

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN – ĐIỆN TỬ**

**BỘ MÔN VIỄN THÔNG**



**BÁO CÁO ĐỒ ÁN 1**

**THIẾT KẾ HỆ THỐNG GIÁM SÁT VÀ CẢNH BÁO CHẤT  
LƯỢNG KHÔNG KHÍ**

**GVHD: Thầy Võ Tuấn Kiệt**

**SVTH: Huỳnh Quang Thắng 2213190**

*Thành phố Hồ Chí Minh – tháng 6 năm 2025*

## **LỜI CẢM ƠN**

Trước tiên, em xin gửi lời cảm ơn chân thành đến giảng viên hướng dẫn em là thầy Võ Tuấn Kiệt hỗ trợ góp ý cho em để thực hiện đồ án này.

Em cũng xin cảm ơn các thầy cô trong khoa đã truyền đạt cho em những nền tảng kiến thức vững chắc trong suốt quá trình học tập tại trường. Nhờ đó, em mới có thể triển khai ý tưởng và hoàn thiện sản phẩm một cách cụ thể và có định hướng rõ ràng hơn.

Dự án lần này là cơ hội để em tiếp cận thực tế, áp dụng kiến thức về phân cứng, lập trình và xử lý dữ liệu vào một sản phẩm có tính ứng dụng. Dù trong quá trình thực hiện còn gặp nhiều khó khăn do có một số thứ là lần đầu em tiếp xúc và học để phục vụ cho đồ án, sản phẩm vẫn còn một số điểm hạn chế nhưng em cảm thấy rất vui vì đã hoàn thành đồ án và hi vọng rằng những trải nghiệm và bài học trong quá trình thực hiện đồ án này sẽ là nền tảng tốt cho em trong các dự án sau này.

**GVHD**

**Sinh viên**

## **TÓM TẮT ĐỒ ÁN**

Đồ án trình bày về một hệ thống IoT có chức năng giám sát và cảnh báo chất lượng không khí. Đồ án sử dụng cảm biến DHT22 để đo nhiệt độ, độ ẩm và MQ135 để đo nồng độ các chất khí trong không khí ( $\text{CO}$ ,  $\text{CO}_2$ ,  $\text{NH}_3$ ,...). Dữ liệu cảm biến trả về mỗi 5 giây sẽ được Esp32 hiển thị ra LCD đồng thời gửi tới server backend để lưu xuống database. Dữ liệu này cũng sẽ được server gửi lên giao diện người dùng và hiển thị dưới dạng biểu đồ để dễ dàng quan sát

## Mục lục

<b>CHƯƠNG I. GIỚI THIỆU .....</b>	<b>1</b>
1.1. Lý do chọn đề tài.....	1
1.2. Mục tiêu đề tài.....	1
1.3. Phạm vi thực hiện .....	1
1.4. Công cụ hỗ trợ.....	2
<b>CHƯƠNG II. TỔNG QUAN HỆ THỐNG.....</b>	<b>3</b>
2.1. Kiến trúc tổng thể hệ thống .....	3
2.2. Sơ đồ khối hệ thống.....	4
<b>CHƯƠNG III. CƠ SỞ LÝ THUYẾT.....</b>	<b>5</b>
3.1. ESP32 .....	5
3.2. LCD .....	6
3.3. Cảm biến .....	7
3.3.1. DHT22 .....	7
3.3.2. MQ135 .....	7
3.4. RESTful API.....	7
3.5. Giao thức HTTP và WebSocket .....	8
3.5.1. Giao thức HTTP.....	8
3.5.2. WebSocket .....	9
3.6. Cơ sở dữ liệu SQL Server.....	9
3.7. ReactJS và JSX .....	10
<b>CHƯƠNG IV. THIẾT KẾ .....</b>	<b>11</b>
4.1. Thiết kế phần cứng .....	11
4.1.1. Chuẩn hóa MQ135.....	11
4.1.2. Kết nối Esp32 với server backend .....	13
4.2. Thiết kế phần mềm .....	13

4.2.1. Server backend.....	13
4.2.2. Giao diện người dùng (frontend) .....	16
<b>CHƯƠNG V. TRIỂN KHAI VÀ KẾT QUẢ.....</b>	<b>19</b>
<b>5.1. Luồng hoạt động của toàn bộ hệ thống.....</b>	<b>19</b>
<b>CHƯƠNG VI: KẾT LUẬN.....</b>	<b>25</b>
<b>6.1. Kết quả đạt được.....</b>	<b>25</b>
<b>6.2. Nhận xét .....</b>	<b>25</b>
<b>CHƯƠNG VII: TÀI LIỆU THAM KHẢO .....</b>	<b>26</b>
<b>CHƯƠNG VII: PHỤ LỤC .....</b>	<b>27</b>

