Laboratorio Integrato

Alto Apprendistato – Backend System Integrator

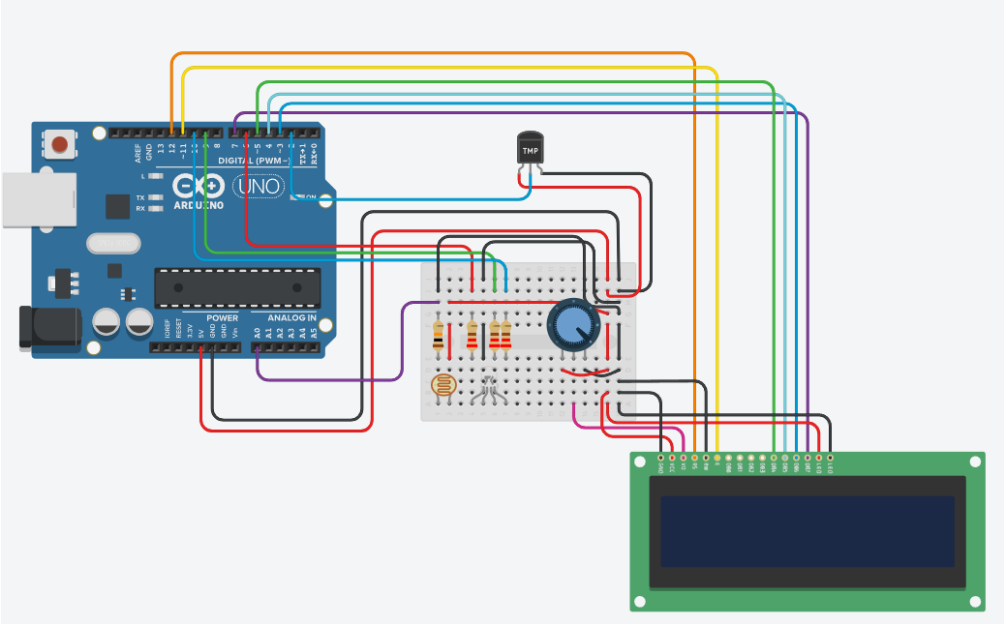
Cinzia Perona

L'obiettivo principale è fornire una soluzione efficiente e accessibile per monitorare le condizioni termoigrometriche di un ambiente utilizzando tecnologie open-source.

Il progetto è nato da un idea mia e di un mio collega ed è stata sviluppata interamente dalla sottoscritta.

Il progetto, Wine Guardian, è diviso in tre macro-parti: Arduino, NodeJS e MySQL.

Arduino



Componenti hardware:

- Scheda Arduino UNO R3
- Potenziometro
- Schermo LCD (16x2):
  Collegamenti (lcd->scheda): VSS -> GND, VDD -> 5v, V0 -> potenziometro, RS -> 12, RW -> GND, E -> 11, D4 -> 5, D5 -> 4, D6 -> 3, D7 -> 7, A -> 5v, K -> GND
- Led RGB: pin 7, 9, 10
- Fotoresistenza: pin A0
- Sensore umidità e temperatura DHT11: pin 2
- Resistenze: 3x220Ω, 1x10kΩ

Software:

Arduino va programmato con il proprio linguaggio che deriva dal C/C++.

Librerie aggiuntive:

- LiquidCrystal: libreria ufficiale necessaria per lo schermo lcd, consente di stampare direttamente dei caratteri sullo schermo.
- DHT Sensor Library: libreria mantenuta da Adafruit per la gestione di sensori DHT, consente di leggere facilmente i valori di temperatura e umidità.

