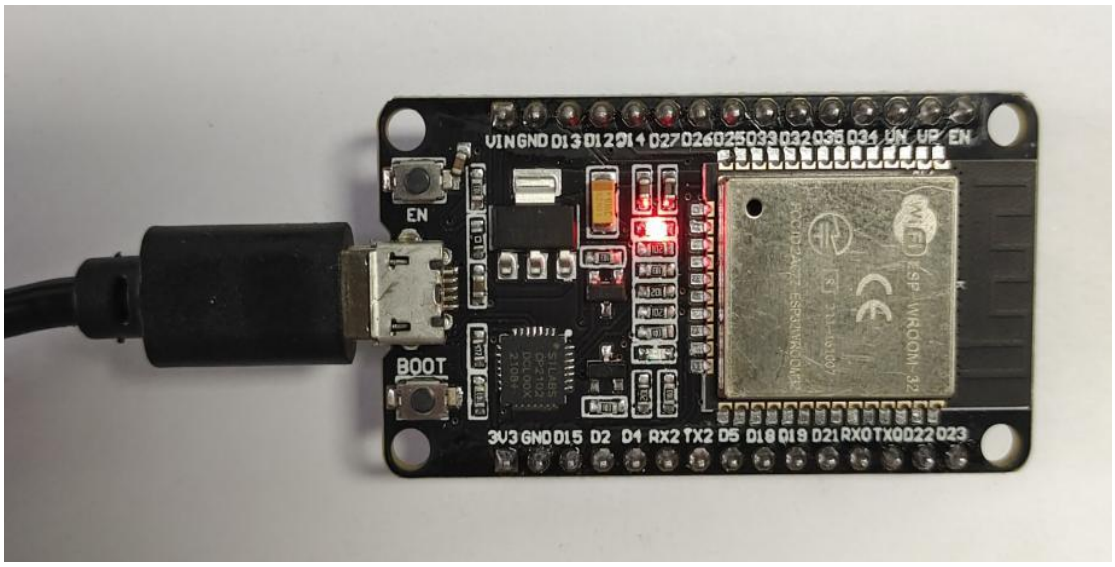
A döntés a méret és az ár miatt egy ESP32 rendszerre esett, mely jóval túlszárnyalta az eredeti elvárásokat, valamint anyagilag is körülbelül 3-4 ezer forintból megvalósítható. A nagyfeszültségű rendszer kapcsolása egy DC 5V feszültséggel működő relay-vel lett megvalósítva, melyet az ESP32 közvetlenül tud vezérelni. A mintaprojekt egy szerelőlapon lett megvalósítva, ahol a mozgásérzékelő jelét egy nyomógomb helyettesíti.

A rendszerhez felhasznált alkatrészek:

