

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ**  
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**



**ЗАПИСКА ДО КУРСОВОЇ РОБОТИ**  
**З дисципліни «Мікроконтролери, частина 1»**  
**за темою «Клімат контроль»**

**Виконала:**

ст. гр. ІР-23

ІКТА

Демкович М. С.

**Прийняв:**

доцент КСА

Павельчак А.Г.

**Львів 2022**

## **Зміст**

1. Мета та завдання курсового проекту
2. Вступ
3. Аналіз питання
4. Загальний опис складових
5. Технології, які використовувалися
6. Структурна схема проекту
7. Детальні описи проекту
8. Загальна ілюстрація працездатності проекту
9. Висновки

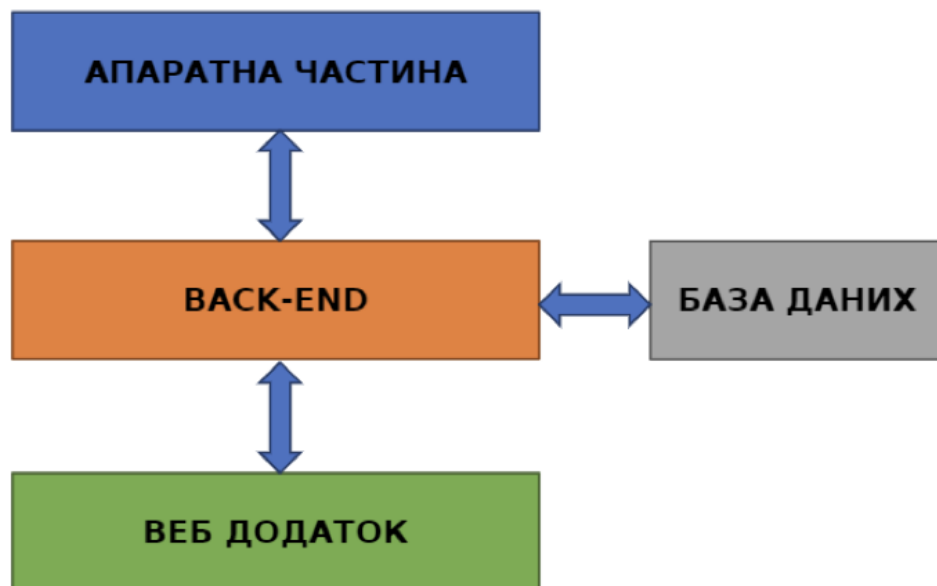
## Мета та завдання курсового проекту

### Завдання

Даний курсовий проект має на меті закріплення знань та навичок, здобутих під час навчання, та охоплює мікроконтролери, схемотехніку, бази даних, створення графічного інтерфейса та інші.

Проект має виконуватися в Proteus і реально працювати.

#### Основні складові курсового проекту:



\*У моєму випадку замість “Веб додатку” буде Graphical User Interface (GUI)

# Вступ

Мій проект : “Клімат контроль”

Клімат контроль - проект за допомогою якого ви можете віддалено вимірювати температуру у вашій квартирі/будинку та контролювати за допомогою кондиціонера .

*Переваги:*

- *Можливість віддалено перевірити температуру у квартирі/будинку;*
- *Можливість вмикати кондиціонер віддалено , щоб врегулювати температуру;*
- *Зручний застосунок для керування.*

*Недоліки:*

- *Не гнучкий функціонал;*
- *Потрібно більше функціоналу для керування температурою у квартирі/будинку;*

Необхідність віддалено контролювати температуру влітку , коли жарко , або взимку , коли холодно , щоб при поверненні додому була комфортна температура/вологість . Для того , щоб зробити життя комфортнішим , я вирішив обрати цю тему для курсового проекту.

Також, це дало змогу покращити свої знання з таких предметів, як мікроконтролери , схемотехніка , бази даних та інші.

## **Аналіз питання**

Зрозумівши основну ідею та проблематику , яку мій проект може вирішити , можна зазначити , що мій курсовий проект допоможе зрозуміти , як саме може відбуватися взаємодія між процесами контролю температури та вологості віддалено у вашій квартирі/будинку.

Для цієї лабораторної роботи, щоб реалізувати поставлене завдання та всі його складові (апаратну частину, back-end, базу даних та front-end , а саме Graphical User Interface (GUI)), потрібно було ознайомитись з певними технологіями та використовувати знання з різних сфер програмування.

Клімат контроль - це проект побудований за допомогою мікроконтролера та з такою периферією, як : датчик вимірювання температури та вологи . Також проект який можна користуватися віддалено завдяки графічному інтерфейсі.

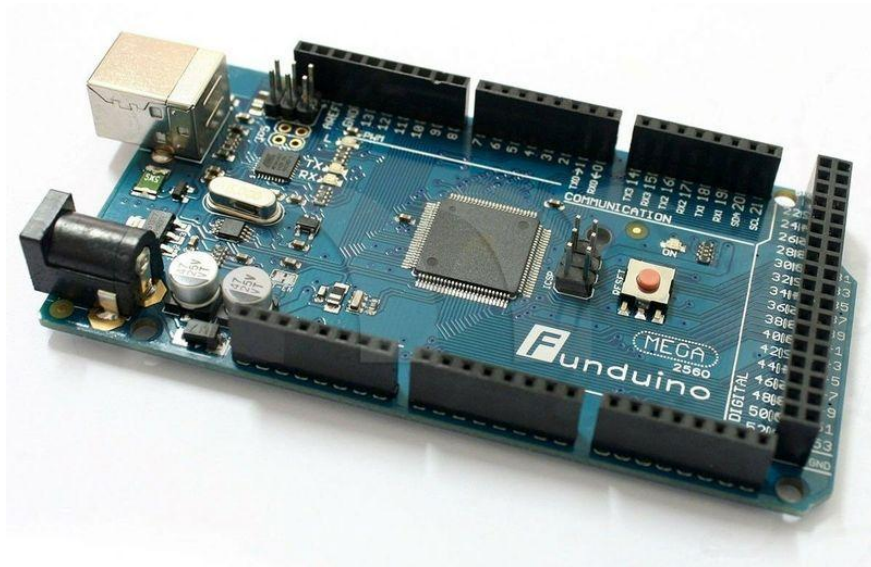
Отже, перейдемо до частин курсового проекту та технологій, використаних для реалізації.

## Апаратна частина

Я використовував Protheus для симуляції роботи електронної схеми з потрібною периферією.