Lettura dati:

Per leggere l'umidità e la temperatura bisogna leggere il relativo evento:
```
dht.humidity().getEvent(&event);
```

E successivamente dall'evento letto si può accedere al valore già convertito:

```
event.relative_humidity
```

La lettura dell'illuminamento viene invece fatta manualmente tramite pin analogico, bisogna quindi leggere il valore del pin: `sensorVal = analogRead(pin_photor);`

Per convertire il valore letto in Lumen bisogna:

- Convertire da analogico a voltaggio: `float Vout = float(sensorVal) * (VIN / float(1023));`
- Convertire da voltaggio a resistenza: `float RLDR = (R * (VIN - Vout))/Vout;`
- Convertire da resistenza a lumen: `int phys=500/(RLDR/1000);`

Con VIN = 5, R = 10000, pin_photor = A0.

Collegamento con NodeJS:

Una volta letti i dati vengono mandati a NodeJS tramite porta seriale.

I metodi relativi alla porta seriale sono interni di Arduino.

La porta seriale va inizializzata, è importante che la frequenza scelta sia la stessa utilizzata anche in Node:
```
Serial.begin(9600);
```

Una volta letti i dati li mando a Node tramite una stringa unica:

```
Serial.println(String(nSens)+ "-"+String(lux)+"-"+String(temp)+"-"+String(hum));
```

NodeJS

Node Modules richiesti:

• mysql	• websocket	• http	• express
• body-parser	• path	• cors	• hbs
• SerialPort	• ReadlineParser	• Chart.js	

Altri Node Modules:

- **config.js**: definisce variabili di configurazione come indirizzo ip dei server e connessione al db

- **Arduino.js**: comunicazione bidirezionale con Arduino. Per leggere e mandare dati serve inizializzare la porta seriale a cui è collegato il dispositivo utilizzando SerialPort e ReadlineParser. Nel caso di lettura di dati viene definita una funzione lambda, che manda i dati al server tramite websocket, richiamata automaticamente quando viene rilevata una nuova stringa in arrivo dalla porta seriale.
  Nel caso di scrittura di dati bisogna definire dei metodi da richiamare che mandano il dato tramite la porta, esempio di un metodo:
  ```
  sendColor(hex){
      console.log(`send to arduino: ${hex}`);
      this.arduinoSerialPort.write(hex,"hex");}
  ```
- **Store.js**: connessione con database. Nel costruttore viene inizializzato un oggetto di connessione al db. Le query vengono tutte eseguite tramite il metodo execQuery che riceve come parametri la stringa di query e i valori richiesti alla query, il metodo restituisce una Promise con i dati restituiti dalla query in caso di successo oppure l'errore in caso di fallimento. Sono presenti poi altri metodi che definiscono la query e la fanno eseguire come:

  ```
  getWine(key){

      const query = `SELECT * FROM ${config.table_wine}
      WHERE name = ?`;

      return this.execQuery(query,key);

      }
  ```

```
.
└── Wine_Guardian/
    ├── arduino-code/
    │   └── smart-sensor.ino
    ├── db/
    │   └── dump.sql
    ├── lib/
    │   ├── css/
    │   │   ├── bootstrap.min.css
    │   │   └── ...
    │   └── js/
    │       ├── bootstrap.min.js
    │       ├── handlebars-v4.7.8.js
    │       └── ...
    ├── node_modules
    ├── views/
    │   ├── css/
    │   ├── js/
    │   ├── partials/
    │   ├── index.hbs
    │   ├── layout.hbs
    │   ├── tab_sensors.hbs
    │   └── ...
    ├── app.js
    ├── Arduino.js
    ├── config.js
    ├── package-lock.json
    ├── package.json
    └── Store.js
```

App.js è il file principale, da eseguire tramite Node, gestisce il server Express, il server Websocket e definisce gli Helper di Handlebars utilizzati nelle pagine hbs.

Server Express:

Vengono definite come *static* le cartelle: views, lib, node_modules.

Inizializzazione server:

```
app.listen(config.node_port, config.ip, function () {
    console.log(`Wine Guardian listening on ${config.node_port}`);
});
```

Vengono messi a disposizione vari endpoint get, post o delete. Esempio di un endpoint che carica una nuova pagina hbs con dati letti da db:

```
app.get('/tab_sensors', async (req,res)=>{
    const data = await store.getAllSensor();
    res.locals = data;
    res.render('tab_sensors');
});
```

Server WebSocket:

Ci si collega l'oggetto Arduino e la pagina web.

