Siltumnīca // Pilnā Atskaita

Pilnais kods, atskaite un vairāki pielikumi ir arī pieejami GitHub:  
<https://github.com/Gustavs-K/1.k-1.s-Siltumnica-GK>

Pilnais kods satur komentārus, kas paskaidros koda darbību daudz dziļākā līmenī. Visi komentāri ir angļu valodā.

# Īss aprakts

Projekts iekļauj ESP32, DHT22, servo motoru, vairākas elektronikas detaļas, paštaisītu 4 kanālu releju moduļa atdarinājumu ar PCB dizainu, vairākas 3D printētas detaļas, web interfeisu, caur kuru lietotājs var kontrolēt iekārtas darbību, automatizētu LED apgaismojuma, gaisa sildītāja, gaisa ventilatora, ūdens laistīšanas sistēmu.

Projekta daļas, kas ir “work in progress”:

4 kanāli releju modulis – optocoupleru un paštaisītā PCB pieejamības dēļ,

3D printētās detaļas – ilgs printēšanās laiks.

# Galvenās funkcijas

Mikrokontrolleris – ESP32\_Devtikc\_V4

“Web interface”

Nolasīt gaisa temperatūru

Nolasīt gaisa mitrumu

Sildīt gaisu, ja pārāk auksts

Ventilēt gaisu, ja pārāk karsts vai mitrs

Laistīt augus, pēc maināma grafika un daudzuma

Ieslēgt / izslēgt LED, pēc maināma grafika

# Saturs

[Īss aprakts 1](#_Toc157046898)

[Galvenās funkcijas 1](#_Toc157046899)

[Saturs 2](#_Toc157046900)

[Elektronika 4](#_Toc157046901)

[Ūdens kontrole – Servo motors 4](#_Toc157046902)

[Gaisa temperatūras un mitruma mērīšanas – DHT22 modulis 5](#_Toc157046903)

[Sildītāja, ventilatora un LED kontrole – Releju modulis 5](#_Toc157046904)

[Cenas aprēķins un cita informācija: 7](#_Toc157046905)

[Web interfeisa un iekārtas vizuālais izskats 8](#_Toc157046906)

[Atkārtoti lietoti jēdzieni un informācija 11](#_Toc157046907)

[Setup() funkcija 12](#_Toc157046908)

[Loop() funkcija 13](#_Toc157046909)

[Kods: 13](#_Toc157046910)

[Paskaidrojums: 13](#_Toc157046911)

[“Web interface” 14](#_Toc157046912)

[1. Setup – SPIFFS failu sagatavošana un informācijas dublēšana. 14](#_Toc157046913)

[2. Setup – WiFi un lietotāja savienošanās ar interneta lapu. 14](#_Toc157046914)

[3. Setup / Loop – Lietotāju savienošanās vadība. 15](#_Toc157046915)

[4. Loop – SPIFFS faili tiek pārrakstīti vairākās loop funkcijās. 15](#_Toc157046916)

[Nolasīt gaisa temperatūru 16](#_Toc157046917)

[Kods: 16](#_Toc157046918)

[Paskaidrojums – readTemp() 16](#_Toc157046919)

[Nolasīt gaisa mitrumu 17](#_Toc157046920)

[Kods: 17](#_Toc157046921)

[Paskaidrojums – readHumid() 17](#_Toc157046922)

[Sildīt gaisu, ja pārāk auksts 18](#_Toc157046923)

[Kods: 18](#_Toc157046924)

[1. Gaisa sildīšanas pārbaudes funkcija – tempHandling() 19](#_Toc157046925)

[2. Gaisa sildītāja ieslēgšanas funkcija – heaterHandling(int heatStateCall) 19](#_Toc157046926)

[Ventilēt gaisu, ja pārāk karsts vai mitrs 20](#_Toc157046927)

[Kods: 20](#_Toc157046928)

[1. Gaisa ventilācijas pārbaudes funkcija – humidHandling() 21](#_Toc157046929)

[2. Gaisa ventilācijas ieslēgšanas funkcija – fanHandling(int fanStateCall) 21](#_Toc157046930)

[Ieslēgt / izslēgt LED, pēc maināma grafika 22](#_Toc157046931)

[Kods: 22](#_Toc157046932)

[1. LED galvenais vadītājs – ledHandling() 22](#_Toc157046933)

[2. LED taimeri – startLedTimeTillON() ; checkLedTimeTillON() / ...OFF() 23](#_Toc157046934)

[Laistīt augus, pēc maināma grafika un daudzuma 24](#_Toc157046935)

[Kods: 24](#_Toc157046936)

[1. Ūdens galvenais vadītājs – waterHandling() 24](#_Toc157046937)

[2. Ūdens automātikas kods 25](#_Toc157046938)

[softDelay() funkcija 26](#_Toc157046939)

[Kods: 26](#_Toc157046940)

[Paskaidrojums: 26](#_Toc157046941)

[Pilnais kods 27](#_Toc157046942)

# Elektronika

## Ūdens kontrole – Servo motors

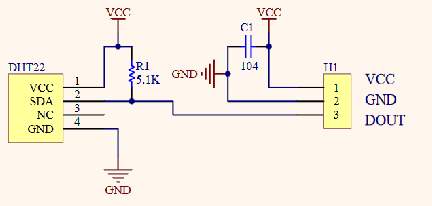
Servo motors ir TD-6620MG, pielieto 5 voltus, no 5V enerģijas padeves.

DATA vads ir pievienots ESP32 pinam 16.

## Gaisa temperatūras un mitruma mērīšanas – DHT22 modulis

DHT22 ir ietverts modulī, kas satur 5.1kΩ rezistoru un nepolarizētu kondensatoru.

Moduļa shēma:



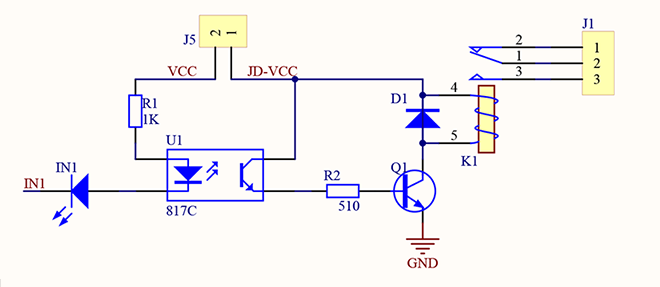
VCC ir 3.3V no ESP32, GRD ir pievienots ESP32 GRD, DATA (vidējais vads) ir pievienots ESP32 pinam 23.

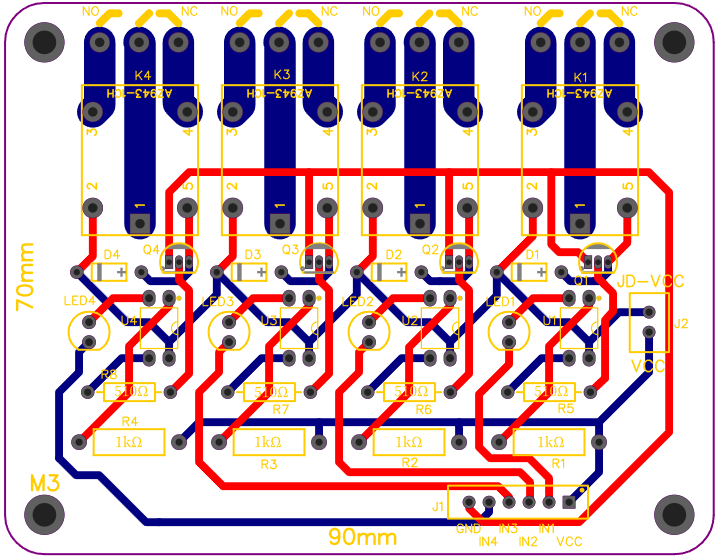
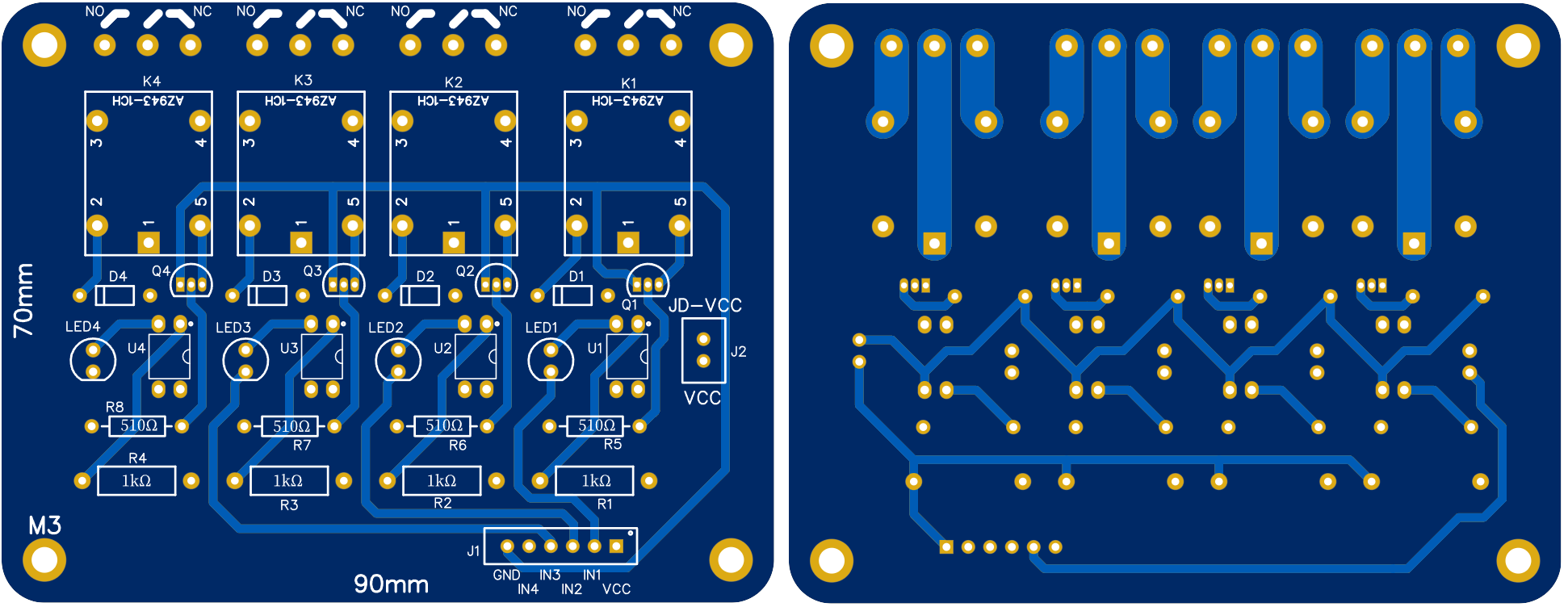
## Sildītāja, ventilatora un LED kontrole – Releju modulis

Sildītājs, ventilators un LED kontrole ir bināra (ON vai OFF), jo tiek pielietoti releji.

Releja modulis sastāv no 1kΩ un 510Ω rezistoriem, optocouplera, NPN tranzistora, diodes, sarkanas LED diodes un releja.

Moduļa shēma (atkārtojas 4 reizes):



Mans 4 kanālu releja moduļa PCB atdarinājums (2 slāņu PCB / THT detaļas):

Šajā modulī iekļautie optocoupleri, atļauj releju kontroles un mikrokontrollera spriegumiem būt atšķirīgiem. Šai gadījumā ESP32 3.3V, kas darbina optocoupleri un sarkano LED diodi, kas norāda uz releja darbību, un 5V 3A PSU, kas darbina releju, kad optocouplera fototranzistors tiek iedarbināts.