Перш за все, мені знадобилось таке обладнання, як :

### ATMega2560



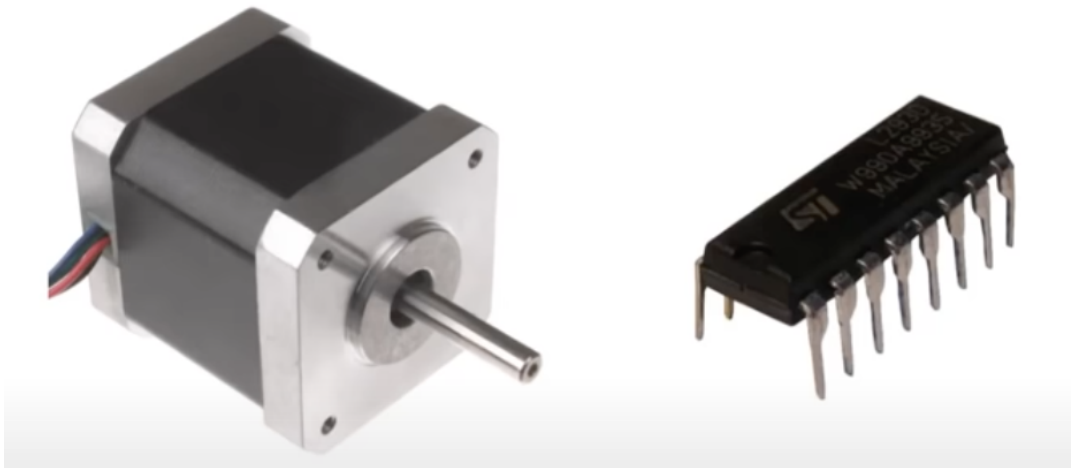
ATMega2560 фірми ATMel. Він служить основою платформи Arduino Mega 2560, що об'єднує на єдиній платі не тільки сам процесор, а й додатково програмований адаптер USB-COM портів ATMega16u2, що дозволяє використовувати ATMega2560 як USB-HID-пристрій.

Мікроконтролер представлений RISC процесором, розробленим AVR і функціонуючим на частоті 16МГц, яка є максимальною з усієї лінійки продуктів ATMel. На кристалі його чіпа розташовані всі пристрої, що відносяться до загального поняття комп'ютерної системи: оперативна і постійна, що перепрограмується, а також flash пам'ять, інтерфейсні мости, помножувач.

Процесор характеризується, як обчислювач одного за часом відгуку, на виконання будь-якої команди, незалежно від її складності. Розрядність шини адрес та внутрішніх регістрів - 8 біт. Максимальний розмір

зовнішньої пам'яті SRAM, що підключається, - 64 Кбайт. Який задає частоту генератор перебуває у складі самої мікросхеми контролера.

## Біполярний кроковий мотор



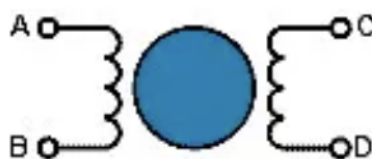
Біполярний кроковий двигун

Драйвер керування двигуном(L239D)

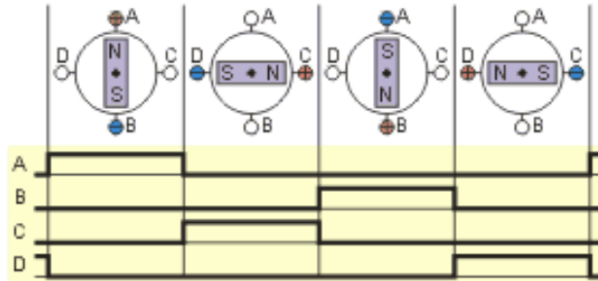
## Bipolar stepper

**Кроковий двигун** - це синхронний безколекторний з кількома обмотками електричний двигун, в якому подача електричного струму на одну з обмоток приводить до того, що його ротор фіксується в строго певному положенні. Послідовне підключення обмоток приводить до обертального руху на заданий кут. Завдяки цьому кут повороту ротора залежить від кількості послідовних перемикань обмоток, а швидкість обертання ротора дорівнює частоті перемикання обмоток, помноженої на кут повороту ротора за одне перемикання.

Біполярний мотор - характерний 2 обмотками



Кроки у мене здійснюються за допомогою способу “one phase on” у повнокроковому режимі



Переваги:

- Головна перевага крокових приводів - точність. При подачі струму в обмотки кроковий двигун повернеться строго на певний кут. Помилка позиціонування в межах 3 - 5% кроку і ця помилка не накопичується від кроку до кроку.
- Залежність обертів двигуна від дискретних імпульсів дозволяє управляти двигуном без зворотного зв'язку.
- Вартість крокових приводів в середньому в 1,5-2 рази нижча сервоприводів. Кроковий привід, як недорога альтернатива сервоприводу, найкращим чином підходить для автоматизації окремих вузлів і систем, де не потрібна висока динаміка

Недоліки:

- Крокові двигуни створюють порівняно високий момент при низьких швидкостях обертання. Момент істотно падає при збільшенні швидкості обертання. Однак, динамічні характеристики двигуна можуть бути істотно поліпшені при використанні драйверів зі стабілізацією струму на основі ШІМ.
- Дискретність кроку створює істотні вібрації, які в ряді випадків можуть призводити до зниження крутного моменту і порушення механічних резонансів в системі. Рівень вібрацій вдається знижувати при використанні режиму дроблення кроку.



## L293D

**L293D IC** відомий як драйвер двигуна. Це низьковольтний робочий пристрій, як і інші мікросхеми. L293D забезпечує безперервний двонаправлений постійний струм до двигуна. Полярність струму може змінитися в будь-який момент, не впливаючи на всю мікросхему або будь-який інший пристрій у ланцюзі. L293D має внутрішній Н-міст, встановлений для двох двигунів постійного струму або одного крокового двигуна.

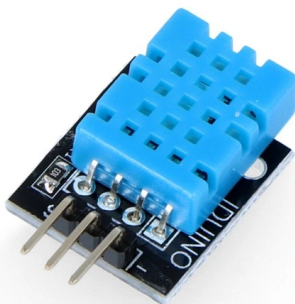
H-Bridge - це електрична ланцюг, що включає навантаження в двонаправленому напрямку. Міст L293D керується зовнішніми сигналами низької напруги. Він може бути невеликим за розміром, але його вихідна потужність вища, ніж ми очікували. Він міг керувати будь-якою швидкістю та напрямком двигуна з діапазоном напруги від 4,5 до 36 вольт.

З'єднання досить просте.

Почнемо з підключення виходу 5 В на Arduino до контактів Vcc2 і Vcc1. Підключаємо заземлення до землі, підключаємо обидва контакти ENA та ENB до виходу 5 В, щоб двигун завжди був увімкненим.

Вхідні контакти (IN1, IN2, IN3 і IN4) мікросхеми L293D до чотирьох цифрових вихідних контактів на платі мікропроцесора.

## DHT11



**DHT11** — це базовий, цифровий датчик температури та вологості.

Він використовує ємнісний датчик вологості та термістор для вимірювання навколишнього повітря та видає цифровий сигнал

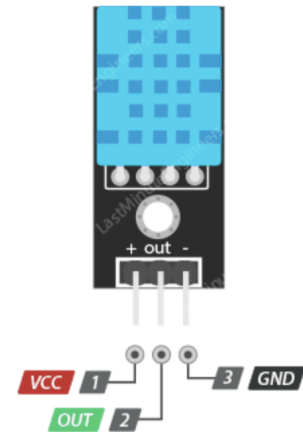
Підключення датчика здійснюється просто :

**+** (VCC) забезпечує живлення датчика.

Рекомендується живлення 5 В.

**Out** використовується для зв'язку між датчиком і Arduino.

**-** (GND) використовується для заземлення.



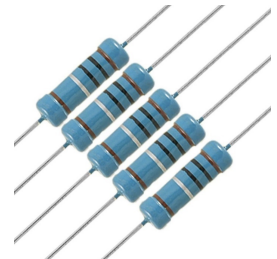
Також я використовував **Кнопки** , **Світлодіод** , **Резистори** та **З'єднувальні кабелі** для підключення до мікроконтролера.