## DANH SÁCH HÌNH MINH HỌA

Hình 2.1: Sơ đồ khối hệ thống.....	4
Hình 3.1: Sơ đồ chân Esp32 .....	5
Hình 3.2: LCD 20x4 .....	6
Hình 3.3: Cảm biến nhiệt độ, độ ẩm DHT22.....	7
Hình 3.4: Cảm biến MQ135 .....	7
Hình 3.5: So sánh giữa HTTP và WebSocket.....	9
Hình 4.1: Biểu đồ nồng độ khí (ppm) theo tỷ lệ $\frac{R_S}{R_0}$ .....	11
Hình 4.2: Đoạn code để tính nồng độ khí dựa vào sự biến thiên của RS .....	12
Hình 4.3: Các file chính của server backend .....	13
Hình 4.4: Các file chính của giao diện người dùng (frontend).....	16
Hình 5.1: Khởi động database .....	22
Hình 5.2: Khởi động backend.....	22
Hình 5.3: Phân cứng .....	23
Hình 5.4: Data thu được.....	24
Hình 5.5: Khởi động frontend.....	24
Hình 5.6: Màn hình hiển thị lỗi.....	25
Hình 5.7: Màn hình loading.....	25
Hình 5.8: Load thành công .....	26
Hình 5.9: Bảng chọn phiên .....	26
Hình 5.10: Web hiển thị dữ liệu của phiên 33 .....	27
Hình 5.11: Thông số CO <sub>2</sub> của phiên 33 .....	27

# CHƯƠNG I. GIỚI THIỆU

## 1.1. Lý do chọn đề tài

Trong những năm gần đây, vấn đề ô nhiễm không khí ngày càng được quan tâm, đặc biệt tại các khu vực đô thị, khu công nghiệp và môi trường học đường. Chất lượng không khí suy giảm không chỉ ảnh hưởng trực tiếp đến sức khỏe con người mà còn tác động tiêu cực đến hệ sinh thái. Dù đã có các hệ thống giám sát, nhưng phần lớn đều có chi phí cao, khó tiếp cận với người dùng phổ thông và chưa thực sự đáp ứng nhu cầu theo dõi dữ liệu theo thời gian thực.

Nhờ sự phát triển mạnh mẽ của công nghệ Internet of Things (IoT), việc xây dựng một hệ thống giám sát chất lượng không khí với khả năng theo dõi liên tục, lưu trữ dữ liệu có tổ chức, hiển thị trực quan và chi phí thấp đã trở nên khả thi hơn. Xuất phát từ nhu cầu thực tiễn, em quyết định thực hiện đề tài “Hệ thống giám sát và cảnh báo chất lượng không khí” nhằm khắc phục những hạn chế hiện có, đồng thời mang lại một giải pháp hiệu quả và dễ triển khai trong thực tế.

## 1.2. Mục tiêu đề tài

Đề tài hướng đến việc xây dựng một hệ thống hoàn chỉnh gồm ba phần: thiết bị đo (ESP32 + cảm biến), server backend và giao diện người dùng (frontend). Cụ thể hơn như sau:

- Thu thập các dữ liệu về nồng độ khí (CO<sub>2</sub>, CO...), nhiệt độ, độ ẩm thông qua các cảm biến.
- Truyền dữ liệu từ thiết bị đến server backend theo thời gian thực.
- Lưu trữ dữ liệu vào database một cách có hệ thống nhằm quản lý tốt hơn
- Hiển thị dữ liệu cảm biến trực quan dưới dạng biểu đồ trên giao diện web.

## 1.3. Phạm vi thực hiện

Phần cứng: Sử dụng vi điều khiển ESP32 kết nối với các cảm biến môi trường như MQ135, DHT22 để đo khí độc, nhiệt độ, độ ẩm.

Phần mềm:

- Firmware trên ESP32 gửi dữ liệu qua HTTP hoặc WebSocket đến server.

- Backend sử dụng API để nhận, xử lý và lưu dữ liệu vào SQL Server.
- Frontend sử dụng ReactJS hiển thị dữ liệu theo thời gian thực và vẽ biểu đồ cảm biến.

#### **1.4. Công cụ hỗ trợ**

- Vscode với phần mở rộng PlatformIO để lập trình ESP32
- Webserver Uvicorn và Restful API
- SQL Server để quản lý database
- AI (ChatGPT) để giúp viết code frontend



## CHƯƠNG II. TỔNG QUAN HỆ THỐNG

### 2.1. Kiến trúc tổng thể hệ thống

Hệ thống giám sát và cảnh báo chất lượng không khí được xây dựng với 3 khối chính:

- Phần cứng: vi xử lý ESP32, LCD, nút bấm, mạch nguồn và các cảm biến
- Server backend (FastAPI và SQL Server)
- Giao diện người dùng (frontend) để hiển thị trực quan các giá trị đo được

#### Phần cứng

- Vi xử lý ESP32 có vai trò thu thập dữ liệu từ các cảm biến: DHT22 (độ ẩm, nhiệt độ), MQ135 (nồng độ các khí CO<sub>2</sub>, CO, NH<sub>3</sub>, Toluen và C<sub>6</sub>H<sub>6</sub>), đồng thời hiển thị dữ liệu nồng độ các khí trên LCD, có thể bấm nút để chuyển sang hiển thị loại khí khác
- Dữ liệu thu được sẽ được ESP32 gửi đến server backend thông qua giao thức HTTP
- Dữ liệu sẽ được gửi kèm với thời gian đo và id của phiên đo (session\_id)

#### Server backend (REST API và SQL Server)

- Ở đây server backend đóng vai trò trung gian giữa phần cứng và giao diện người dùng (frontend)
- Sử dụng framework FAST API với ngôn ngữ là python, API sau khi nhận được dữ liệu thông qua giao thức HTTP thì sẽ lưu dữ liệu vào database quản lý bởi SQL Server
- Giữa server backend và frontend sẽ có WebSocket để giúp dữ liệu được cập nhật theo thời gian thực