## Cenas aprēķins un cita informācija:

ESP32-DEVKITC-S1 – 17.13 EUR

DHT22 modulis – 9.44 EUR

Servo motors 6620MG – 16 EUR (LOCAL) – 9 EUR (CHINA)

Akvārijs (Kaste) – ≈ 30 EUR

LED apgaismojums 12V 9.6Wm IP65   
– ≈ 11.03 EUR (1m)

12V 150W PSU – 19.05 EUR

5V 3A PSU – 6.83 EUR

Breadboard – 4.83 EUR

3D nepieciešamais printera materiāls (SLICER) – ≈ 605g (11.37 EUR)

3D printera laiks (SLICER)   
– ≈ 104 h; 7 min

Kopējā cena – ≈ 163.98 EUR

Kodēšanas stundas – 35 h

Kopējais koda rindu skaits (Ieskaitot komentārus un spraugas) – 1030

Kopējais simbolu skaits kodā (Ieskaitot komentārus un spraugas)   
– 35’150

Kopējais funkciju skaits kodā – 25

4 kan. 5V Releja modulis – Kopā   
– 10.60 EUR (Prieks 1)

Rezistors 510Ω 0.25W – 0.01 x4

Rezistors 1k0Ω 1W – 0.05 x 4

Optocoupler – 0.15 x 4

Diode – 0.04 x 4

Sarkans LED – 0.06 x 4

Relejs – 1.04 x 4

Tranzistors – 0.05 x 4

PCB termināla bloks – 0.70 x 4

PCB pin headers– 0.10

PCB – 2.10

Infrasarkanais spuldze 250W   
– ≈ 10 EUR

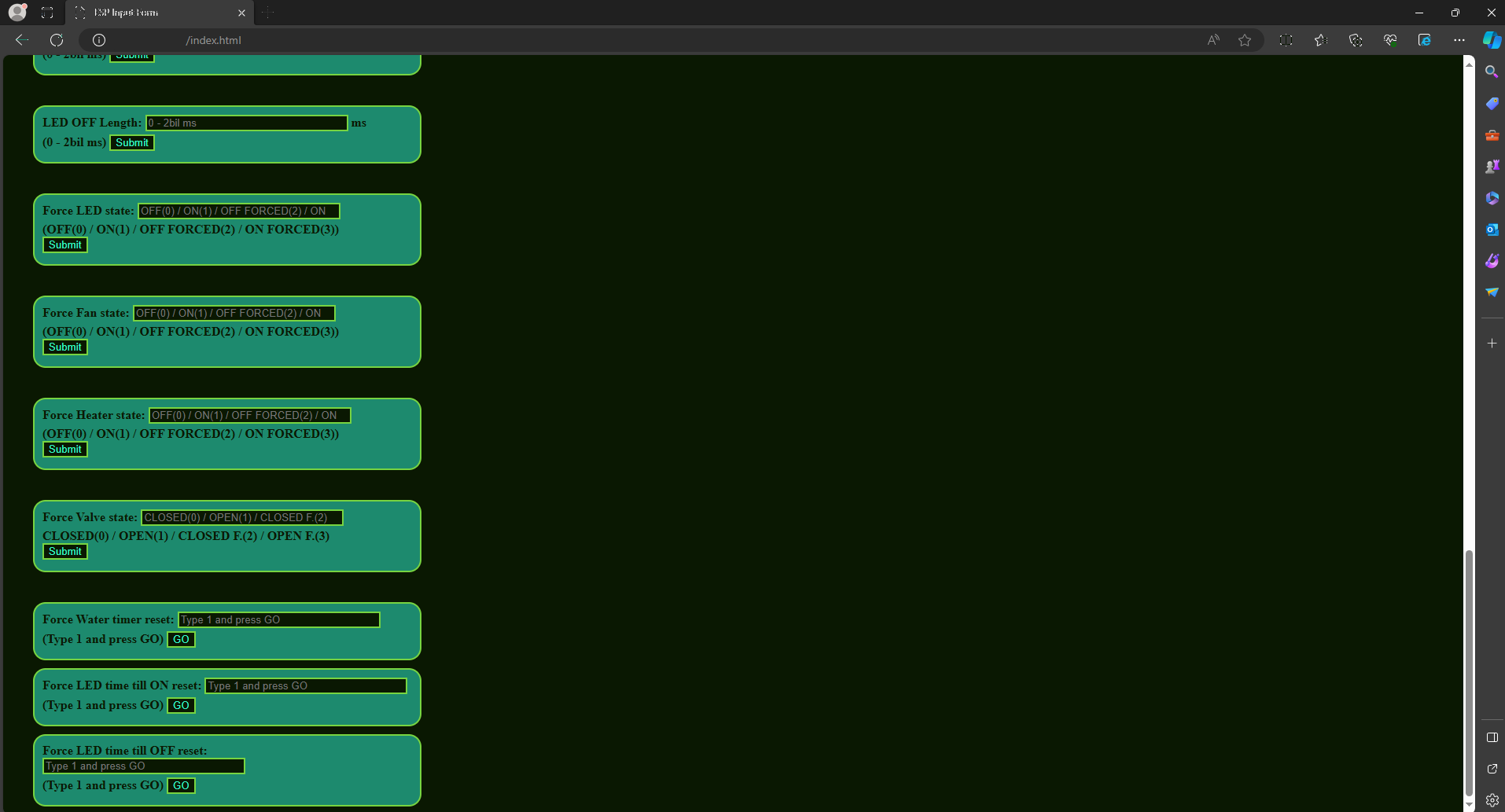
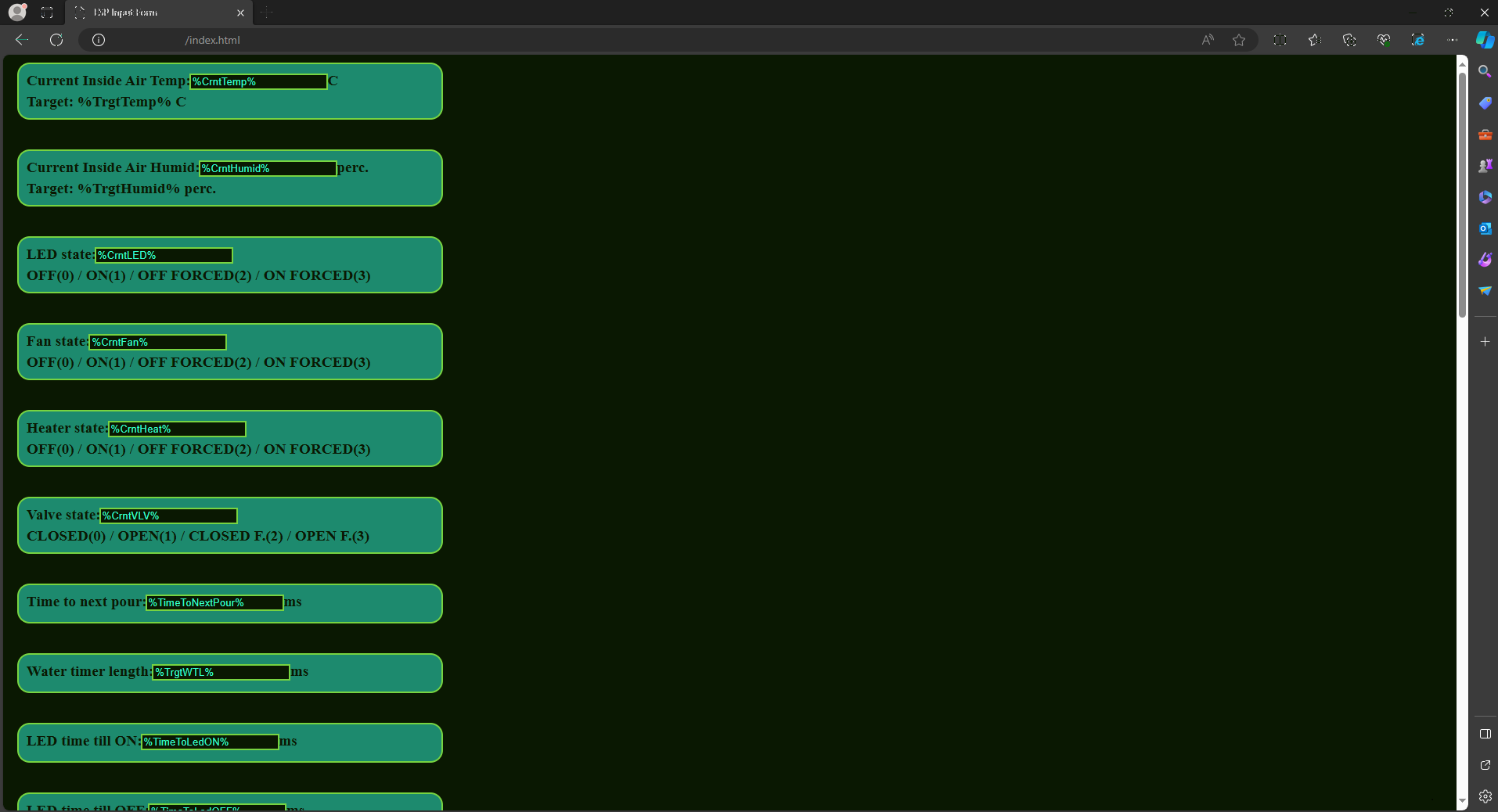
PC fēns (220VAC servera fēns)  
 – ≈ 15 EUR