*кнопка*



*світлодіод*



*резистор*



*з'єднувальні кабелі*

## Технології, які використовувалися

### Програмування мікроконтролера

1. Скрипт, написаний мовою C

Середовище розробки **Arduino IDE**



Скрипт:

[https://github.com/DemkovychMaks/kursova/tree/main/kursova\\_project/arduino](https://github.com/DemkovychMaks/kursova/tree/main/kursova_project/arduino)

2. Симуляція і моделювання схеми здійснювалося в **Proteus**



Схема:

[https://github.com/DemkovychMaks/kursova/tree/main/kursova\\_project/protheus](https://github.com/DemkovychMaks/kursova/tree/main/kursova_project/protheus)

### *GUI (Graphical User Interface)*

3. Програма написана мовою програмування Python

Середовище розробки **PyCharm**



використовувалась бібліотека **PyQt**



4. Для дизайну та скрипту GUI використовувалась програма

**Qt Designer**



**Все по GUI:**

[https://github.com/DemkovychMaks/kursova/tree/main/kursova\\_project/python/python\\_kursova](https://github.com/DemkovychMaks/kursova/tree/main/kursova_project/python/python_kursova)

**Бази Даних (Database)**

5. Використовувалась база даних **MySQL**



6. Для проектування баз даних використовувався

## MySQL Workbench

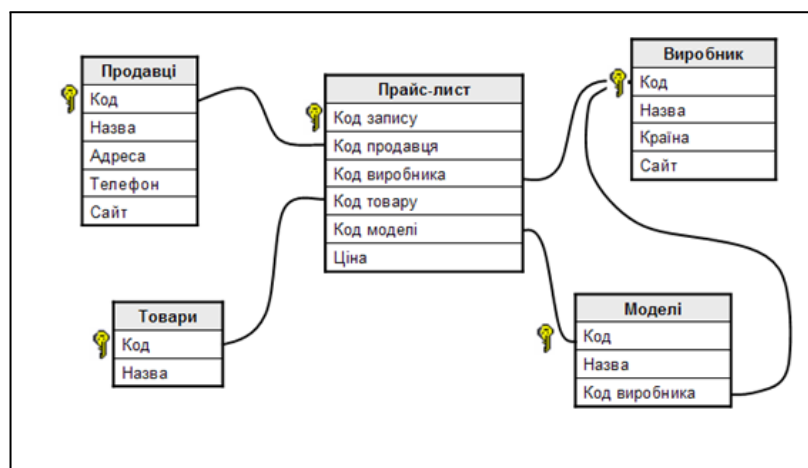


**База даних (database)** – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами.

В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім самих даних, містить їх опис та може містити засоби для їх обробки.

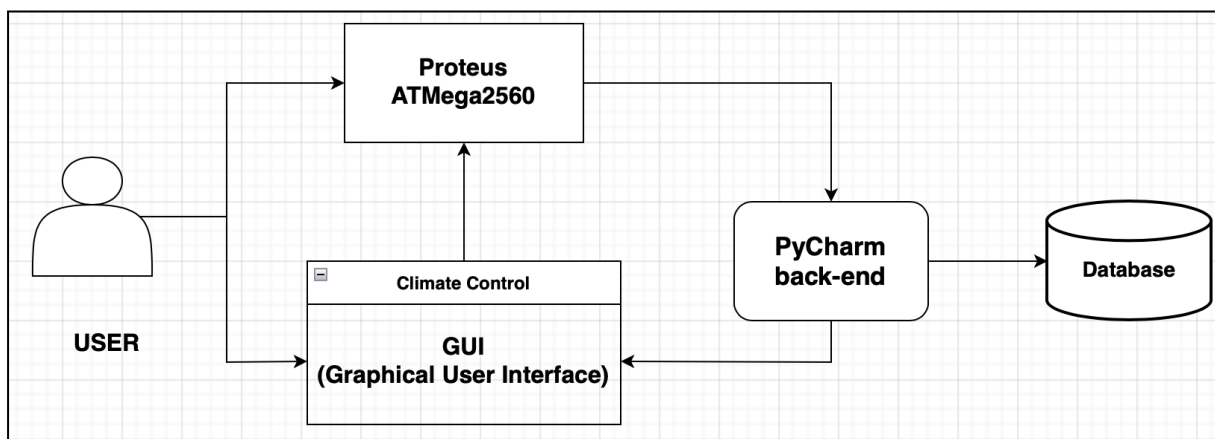
Бази даних можна вважати будь-який впорядкований набір даних.

**MySQL** – це реляційна база даних (СУБД). Реляційна — отже, всередині неї є дані, пов'язані між собою, і цей зв'язок можна подати у вигляді таблиць.



Для роботи з базами даних, я надав перевагу **MySQL**, оскільки вона проста у користуванні, реляційна та багато користувацького функціоналу.

## Структурна схема проекту



Як ми можемо побачити на діаграмі, у нас є чотири основних компоненти, з двома з яких у нас взаємодіє кінцевий користувач, а саме з апаратною частиною та графічним інтерфейсом, який реалізований за допомогою **PyQt**, який власне сприймається користувачем. З іншого боку, є ще така частина як **back-end**, яка прихована від користувача, займається зберіганням, обробкою даних.

Отже, користувач може працювати як і з самим Proteus`ом, щоб керувати мікроконтролером, так і з графічним інтерфейсом GUI, який може також керувати мікроконтролером.

Сам мікроконтрол передає дані на **back-end**, який написаний в PyCharm, а **back-end** вже в свою чергу передає всю інформацію на GUI. Передача даних здійснюється за допомогою **Virtual Serial Ports** (віртуальних послідовних портів).

Вся оброблена інформація знаходиться на **back-end** в PyCharm та ця інформація в свою чергу може бути передана в базу даних. Передача даних здійснюється за допомогою підключення бази даних(MySQL) до **back-end** (PyCharm).

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

## Детальні описи проекту

### 1) Апаратна частина

Почнемо з того, що для реалізації проекту, потрібно скласти схему, яка містить мікроконтролер **ATMega2560**, датчик температури та вологості **DHL11**, біполярний мотор **Bipolar stepper** та драйвер для керування ним **L293D** (ця взаємодія реалізує роботу кондиціонера), дві кнопки для запуску та зупинки мотора, червоний світлодіод, який показує стан кондиціонера.

Для реалізації схеми, нам потрібно було знати базові закони складання фізичних схем, оскільки на кожен плату потрібно подати напругу та заземлити її, ці правила були дотримані при складанні схеми.

Також, для до світлодіода, ми під'єднуємо резистор, тому що напруга яка подається на плату є вищою за допустиму для світлодіодів, тому нам потрібно додатковий опір.

При під'єднанні кнопки, ми повинні приєднувати кнопку до плати, додаючи два з'єднувальних кабелі – один на заземлення, інший в пін мікроконтролера.

При під'єднанні датчика температури та вологості нас було важливо забезпечити живлення датчика (5V), при з'єднанні датчика з мікроконтролером потрібно було додати резистор, який стабілізував напругу на лінії та дані не були спотворені.

При під'єднанні мотора, нам було важливо правильно давати струм на драйвер, який в свою чергу перенаправляє цю напругу на порти мотора, який після цього починав працювати. На драйвер важливо також було правильно подати напругу та заземлити його.

Таким чином ми забезпечуємо механізм правильного складання схеми та симуляція була робочою.