#### Database (SQL Server)

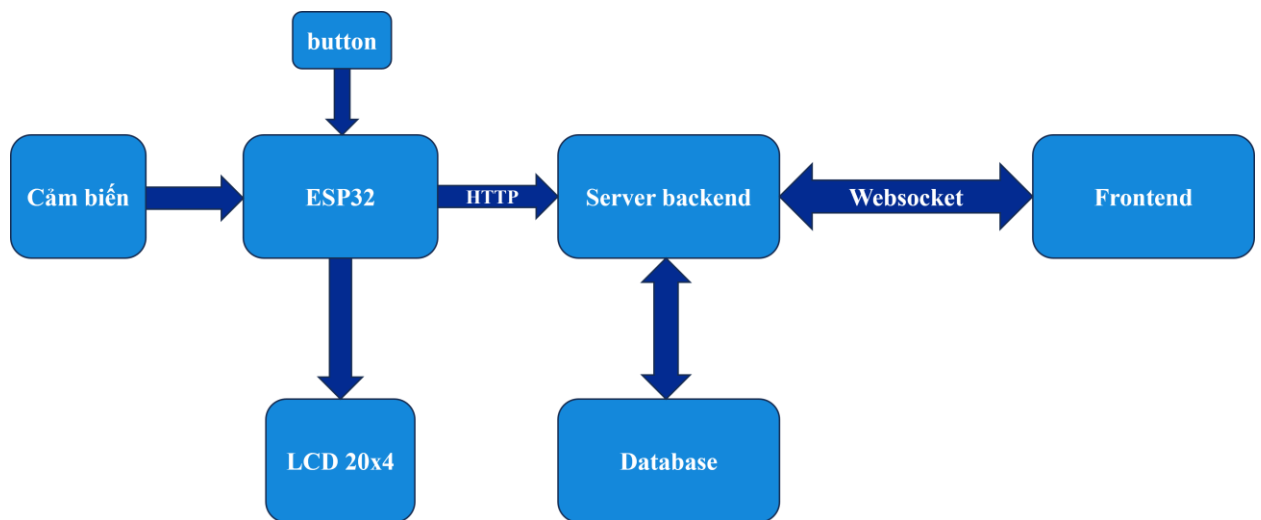
Dữ liệu được gửi với 2 thành phần chính:

- Session: dùng để lưu thông tin về các phiên đo (id, thời gian đo)
- SensorData: dùng để lưu giá trị cảm biến và thời gian đo

#### Giao diện người dùng (Frontend)

- Được xây dựng dựa trên framework ReactJS viết bằng JSX
- Hiển thị các giá trị đo dưới dạng thông số và biểu đồ
- Giao tiếp với server thông qua WebSocket để đảm bảo tính real-time

## 2.2. Sơ đồ khối hệ thống



Hình 2.1: Sơ đồ khối hệ thống

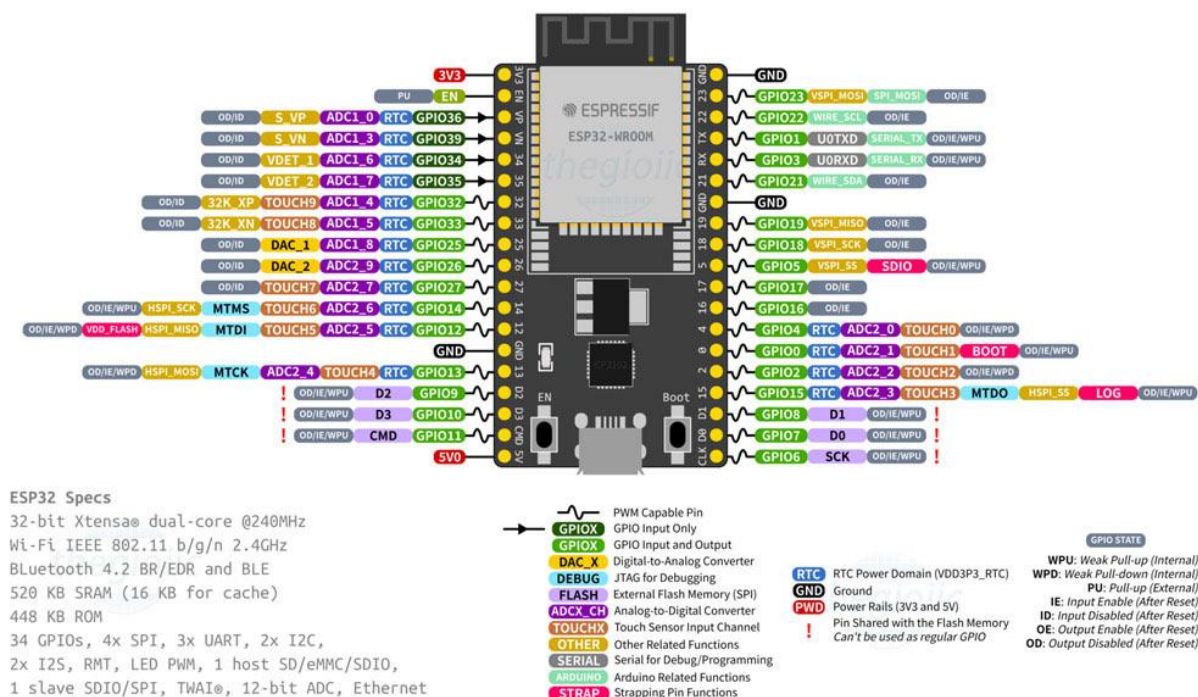
## CHƯƠNG III. CƠ SỞ LÝ THUYẾT

### 3.1. ESP32

Vi điều khiển ESP32-DevKitC-32D Module WiFi Bluetooth 2.4GHz là một vi điều khiển có tích hợp wifi và bluetooth. Với các đặc điểm là chi phí thấp, kích thước nhỏ gọn, có các chân ngoại vi đưa ra ngoài giúp dễ dàng giao tiếp với nhiều kiểu giao tiếp được hỗ trợ như ADC, I<sup>2</sup>C, SPI, UART. ESP32 rất phù hợp cho đề án này.