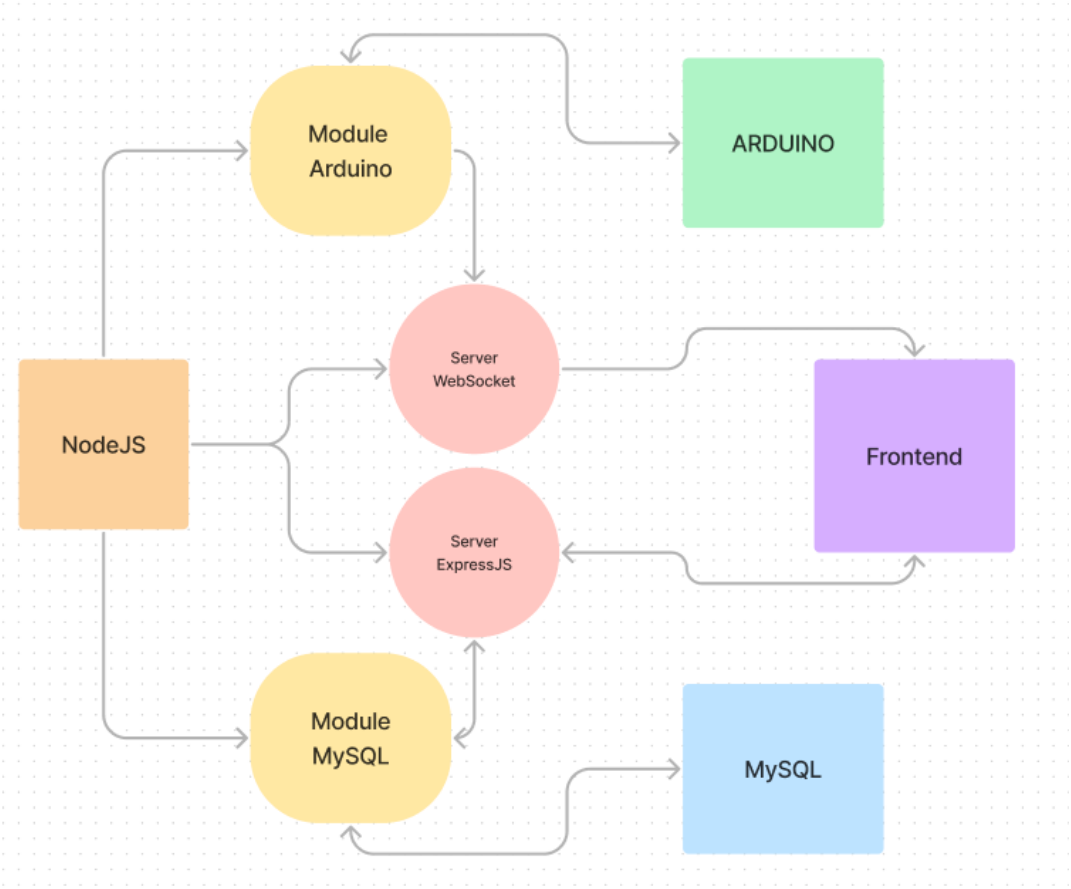
Quando riceve dati controlla se sono i dati dei sensori di Arduino; in caso affermativo li salva su db e fa partire una query che controlla se i dati rientrano nei parametri accettati, poi richiama il metodo per mandare un colore in esadecimale ad arduino: rosso se almeno uno dei dati non va bene, verde se vanno tutti bene. Successivamente manda a tutti i client collegati un segnale per indicare che sono arrivati nuovi dati così da far ricaricare le pagine per visualizzarli.