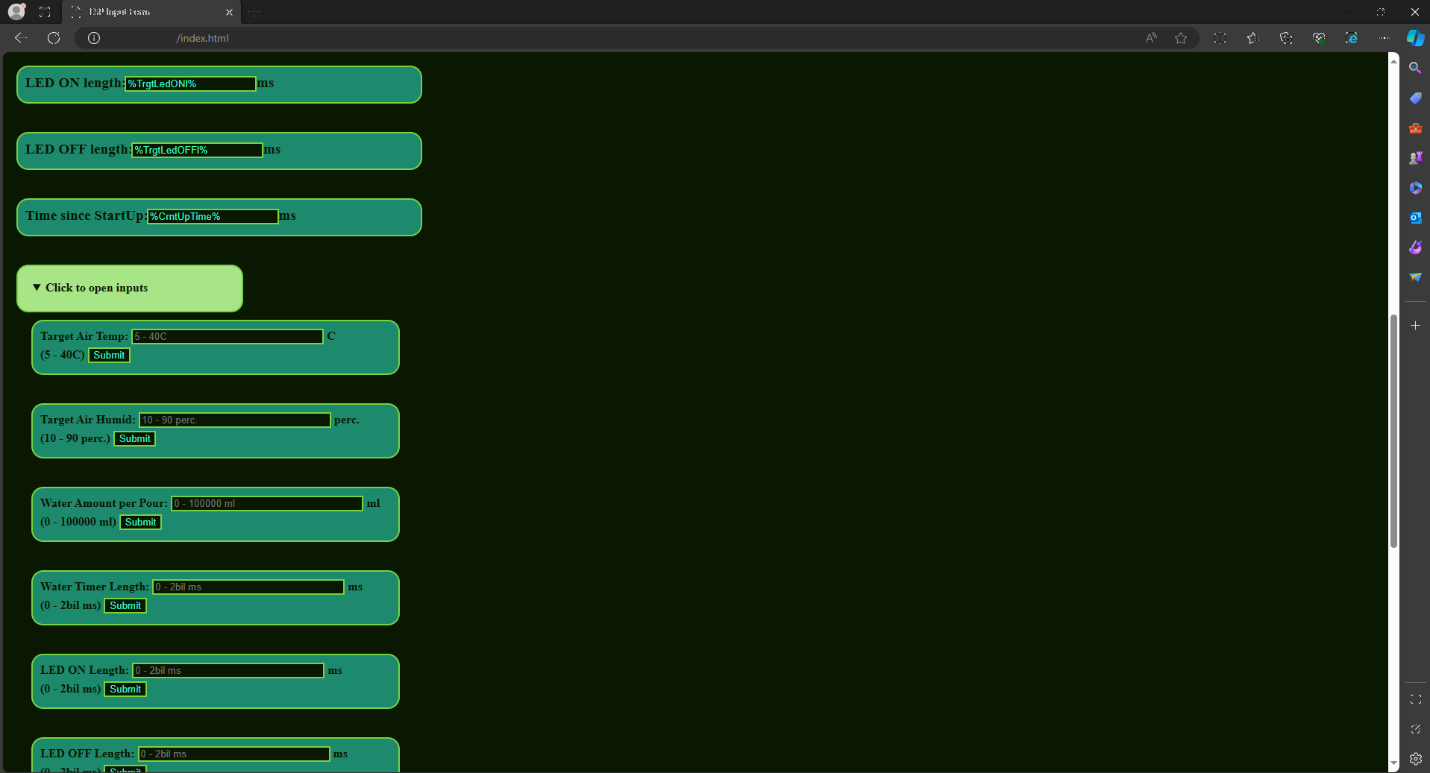
Ūdens vārsts – ≈ 5 EUR

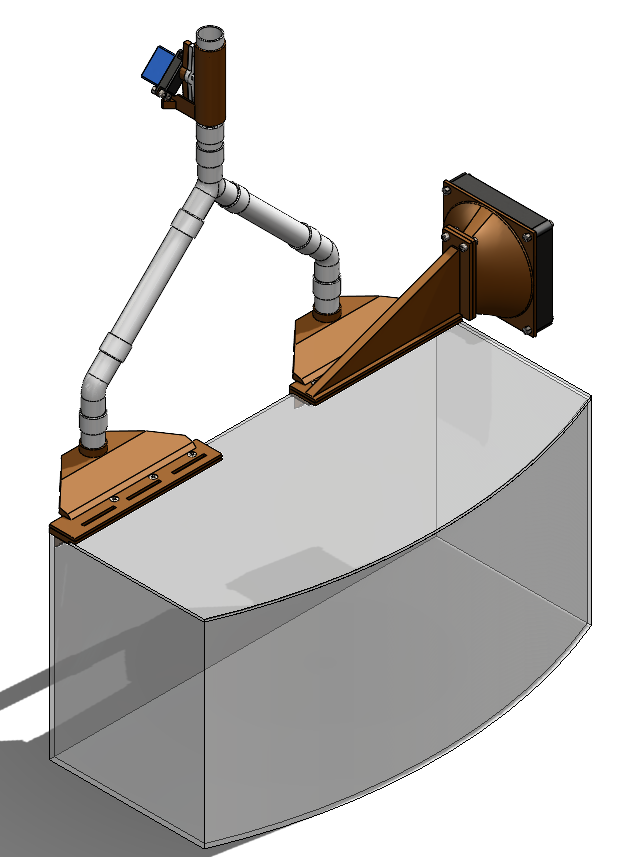
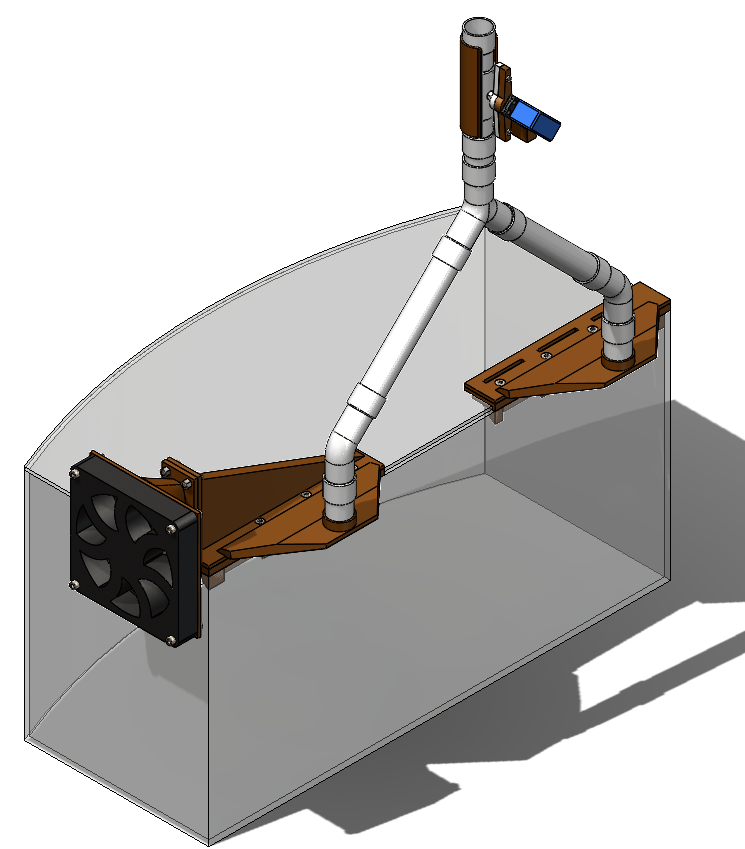
PVC trubas – ≈ 10 EUR

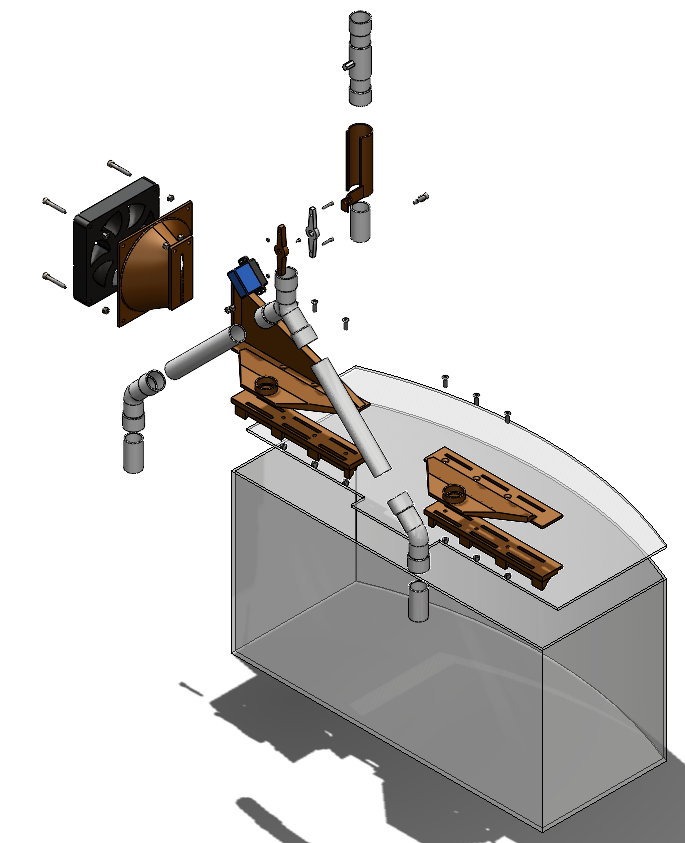
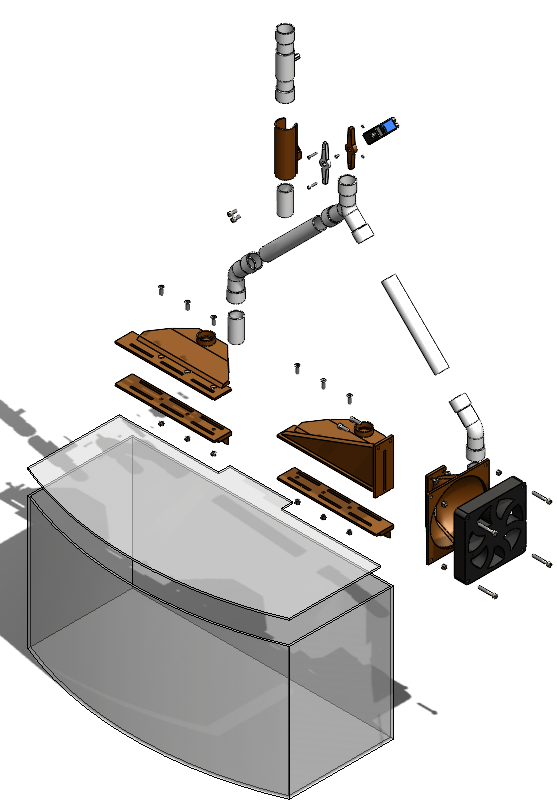
# Web interfeisa un iekārtas vizuālais izskats

Web interfeisa simulācija izmantojot VS Code “Live Server” paplašinājumu.

Visas vērtības, kas ir ietvertas procentu zīmēs ( %EXAMPLE% ), tiek ESP32 loop darbības laikā aizstātas ar attiecīgajās vērtībām.



Vienkāršots fiziskās iekārtas SolidWorks 3D modelis, kas satur 3D printētās detaļas:



# Atkārtoti lietoti jēdzieni un informācija

1. Visi mainīgie, kas attēlo kādas iekārtas vai procesa stāvokli, piemēram, vai sildītājas ir ieslēgts vai izslēgt, ir int tipa mainīgie un ir ar vienu no 4 vērtībām:

0 (Automātiski OFF); 1 (automātiski ON); 2 (piespiedu OFF); 3 (piespiedu ON)

Piespiedu ir stāvoklis, ko lietotājs iestata un ir nemainīgs.

Automātisks ir stāvoklis, kurš mainās atkarībā no sensoru no lasītā, taimeriem, u.t.t.

2. Pārsvarā tiek lietotas 2 galveni informācijas glabāšanas veidi:

Globālie mainīgie – Atrodas operatīvajā atmiņā (RAM), un tiek pielietoti ESP32 koda vadībā. Pazūd, kas sistēma tiek pārstartēta.   
To nosaukumi sākas ar GV. Piemēram, GVCrntTemp.

SPIFFS faili – Atrodas zibatmiņā (Flash memory), un tajos tiek dublēta globālo mainīgo informācija. Šis solis nodrošina saglabātās informācijas atgūšanu sistēmas restartēšanas gadījumā un ir nepieciešams, lai parādītu informāciju interneta lapā. Tie ir .txt faili. Piemēram, /CrntTemp.txt

# Setup() funkcija

Setup funkcijā tiek:

1. uzsāks serial monitors ar 115200 baud tempu,

2. uzsākti LED, sildītāja un ventilatora pini, izmantojot koda sākumā definēto informāciju,

3. uzsākts servo un DHT22,

4. uzsākta SPIFFS failu sistēma,

5. serial monitorā tiek izvadīti visu eksistējošo SPIFFS failu nosaukumi,

6. SPIFFS failu vērtības tiek ierakstītas globālajos mainīgajos,

7. izveidots WiFi savienojums, izmantojot koda sākumā definēto informāciju,

8. ietvertas interneta interfeisa pieprasījumu funkcijas  
(Skatīt “[Web interface // 3.](#_3._Setup_/)”)

9. izvadīts cik ms aizņēma setup funkcijas izpilde.