Thông số kỹ thuật:

- Vi xử lý: ESP32 – WROOM – 32D
- Core: ESP32 – D0WD
- Bộ nhớ flash: 4MB
- Thạch anh: 40 MHz
- Chuẩn wifi: 802.11 b/g/n
- Tần số hoạt động: 2.4 GHz
- Chuẩn bluetooth: Bluetooth v4.2
- Giao tiếp hỗ trợ: ADC, I<sup>2</sup>C, SPI, UART, GPIO



Hình 3.1: Sơ đồ chân Esp32

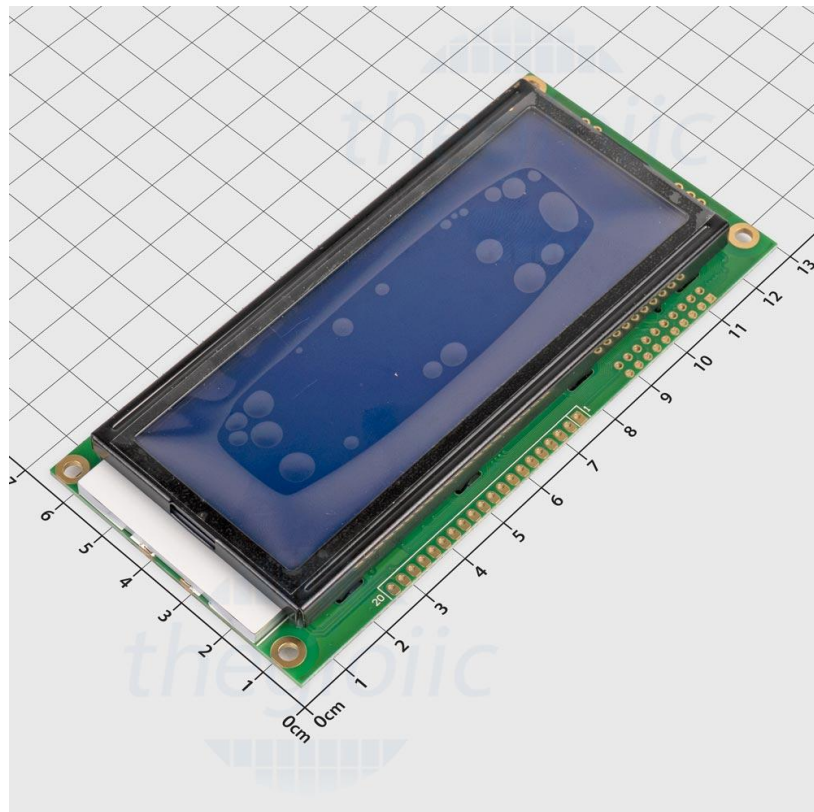
Trong đồ án này thì ESP32 sẽ có vai trò thu thập dữ liệu từ cảm biến (DHT22, MQ135). Sau đó hiển thị thông số lên LCD đồng thời gửi tới server bằng wifi thông qua phương thức HTTP.

### 3.2. LCD

LCD 20x4 nền xanh dương chữ trắng kèm theo I2C driver

Thông số kỹ thuật:

- Giao tiếp: I2C
- Địa chỉ I2C: 0x27
- SDA (DATA) - đầu vào Analog chân 4
- SCL (CLOCK) - đầu vào Analog chân 5
- Điện áp cung cấp: 5VDC
- Kích thước Pcb: 60mm × 99mm



Hình 3.2: LCD 20x4

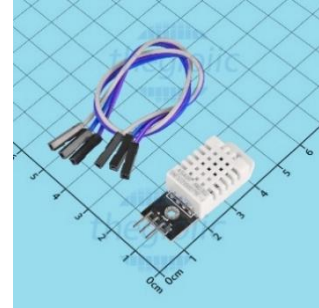
### 3.3. Cảm biến

#### 3.3.1. DHT22

Cảm biến DHT22 là một loại cảm biến đo nhiệt độ và độ ẩm

Thông số kỹ thuật:

- Điện áp làm việc: 3 ~ 5.5 VDC
- Dòng tiêu thụ: tối đa 2.5mA
- Phạm vi đo nhiệt độ: -40°C tới 80°C, độ chính xác  $\pm 0.5^\circ\text{C}$
- Phạm vi đo độ ẩm: 0% tới 100% RH (Relative Humidity), độ chính xác  $\pm 2\%$  RH
- Tần số lấy mẫu: 0.5Hz