Handlebars Helper:

Registro nuovi Helper da utilizzare in tutte le pagine hbs, utili per confrontare dati oppure per modificare come vengono mostrati a schermo, esempio Helper per convertire un timestamp in formato classico europeo:

```
hbs.registerHelper("parseDate", function(ts) {
    let date = new Date(ts);
    return `${date.getDate()}/${date.getMonth()+1}/${date.getFullYear()}
${date.getHours()}:${date.getMinutes()}:${date.getSeconds()}`
});
```

Comunicazione Arduino – NodeJS – Frontend:



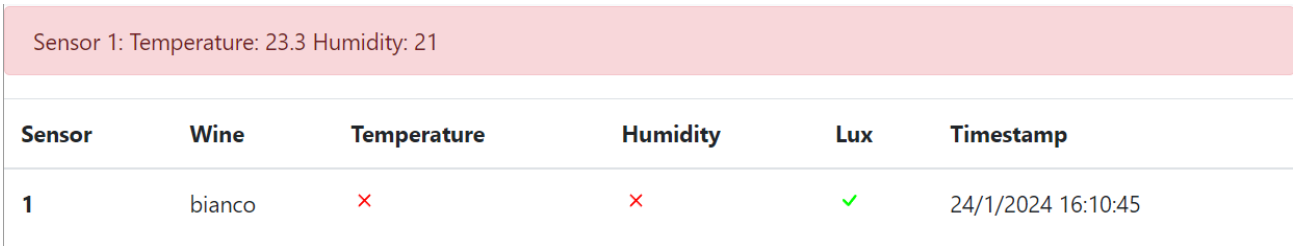
Frontend:

Il progetto utilizza le librerie di: Bootstrap, DataTables, Handlebars e Chart.js.

Il sito è diviso in:

- **Homepage**

Mostra una tabella con gli ultimi dati letti per ciascun sensore per poter facilmente sapere se i valori che stanno venendo letti vanno bene o no. Nel caso in cui uno dei valori non vada bene viene anche mostrato un alert:



Per generare la tabella viene eseguita una fetch per ricevere gli ultimi dati di tutti i sensori prendendo anche i relativi valori di confronto assegnati per poter facilmente eseguire il controllo.

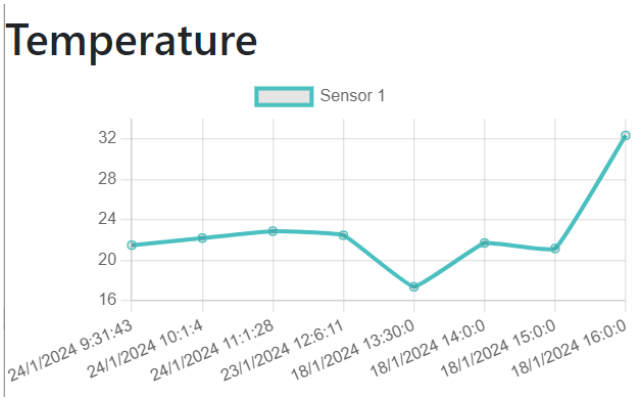
La `<thead>` della tabella la inizializzo in modo statico invece il `<tbody>` lo genero in modo dinamico eseguendo:

```
for (const sensor of lastValue) .
```

Vengono mostrati anche due grafici, generati utilizzando Chart.js: uno per la temperatura e uno per l'umidità. Per generarli viene eseguita una fetch per raccogliere tutti i dati, la query restituisce i dati raggruppati per sensore ed esegue la media per ogni ora. Viene poi eseguito `for(const value in data)` generando il JSON con i dati per la riga quando viene cambiato sensore.

JSON per la riga del sensore:

```
dataTemp.push({
   label: `Sensor ${data[value].id_sensor}`,
   data: temp,
   fill: false,
   borderColor: color[data[value].id_sensor-1],
   tension: 0.1
});
```



Una volta finiti i dati crea il grafico con i dati letti:

```
new Chart(
   document.getElementById('sensor_temp'),
   {
      type: 'line',
      data: {
            labels: labels,
            datasets: dataTemp
      }
   }
);
```

Questa pagina stabilisce anche un collegamento col server Websocket in background per ricaricarsi automaticamente quando vengono ricevuti nuovi dati dai sensori. Le altre pagine non lo eseguono per non disturbare l'analisi dati.