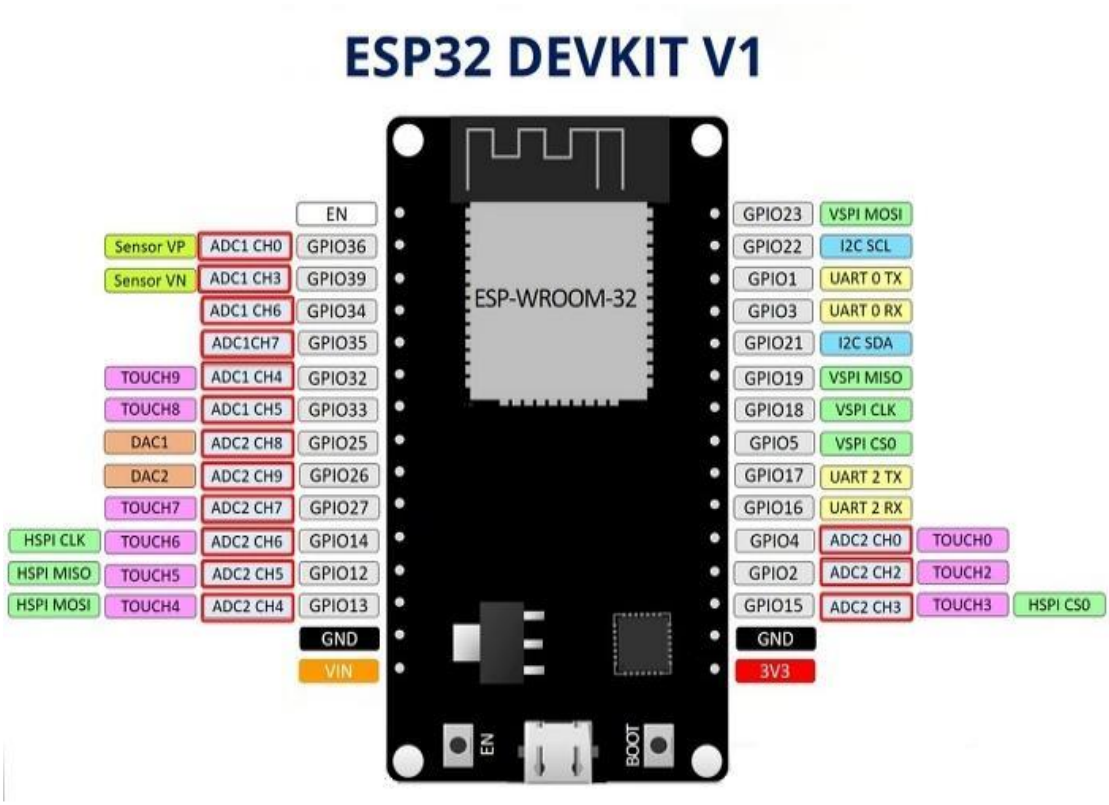
- ESP32 Wifi Board
- 5 mm LED
- 220 Ohm ellenállás
- 10 kOhm ellenállás
- DC 5V AC220 relay
- HC-SR501 PIR Motion sensor
- nyomógomb
- szerelőlap

A mintaprojekt elkészítéséhez szükséges alkatrészek egy Espressif ESP32 Starter kitből lettek felhasználva.