# Loop() funkcija

## Kods:

// Loop function is executed continuously

void loop() {

  // Denotes the start of a new loop in the serial monitor and starts a timer

  Serial.println("\n// New loop //");

  unsigned long loopMillis = millis();

  // Calls the following functions:

  readTemp();

  readHumid();

  tempHandling();

  humidHandling();

  ledHangling();

  waterHandling();

  // Denotes the end of a loop in the serial monitor and prints how long this loop took, to execute

  Serial.print("// Loop done in ");

  loopMillis = millis() - loopMillis;

  Serial.print(loopMillis);

  Serial.println("ms //\n");

}

## Paskaidrojums:

Šis kods tiek cikliski atkārtots.

Loop cikla sākumā tiek pierakstīts millis laiks, cikos sākas pašreizējais cikls.

Loop cikla beigās, tiek serial monitorā tiek izrakstīts, cik ms aizņēma šis loop cikls.

Šis laika kods **nav** noturīgs pret millis pārplūšanu, kas nozīmē, ka katras ≈ 49.7 dienas, viens cikls tiks izrakstīts ar nepareizu laiku.

# “Web interface”

Lai savienotos ar internetu, tiek izmantota “WiFi.h” bibliotēka.

Interneta interfeiss izmanto “ESPAsyncWebServer” un “AsyncTCP” bibliotēkas.

Saziņai starp interneta mājas lapu un ESP32 kodu, tiek pielietota “SPIFFS.h” bibliotēka un failu sistēma.

Interneta interfeiss un ar to asociētās detaļas aizņem lielāko daļu no koda un tā funkcijas tiek izsauktas ļoti daudzās vietās.

Darbības process:

## 1. Setup – SPIFFS failu sagatavošana un informācijas dublēšana.

SPIFFS failu sistēma tiek uzsākta, ja tas nesanāk serial monitorā parādās brīdinājums.

Serial monitorā tiek izvadīti visi pašlaik eksistējošo SPIFFS failu nosaukumi.

SPIFFS faili tiek nolasīti, un to vērtības tiek piesaistītas ESP32 kodā esošiem globāliem manīgajiem. Šis process nodrošina SPIFFS failos saglabātās informācijas pieejamību, pēc sistēmas pārstartēšanas.   
(SPIFFS fails) /EXAMPLE.txt --> int GVEXAMPLE (globāls mainīgais)

## 2. Setup – WiFi un lietotāja savienošanās ar interneta lapu.

Savienojas ar WiFi tīklu izmantojot kodā ievadītos datus (WiFi nosaukumu un paroli). Ja neizdodas savienoties ar internetu, serial monitorā parādās brīdinājums, un pārējais darba process turpinās bez interneta lapas darbības. Ja izdodas savienoties ar WiFi, serial monitorā parādās ESP32 lokālā IP adrese, kura ir jāievada interneta pārlūkā, lai piekļūtu pie interneta lapas. Gan ESP32, gan lietotāja ierīcei (datoram, telefonam, …) ir jābūt tajā pašā WiFi tīklā.

## 3. Setup / Loop – Lietotāju savienošanās vadība.

Setup funkcija satur divas ESPAsyncWebServer funkcijas:

server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request){…

});

un

server.on("/get", HTTP\_GET, [] (AsyncWebServerRequest \*request) {…

});

Kaut arī šīs funkcijas atrodas setup funkcijā, tās tiek izsauktas katru reizi kāds lietotājs savienojas ar atbilstošo lapu.

Pirmā “/” funkcija nosūta ESP32 kodā ietverto HTML kodu uz lietotāja pārlūk programmu, un atjauno noteiktus SPIFFS failus ar tiem atbilstošajiem globālajiem mainīgajiem.

Kad lietotājs ievada un iesniedz kādu vērtību viņš tiek novirzīts uz “EXAMPLE\_IP/get?INPUT\_NAME=INPUT\_MESSAGE” lapu, kas satur ievadīto informāciju, kura tiek ierakstīta atbilstošajā globālajā mainīgajā (ESP32 loop funkciju izmantošanai) un SPIFFS failā (interneta lapā redzamā informācija un informācijas dublēšana).

## 4. Loop – SPIFFS faili tiek pārrakstīti vairākās loop funkcijās.

Loop funkcijas, kas maina kādu globālo mainīgo, arī maina SPIFFS failu informāciju, piemēram, “readTemp” funkcija, kas nolasa temperatūru ar DHT22 un pieraksta šo vērtību “GVCrntTemp” mainīgajā un “/CrntTemp.txt” failā.

# Nolasīt gaisa temperatūru

## Kods:

// Reads temperature with DHT22 and, if successful, updates current temperatures global variable and SPIFFS file

void readTemp(){

  float t = dht.readTemperature();

  if(isnan(t)){

    return;

  }

  else{

    GVCrntTemp = t;

    inputForUpload = GVCrntTemp;

    writeFile(SPIFFS, "/CrntTemp.txt", inputForUpload.c\_str());

    Serial.print("Temperature set to:" );

    Serial.println(t);

  }

}

## Paskaidrojums – readTemp()

Temperatūra nolasīšanas funkcija tiek izsaukta pirmā katrā loop ciklā.

Gaisa temperatūra tiek nolasīta ar DHT22 un “DHT.h” bibliotēku.

Temperatūra tiek nolasīta ar dht.reatTemperature() funkciju, un nolasītā vērtība tiek piesaistīta float mainīgajam “ t ”.

If(isnan(t)){…} pārbauda vai “ t ” nesatur vērtību. Ja “ t ” ir tukšs, temperatūras nolasīšanas funkcija tiek pārtraukta, bez rezultātiem. (Nolasīšana neizdevās)

Ja “ t ” nav tukšs, ar “ t ” vērtību pārrakstīts GVCrntTemp, kas ir globālais mainīgais pašreizējai temperatūrai, /CrntTemp.txt, kas ir SPIFFS fails un “ t ” tiek izvadīts serial monitorā.

# Nolasīt gaisa mitrumu

## Kods:

// Reads humidity with DHT22 and, if successful, updates current humidities global variable and SPIFFS file

void readHumid(){

  float h = dht.readHumidity();

  if(isnan(h)){

    return;

  }

  else{

    GVCrntHumid = h;

    inputForUpload = GVCrntHumid;

    writeFile(SPIFFS, "/CrntHumid.txt", inputForUpload.c\_str());

    Serial.print("Humidity set to:" );

    Serial.println(h);

  }

}

## Paskaidrojums – readHumid()

Loop ciklā gaisa mitruma nolasīšanas funkcija ir 2.

Gaisa mitruma nolasīšana ir gandrīz identiska temperatūras nolasīšanai.   
(Skatīt ”[Nolasīt gaisa temperatūru](#_Nolasīt_gaisa_temperatūru)”)

# Sildīt gaisu, ja pārāk auksts

## Kods:

// Compares if current temperature global variable is too high, too low or close enough to target temperature global variable

// Controls heater accordingly

void tempHandling(){

  Serial.println("Handling temp");

  if(GVCrntTemp > GVTrgtTemp - 1 && GVCrntTemp < GVTrgtTemp + 1){

    heaterHandling(0);

  }

  else if (GVCrntTemp < GVTrgtTemp - 2){

    heaterHandling(1);

  }

  else if (GVCrntTemp > GVTrgtTemp + 2){

    heaterHandling(0);

  }

}

// Starts or stops heater, depending on the global heater state variable and functions called variable

// Almost identical to fanHandling

void heaterHandling(int heatStateCall){

  Serial.print(GVCrntHeat);

  Serial.print(" <- Global var | Heater handling | Call var -> ");

  Serial.println(heatStateCall);

  if(GVCrntHeat == 3){

    digitalWrite(PIN\_HEATER, HIGH);

  }

  else if(GVCrntHeat == 2){

    digitalWrite(PIN\_HEATER, LOW);

  }

  else{

    if(heatStateCall == 1)

    {

      GVCrntHeat = 1;

      digitalWrite(PIN\_HEATER, HIGH);

    }

    else if(heatStateCall == 0){

      GVCrntHeat = 0;

      digitalWrite(PIN\_HEATER, LOW);

    }

    inputForUpload = GVCrntHeat;

    writeFile(SPIFFS, "/CrntHeat.txt", inputForUpload.c\_str());}}

## 1. Gaisa sildīšanas pārbaudes funkcija – tempHandling()

Loop ciklā gaisa sildīšanas pārbaudes funkcija ir 3.

tempHandling() pārbauda, kāda ir atšķirība starp pašreizējo temperatūru un mērķa temperatūru.   
Ja pašreizējā temperatūra ir mazāka par mērķa temperatūru - 2°C, tiek ieslēgts sildītājs.

## 2. Gaisa sildītāja ieslēgšanas funkcija – heaterHandling(int heatStateCall)

heaterHandling( int heatStateCall ) tiek izsaukts ar int vērtību 0 vai 1.

Atkarībā no globālā mainīgā GVCrntHeat, kas nosaka vai sildītājs ir automātiskā režīmā (vērtības 0 vai 1), vai piespiedu režīmā (vērtības 2 vai 3).

Ja ir piespiedu režīmā, funkcijas izsauktā vērtība (int heatStateCall) netiek ņemta vērā. Globālā mainīgā vērtība 3 nozīmē piespiedu ON režīmu, un 2 nozīmē piespiedu OFF.

Ja globālā mainīgā vērtība neatbilst 2 vai 3, tiek ieslēgs automātiskais režīms, kurā tiek pārbaudīts mainīgais heatStateCall. Vērtība 1 nozīmē ON un 0 nozīmē OFF.

Beigās, ja ir automātiskais režīms, tiek atjaunota sildītāja stāvokļa SPIFFS faila vērtība ar globālā mainīgā palīdzību.

# Ventilēt gaisu, ja pārāk karsts vai mitrs

## Kods:

// Compares if current humidity global variable is too high or current temperature global variable is too high

// Controls fan accordingly

void humidHandling(){

  Serial.println("Handling humidity");

  if(GVCrntTemp > GVTrgtTemp + 2 || GVCrntHumid > GVTrgtHumid + 3){

    fanHandling(1);

  }

  else{

    fanHandling(0);

  }

}

// Starts or stops fan, depending on the global fan state variable and functions called variable

void fanHandling(int fanStateCall){

  Serial.print(GVCrntFan);

  Serial.print(" <- Global var | Fan handling | Call var -> ");

  Serial.println(fanStateCall);

  // if global fan state variable is 3 or 2, fan is forced ON or OFF

  if(GVCrntFan == 3){

    digitalWrite(PIN\_FAN, HIGH);

  }

  else if(GVCrntFan == 2){

    digitalWrite(PIN\_FAN, LOW);

  }

  else{

    // else if function is called with 1 or 0, fan is auto ON or OFF

    if(fanStateCall == 1)

    {

      GVCrntFan = 1;

      digitalWrite(PIN\_FAN, HIGH);

    }

    else if(fanStateCall == 0){

      GVCrntFan = 0;

      digitalWrite(PIN\_FAN, LOW);

    }

    // if fan value was updated automatically, update fan state SPIFFS file with new value

    inputForUpload = GVCrntFan;

    writeFile(SPIFFS, "/CrntFan.txt", inputForUpload.c\_str());}}

## 1. Gaisa ventilācijas pārbaudes funkcija – humidHandling()

Loop ciklā gaisa ventilācijas pārbaudes funkcija ir 4.

humidHandling() pārbauda vai gaisa mitrums pašreizējais gaisa mitrums ir lielāks par mērķa gaisa mitrumu + 3%, **vai** pašreizējā gaisa temperatūra ir lielākā par mērķa gaisa temperatūru + 2°C. Ja kāds no šiem nosacījumiem izpildās, tiek iedarbināta ventilācijas. Ja neviena neizpildās, ventilācija tiek izslēgta.