- **Sensors / Wines**

Le pagine per visualizzare i sensori e i vini registrati sono strutturalmente uguali, entrambe sono composte da una DataTable.

I dati della tabella vengono direttamente passati dal server Express quando viene richiamato l'endpoint per richiamare la pagina:

```
app.get('/tab_wines', async(req,res)=>{
    const data = await store.getAllWine();
    res.locals = data;
    res.render('tab_wines');
});
```

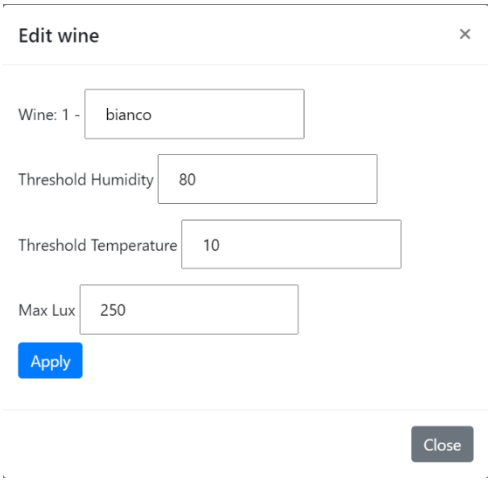
Per generare le righe della tabella viene utilizzata la seguente sintassi di Handlebars:

```
<tbody>
        {{#each _locals}}
            <tr>
                <td>{{id}}</td>
                <td>{{name}}</td>
                <td>{{t_humidity}}%</td>
                <td>{{t_temperature}}°C</td>
                <td>{{t_lux}}</td>
            </tr>
        {{/each}}
    </tbody>
```

E la tabella viene inizalizzata con i seguenti parametri:

```
let table = new DataTable('#tab_wines',{
        dom: "<'row'<'col-sm-12 col-md-6'B><'col-sm-12 col-md-6'f>>" +
            "<'row'<'col-sm-12'tr>>" +
            "<'row'<'col-sm-12 col-md-5'li><'col-sm-12 col-md-7'p>>",
        buttons:[{
            text: 'Edit',
            action: async function(){...}
        },{
            text: 'Add',
            action: async function(){...}
        }],
        select: true
    });
```

Notare che vengono generati due pulsanti: Edit e Add; che permettono di modificare una linea selezionata oppure aggiungerne una nuova.



Le action mostrano un Modal che contiene un form, il Modal è unico per Edit/Add, i valori mostrati vengono aggiunti nella action.

In base a quale titolo viene mostrato il pulsante 'Apply' richiama una fetch diversa, le due fetch sono entrambe di tipo POST, cambia l'endpoint richiamato. Esempio di una fetch:

```
await fetch('/addWine', {
      method: 'POST',
      headers: {
         "Content-type": "application/json; charset=UTF-8"
      },
      body: JSON.stringify({
          name: document.getElementById('inpNameWine').value,
          t_hum: document.getElementById('inpThum').value,
          t_temp: document.getElementById('inpTtemp').value,
          t_lux: document.getElementById('inpTlux').value
       })
 });
```

- **Values**

Stessa struttura delle pagine Sensors e Wines con alcune differenze significative.

La tabella non permette la modifica o l'aggiunta manuale di dati; quindi, non ha pulsanti ma viene generata applicando un ordinamento descrescente in base alla data in questo modo:

```
new DataTable('#tab_values',{
        order: [[0,'desc']]
    });
```

Datatables implementa automaticamente l'ordinamento delle colonne, ma il confronto viene fatto come stringa quindi per la colonna della data è importante specificare una stringa diversa da quella mostrata. Il codice è il seguente:

```
<td data-sort={{orderDate timestamp}}>{{parseDate timestamp}}</td>
```

Utilizza due Helper definiti nel file app.js per convertire il timestamp in due diversi formati.