Проте, однієї схеми недостатньо. Для того, щоб певні дії виконувались при натисканні кнопки, щоб передавалась інформація з датчика, щоб працював мотор та світився світлодіод, нам потрібно правильно прошити мікроконтролер. Для цього я написав скрипт мовою C у середовищі розробки Arduino IDE.

Що саме програмується у цьому скрипті?

1. Ініціалізуємо правильно змінні та порти, які будуть на вхід, а які на вихід.
2. Вказуємо за якими номерами пінів, ми будемо звертатись до світлодіодів та кнопок.
3. Складаємо алгоритм передавання напруги на порти мотора.
4. Реалізовуємо переривання.
5. Даємо дозвіл на обмін даними з мікроконтролера за допомогою COM-port.

```
float oldTemp = 0; // ініціалізація змінних
float oldHum = 0;

const int buttonPin1 = 20; // ініціалізація констант
const int buttonPin2 = 21;
const int led = 30;

int val1;
int val2;
int on;

ISR(INT0_vect){ // переривання
on = 0;
}

void setup() {
  Serial.begin(9600); // підключення COM-port`а

  noInterrupts();

  EIMSK |= (1<<INT0); //
  EICRA &= ~(1<<ISC00); // блок переривання
  EICRA |= (1 <<ISC01); //

  interrupts(); // дозволяємо переривання

  DDRC = 0x0F; // ініціалізуємо порти
  DDRC |= (1<<7);

  pinMode(buttonPin1, INPUT_PULLUP); // ініціалізуємо кнопку на вихід
  digitalWrite(buttonPin1, HIGH); // даємо високу напругу і відповідно коли
  // вона буде натиснутою, напруга буде низькою
  pinMode(buttonPin2, INPUT_PULLUP);
  digitalWrite(buttonPin2, HIGH);
}
```



Щоб працювати з датчиком температури та вологості нам потрібно підключити бібліотеку, яка це все опрацьовує:

```
include <dht.h>

#define outPinDHT 22 // номер піна, до якого під'єднаний датчик
```

Як вже раніше згадувалося, дані з мікроконтролера надсилаються на back-end, в свою чергу дані можуть надсилатися з back-end на мікроконтролер.

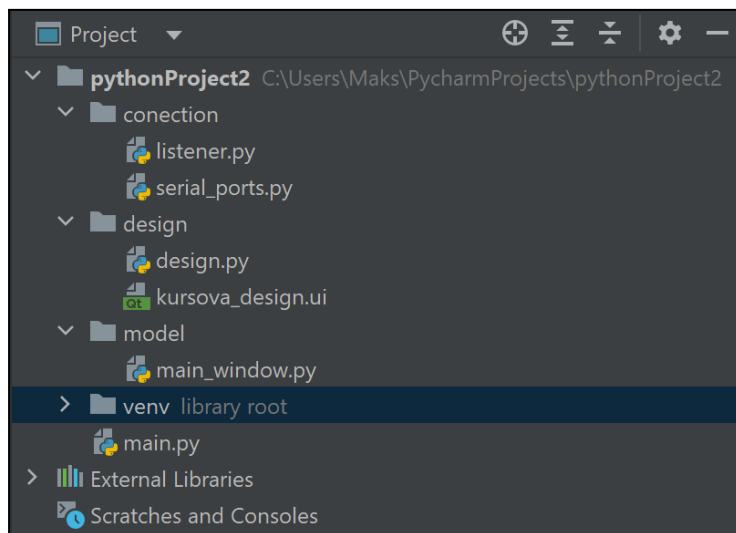
Це здійснюється за допомогою:

```
Serial.print()
Serial.read()
Serial.write()
```

## 2) Back-end

**Back-end** написаний мовою **Python** в середовищі розробки **PyCharm**.

*Структура проекту:*



Отже, почнемо з директорії “connection”.

У цій директорії реалізовано з’єднання віртуальних послідовних портів (**virtual serial ports**).

У директорії “**design**” знаходиться сам дизайн графічного інтерфейсу ( **GUI**), файл “kursova\_design.ui” - це макет, а “design.py” - це також макет дизайну, але переведений у скрипт зрозумілий інтерпретатору Python.

У директорії “**model**” знаходиться **back-end**, завдяки якому ми можемо встановлювати з’єднання з мікроконтролером та базою даних.

Файл “**main**” запускає весь цей проект.

Якщо коротко, про сам **back-end**, то в файлі “main\_window” ініціалізовані змінні, які відповідають певним елементам на клієнтському графічному інтерфейсі, ми створюємо функції, які викликають після натиску на кнопки, в який реалізовані зчитування інформації з **virtual serial ports**, запис даних в ці порти, що в свою чергу відслідковується в процесорі мікроконтролера (процесор ми запрограмували в **Arduino IDE**), що призводить до виконання певних завдань як і в графічному інтерфейсі, так і в самій периферії мікроконтролера.

Надсилення даних з back-end відбувається наступним способом:

```
def AirStart(self):
    message = 11
    self.conditioner = "is working"
    self.conditionerStatus.setText("Conditioner " + self.conditioner)
    if self.port:
        self.port.write(chr(message).encode('utf-8'))
def AirStop(self):
    message = 12
    self.conditioner = "stopped"
    self.conditionerStatus.setText("Conditioner " + self.conditioner)
    if self.port:
        self.port.write(chr(message).encode('utf-8'))
```

Зчитування в процесорі мікроконтролера відбувається ось так:

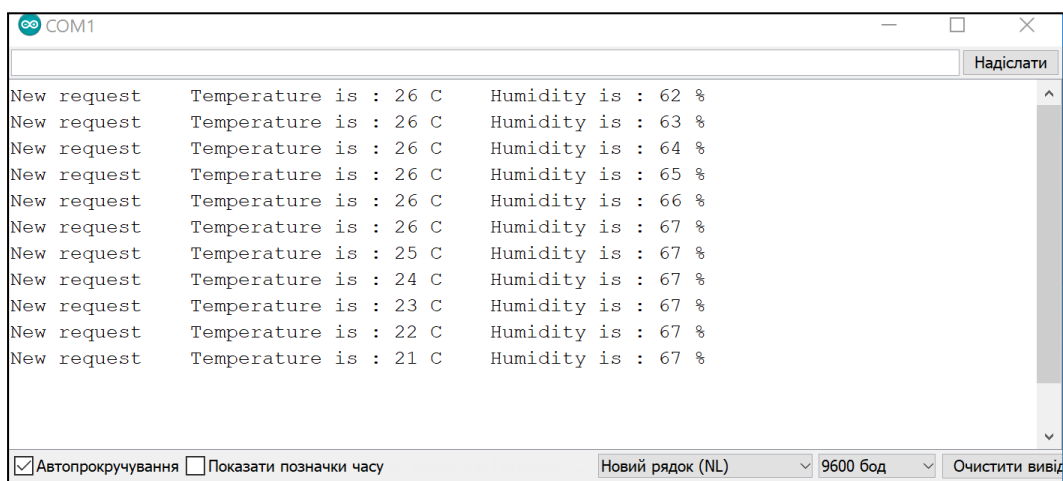
```
int byte = Serial.read();
if (byte == 11) {
    val1 = LOW;
}

int byte = Serial.read();
if (byte == 12) {
    val1 = HIGH;
}
```