## 2. Gaisa ventilācijas ieslēgšanas funkcija – fanHandling(int fanStateCall)

Gaisa sildītāja ieslēgšanas funkcija un gaisa ventilācijas ieslēgšanas funkcija ir gandrīz identiskas. (Skatīt “[2. Gaisa sildītāja ieslēgšanas funkcija](#_2._Gaisa_sildītāja)”)

# Ieslēgt / izslēgt LED, pēc maināma grafika

## Kods:

Kods ir pārāk garš. (Skatīt “[Pilnais kods](#_Pilnais_kods)”)

// Start timer, that goes while LED is OFF

void startLedTimeTillON(){...}

// Start timer, that goes while LED is ON

void startLedTimeTillOFF(){...}

// Checks if LED time till ON timer has expired and returns: Expired(true) , In progress(false)

bool checkLedTimeTillON(){...}

// Checks if LED time till OFF timer has expired and returns: Expired(true) , In progress(false)

bool checkLedTimeTillOFF(){...}

// Handles LED lighting

void ledHangling(){...}

## 1. LED galvenais vadītājs – ledHandling()

Loop ciklā LED vadīšanas funkcija ir 5.

LED vadīšanas funkcija ar LED taimeru restartēšanas pārbaudi. Ja kāds no globālajiem mainīgajiem “GVFrcTTOffR” vai ”GVFrcTTOnR”   
(GV Force Time Till On/Off Restart) ir ar vērtību 1, tad tas taimeris tiek restartēts, un attiecīgais mainīgais tiek iestatīts atpakaļ uz 0.

Tad tiek pārbaudīts vai LED ir piespiedu režīmā, un ja ir, tad tiek attiecīgai ieslēgti vai izslēgti LED, un pārējais tiek izlaists.

Ja LED nav piespiedu režīmā, tad tiek pārbaidīts vai LED ir automātiski OFF vai ON **un** vai attiecīgais taimeris ir iztecējis. Ja kāds no šiem diviem gadījumiem izpildās, tad LED tiek nomainīts uz pretējo stāvokli, iztecējušais taimeris tiek iestatīts uz 0 (lai būtu vieglāk saprotams interneta lapā) un tiek ieslēgts attiecīgais taimeris, kas noteiks, kad atkal notiks LED maiņa uz pretējo.

## 2. LED taimeri – startLedTimeTillON() ; checkLedTimeTillON() / ...OFF()

LED automātiskā sistēma izmanto 2 taimerus. Viens nosaka cik ilgi līdz LED izslēgsies un otrs cik ilgi līdz ieslēgsies. Tie tiek uzsākti un pārstartēti ar startLedTimeTillON() un startLedTimeTillOFF() funcijām.

Šie taimeri tiek pārbaudīti ar checkLedTimeTillON() un checkLedTimeTillOFF() funkcijās, kas atgriež bool vērtību true, ja taimeris ir iztecējis, un false, ja taimeris vēl iet. Gan LED, gad ūdens, gan softDelay() taimeri ir sarakstīti tā, lai būtu noturīgi pret millis pārplūdumu, kas notiek katras ≈ 49.7 dienas.

Laistīt augus, pēc maināma grafika un daudzuma

## Kods:

Kods ir pārāk garš. (Skatīt “[Pilnais kods](#_Pilnais_kods)”)

// Starts global water timer, also used to restart the timer

void startWaterTimer(){...}

// Checks if water timer has expired

// Also updates automatic water handling progress

void checkWaterTimer(){...}

// Opens water valve

void startWater(){...}

// Closes water valve

void endWater(){...}

// Forced and automated water and water timer handler

void waterHandling(){...}

## 1. Ūdens galvenais vadītājs – waterHandling()

Loop ciklā ūdens vadības funkcija ir 6 un pēdējā.

Ūdens vadība strādā līdzīgi LED vadībai, tikai ar 1 taimeri un nevis tiek kontrolēts relejs, kas ieslēdz vai izslēdz LED lampas, šeit tiek kontrolēts servo motors, kas atgriež vai aizgriež ūdens vārstu.   
(Skatīt “[Ieslēgt / izslēgt LED, pēc maināma grafika](#_Ieslēgt_/_izslēgt)”)

Sākumā tiek pārbaudīts vai vārsts ir automātiskajā vai piespiedu režīmā.

Tad tiek pārbaudīts vai ūdens taimeris ir jāpārstartē.

Tad tiek palaists vai izlaists automātikas kods.

## 2. Ūdens automātikas kods

Automātikas kods ir sadalīts daļās, kas cikliski atkārtojas. Šīs daļas ir numurētas sākot ar 0.

0. daļa – Ieslēdz ūdens taimeri un nomaina ūdens cikla progresu uz 1. (Tas tiks izpildīts nākošajā loop ciklā)

1. daļa – Tiek pārbaudīts ūdens taimeris.

checkWaterTimer() funkcija pārbauda vai ūdens taimeris ir iztecējis. Ja tas ir iztecējis, ūdens cikla progress tiek nomainīts uz 2, savādāk atlikušais laiks tiek ierakstīts atbilstošajā SPIFFS failā un serial monitorā.

Gadījumā, kad taimeris vēl nav iztecējis, kopējais loop cikla garums var būt ļoti mazs, kas ļoti apgrūtina serial monitora lasīšanu, tāpēc tiek izsaukta 500ms gara softDelay() funkcija. (Skatīt “[softDelay() funkcija](#_softDelay()_funkcija)”)

2. daļa – Aprēķina vārsta ilgumu, atver vārstu, softDelay aprēķināto ilgumu, aizver vārstu.

Tiek aprēķināts cik ilgi ūdens vārstam jābūt atvērtam ar “ delayTime = GVTrgtWPP / (WATER\_FLOWRATE \* 1000); ” funkciju. delayTime ir laiks, cik ilgi jāgaida,   
GVTrgtWPP ir lietotāja iestatītais ūdens daudzums un WATER\_FLOWRATE ir iepriekš iestatīts ūdens plūšanas ātrums (m3/s).

Vārsts tiek atvērts ar startWater() funkciju, kas pakāpeniski pārvieto sevro motora pozīciju pa 1 grādu katras ≈ 15ms.

Tiek izsaukts softDelay() ar aprēķināto laika intervālu.

Vārsts tiek aizvērts ar endWater() funkciju, kas strādā gandrīz identiski startWater() funkcijai.

Gan startWater(), gan endWater() abi aptur programmas trupināšanos, līdz tie pabeidz savu darbu, bet, tādēļ ka tie pielieto vairākus mazus delay intervālus, šīs funkcijas ļauj interneta interfeisam strādāt bez atpurēm. (Līdzīgs princips kā softDelay() funkcijai)

Pēc vārsta aizgriešanas, ūdens cikla daļa tiek iestatīta atpakaļ uz 0.

# softDelay() funkcija

## Kods:

// Delay function, that delays in 10 ms increments, so that Async webserver can check for user requests

void softDelay(unsigned long delayLength){

  Serial.print("Soft delaying loop for ");

  Serial.print(delayLength);

  Serial.print(" ms");

  unsigned long delayStart = millis();

  while(millis() - delayStart <= delayLength){

    Serial.print(".");

    delay(10);

  }

  Serial.println("Resuming");

}

## Paskaidrojums:

softDelay() funkcija tiek pielietota standarta delay funkcijas vietā, jo tā ļauj AsyncWebServerim darboties, kamēr aizkavē laiku.

softDelay( unsigned long delayLength ) tiek izsaukts ar laiku, kurš ir jāaizkavē, līdzigi parastajai delay funkcijai.

Atlikušais aizkavēšanas laiks tiek pārbaudīts katras ≈ 10 ms. Kad beidzas taimeris, funkcija beidzas un turpinās pārējā programma.

# Pilnais kods

Kods ir arī pieejams GitHub:   
<https://github.com/Gustavs-K/1.k-1.s-Siltumnica-Kruzmanis>

#include <Arduino.h>            // General

#include <WiFi.h>               // WIFI and Web server

#include <AsyncTCP.h>           // WIFI and Web server

#include <SPIFFS.h>             // WIFI and Web server

#include <ESPAsyncWebServer.h>  // WIFI and Web server

#include <DHT.h>                // DHT 22

#include <ESP32Servo.h>         // Servo

#include <ESP32PWM.h>           // Servo

//#include <analogWrite.h>      // NOT USED

//#include <ESP32Tone.h>        // NOT USED

AsyncWebServer server(80);

// USER SPECIFIED

// NETWORK CREDENTIALS

const char\* ssid = "REPLACE\_WITH\_WIFI\_NAME";          // BUFALO24

const char\* password = "REPLACE\_WITH\_WIFI\_PASSWORD";  // arnis070668

// VALVE POSITIONS IN DEGREESE

#define VLV\_POS\_OPEN 0

#define VLV\_POS\_CLOSED 180

// WATER FLOW RATE (m^3/s)

#define WATER\_FLOWRATE 0.00544 // Currently with pipe diam - 3cm ; pipe lenght - 0.3m ; drop - 0.5m

// Pins

#define PIN\_DHT22 23

#define PIN\_LED 19

#define PIN\_FAN 18

#define PIN\_HEATER 17

#define PIN\_SERVO 16

// Defines servo

Servo waterServo;

// Defines DHT22

#define DHTTYPE DHT22

// Attaches DHT22

DHT dht(PIN\_DHT22, DHTTYPE);

// Input params for webserver upload process

const char\* PARAM\_TEMP = "TrgtTemp";

const char\* PARAM\_HUMD = "TrgtHumid";

const char\* PARAM\_WPP = "TrgtWPP";

const char\* PARAM\_WTL = "TrgtWTL";

const char\* PARAM\_LED = "CrntLED";

const char\* PARAM\_FAN = "CrntFan";

const char\* PARAM\_HEAT = "CrntHeat";

const char\* PARAM\_VLV = "CrntVLV";

const char\* PARAM\_WTR = "FrcWTR";

const char\* PARAM\_LNL = "TrgtLedONl";

const char\* PARAM\_LFL = "TrgtLedOFFl";

const char\* PARAM\_TTNR = "FrcLedTTOnR";

const char\* PARAM\_TTFR = "FrcLedTTOffR";

// INPUTS FROM WEBPAGE

// TrgtTemp  TrgtHumid  TrgtWPP  TrgtWTL  CrntLED  CrntFan  CrntHeat  CrntVLV(replaced FrcWP)  FrcWTR