- orderDate formatta la data secondo lo standard UTC
- parseDate la formatta secondo lo stile: DD/MM/YYYY hh:mm:ss

Altro elemento importante di visualizzazione è il simbolo × o √ vicino ai valori letti. Per generarli utilizzo il seguente codice:

```
<td>{{humidity}}%
      {{#if (confHum humidity t_humidity)}}svg ...></svg>
       {{else}}<svg ...></svg>
{{/if}}</td>
```

Col seguente risultato:

Timestamp	Id sensor	Wine	Humidity	Temperature	Lux
24/1/2024 9:59:23	1	bianco	24% ✗	21.6° ✗	158 ✓

MySQL

Il DataBase è composto da tre tabelle:

- Wine: tabella per registrare dei valori di soglia di confronto per i valori letti dai sensori. Ha le seguenti colonne:
  - id: valore int autoincrementale, chiave primaria
  - name: nome per identificare la configurazione
  - t_temperature: è il valore di soglia della temperatura
  - t_humidity: è il valore di soglia dell'umidità
  - t_lux: è il valore di soglia dell'illuminamento, misurato in Lumen.
- Sensor: tabella per registrare nuovi sensori e associargli una configurazione. Ha le seguenti colonne:
  - id: valore int autoincrementale, chiave primaria
  - id_wine: chiave esterna associata alla tabella wine
- Value: tabella per registrare i valori letti dai sensori. Ha le seguenti colonne:
  - id: valore int autoincrementale, chiave primaria
  - id_sensor: chiave esterna associata al sensore, permette il collegamento con la configurazione associata al sensore per controllare se i valori vanno bene
  - temperature: temperatura letta in gradi Celsius
  - humidity: umidità letta in percentuale
  - lux: illuminamento letto in Lumen
  - timestamp: momento di lettura del valore

Query principali:

- Ultimi valori letti per ogni sensore:

```
select dht.timestamp, dht.id_sensor, dht.humidity, dht.temperature, dht.lux, w.name,
w.t_humidity,w.t_temperature,w.t_lux
from ${config.table_value} as dht
inner join ${config.table_sensor} as s on s.id = dht.id_sensor
inner join ${config.table_wine} as w on w.id = s.id_wine
where dht.timestamp = (select timestamp from ${config.table_value} order by
${config.table_value}.timestamp DESC limit 1);
```

- Controllo se il valore letto da un sensore in un certo momento va bene (con id_sensor e timestamp passati come parametri):