Надсилаємо дані з мікроконтролера на COM-port:

```
Serial.print("New request");
Serial.print(" ");
Serial.print("Temperature is : ");
Serial.print(t);
Serial.print(" C");
Serial.print(" ");
Serial.print("Humidity is : ");
Serial.print(h);
Serial.print(" %");
Serial.println("");
```

Дані, які надійшли на COM-port:



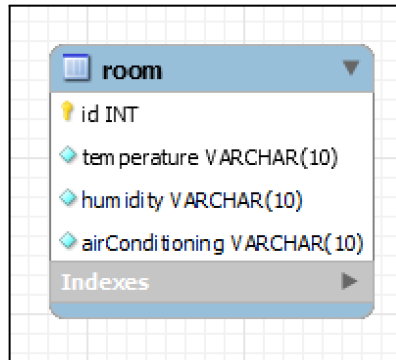
Зчитування даних на back-end з COM-port:

```
def click(self):
    data = str(self.port.readlines()[-1])
    print(data)
    self.temperature = data[34:36]
    self.humidity = data[56:58]
    self.dispTemp.setText(self.temperature + " \N{DEGREE SIGN}C")
    self.dispHum.setText(self.humidity + " %")
    self.dispTemp_2.setText(self.temperature + " \N{DEGREE SIGN}C")
    self.dispHum_2.setText(self.humidity + " %")
```

### 3) База даних

Як згадувало раніше, база даних реалізована на основі MySQL, проектування таблиці здійснювалося в програмі MySQL Workbench.

#### Вигляд таблиці:



Можемо побачити, які поля та типи даних я використовував.

*поле id* - унікальний номер запису в базу даних,

це поле я зробив РК(первинний ключ),

INT (приймаються тільки цілі числа),

NOT NULL (поле повинно бути заповнене),

AUTO\_INCREMENT (автоінкрементним).

*поле temperature* - температура

*поле humidity* - вологість

*поле airConditioning* - вказує на стан кондиціонера (чи він працює, чи ні)

Ці поля просто VARCHAR(10) - можуть вмістити 10 символів,

та NOT NULL (поле повинно бути заповнене).

#### Зберігання даних в базі даних:

|   | id   | temperature | humidity | airConditioning |
|---|------|-------------|----------|-----------------|
| ▶ | 1    | 24          | 51       | is working      |
|   | 2    | 24          | 51       | stopped         |
| ✱ | NULL | NULL        | NULL     | NULL            |

З базою даних ми працюємо за допомогою back-end.

Ось так виглядає підключення та передавання даних до бази даних:

```
def DataBase(self):
    try:
        connection = mysql.connector.connect(
            host="localhost",
            database="db_kursova",
            user="root",
            password="root"
        )
        if connection.is_connected():
            db_info = connection.get_server_info()
            print("Connected to MySQL Server version ", db_info)
            cursor = connection.cursor()

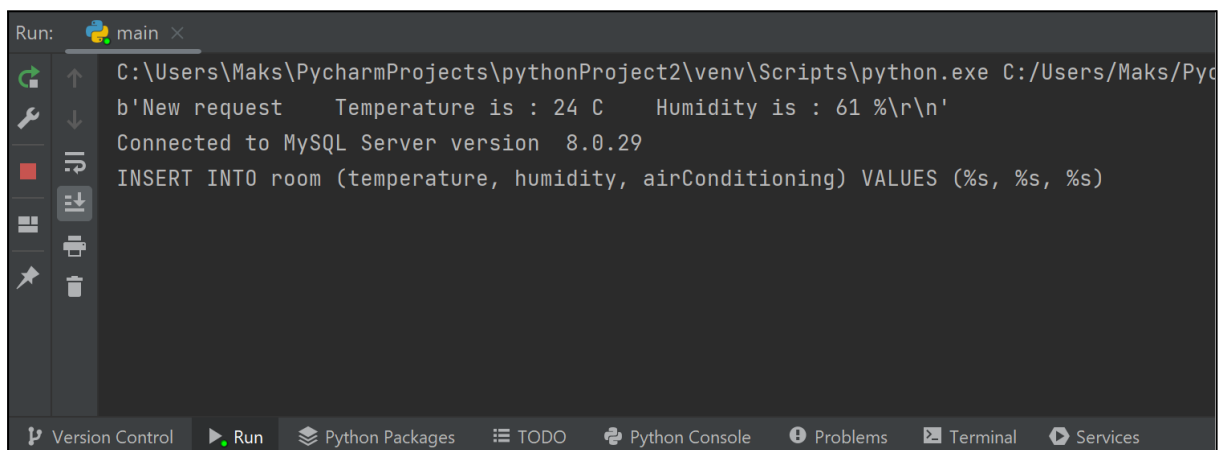
            sql = "INSERT INTO room (temperature, humidity, airConditioning)
VALUES (%s, %s, %s)"
            print(sql)
            val = (self.temperature, self.humidity, self.conditioner)
            cursor.execute(sql, val)

            connection.commit()
            cursor.close()
            connection.close()
        except mysql.connector.Error:
            print("Error while connection to database")
```

Все можемо побачити в консолі:

Спочатку ми витягли дані, потім встановили з'єднання з базою даних про успішне підключення бази даних свідчить повідомлення : "Connected ..."  
та, щоб записати дані в базу даних ми використовуємо команду :

***"INSERT INTO <table\_name> <values> "***



```
Run: main x
C:\Users\Maks\PycharmProjects\pythonProject2\venv\Scripts\python.exe C:/Users/Maks/Pyd
b'New request Temperature is : 24 C Humidity is : 61 %\r\n'
Connected to MySQL Server version 8.0.29
INSERT INTO room (temperature, humidity, airConditioning) VALUES (%s, %s, %s)
```

## 4) GUI (графічний користувацький інтерфейс)

Саме ця частина є найбільш важливою для кінцевого користувача, оскільки вона дає змогу побачити, «натиснути», протестувати функціонал, написаний девелопером.

Весь дизайн графічного інтерфейсу знаходиться у директорії “**design**”.

Як говорилося раніше, макет розроблявся в програмі **QtDesigner**, після розроблення макету потрібно зберегти цей файл, далі нам потрібно цей файл перетворити у зрозумілу “мову” для Python інтерпретатора, це потрібно зробити за допомогою команди:

```
pyuic5 -x kursova_design.ui
```

Після чого, нам потрібно скопіювати з консолі, все що нам вивело, саме це код який потрібний, щоб Python вмів працювати з цим дизайном.