// INPUTS FROM SENSORS

// CrntTemp  CrntHumid

// CALCULATED

// TimeToNextPour  CrntUpTime

// Global variables

int GVTrgtTemp;        // Target Temperature

int GVTrgtHumid;       // Target Humidity

int GVTrgtWPP;         // Target Water Per Pour

int GVTrgtWTL;         // Target Water Timer Length

int GVTrgtLedONl;      // How long will LEDs stay on

int GVTrgtLedOFFl;     // How long will LED stay off

int GVCrntLED = 3;     // Current LED state           (0 auto off, 1 auto on, 2 forced off, 3 forced on)

int GVCrntFan;         // Current Fan state           (0 auto off, 1 auto on, 2 forced off, 3 forced on)

int GVCrntHeat;        // Current Heater state        (0 auto off, 1 auto on, 2 forced off, 3 forced on)

int GVCrntVLV;         // Current Valve state         (0 auto closed, 1 auto open, 2 forced closed, 3 forced open)

int GVFrcWTR;          // Force Water Timer Restart

int GVFrcLedTTOnR;     // Force restart LED Time till ON timer

int GVFrcLedTTOffR;    // Force restart LED Time till OFF timer

float GVCrntTemp;      // Current Temperature

float GVCrntHumid;     // Current Humidity

int GVTimeToNextPour;  // Time left until Next water Pour

int GVTimeToLedON;     // Time left until LED turns ON

int GVTimeToLedOFF;    // Time left until LED turns OFF

// NOT SHOWN ON WEBPAGE

unsigned long GVTTNPstart; // Millis at which "GVTimeToNextPour" timer starts

unsigned long GVLedTimeTillONStart; // Millis at which "GVTimeToLedON" timer starts

unsigned long GVLedTimeTillOFFStart; // Millis at which "GVTimeToLedOFF" timer starts

int GVWaterHandlingState = 0; // 0 - Set GVTTNPstart ; 1 - Wait ; 2 - Start water ; 3 - Wait ; 4 - End water and Run 0

int GVifAutoWater = 1; // Determines if water handling is automated or forced (0 - FORCED ; 1 - AUTOMATED)

int pos = 0; // Current servo position in degrees

String inputForUpload; // String variable, global variables write to this variable -> this variable writes to SPIFFS

// HTML web page code, anything between two precent symbols " %EXAMPLE% " is a placeholder, that is later replaced by SPIFFS file values, by the same name " EXAMPLE.txt "

// submitMessage() function is called when user presses any submit button. The function gives a pop-up alert and afterwards refreshes the page

// <style> contains CSS code, for styling the page

// The web server variable updating works like this:

// Placeholders in the HTML code are replaced by SPIFFS values, when the main page is loaded, but the SPIFFS values, that are placed in, have to be updated in the ESP32 code.

// When a user accesses the main page "CURRENT\_IP/" the "server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request)" function is called, that is located in the "setup" function and updates multiple SPIFFS files.

// When any value is inputted and submitted this page is accessed "CURRENT\_IP/get?INPUTNAME=INPUTVALUE", where "get" calls "server.on("/get", HTTP\_GET, [] (AsyncWebServerRequest \*request)", "INPUTNAME" is the name of the input getting submitted (this is also what the input value will get written to), "INPUTVALUE" is the inputted value

// UNUSED FONT

// @import url('https://fonts.googleapis.com/css?family=Courier%20Prime:700|Courier%20Prime:400');

const char index\_html[] PROGMEM = R"rawliteral(

<!DOCTYPE HTML>

<html>

    <head>

        <title>ESP Input Form</title>

        <meta name="viewport" content="width=device-width, initial-scale=1">

        <script>

            function submitMessage() {

                alert("Saved value to ESP SPIFFS");

                setTimeout(function(){ document.location.reload(false); }, 500);

            }

        </script>

        <style>

            body{

                background-color: #0A1802;

                font-family: 'Courier Prime';

                font-weight: 700;

            }

            summary{

                width: 260px;

                height: 20px;

                cursor: pointer;

                background-color: #A7E587;

                color: #0A1802;

                border: 2px solid #77D441;

                margin: 10px;

                margin-left: 10px;

                padding: 20px;

                border-radius: 16px;

            }

            form{

                width: 470px;

                height: auto;

                background-color: #1D8A6E;

                color: #0A1802;

                border: 2px solid #77D441;

                margin: 10px;

                margin-left: 30px;

                padding: 10px;

                border-radius: 16px;

            }

            form.Output-form{

                width: 520px;

                height: auto;

                font-size:larger;

                margin-left: 10px;

            }

            input{

                color: #37ffcd;

                background-color: #0A1802;

                border: 2px solid #77D441;

                margin-bottom: 4px;

            }

            input.Input-box{

                width: 250px;

            }

            input.Output-box{

                width: auto;

                margin-left: -7px;

                margin-right: -5px;

            }

        </style>

    </head>

    <body>

        <!-- OUTPUTS -->

        <form class="Output-form">

            Current Inside Air Temp: <input type="text" disabled value="%CrntTemp%" class="Output-box" name="CrntTemp"> C <br> Target: %TrgtTemp% C

        </form><br>

        <form class="Output-form">

            Current Inside Air Humid: <input type="text" disabled value="%CrntHumid%" class="Output-box" name="CrntHumid"> perc. <br> Target: %TrgtHumid% perc.

        </form><br>

        <form class="Output-form">

            LED state: <input type="text" disabled value="%CrntLED%" class="Output-box" name="CrntLED"> <br> OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)

        </form><br>

        <form class="Output-form">

            Fan state: <input type="text" disabled value="%CrntFan%" class="Output-box" name="CrntFan"> <br> OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)

        </form><br>

        <form class="Output-form">

            Heater state: <input type="text" disabled value="%CrntHeat%" class="Output-box" name="CrntHeat"> <br> OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)

        </form><br>

        <form class="Output-form">

            Valve state: <input type="text" disabled value="%CrntVLV%" class="Output-box" name="CrntVLV"> <br> CLOSED(0) / OPEN(1) / CLOSED F.(2) / OPEN F.(3)

        </form><br>

        <form class="Output-form">

            Time to next pour: <input type="text" disabled value="%TimeToNextPour%" class="Output-box" name="TimeToNextPour"> ms

        </form><br>

        <form class="Output-form">

            Water timer length: <input type="text" disabled value="%TrgtWTL%" class="Output-box" name="CrntWTL"> ms

        </form><br>

        <form class="Output-form">

            LED time till ON: <input type="text" disabled value="%TimeToLedON%" class="Output-box" name="TimeToLedON"> ms

        </form><br>

        <form class="Output-form">

            LED time till OFF: <input type="text" disabled value="%TimeToLedOFF%" class="Output-box" name="TimeToLedOFF"> ms

        </form><br>

        <form class="Output-form">

            LED ON length: <input type="text" disabled value="%TrgtLedONl%" class="Output-box" name="TrgtLedONl"> ms

        </form><br>

        <form class="Output-form">

            LED OFF length: <input type="text" disabled value="%TrgtLedOFFl%" class="Output-box" name="TrgtLedOFFl"> ms

        </form><br>

        <form class="Output-form">

            Time since StartUp: <input type="text" disabled value="%CrntUpTime%" class="Output-box" name="CrntUpTime"> ms <br>

        </form><br>

        <!-- INPUTS -->

        <details>

            <summary class="pointer">Click to open inputs</summary>

            <span>

                <form action="/get" target="hidden-form">

                    Target Air Temp: <input type="number" value="%TrgtTemp%" class="Input-box" placeholder="5 - 40C" min="5" max="40" name="TrgtTemp"> C <br> (5 - 40C)

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    Target Air Humid: <input type="number" value="%TrgtHumid%" class="Input-box" placeholder="10 - 90 perc." min="10" max="90" name="TrgtHumid"> perc. <br> (10 - 90 perc.)

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    Water Amount per Pour: <input type="number" value="%TrgtWPP%" class="Input-box" placeholder="0 - 100000 ml" min="0" max="100000" name="TrgtWPP"> ml <br> (0 - 100000 ml)

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    Water Timer Length: <input type="number" value="%TrgtWTL%" class="Input-box" placeholder="0 - 2bil ms" min="0" max="2000000000" name="TrgtWTL"> ms <br> (0 - 2bil ms)

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    LED ON Length: <input type="number" value="%TrgtLedONl%" class="Input-box" placeholder="0 - 2bil ms" min="0" max="2000000000" name="TrgtLedONl"> ms <br> (0 - 2bil ms)

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    LED OFF Length: <input type="number" value="%TrgtLedOFFl%" class="Input-box" placeholder="0 - 2bil ms" min="0" max="2000000000" name="TrgtLedOFFl"> ms <br> (0 - 2bil ms)

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    Force LED state: <input type="number" value="%CrntLED%" class="Input-box" placeholder="OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)" min="0" max="3" name="CrntLED"> <br> (OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)) <br>

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    Force Fan state: <input type="number" value="%CrntFan%" class="Input-box" placeholder="OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)" min="0" max="3" name="CrntFan"> <br> (OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)) <br>

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    Force Heater state: <input type="number" value="%CrntHeat%" class="Input-box" placeholder="OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)" min="0" max="3" name="CrntHeat"> <br> (OFF(0) / ON(1) / OFF FORCED(2) / ON FORCED(3)) <br>

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    Force Valve state: <input type="number" value="%CrntVLV%" class="Input-box" placeholder="CLOSED(0) / OPEN(1) / CLOSED F.(2) / OPEN F.(3)" min="0" max="3" name="CrntVLV"> <br> CLOSED(0) / OPEN(1) / CLOSED F.(2) / OPEN F.(3) <br>

                    <input type="submit" value="Submit" onclick="submitMessage()">

                </form><br>

                <form action="/get" target="hidden-form">

                    Force Water timer reset: <input type="number" value="%FrcWTR%" class="Input-box" placeholder="Type 1 and press GO" min="0" max="1" name="FrcWTR"> <br> (Type 1 and press GO)

                    <input type="submit" value="GO" onclick="submitMessage()">

                </form>

                <form action="/get" target="hidden-form">

                    Force LED time till ON reset: <input type="number" value="%FrcLedTTOnR%" class="Input-box" placeholder="Type 1 and press GO" min="0" max="1" name="FrcLedTTOnR"> <br> (Type 1 and press GO)

                    <input type="submit" value="GO" onclick="submitMessage()">

                </form>

                <form action="/get" target="hidden-form">

                    Force LED time till OFF reset: <input type="number" value="%FrcLedTTOffR%" class="Input-box" placeholder="Type 1 and press GO" min="0" max="1" name="FrcLedTTOffR"> <br> (Type 1 and press GO)

                    <input type="submit" value="GO" onclick="submitMessage()">

                </form>

            </span>

        </details>

        <iframe style="display:none" name="hidden-form"></iframe>

    </body>

</html>

)rawliteral";

// Function for displaying web pages and requests, that dont exist

void notFound(AsyncWebServerRequest \*request) {

  request->send(404, "text/plain", "Not found");

}

// Function for reading SPIFFS files

String readFile(fs::FS &fs, const char \* path){

  Serial.printf("Reading file: %s\r\n", path);

  File file = fs.open(path, "r");

  if(!file || file.isDirectory()){

    Serial.println("- empty file or failed to open file");

    return String();

  }

  Serial.println("- read from file:");

  String fileContent;

  while(file.available()){

    fileContent+=String((char)file.read());

  }

  file.close();

  Serial.println(fileContent);

  return fileContent;

}

// Function for writing SPIFFS files

void writeFile(fs::FS &fs, const char \* path, const char \* message){

  Serial.printf("Writing file: %s\r\n", path);