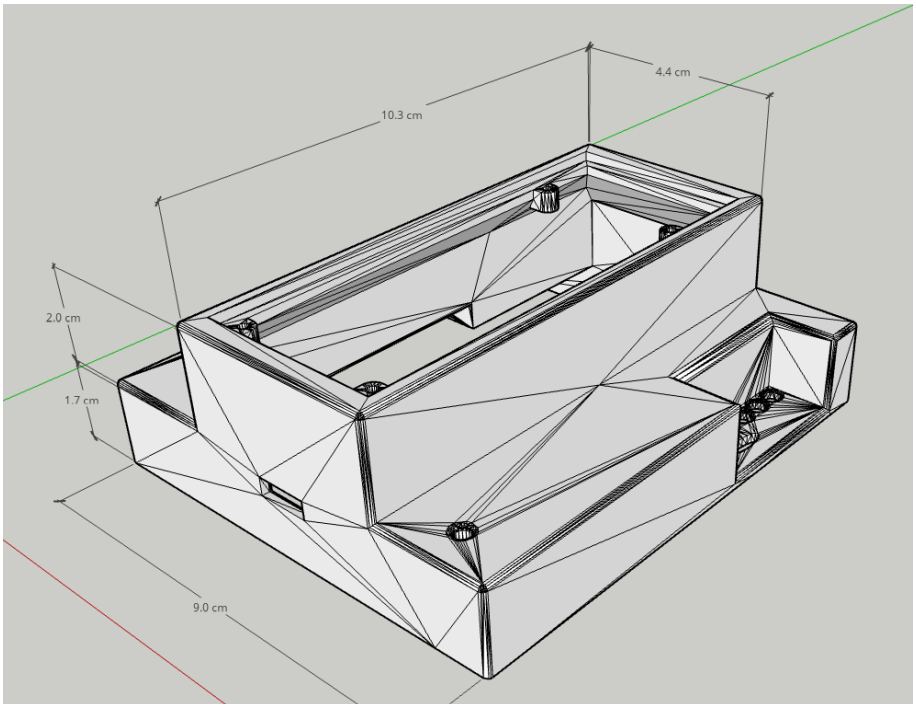
```
select *
```

```
from ${config.table_value} as dht
inner join ${config.table_sensor} as s on s.id=dht.id_sensor
inner join ${config.table_wine} as w on w.id=s.id_wine
where ((dht.humidity>w.t_humidity+3 OR dht.humidity<w.t_humidity-3) OR
(dht.temperature>w.t_temperature+2 OR dht.temperature<w.t_temperature-2) OR
(dht.lux>w.t_lux) ) AND dht.id_sensor = ? AND dht.timestamp = ?;
```

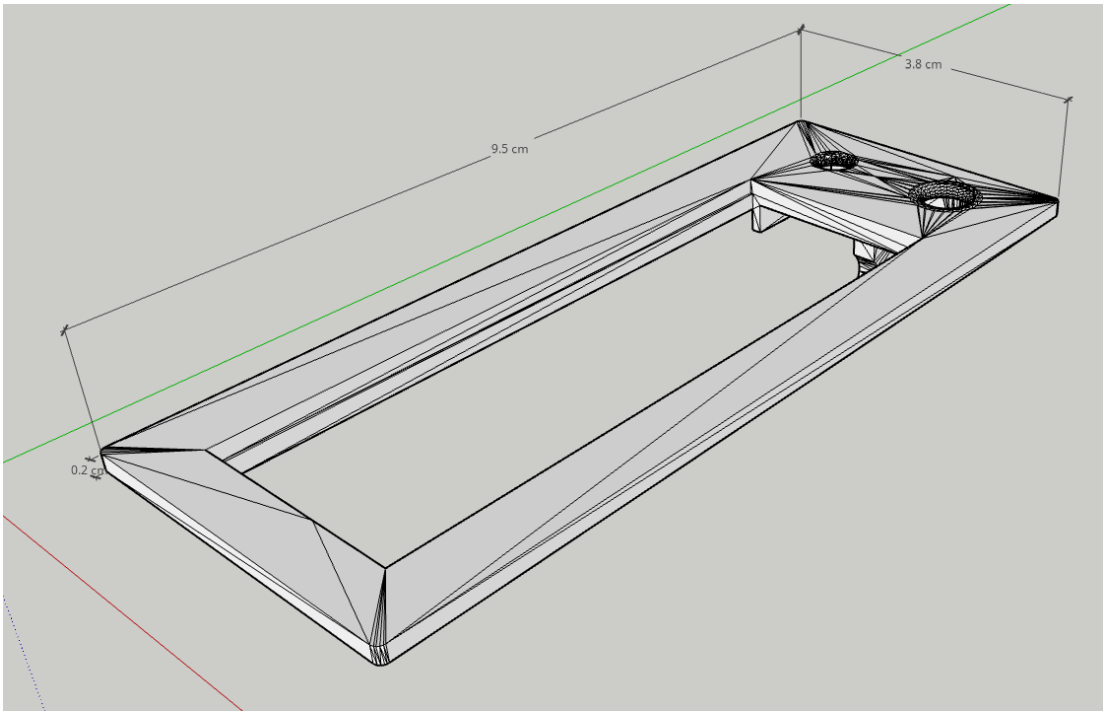
- Valori letti dai sensori raggruppati per sensore ed eseguendo una media dei dati per ogni ora:

```
select dht.id_sensor, dht.timestamp, AVG(dht.temperature) as AVG_TEMP, AVG(dht.humidity)
as AVG_HUM, AVG(dht.lux) as AVG_LUX, w.name as W_NAME, w.t_humidity as W_HUM,
w.t_temperature as W_TEMP, w.t_lux as W_LUX
from ${config.table_value} as dht
inner join ${config.table_sensor} as s on s.id = dht.id_sensor
inner join ${config.table_wine} as w on w.id = s.id_wine
group by dht.id_sensor, HOUR(dht.timestamp)
order by dht.id_sensor
```