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(920, 650)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
```

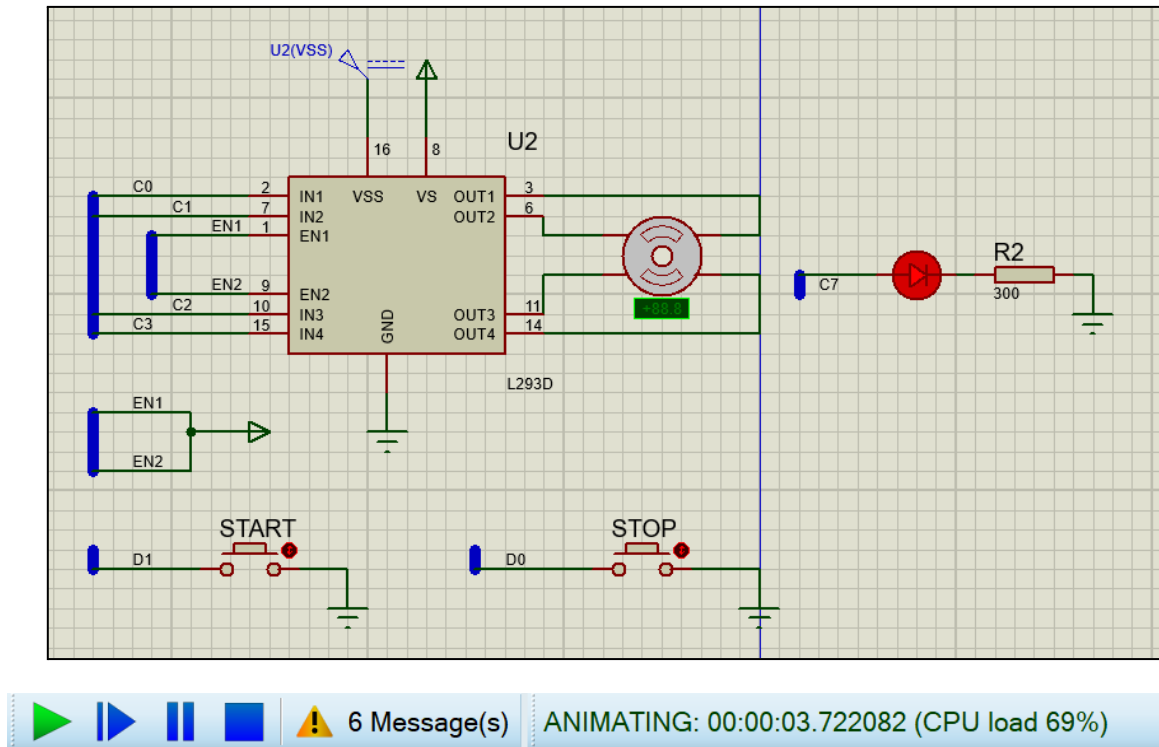
Саме ці функції відповідальні за створення основного вікна графічного інтерфейсу та все інше наповнення яке потрібно.

У моєї аплікації досить мінімалістичний дизайн, оскільки головним досягненням та метою є саме обробка даних, які приходять від мікроконтролера та надсилання даних до нього.

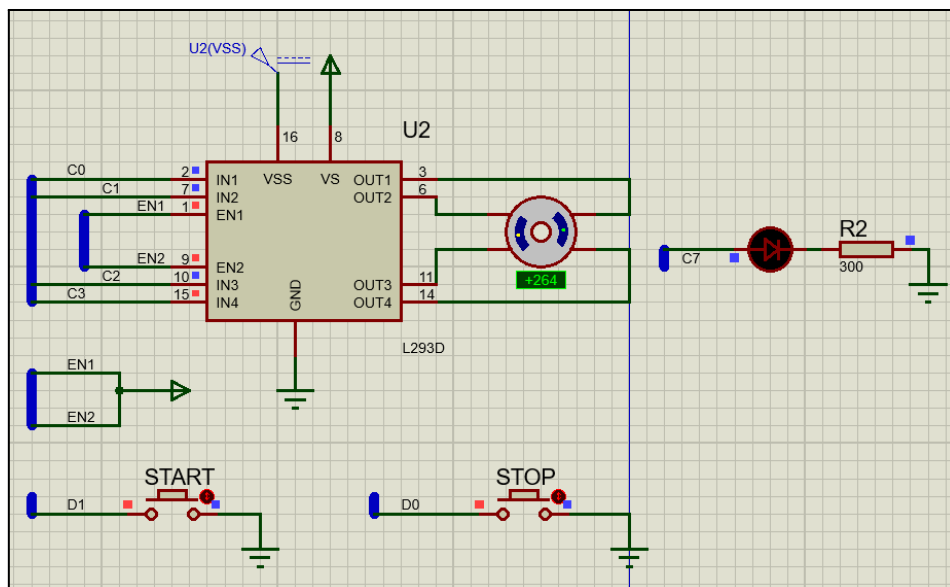
Аплікація запустилась успішно, коли після компіляції основного файлу “**main**”, який безпосередньо зв'язаний з усім проектом, без жодних помилок запустився наш графічний інтерфейс.

# Загальна ілюстрація працездатності проекту

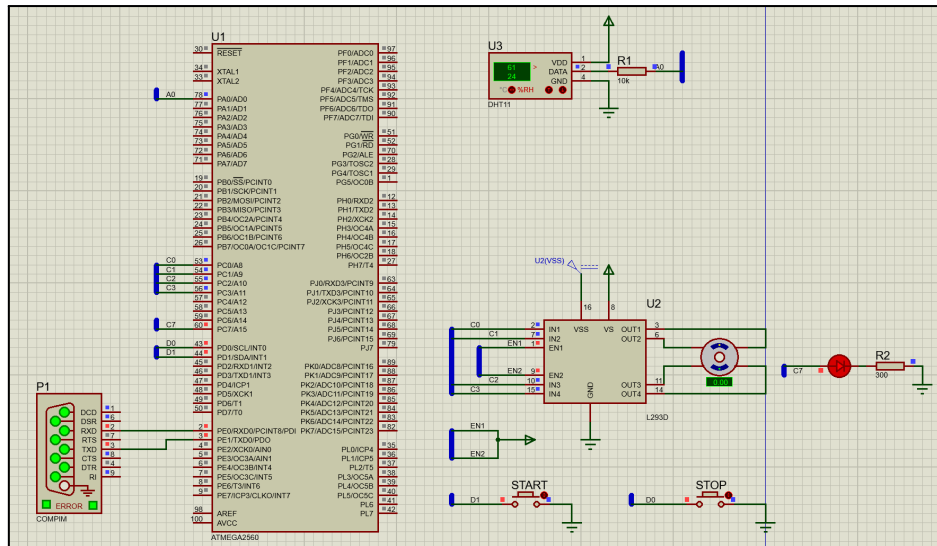
## 1. Спочатку потрібно запустити **Proteus**.



## 2. Увімкнемо та вимкнемо кондиціонер в самій симуляції за допомогою кнопок Start/Stop

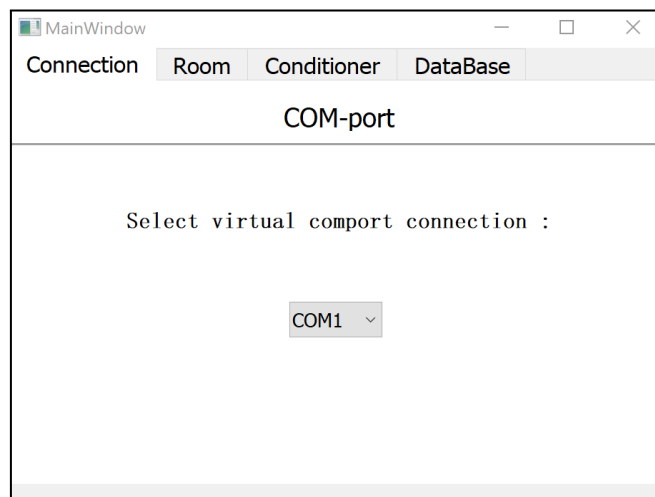


кнопку **Start** натиснуто та кондиціонер запрацював (червоний ледик погас)



кнопку **Stop** натиснуто та кондиціонер вимкнувся (загорівся червоний ледик)

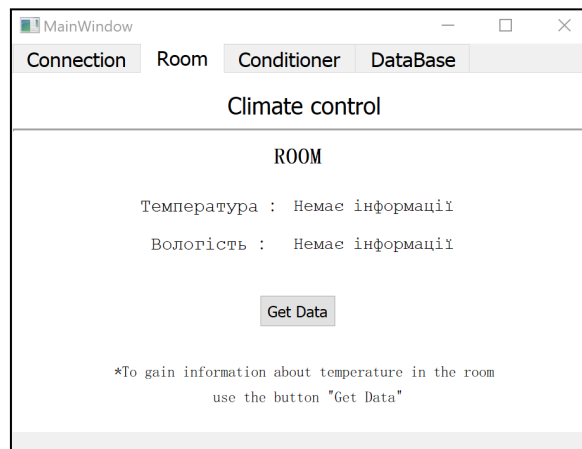
- Для роботи з **GUI** нам всього лише потрібно запустити код в **PyCharm**.  
(також потрібно перевірити вибрані віртуальні порти, щоб все коректно працювало)



перша сторінка графічного інтерфейсу призначена для встановлення з'єднання на віртуальних портах

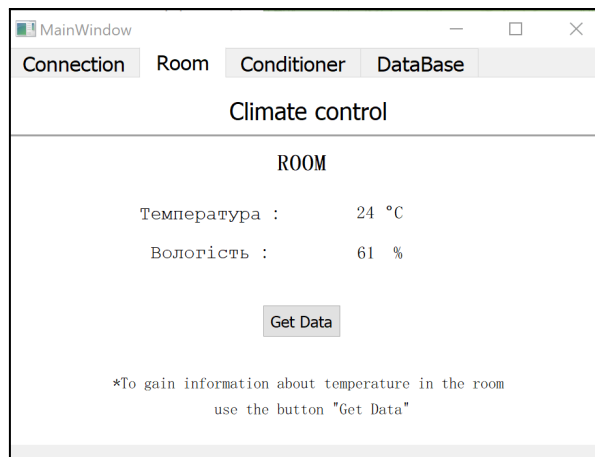


#### 4. Друга сторінка клієнтського інтерфейсу (**Room**)

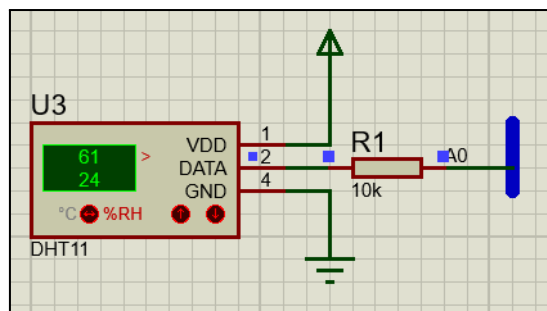


*друга сторінка призначена для виведення інформації з датчика температури та вологості (дані до отримання інформації)*