  File file = fs.open(path, "w");

  if(!file){

    Serial.println("- failed to open file for writing");

    return;

  }

  if(file.print(message)){

    Serial.println("- file written");

  } else {

    Serial.println("- write failed");

  }

  file.close();

}

// Function for returning requested SPIFFS file values

String processor(const String& var){

  //Serial.println(var);

  if(var == "TrgtTemp"){

    return readFile(SPIFFS, "/TrgtTemp.txt");

  }

  else if(var == "TrgtHumid"){

    return readFile(SPIFFS, "/TrgtHumid.txt");

  }

  else if(var == "TrgtWPP"){

    return readFile(SPIFFS, "/TrgtWPP.txt");

  }

  else if(var == "TrgtWTL"){

    return readFile(SPIFFS, "/TrgtWTL.txt");

  }

  else if(var == "CrntLED"){

    return readFile(SPIFFS, "/CrntLED.txt");

  }

  else if(var == "CrntFan"){

    return readFile(SPIFFS, "/CrntFan.txt");

  }

  else if(var == "CrntHeat"){

    return readFile(SPIFFS, "/CrntHeat.txt");

  }

  else if(var == "CrntVLV"){

    return readFile(SPIFFS, "/CrntVLV.txt");

  }

  else if(var == "FrcWTR"){

    return readFile(SPIFFS, "/FrcWTR.txt");

  }

  else if(var == "CrntTemp"){

    return readFile(SPIFFS, "/CrntTemp.txt");

  }

  else if(var == "CrntHumid"){

    return readFile(SPIFFS, "/CrntHumid.txt");

  }

  else if(var == "TimeToNextPour"){

    return readFile(SPIFFS, "/TimeToNextPour.txt");

  }

  else if(var == "TrgtLedONl"){

    return readFile(SPIFFS, "/TrgtLedONl.txt");

  }

  else if(var == "TrgtLedOFFl"){

    return readFile(SPIFFS, "/TrgtLedOFFl.txt");

  }

  else if(var == "TimeToLedON"){

    return readFile(SPIFFS, "/TimeToLedON.txt");

  }

  else if(var == "TimeToLedOFF"){

    return readFile(SPIFFS, "/TimeToLedOFF.txt");

  }

  else if(var == "FrcLedTTOnR"){

    return readFile(SPIFFS, "/FrcLedTTOnR.txt");

  }

  else if(var == "FrcLedTTOffR"){

    return readFile(SPIFFS, "/FrcLedTTOffR.txt");

  }

  else if(var == "CrntUpTime"){

    return readFile(SPIFFS, "/CrntUpTime.txt");

  }

  return String();

}

// Function for listing all existing SPIFFS files

void listAllFiles(){

  Serial.println("\nSPIFFS file check:");

  // Accsseses the "SPIFFS/" directory

  File root = SPIFFS.open("/");

  File file = root.openNextFile();

  // If any file exists in the current directory print "FILE: FILENAME" and move to next file

  while(file){

    Serial.print("FILE: ");

    Serial.println(file.name());

    file = root.openNextFile();

  }

  // When no more files remain, print empty line

  Serial.println("");

}

// Converts the values of these SPIFFS file to global variables for easier use in ESP32 code

void SpiffsToGlobalVar(){

  // Temporary middle man file

  String inputMessage;

  // Write "/TrgtTemp.txt" value to temporary file.

  inputMessage = readFile(SPIFFS, "/TrgtTemp.txt");

  // Write temporary file to global variable

  GVTrgtTemp = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/TrgtHumid.txt");

  GVTrgtHumid = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/TrgtWPP.txt");

  GVTrgtWPP = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/TrgtWTL.txt");

  GVTrgtWTL = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/CrntLED.txt");

  GVCrntLED = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/CrntFan.txt");

  GVCrntFan = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/CrntHeat.txt");

  GVCrntHeat = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/CrntVLV.txt");

  GVCrntVLV = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/FrcWTR.txt");

  GVFrcWTR = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/TimeToNextPour.txt");

  GVTimeToNextPour = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/TrgtLedONl.txt");

  GVTrgtLedONl = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/TrgtLedOFFl.txt");

  GVTrgtLedOFFl = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/FrcLedTTOnR.txt");

  GVFrcLedTTOnR = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/FrcLedTTOffR.txt");

  GVFrcLedTTOffR = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/TimeToLedON.txt");

  GVTimeToLedON = inputMessage.toInt();

  inputMessage = readFile(SPIFFS, "/TimeToLedOFF.txt");

  GVTimeToLedOFF = inputMessage.toInt();

}

// setup() is called only once, when system is powered on

// However "server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request)" and "server.on("/get", HTTP\_GET, [] (AsyncWebServerRequest \*request)" parts of the setup are called any time the user accesses the corresponding web page

void setup() {

  // Starts serial monitor at 115200 baud rate

  Serial.begin(115200);

  // Initializes pins using predefined values

  pinMode (PIN\_LED, OUTPUT);

  pinMode (PIN\_FAN, OUTPUT);

  pinMode (PIN\_HEATER, OUTPUT);

  // Initializes servo control pin

  waterServo.attach(PIN\_SERVO);

  // Starts DHT22

  dht.begin();

  // Initializes SPIFFS

  #ifdef ESP32

    if(!SPIFFS.begin(true)){

      Serial.println("An Error has occurred while mounting SPIFFS");

      return;

    }

  #else

    if(!SPIFFS.begin()){

      Serial.println("An Error has occurred while mounting SPIFFS");

      return;

    }

  #endif

  // Lists all saved SPIFFS files, these files persist through system restarts

  listAllFiles();

  // SPIFFS are written to global variable, ensuring that any saved values are carried over, when system is restarted

  SpiffsToGlobalVar();

  // Starts and connects to WIFI

  WiFi.mode(WIFI\_STA);

  WiFi.begin(ssid, password);

  if (WiFi.waitForConnectResult() != WL\_CONNECTED) {

    Serial.println("WiFi Failed!");

    return;

  }

  Serial.println();

  // PRINTS IP ADDRESS, FOR CONNECTING TO WEB PAGE

  Serial.print("IP Address: ");

  Serial.println(WiFi.localIP());

  // Function is called when user connects to main web page "CURRENT\_IP/"

  server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request){

    // Sends HTML code to user

    request->send\_P(200, "text/html", index\_html, processor);

    Serial.println("Client connected to main page");

    // Updates the following values form DHT22 sensor and ESP32 millis clock:

    // Current temperature

    inputForUpload = GVCrntTemp;

    writeFile(SPIFFS, "/CrntTemp.txt", inputForUpload.c\_str());

    // Current humidity

    inputForUpload = GVCrntHumid;

    writeFile(SPIFFS, "/CrntHumid.txt", inputForUpload.c\_str());

    // Time till next pour

    inputForUpload = GVTimeToNextPour;

    writeFile(SPIFFS, "/TimeToNextPour.txt", inputForUpload.c\_str());

    // Times since last system start up

    inputForUpload = millis();

    writeFile(SPIFFS, "/CrntUpTime.txt", inputForUpload.c\_str());

    // Time till LED turns ON

    inputForUpload = GVTimeToLedON;

    writeFile(SPIFFS, "/TimeToLedON.txt", inputForUpload.c\_str());

    // Time till LED turns OFF

    inputForUpload = GVTimeToLedOFF;

    writeFile(SPIFFS, "/TimeToLedOFF.txt", inputForUpload.c\_str());

  });

  // Send a GET request to <ESP\_IP>/get?inputString=<inputMessage>

  server.on("/get", HTTP\_GET, [] (AsyncWebServerRequest \*request) {

    String inputMessage;

    // GET inputString value on <ESP\_IP>/get?inputString=<inputMessage>

    if (request->hasParam(PARAM\_TEMP)) {

      inputMessage = request->getParam(PARAM\_TEMP)->value();

      GVTrgtTemp = inputMessage.toInt();

      writeFile(SPIFFS, "/TrgtTemp.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_HUMD)) {

      inputMessage = request->getParam(PARAM\_HUMD)->value();

      GVTrgtHumid = inputMessage.toInt();

      writeFile(SPIFFS, "/TrgtHumid.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_WPP)) {

      inputMessage = request->getParam(PARAM\_WPP)->value();

      GVTrgtWPP = inputMessage.toInt();

      writeFile(SPIFFS, "/TrgtWPP.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_WTL)) {

      inputMessage = request->getParam(PARAM\_WTL)->value();

      GVTrgtWTL = inputMessage.toInt();

      writeFile(SPIFFS, "/TrgtWTL.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_LED)) {

      inputMessage = request->getParam(PARAM\_LED)->value();

      GVCrntLED = inputMessage.toInt();

      writeFile(SPIFFS, "/CrntLED.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_FAN)) {

      inputMessage = request->getParam(PARAM\_FAN)->value();

      GVCrntFan = inputMessage.toInt();

      writeFile(SPIFFS, "/CrntFan.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_HEAT)) {

      inputMessage = request->getParam(PARAM\_HEAT)->value();

      GVCrntHeat = inputMessage.toInt();

      writeFile(SPIFFS, "/CrntHeat.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_VLV)) {

      inputMessage = request->getParam(PARAM\_VLV)->value();

      GVCrntVLV = inputMessage.toInt();

      writeFile(SPIFFS, "/CrntVLV.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_WTR)) {

      inputMessage = request->getParam(PARAM\_WTR)->value();

      GVFrcWTR = inputMessage.toInt();

      writeFile(SPIFFS, "/FrcWTR.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_LNL)) {

      inputMessage = request->getParam(PARAM\_LNL)->value();

      GVTrgtLedONl = inputMessage.toInt();

      writeFile(SPIFFS, "/TrgtLedONl.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_LFL)) {

      inputMessage = request->getParam(PARAM\_LFL)->value();

      GVTrgtLedOFFl = inputMessage.toInt();

      writeFile(SPIFFS, "/TrgtLedOFFl.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_TTNR)) {

      inputMessage = request->getParam(PARAM\_TTNR)->value();

      GVFrcLedTTOnR = inputMessage.toInt();

      writeFile(SPIFFS, "/FrcLedTTOnR.txt", inputMessage.c\_str());

    }

    else if (request->hasParam(PARAM\_TTFR)) {

      inputMessage = request->getParam(PARAM\_TTFR)->value();

      GVFrcLedTTOffR = inputMessage.toInt();

      writeFile(SPIFFS, "/FrcLedTTOffR.txt", inputMessage.c\_str());

    }

    else {

      inputMessage = "No message sent";

    }

    Serial.println(inputMessage);

    request->send(200, "text/text", inputMessage);

  });

  server.onNotFound(notFound);

  server.begin();

  Serial.print("// Start up done in ");

  Serial.print(millis());

  Serial.println("ms //");

}

// Reads temperature with DHT22 and, if successful, updates current temperatures global variable and SPIFFS file

void readTemp(){

  float t = dht.readTemperature();

  if(isnan(t)){

    return;

  }

  else{

    GVCrntTemp = t;

    inputForUpload = GVCrntTemp;

    writeFile(SPIFFS, "/CrntTemp.txt", inputForUpload.c\_str());

    Serial.print("Temperature set to:" );

    Serial.println(t);

  }

}

// Reads humidity with DHT22 and, if successful, updates current humidities global variable and SPIFFS file

void readHumid(){

  float h = dht.readHumidity();

  if(isnan(h)){

    return;