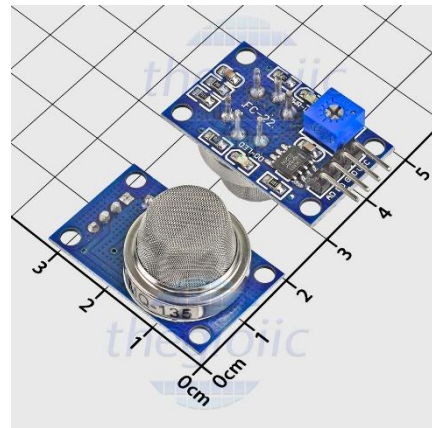
Hình 3.3: Cảm biến nhiệt độ độ ẩm DHT22

#### 3.3.2. MQ135

MQ135 là một cảm biến khí dùng để đo chất lượng không khí, phản ứng với các loại khí nhất định như CO<sub>2</sub>, CO, NH<sub>3</sub>, Benzen, ...

Thông số kỹ thuật:

- Điện áp làm việc: 5V DC
- Điện trở tải: 36k $\Omega$
- Ngõ ra DO: 0.1V / 5V
- Ngõ ra AO: 0.1~0.3V



Hình 3.4: Cảm biến MQ135

### 3.4. RESTful API

API (Application Programming Interface – giao diện lập trình ứng dụng) là các phương thức, giao thức kết nối với các thư viện và ứng dụng khác. API là một trung gian

phần mềm cho phép hai ứng dụng giao tiếp với nhau. API ở đây sẽ được xây dựng trên FAST API – một framework để xây dựng API bằng python

REST (REpresentational State Transfer) là một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy. Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE, ... đến một URL để xử lý dữ liệu.

Các nguyên tắc cốt lõi của REST

- Client-Server: Phân tách giao diện người dùng (client) và xử lý dữ liệu (server). Giúp phát triển độc lập và mở rộng dễ hơn
- Stateless: Mỗi yêu cầu từ client đến server phải chứa đầy đủ thông tin để xử lý. Server không lưu trạng thái giữa các lần request
- Cacheable: Phản hồi từ server có thể được cache để cải thiện hiệu suất.
- Uniform Interface: Giao tiếp qua API phải thống nhất – mọi tài nguyên được truy cập theo một chuẩn chung (URL, phương thức HTTP, mã trạng thái...).
- Layered System: Hệ thống có thể gồm nhiều tầng trung gian (proxy, gateway), client không cần biết chi tiết cấu trúc bên trong.
- REST thường gắn với các thao tác CRUD thông qua các phương thức HTTP chuẩn: Create (POST), Read (GET), Update (PUT) và Delete (DELETE)

RESTful API là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web, được xây dựng dựa trên nguyên lý REST, RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile...) khác nhau giao tiếp với nhau.

### **3.5. Giao thức HTTP và Websocket**

#### **3.5.1. Giao thức HTTP**

HTTP (HyperText Transfer Protocol – Giao thức truyền tải siêu văn bản) là một giao thức thuộc lớp Application, nó là nền tảng để liên lạc và trao đổi dữ liệu với World Wide Web, hoạt động trên nền tảng TCP/IP và thường được truyền qua kết nối mã hóa TLS để bảo vệ dữ liệu

HTTP là một giao thức Request – Response được xây dựng dựa trên cơ sở cấu trúc Client – Server. Các Request và Response là các message được gửi bởi client và server.

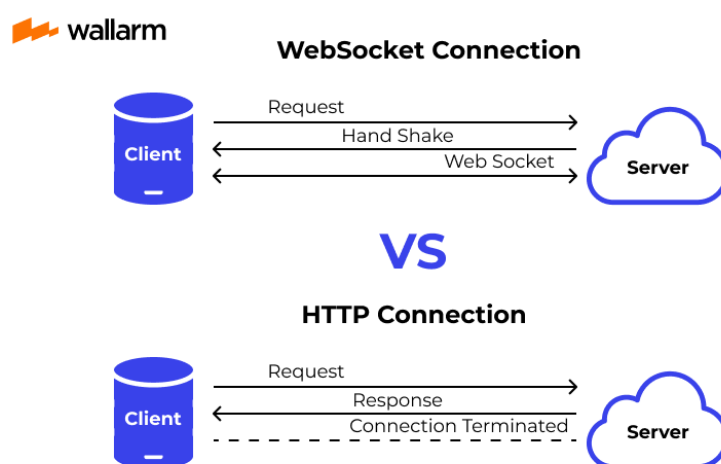
Client có thể là một web được người truy cập yêu cầu, còn Server có thể là một máy chủ web trên đám mây hoặc lưu trữ cục bộ trên một chiếc máy tính. Điều này cho phép chúng ta truy cập vào các trang web, tải xuống tập tin, xem video, hình ảnh, ...

### 3.5.2. WebSocket

Dữ liệu được truyền tải thông qua HTTP chứa quá nhiều dữ liệu không cần thiết trong phần header. Một header request/response của HTTP có kích thước khoảng 871 byte và chỉ riêng phần header này thôi đã chiếm một phần lưu lượng đáng kể.

Để tiết kiệm lưu lượng, ta sử dụng WebSocket. WebSocket là một giao thức truyền thông cho phép giao tiếp hai chiều (full-duplex) giữa client và server thông qua một kết nối TCP duy nhất. Không giống như HTTP chỉ có thể kết nối một chiều (client gửi – server trả lời), mỗi lần gửi là một lần kết nối với và chỉ thích hợp với các trang web thông thường.