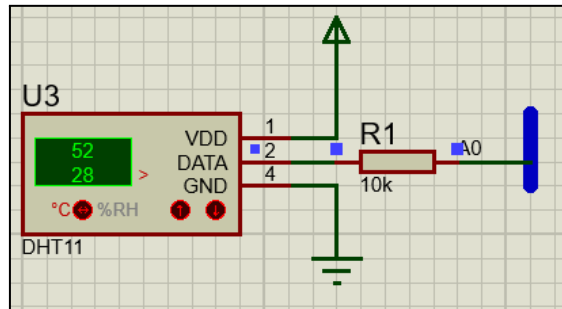
*Натиснемо на кнопку “**Get Data**” для отримання інформації*



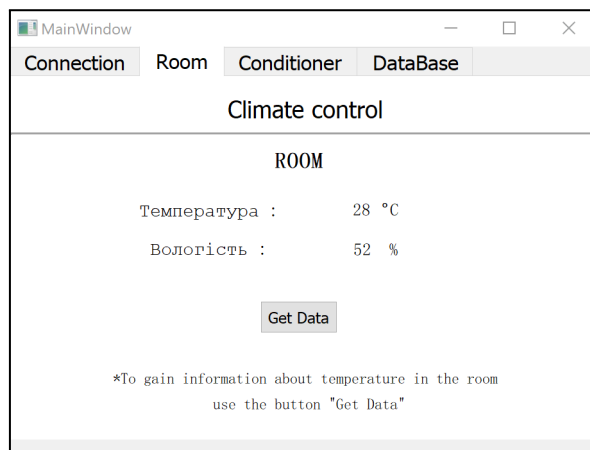
*друга сторінка після отримання даних з датчика*



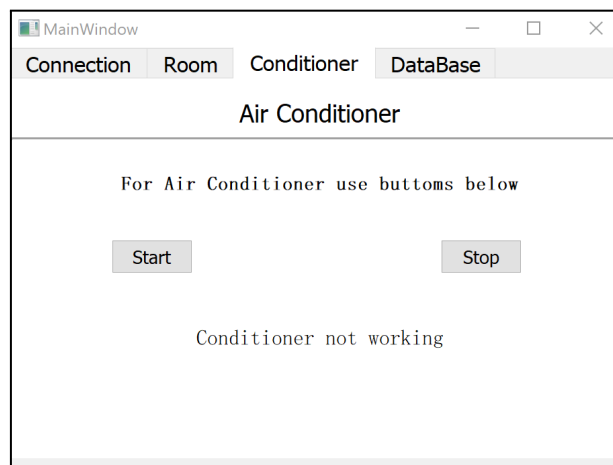
дані з датчика, можемо звірити ці дані з тими, що отримали  
Змінимо дані на датчику



Натиснемо “**Get Data**”, та отримаємо нові дані (дані відповідні)

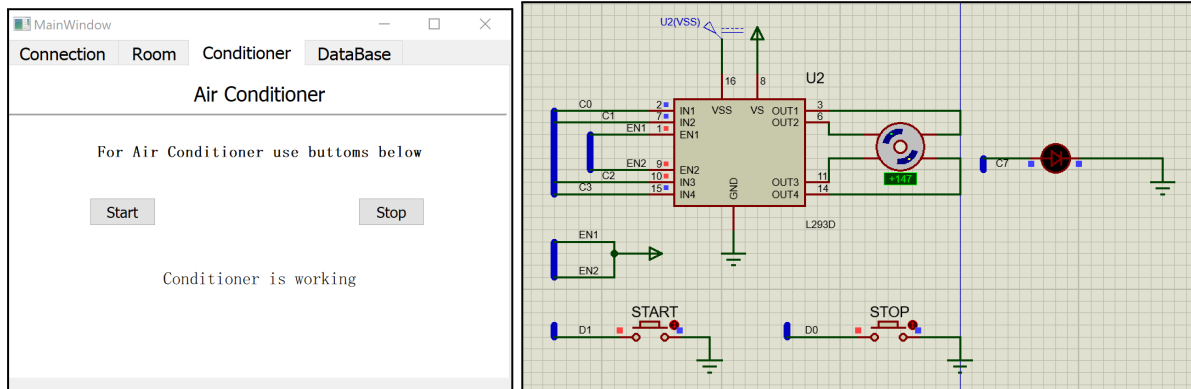


## 5. Третя сторінка клієнтського інтерфейсу (**Conditioner**)

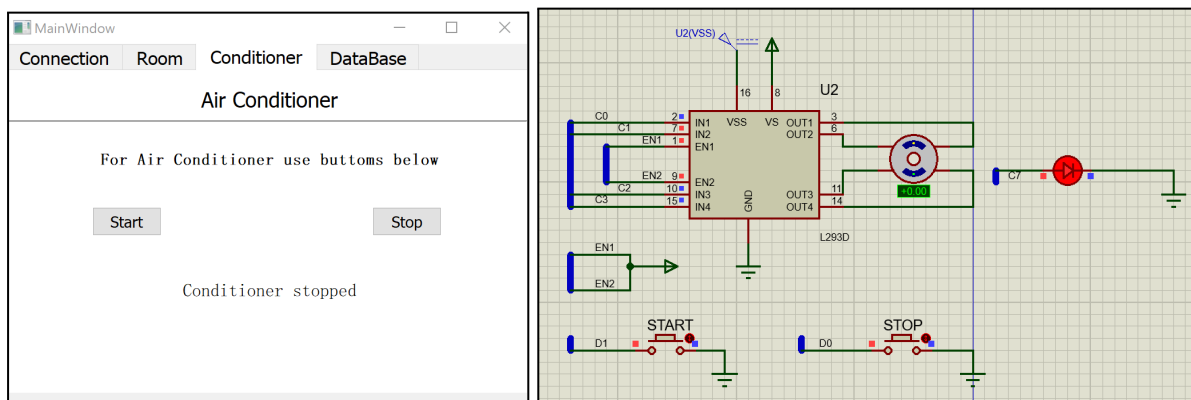


третя сторінка призначена, щоб запускати та зупиняти кондиціонер за  
допомогою кнопок **Start/Stop**

натиснемо **Start**, щоб запустити кондиціонер, та побачимо, що він працює

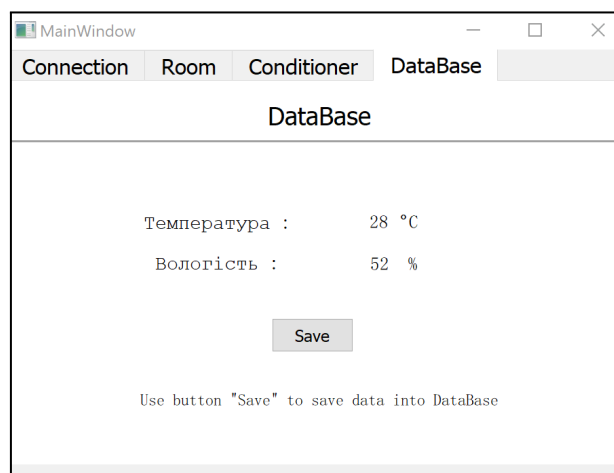


натиснемо **Stop**, щоб тепер його запустити



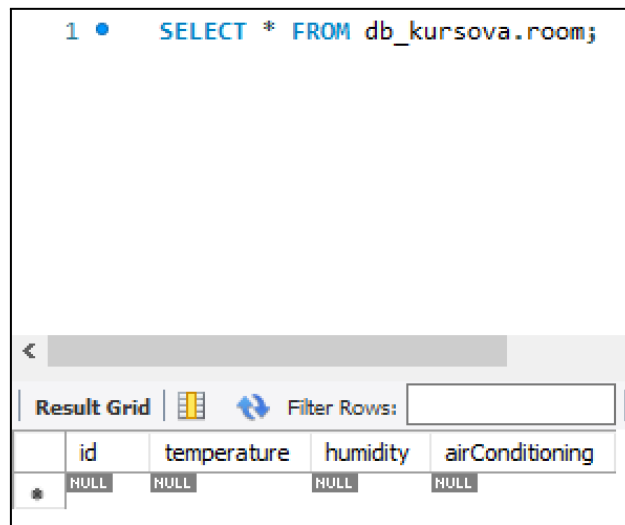
*червоний ледик засвітився, це означає, що кондиціонер не працює*

## 6. Четверта сторінка клієнтського інтерфейсу (DataBase)



четверта сторінка призначення для запису даних у базу даних

База даних без жодних записів



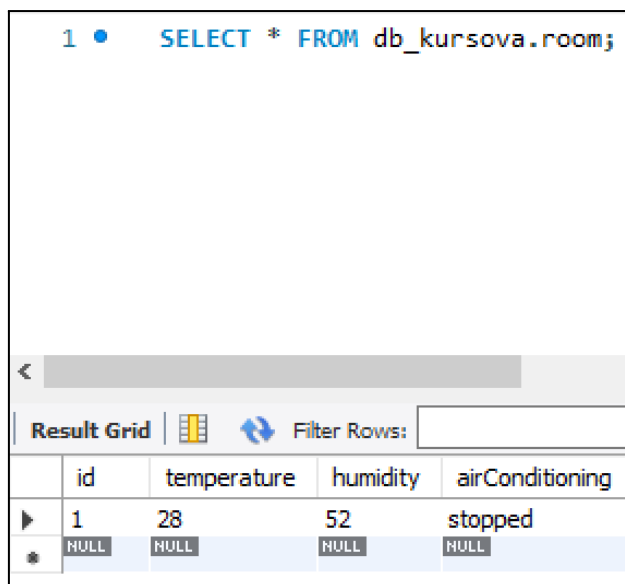
The screenshot shows a database query editor with the SQL statement `SELECT * FROM db_kursova.room;`. Below the query, there is a "Result Grid" section. The grid has a header row with columns: `id`, `temperature`, `humidity`, and `airConditioning`. The first data row shows all four columns with the value `NULL`. A scrollbar is visible at the top of the result grid.

|   | id   | temperature | humidity | airConditioning |
|---|------|-------------|----------|-----------------|
| * | NULL | NULL        | NULL     | NULL            |