  }

  else{

    GVCrntHumid = h;

    inputForUpload = GVCrntHumid;

    writeFile(SPIFFS, "/CrntHumid.txt", inputForUpload.c\_str());

    Serial.print("Humidity set to:" );

    Serial.println(h);

  }

}

// Compares if current temperature global variable is too high, too low or close enough to target temperature global variable

// Controls heater accordingly

void tempHandling(){

  Serial.println("Handling temp");

  if(GVCrntTemp > GVTrgtTemp - 1 && GVCrntTemp < GVTrgtTemp + 1){

    heaterHandling(0);

  }

  else if (GVCrntTemp < GVTrgtTemp - 2){

    heaterHandling(1);

  }

  else if (GVCrntTemp > GVTrgtTemp + 2){

    heaterHandling(0);

  }

}

// Compares if current humidity global variable is too high or current temperature global variable is too high

// Controls fan accordingly

void humidHandling(){

  Serial.println("Handling humidity");

  if(GVCrntTemp > GVTrgtTemp + 2 || GVCrntHumid > GVTrgtHumid + 3){

    fanHandling(1);

  }

  else{

    fanHandling(0);

  }

}

// Starts or stops fan, depending on the global fan state variable and functions called variable

void fanHandling(int fanStateCall){

  Serial.print(GVCrntFan);

  Serial.print(" <- Global var | Fan handling | Call var -> ");

  Serial.println(fanStateCall);

  // if global fan state variable is 3 or 2, fan is forced ON or OFF

  if(GVCrntFan == 3){

    digitalWrite(PIN\_FAN, HIGH);

  }

  else if(GVCrntFan == 2){

    digitalWrite(PIN\_FAN, LOW);

  }

  else{

    // else if function is called with 1 or 0, fan is auto ON or OFF

    if(fanStateCall == 1)

    {

      GVCrntFan = 1;

      digitalWrite(PIN\_FAN, HIGH);

    }

    else if(fanStateCall == 0){

      GVCrntFan = 0;

      digitalWrite(PIN\_FAN, LOW);

    }

    // if fan value was updated automaticaly, update fan state SPIFFS file with new value

    inputForUpload = GVCrntFan;

    writeFile(SPIFFS, "/CrntFan.txt", inputForUpload.c\_str());

  }

}

// Starts or stops heater, depending on the global heater state variable and functions called variable

// Almost identical to fanHandling

void heaterHandling(int heatStateCall){

  Serial.print(GVCrntHeat);

  Serial.print(" <- Global var | Heater handling | Call var -> ");

  Serial.println(heatStateCall);

  if(GVCrntHeat == 3){

    digitalWrite(PIN\_HEATER, HIGH);

  }

  else if(GVCrntHeat == 2){

    digitalWrite(PIN\_HEATER, LOW);

  }

  else{

    if(heatStateCall == 1)

    {

      GVCrntHeat = 1;

      digitalWrite(PIN\_HEATER, HIGH);

    }

    else if(heatStateCall == 0){

      GVCrntHeat = 0;

      digitalWrite(PIN\_HEATER, LOW);

    }

    inputForUpload = GVCrntHeat;

    writeFile(SPIFFS, "/CrntHeat.txt", inputForUpload.c\_str());

  }

}

// Delay function, that delays in 10 ms increments, so that Async webserver can check for user requests

void softDelay(unsigned long delayLength){

  Serial.print("Soft delaying loop for ");

  Serial.print(delayLength);

  Serial.print(" ms");

  unsigned long delayStart = millis();

  while(millis() - delayStart <= delayLength){

    Serial.print(".");

    delay(10);

  }

  Serial.println("Resuming");

}

// Starts global water timer, also used to restart the timer

void startWaterTimer(){

  Serial.println("Starting Water Timer");

  GVTTNPstart = millis();

}

// Checks if water timer has expired

// Also updates automatic water handling progress

void checkWaterTimer(){

  unsigned long duration = GVTrgtWTL;

  unsigned long passed = millis() - GVTTNPstart;

  //Serial.print("Water check - Duration - ");

  //Serial.println(duration);

  //Serial.print("Water check - Passed - ");

  //Serial.println(passed);

  // If timerNow - timerStart >= timerDuration

  if(passed >= duration){

    Serial.println("Water timer has expired");

    GVTTNPstart = 0;

    GVWaterHandlingState = 2;

  }

  else{

    GVTimeToNextPour = duration - passed;

    Serial.print("Water timer remaining duration: " );

    Serial.println(GVTimeToNextPour);

    softDelay(500);

  }

}

// Opens water valve

void startWater(){

  Serial.println("Opening Water Valve // START");

  // Opens water valve servo at 1 degree per 15ms

  if(pos >= 179){

    for (pos = VLV\_POS\_CLOSED; pos >= VLV\_POS\_OPEN; pos -= 1) {

      waterServo.write(pos);

      delay(15);

    }

    Serial.println("Opening Water Valve // END");

  }

}

// Closes water valve

void endWater(){

  Serial.println("Closing Water Valve // START");

  // Closes water valve servo at 1 degree per 15ms

  if(pos <= 1){

    for (pos = VLV\_POS\_OPEN; pos <= VLV\_POS\_CLOSED; pos += 1) {

      waterServo.write(pos);

      delay(15);

    }

    Serial.println("Closing Water Valve // END");

  }

}

// Forced and automated water and water timer handler

void waterHandling(){

  Serial.println("Handling water");

  // If global variable valve state is 3 or 2 -> forced ON or OFF

  if(GVCrntVLV == 3){

    startWater();

    GVifAutoWater = 0;

  }

  else if(GVCrntVLV == 2){

    endWater();

    GVifAutoWater = 0;

  }

  else{

    // Else engage automatic water system

    Serial.println("Starting auto water");

    GVifAutoWater = 1;

  }

  // global variable force water timer reset is 1, set it and automated water system progress to 0 (restart it)

  if(GVFrcWTR == 1){

    Serial.println("Forcing water timer reset");

    GVFrcWTR = 0;

    GVWaterHandlingState = 0;

    inputForUpload = GVFrcWTR;

    writeFile(SPIFFS, "/FrcWTR.txt", inputForUpload.c\_str());

  }

  // If automatic water system is 0 (OFF) skip it

  if(GVifAutoWater == 0){

    Serial.println("Skipping auto water hangling");

    softDelay(500);

  }

  else{

    // Automated water system has 3 stages:

    // Stage 0 - Starts water timer and go to Stage 1

    if(GVWaterHandlingState == 0){

      Serial.println("Auto water state 0 // START");

      startWaterTimer();

      GVWaterHandlingState = 1;

    }

    // Stage 1 - Checks water timer, which, when expired, will set progress to Stage 2

    else if(GVWaterHandlingState == 1){

      Serial.println("Auto water state 1 // START");

      checkWaterTimer();

    }

    else if(GVWaterHandlingState == 2){

      // Stage 2 - Open water valve, Calculates how long water valve has to be open, to drop users set amount of water, soft delays for that long, Closes water valve, Sets Stage to 0

      Serial.println("Auto water state 2 // START");

      startWater();

      int delayTime = GVTrgtWPP / (WATER\_FLOWRATE \* 1000);

      Serial.print("Delay for: ");

      Serial.print(delayTime);

      Serial.println(" ms");

      softDelay(delayTime);

      endWater();

      GVWaterHandlingState = 0;

    }

  }

}

// Start timer, that goes while LED is OFF

void startLedTimeTillON(){

  GVLedTimeTillONStart = millis();

}

// Start timer, that goes while LED is ON

void startLedTimeTillOFF(){

  GVLedTimeTillOFFStart = millis();

}

// Checks if LED time till ON timer has expired and returns: Expired(true) , In progress(false)

bool checkLedTimeTillON(){

  unsigned long LEDTONduration = GVTrgtLedOFFl;

  unsigned long LEDTONpassed = millis() - GVLedTimeTillONStart;

  if(LEDTONpassed >= LEDTONduration){

    Serial.println("LED time till ON expired");

    return true;

  }

  else{

    GVTimeToLedON = LEDTONduration - LEDTONpassed;

    Serial.print("LED OFF remaining: " );

    Serial.println(GVTimeToLedON);

    return false;

  }

}

// Checks if LED time till OFF timer has expired and returns: Expired(true) , In progress(false)

bool checkLedTimeTillOFF(){

  unsigned long LEDTOFFduration = GVTrgtLedONl;

  unsigned long LEDTOFFpassed = millis() - GVLedTimeTillOFFStart;

  if(LEDTOFFpassed >= LEDTOFFduration){

    Serial.println("LED time till OFF expired");

    return true;

  }

  else{

    GVTimeToLedOFF = LEDTOFFduration - LEDTOFFpassed;

    Serial.print("LED ON remaining: " );

    Serial.println(GVTimeToLedOFF);

    return false;

  }

}

// Handles LED lighting

void ledHangling(){

  // Checks if LED timers have to be reset

  if(GVFrcLedTTOffR == 1){

    GVFrcLedTTOffR = 0;

    startLedTimeTillOFF();

    inputForUpload = GVFrcLedTTOffR;

    writeFile(SPIFFS, "/FrcLedTTOffR.txt", inputForUpload.c\_str());

  }

  else if(GVFrcLedTTOnR == 1){

    GVFrcLedTTOnR = 0;

    startLedTimeTillON();

    inputForUpload = GVFrcLedTTOnR;

    writeFile(SPIFFS, "/FrcLedTTOnR.txt", inputForUpload.c\_str());

  }

  // LED is forced ON(3) or OFF(2)

  Serial.println("Handling LEDs");

  if(GVCrntLED == 3){

    Serial.println("LED forced ON");

    digitalWrite(PIN\_LED, HIGH);

  }

  else if(GVCrntLED == 2){

    Serial.println("LED forced OFF");

    digitalWrite(PIN\_LED, LOW);

  }

  // If LED is not forced, then auto LED

  else{

    Serial.println("Auto LED start");

    // If LED is OFF and time till ON has expired -> execute the following

    if(GVCrntLED == 0 && checkLedTimeTillON() == true){

      Serial.println("Auto LED, LED turned ON");

      digitalWrite(PIN\_LED, HIGH);

      startLedTimeTillOFF();

      GVTimeToLedON = 0;

      GVCrntLED = 1;

      inputForUpload = GVCrntLED;

      writeFile(SPIFFS, "/CrntLED.txt", inputForUpload.c\_str());

    }

    // Else if LED is ON and time till OF has expired -> execute the following

    else if(GVCrntLED == 1 && checkLedTimeTillOFF() == true){

      Serial.println("Auto LED, LED turned OFF");

      digitalWrite(PIN\_LED, LOW);

      startLedTimeTillON();

      GVTimeToLedOFF = 0;

      GVCrntLED = 0;

      inputForUpload = GVCrntLED;

      writeFile(SPIFFS, "/CrntLED.txt", inputForUpload.c\_str());

    }

  }

}

// Loop function is executed continuously

void loop() {

  // Denotes the start of a new loop in the serial monitor and starts a timer

  Serial.println("\n// New loop //");

  unsigned long loopMillis = millis();

  // Calls the following functions:

  readTemp();

  readHumid();

  tempHandling();

  humidHandling();

  ledHangling();

  waterHandling();

  // Denotes the end of a loop in the serial monitor and prints how long this loop took, to execute

  Serial.print("// Loop done in ");

  loopMillis = millis() - loopMillis;

  Serial.print(loopMillis);

  Serial.println("ms //\n");

}