Khi hoạt động, đầu tiên client sẽ gửi yêu cầu khởi tạo kết nối WebSocket đến server thông qua liên kết HTTP (WebSocket handshake request – Giao thức bắt tay). Khi server chấp nhận yêu cầu thì sẽ chuyển sang giao thức WebSocket. Khi đó WebSocket giữ kết nối mở liên tục, cho phép server gửi dữ liệu chủ động đến client bất cứ lúc nào (real-time), mà không cần client phải gửi gói tin yêu cầu đến server.



Hình 3.5: So sánh giữa HTTP và WebSocket

## 3.6. Cơ sở dữ liệu SQL Server

Microsoft SQL Server là một hệ quản trị cơ sở dữ liệu quan hệ được phát triển bởi Microsoft. Là một máy chủ cơ sở dữ liệu, nó là một sản phẩm phần mềm có chức năng cho phép người dùng truy vấn, thao tác và quản lý dữ liệu một cách hiệu quả và an toàn theo

yêu cầu của các ứng dụng phần mềm khác. Có thể chạy trên cùng một máy tính hoặc trên một máy tính khác trên mạng (bao gồm cả Internet).

Cơ sở dữ liệu quan hệ là một mô hình cơ sở dữ liệu trong đó dữ liệu được tổ chức thành các bảng (table), và các bảng này có mối quan hệ với nhau thông qua khoá chính (primary key) và khoá phụ (foreign key).

Cấu trúc: gồm các table, mỗi table sẽ có các khóa chính (primary key) và khóa phụ (foreign key)

### **3.7. ReactJS và JSX**

ReactJS là một thư viện JavaScript mã nguồn mở được dùng để xây dựng giao diện người dùng (UI) cho website. ReactJS là một thư viện dựa trên components, khai báo cho phép lập trình viên xây dựng UI component có thể tái sử dụng và tuân theo phương pháp Virtual DOM. Điều này giúp tối ưu hóa hiệu suất render bằng cách giảm thiểu các bản cập nhật DOM.

Tuy nhiên, việc sử dụng JavaScript (JS) thuần có phần dài dòng và khó hiểu. Thay vào đó ta sử dụng JSX. JSX là sự kết hợp của JS và XML, nó transform cú pháp về dạng gần giống như XML thay vì phải sử dụng JS. Việc sử dụng JSX giúp dễ dàng định nghĩa, quản lý các component phức tạp, mang lại một cấu trúc dễ để đọc hiểu hơn.

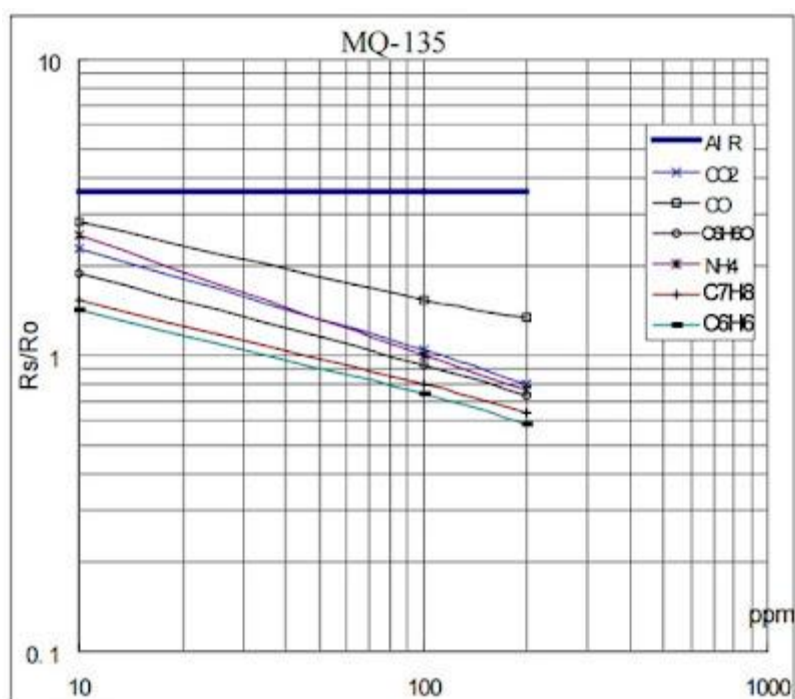


## CHƯƠNG IV. THIẾT KẾ

### 4.1. Thiết kế phần cứng

#### 4.1.1. Chuẩn hóa MQ135

Để sử dụng cảm biến MQ135 đo nồng độ khí thì trước tiên phải chuẩn hóa. Cảm biến có một điện trở biến thiên  $R_S$  sẽ thay đổi giá trị điện trở dựa vào nồng độ khí, nếu nồng độ khí cao thì điện trở giảm và ngược lại thì điện trở tăng. Điện trở  $R_S$  trong điều kiện không khí trong lành (100ppm  $\text{NH}_3$ , nhiệt độ  $20^\circ\text{C}$  và độ ẩm 65%) gọi là  $R_0$ , tuy nhiên việc đạt được môi trường với thông số đó tương đối khó.