Тепер натиснемо **“Save”**, щоб зберегти дані в базу даних

(Температура : 28 C

Вологість : 52 %)



The screenshot shows the same database query editor with the SQL statement `SELECT * FROM db_kursova.room;`. The "Result Grid" now displays a single record. The first data row has values: `1` for `id`, `28` for `temperature`, `52` for `humidity`, and `stopped` for `airConditioning`. The second data row (marked with an asterisk) shows `NULL` for all columns. A scrollbar is visible at the top of the result grid.

|   | id   | temperature | humidity | airConditioning |
|---|------|-------------|----------|-----------------|
| ▶ | 1    | 28          | 52       | stopped         |
| * | NULL | NULL        | NULL     | NULL            |

після натиснення кнопки **“Save”** дані успішно збереглися у базу даних

Посилання на git репозиторій: <https://github.com/DemkovychMaks/kursova>

## **Висновок**

Як висновок, виконуючи цю курсову проектну роботу, я навчився імплементувати GUI(графічний користувацький інтерфейс), який взаємодіють з мікроконтролером, обробляє дані отримані з нього та зберігати цю інформацію в базу даних. За допомогою цієї комплексної роботи, я зміг закріпити знання та навички, здобутих під час навчання, охоплюючи такі дисципліни як мікроконтролери, схемотехніку, бази даних та інші.

Також, хочу зазначити, що виконую цю курсову роботу, я добре навчився використовувати мікроконтролери та вивчив багато корисного, що можна використати для створення комфортних умов.

Завдяки цим навичкам, я зміг реалізувати систему контролю температури, що може бути багатьом людям потрібним у повсякденному житті.