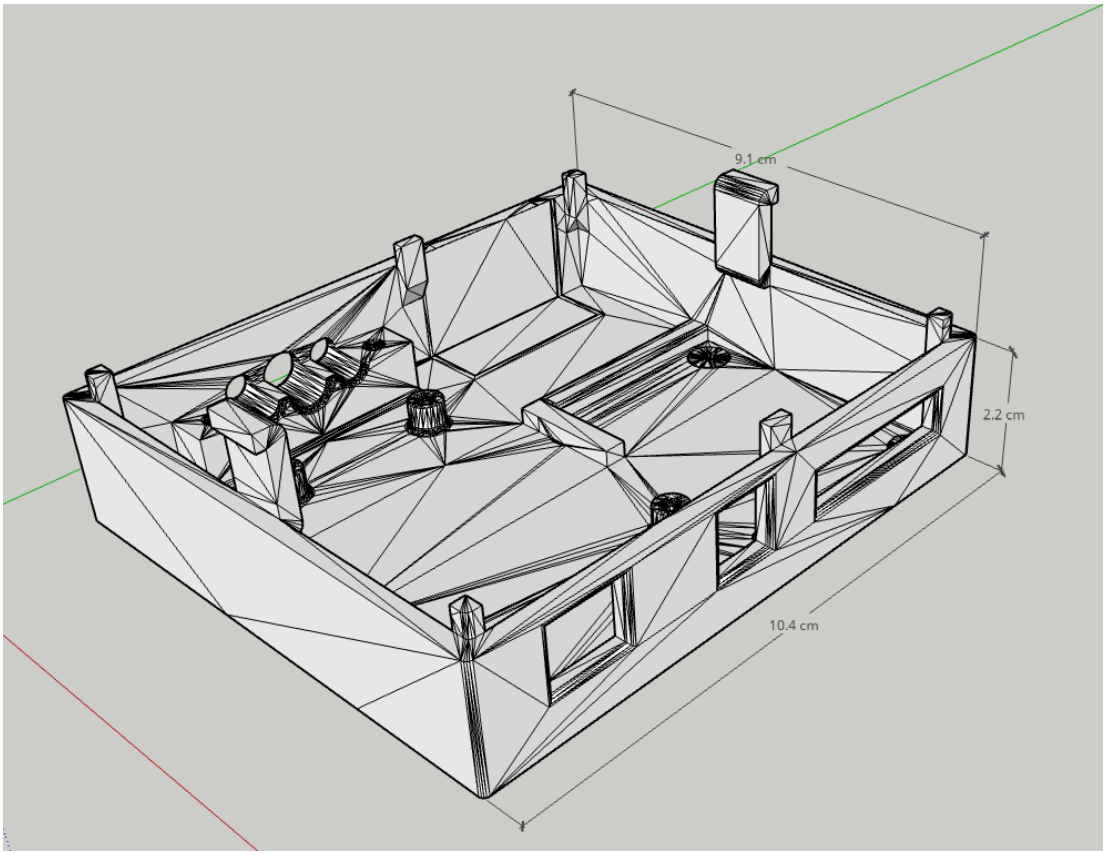
Modelli 3d prototipo



Coperchio



Supporto schermo



Base