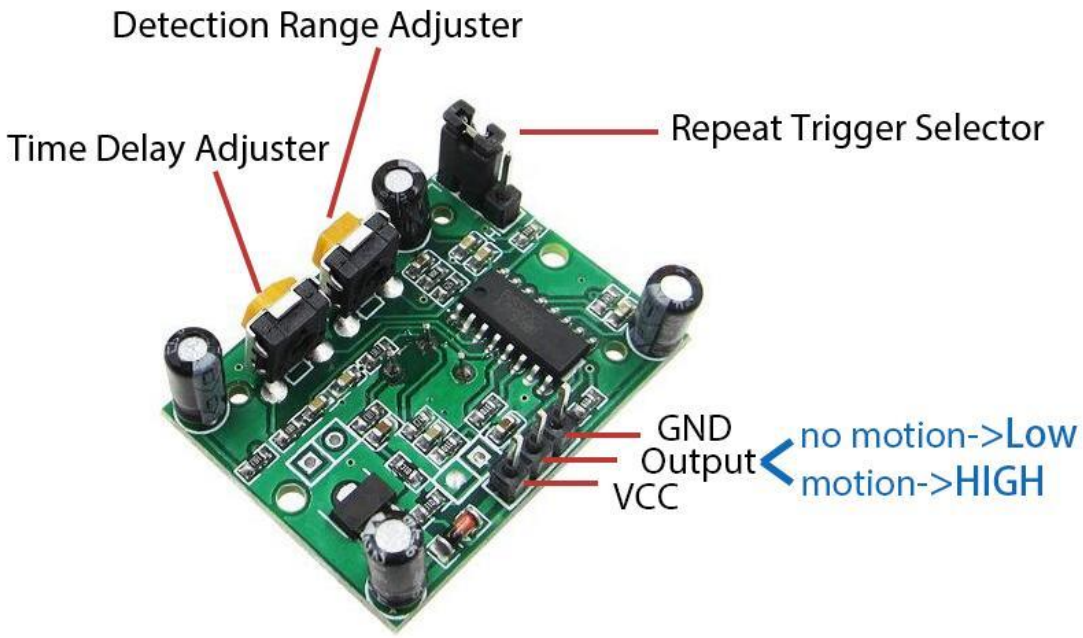
ESP32 vezérlő



ESP32 vezérlő lábkiosztás



HC-SR501 Motion Sensor

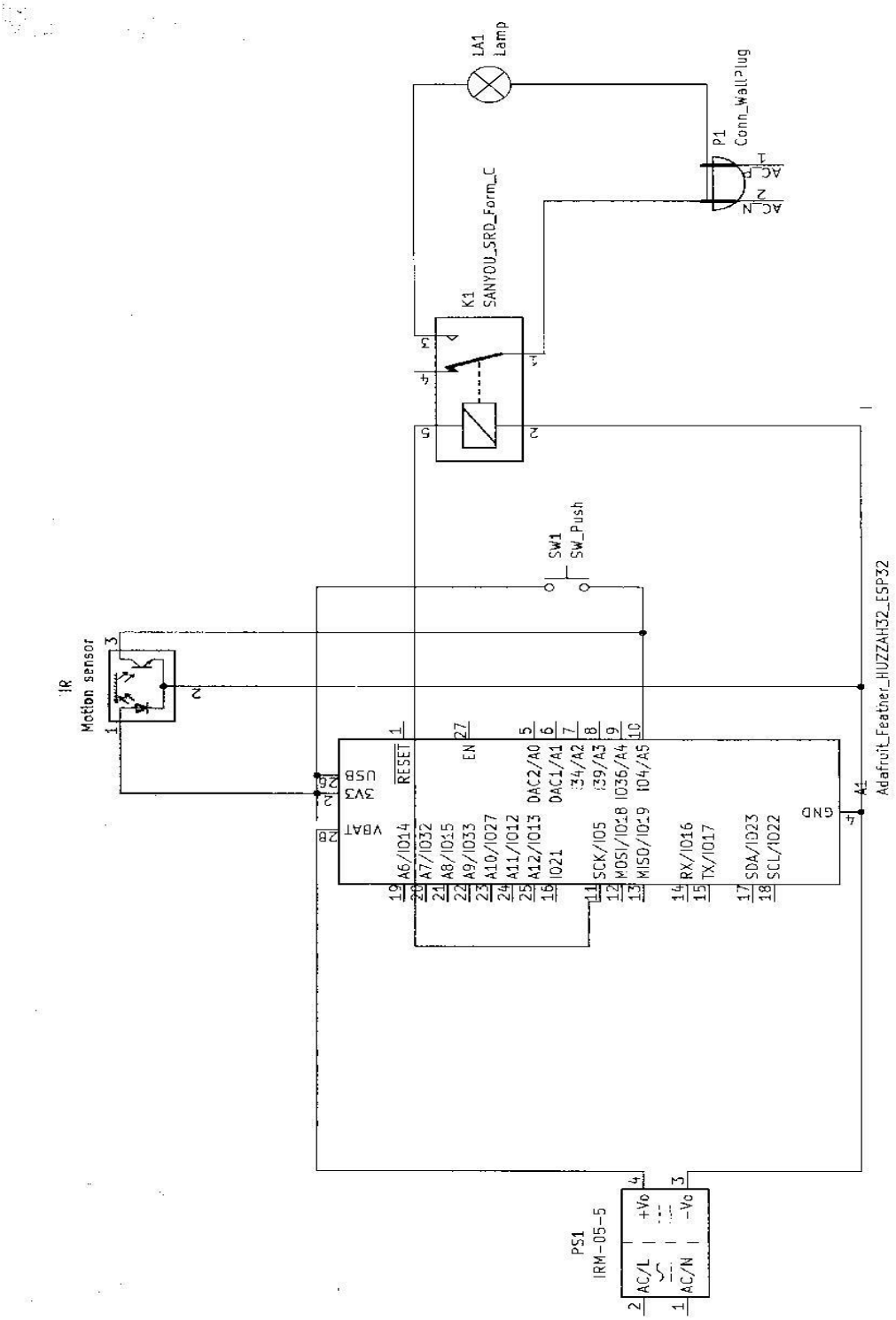


DC 5V AC 220 V Relay

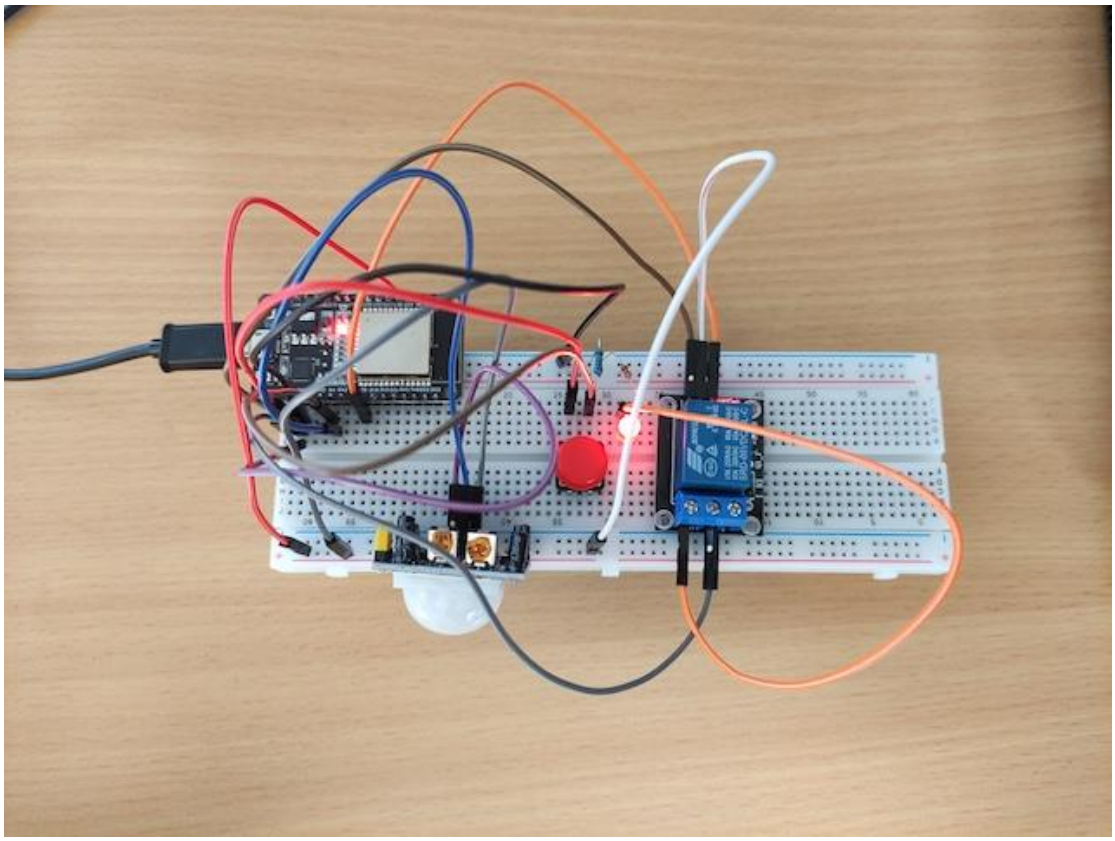
A világítás vezérléséhez az ESP32 alábbi kivezetéseit fogjuk felhasználni:

- GPIO 2 a Relay vezérlésére
- GPIO 4 a mozgásérzékelő illetve a nyomógomb állapotának érzékelésére
- GND földelés
- 3V3 3,3V a vezérléshez
- VIN 5V a Relay és a mozgásérzékelő működtetéséhez

A kapcsolási rajz:



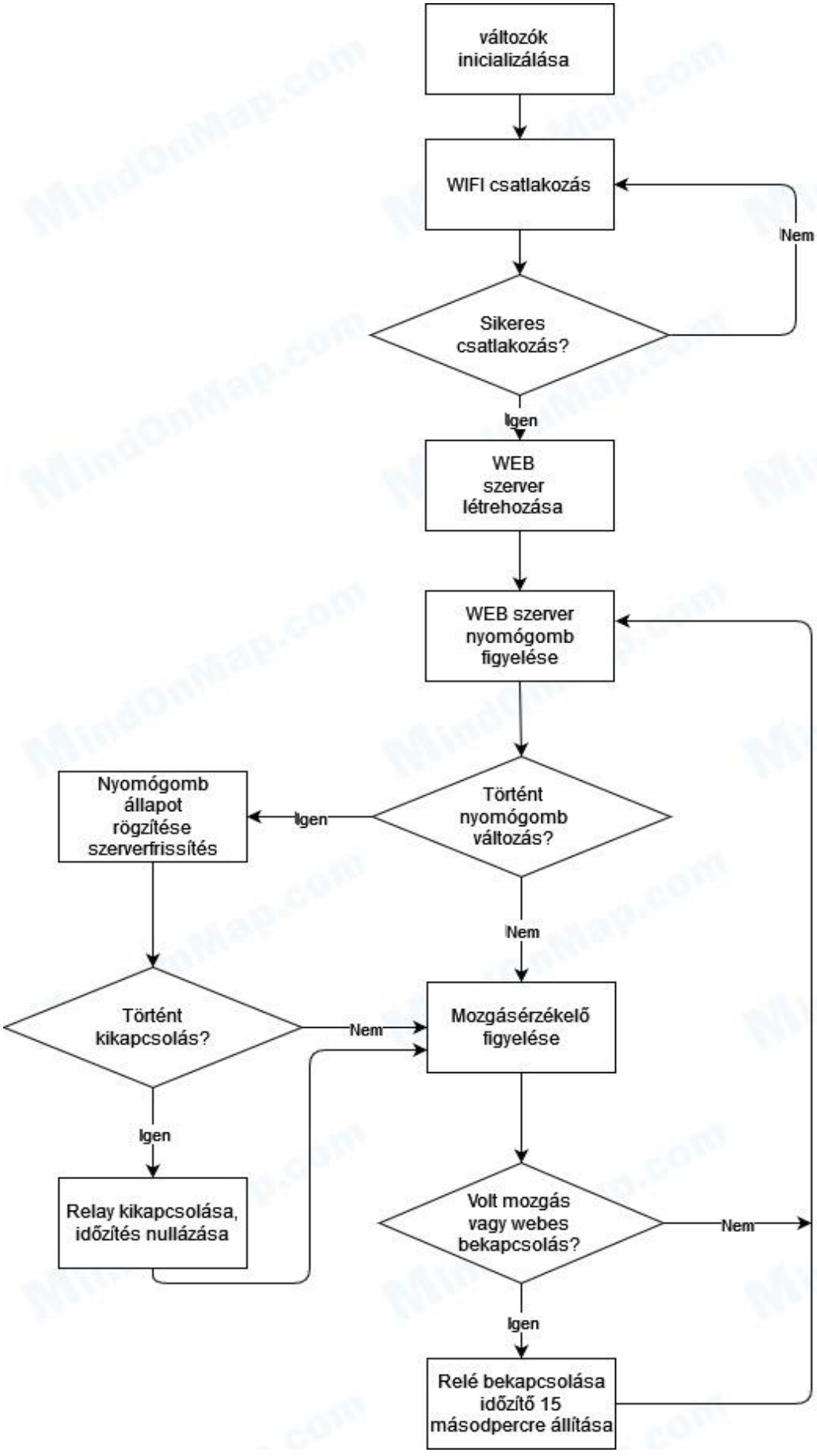
A szerelőpanelen összeállított vezérlés:



A világítást jelen helyzetbe egy LED-el szimuláltam, így nem kellett a nagyfeszültég miatt különleges óvintézkedéseket tenni, valamint a riasztórendszert egy mozgásérzékelő és nyomógomb kombinációja helyettesíti.
Akár a mozgásérzékelőjelére, akár a nyomógomb megnyomására a relay 15 másodpercre bekapcsolja a LED-et. Amennyiben ezen idő alatt újabb mozgás történik, vagy megnyomják a nyomógombot, a 15 másodperces bekapcsolási idő újraindul.
Ezt kiegészítettem egy WEB szerverrel, ahol a LED-et (világítást) saját hálózaton belül bármely a hálózaton lévő eszköz böngészőjéből fel illetve le lehet kapcsolni. A felkapcsolásra is érvényes a 15 másodperces bekapcsolási időzítés, a lekapcsolás azonban azonnal végrehajtódik és ez az összes eszközön aktualizálásra kerül.

A vezérlő program C nyelven készült Arduino IDE programmal.

A program folyamatábrája:



A program felépítése és működése:

Az ESP32 station módban kapcsolódik a hálózathoz és fix IP címet rendelünk hozzá, így más eszközökről a fix IP cím alapján könnyen elérhető.
A helyi hálózatra kliensként kapcsolódik, a bejelentkezéshez szükséges személyes adatokat egy Secrets.h nevű állományban tároljuk, mely a libraries/Secrets almappájában lett létrehozva.

```
# include „Secrets.h”
```

A WEBszerver létrehozásához az ESPAsyncWebServer és az AsyncTCP könyvtárakat használjuk, mivel kényelmes a használatuk és egyidejűleg több klienst is ki tudnak szolgálni. Hátránya, hogy ezeket a könyvtárakat először telepíteni kell.

```
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
```

Létrehozzuk a változókat a mozgásérzékelő és relay állapotának figyeléséhez/rögzítéséhez, a használt GPIO portok számára, valamint beállítjuk a relé kikapcsolási idejét. Ez jelenleg 15 másodperc.

```
const char* PARAM_INPUT_1 = "state";
const int relayPin = 5; // relé a GPIO5 lábra van kötve
const int buttonPin = 4; // riasztó mozgásérzékelő kimenet a GPIO4 lábra kötve
int relay1Status = LOW;          // relé aktuális állapota
int buttonState;             // mozgásérzékelő aktuális állapota
int lastButtonState = LOW;   // mozgásérzékelő utolsó állapota
unsigned long lastDebounceTime = 0;  // az utolsó idő amikor a relay változott
unsigned long debounceDelay = 50;     // várakozási idő, hogy kiszűrjük a zavarokat,
      ha a mozgásérzékelő túl sűrűn változna
// relay bekapcsolásának időzítése
unsigned long relayStartTime = 0;  // A relay utolsó bekapcsolásának ideje
const unsigned long relayOnTime = 15000; // mozgásérzékeléstől számított
      bekapcsolva tartási idő (15 másodperc)
```

Kapcsolódunk a WIFI hálózathoz. A soros porton keresztül tudjuk figyelni a program futását, ezt érdemes rögtön az elején beállítani.

```
WiFi.begin(WIFI_SSID, WIFI_PASS);
delay(1000); // egy kis várakozás, hugy a soros kapcsolat biztos felépüljön az első
kiírás előtt
Serial.print("Connecting to ");
Serial.println(WIFI_SSID);
while (WiFi.status() != WL_CONNECTED) { // addig vár amíg nem csatlakozik
   delay(1000);
   Serial.print(".");
 }
 Serial.println();
 Serial.print("Connected! IP address: ");
 Serial.println(WiFi.localIP()); // kiírja a csatlakozási IP címet
```

Mivel azt szeretnénk, hogy a WeB szerveren folyamatosan aktualizálva legyen a kapcsológomb állapota a relé állapotának megfelelően, ezért azt nem építjük be a HTML kódba, hanem létrehozunk egy placeholder-t %relayPlaceholder% néven.

```
<h2>ESP Home Web Server</h2>
<h3>Folyoso vilagitas</2>
%relayPlaceholder%
```

Ezt a processor() fügvénnyel tudjuk a HTML kódban mindig az aktuális állapotban behelyettesíteni.

```
String processor(const String& var){
if(var == "relayPlaceholder"){
   String buttons ="";
   String outputStateValue = outputState();
   buttons+= "<h4>Relay - GPIO 5 - State <span id=\"outputState\"></span></h4><label class=\"switch\"><input type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"relayPin\" " + outputStateValue + "><span class=\"slider\"></span></label>";
   return buttons;
```

A setInterval függvénnyel másodpercenként lekérdezzük a Webcím állapotát /State, hogy a WEBszervert állandóan frissíteni/pillanatkészen tudjukk tartani.

```
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var inputChecked;
      var outputStateM;
      if( this.responseText == 1){
        inputChecked = true;
        outputStateM = "On";
      }
      else {
        inputChecked = false;
        outputStateM = "Off";
      }
      document.getElementById("relayPin").checked = inputChecked;
      document.getElementById("outputState").innerHTML = outputStateM;
    }
  };
  xhttp.open("GET", "/state", true);
  xhttp.send();
}, 1000 ) ;
```

A következő rész ellenőrzi, hogy érkezett-e kérés valamelyik WEB címről és ha igen akkor annak megfelelően állítja relé státuszát relay1State

```
server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {
   String inputMessage;
   // GET kérés a WEB kapcsoló értékére
   if (request->hasParam(PARAM_INPUT_1)) {
     inputMessage = request->getParam(PARAM_INPUT_1)->value();
     digitalWrite(relayPin, inputMessage.toInt());
     relay1Status = !relay1Status;
     if (inputMessage.toInt() == HIGH) { //ha a WEB-n bekapcsolom, akkor indítom a bekapcsolva tartási időt
       relayStartTime = millis();
     }
   }
   else {
     inputMessage = "No message sent";
   }
   Serial.println(inputMessage);
   request->send(200, "text/plain", "OK");
 });
```

Ha érkezett kérés valamelyik WEBcímről /state akkor annak megfelelően elküldjük az aktuális állapotot.

```
 server.on("/state", HTTP_GET, [] (AsyncWebServerRequest *request) {
   request->send(200, "text/plain", String(digitalRead(relayPin)).c_str());
 });
```

A loop részben beolvassuk a mozgásérzékelő/nyomógomb állapotát és figyelemmel a relay1State változóra be vagy kikapcsoljuk a relét és ezzel a világítást is. Itt vesszük figyelembe azt is, hogy a bekapcsolt állapotú relénél érkezett-e újabb mozgásérzékelés a mozgásérzékelőtől és ha igen, ennek megfelelően újraindítjuk az időzítést.
Az időzítésre nem a delay() funkciót használtuk, mivel az felfüggesztené a program működését, hanem a millis() funkcióval mérjük az eltelt időt.

```
void loop() {
  int reading = digitalRead(buttonPin); // mozgásérzékelő állapotának beolvasása
  // ellenőrizzük, hogy a mozgásérzékelő változott-e
  if (reading != lastButtonState) {
    lastDebounceTime = millis(); // aktualizáljuk az utolsó változás idejét
  }
// ha a megadott zavarszűrési időt figyelembevéve változott a mozgásérzékelő állapota
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == HIGH) { //csak akkor állítjuk a relay állapotot ha a mozgásérzékelő állapota HIGH
        relay1Status = !relay1Status;
        digitalWrite(relayPin, HIGH);
        relayStartTime = millis();
      }
    }
      // relay kikapcsolása, ha elértük a bekapcsolva tartási relayOnTime időt
    if ((millis() - relayStartTime) >= relayOnTime) {
      relay1Status = false; // ez
      digitalWrite(relayPin, LOW);
    }
  }
   // aktualizáljuk az utolsó relay állapotot
  lastButtonState = reading;
}
```

Természetesen az ESP32 lehetőséget ad további mozgásérzékelők figyelésére, illetve a jelenlegi program továbbfejleszthető további funkciókkal, mint például fényérzékelés, hogy ne kapcsoljon fel a villany adott világosság felett, vagy korlátozható időintervallumra is lámpa felkapcsolása. Ez lehet időhöz kötött vagy naplemente napkelte közötti időpont.