Hình 4.1: Biểu đồ nồng độ khí (ppm) theo tỷ lệ  $\frac{R_S}{R_0}$

Trước tiên, sử dụng biểu đồ biểu thị nồng độ khí (ppm) theo tỷ lệ  $\frac{R_S}{R_0}$ , suy ra được

$$\log\left(\frac{R_S}{R_0}\right) = \log \alpha + \beta \log(ppm) \quad (1)$$

Ví dụ với  $\text{CO}_2$ , em chọn 2 điểm. Điểm thứ nhất (ppm = 10,  $\frac{R_S}{R_0} = 2.2$ ), điểm thứ 2 (ppm = 200,  $\frac{R_S}{R_0} = 0.8$ )

$$\text{Ta có } \beta = \frac{\log \frac{2.2}{0.8}}{\log \frac{10}{200}} = -0.339$$

Thay  $\beta$  vào để tìm  $\alpha$

$$\text{Từ phương trình (1)} \Rightarrow \frac{R_S}{R_0} = \alpha \times ppm^\beta \quad (2)$$

$$\Rightarrow \alpha = 4.787$$

$$\text{Từ (2)} \Rightarrow ppm_{CO_2} = \left( \frac{R_S}{R_0 \times \alpha} \right)^{\frac{1}{\beta}} = A \times \frac{R_S^B}{R_0} \text{ với}$$

$$A = 103.286332$$

$$B = -2.961377594$$

Từ đây có được công thức chung để tính ppm theo tỷ lệ  $\frac{R_S}{R_0}$  là

$$ppm = A \times \left( \frac{R_S}{R_0} \right)^B \quad (\text{với } A, B \text{ thay đổi theo từng loại khí}) \quad (3)$$

Theo Statista.com, vào tháng 4/2025 nồng độ khí CO<sub>2</sub> là 429.64ppm. Thay giá trị này cùng với A, B đã tính được ở trên vào (3) có thể tính ra tỷ lệ  $\frac{R_S}{R_0}$  trong môi trường bình thường là 0.6179480415

Sử dụng một đoạn code nhỏ để tính ra  $R_0$  trong môi trường bình thường

$$R_S = 120.05K$$

$$R_0 = 194.27K$$

Từ đó thu thập được giá trị  $R_0 = 194.27$  và thay vào đoạn code chính

Lúc này, việc tính toán nồng độ khí sẽ dựa trên sự biến thiên của  $R_S$ , còn  $R_0$  và các hệ số A, B sẽ được tính toán cố định

```
float readGasPPM(float temp, float humi, GasType gasType)
{
    float RS = mq135_sensor.getResistance();

    float ratio = RS / R0;

    return gasCoefficients[gasType].a * pow(ratio, gasCoefficients[gasType].b);
}
```

Hình 4.2: Đoạn code để tính nồng độ khí dựa vào sự biến thiên của  $R_S$

### 4.1.2. Kết nối Esp32 với server backend

Khi cảm biến thu thập được data sẽ truyền qua cho server backend thông qua phương thức HTTP bằng hàm sendDataToServer như sau:

```
void sendDataToServer(float temp, float humi, GasType displayGasType, int sessionId)
{
    HTTPClient http;
    http.begin(serverUrl);
    http.addHeader("Content-Type", "application/json");
    http.setTimeout(10000);
```

```
void createSession()
{
    HTTPClient http;
    http.begin(serverSessionUrl);
    http.addHeader("Content-Type", "application/json");
    http.setTimeout(10000);
```

Với các đường dẫn Url dẫn tới API của backend

```
const char *serverUrl = "http://192.168.1.8:8000/api/sensor-data";
const char *serverSessionUrl = "http://192.168.1.8:8000/session/";
```

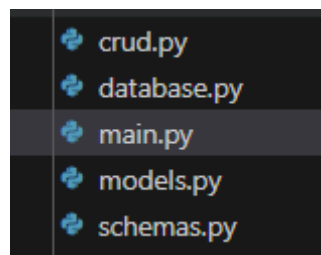
Khi đó data do Esp32 gửi sẽ được API xử lý để lưu xuống database

## 4.2. Thiết kế phần mềm

### 4.2.1. Server backend

Đầu tiên ta xây dựng server backend bằng ngôn ngữ python trong đó sử dụng RESTful API và WebSocket để giao tiếp với frontend

Tổng quan backend gồm 5 file chính:



Hình 4.3: Các file chính của server backend

## database.py

Cấu hình để kết nối với database quản lý bởi SQL Server, file này chỉ có tác dụng kết nối với database thông qua SQLALCHEMY\_DATABASE\_URL

```
# cấu hình kết nối sql
SQLALCHEMY_DATABASE_URL =
"mssql+pyodbc://sa:12346@localhost\\SQLEXPRESS/DOAN1?driver=ODBC+Driver+17+for+SQL+Server"
```

```
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

dùng để tạo session

## models.py

Sau khi kết nối với database, dùng models.py để mô phỏng lại database, sau đó migrate qua database bằng alembic

Về cơ bản, đây là file tượng trưng cho database, từng class trong file này tương ứng với từng bảng trong database

Database ở đây có 2 bảng

- Bảng **session** với khóa chính là **id** dùng để chia data thành từng phiên, mỗi khi ta cho cảm biến đo thì một phiên mới sẽ được thiết lập, mỗi phiên sẽ chứa nhiều data do cảm biến trả về
- Bảng data dùng để lưu giá trị cảm biến trả về và thời gian và id tương ứng

## schemas.py

Đóng vai trò như một file thuộc dạng Data Transfer Object, đây là nơi xử lý để nhận và gửi dữ liệu, đảm bảo dữ liệu được gửi lên API đúng định dạng.

Khi có một request, schemas sẽ nhận file JSON được gửi về từ request đó, sau đó chuyển data thành file JSON để response lại.

File schemas bao gồm các class:

- **DataCreate:** dùng để ESP32 gửi dữ liệu cảm biến lên server qua API với các trường doam, nhietdo, ...
- **Data:** kế thừa từ Datacreate và thêm 2 trường là id và created\_at
- **EmptyPayload và Session:** do API POST theo chuẩn cần payload nên EmptyPayload đóng vai trò như một payload giả, còn session dùng để lưu thông tin về các phiên

### crud.py

File này chứa các hàm CRUD để xử lý logic các data đến và đi

Hàm **get\_product** dùng để lấy toàn bộ dữ liệu từ một phiên cụ thể

Hàm **get\_lastest\_tempurate** dùng để lấy dữ liệu mới nhất của bảng Data

Hàm **get\_all\_session** để lấy danh sách của tất cả các session rồi trả về id và thời gian tạo session đó

Hàm **get\_latest\_session** trả về id của phiên mới nhất, dùng để kiểm tra phiên hiện tại trong WebSocket

### main.py

Là file điều phối chính của backend, bao gồm các cấu hình của server đồng thời quản lý các API và giao tiếp websocket với frontend

- **Endpoint của websocket**

Sử dụng các biến last\_data và last\_session để tránh gửi dữ liệu trùng lặp, mỗi 5 giây websocket sẽ kiểm tra 1 lần xem có dữ liệu mới không

```
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket, db: Session =
Depends(database.get_db)):
    await websocket.accept()
    last_data = None
    last_session = None
```

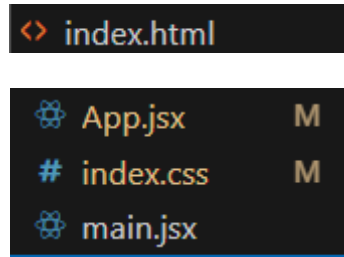
- **Endpoint của REST API**

Các endpoint giúp frontend sử dụng để lấy data mới từ cảm biến, tạo session mới, lấy thông tin về các session, ...

```
@app.get("/data/temperature/latest", response_model=schemas.Data)
def read_latest_temperature(db: Session = Depends(database.get_db)):
    return crud.get_latest_temperature(db)
```

#### 4.2.2. Giao diện người dùng (frontend)

Gồm 4 file chính



Hình 4.4: Các file chính của giao diện người dùng (frontend)

##### **index.html**

Đây là file mà trình duyệt sẽ tải đầu tiên, ứng dụng React chỉ tải duy nhất file html này, sau đó mọi thứ còn lại được vẽ bằng JavaScript

##### **main.jsx**

Đây là nơi khởi động ứng dụng React, render App.jsx vào DOM

```
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

##### **index.css**

Cấu hình cho toàn bộ giao diện như font chữ, căn dòng, độ đậm nhạt của chữ, màu chữ, màu nền. Ở đồ án này vì đã thiết lập giao diện chủ yếu trong file App.jsx, nên ở file index này chỉ có tác dụng cấu hình giao diện khi không tải được dữ liệu phiên.

##### **App.jsx**

Đây là file giao diện chính của frontend, là nơi truy cập vào các API của server backend để lấy data và đồng thời là nơi thiết lập giao diện chính cho các phần tử hiển thị (danh sách chọn phiên, dữ liệu nhiệt độ, độ ẩm và các biểu đồ nồng độ khí)

- **Hàm useEffect:** có tác dụng truy cập vào API để lấy dữ liệu phiên, data từ cảm biến, bên cạnh đó còn được dùng để liên kết websocket

Ví dụ hàm lấy danh sách phiên:

```
useEffect(() => {
  const fetchSessions = async () => {
    try { const response = await axios.get(`${apiUrl}/session`);
      setSessions(response.data);
      setLoading(false);
    } catch (err) {
      setError("Không thể tải danh sách phiên");
      console.error("Lỗi khi gọi API:", err);
      setLoading(false);
    }
  };
});
```

Trong đó **apiUrl** được import từ file **.env** thông qua

```
const apiUrl = import.meta.env.VITE_API_URL;
```

với file **.env** để thay đổi địa chỉ Ipv4 khi truy cập vào mạng khác

```
reactjs_doan1 > doan1 > .env
1  VITE_API_URL=http://192.168.1.8:8000
2  VITE_WEBSOCKET_URL=ws://192.168.1.8:8000/ws
3
```

Đoạn code liên kết websocket

```
useEffect(() => {
  const wsUrl = import.meta.env.VITE_WEBSOCKET_URL;
  const ws = new WebSocket(wsUrl);
```

- **Lọc dữ liệu để hiển thị**

Vì ở phần cứng được thiết lập để đo nồng độ khí 5s một lần, nếu lấy toàn bộ data đó lên biểu đồ thì sẽ rất nhiều, nên ở đây sẽ xử lý logic để cách 10 data thì mới hiển thị một lần, nếu số data không chia hết cho 10 thì sẽ lấy data cuối cùng. Điều này giúp biểu đồ hiển thị gọn hơn.

Phần code còn lại của App.jsx sẽ dùng để cấu hình các thông tin hiển thị trên giao diện bao gồm:

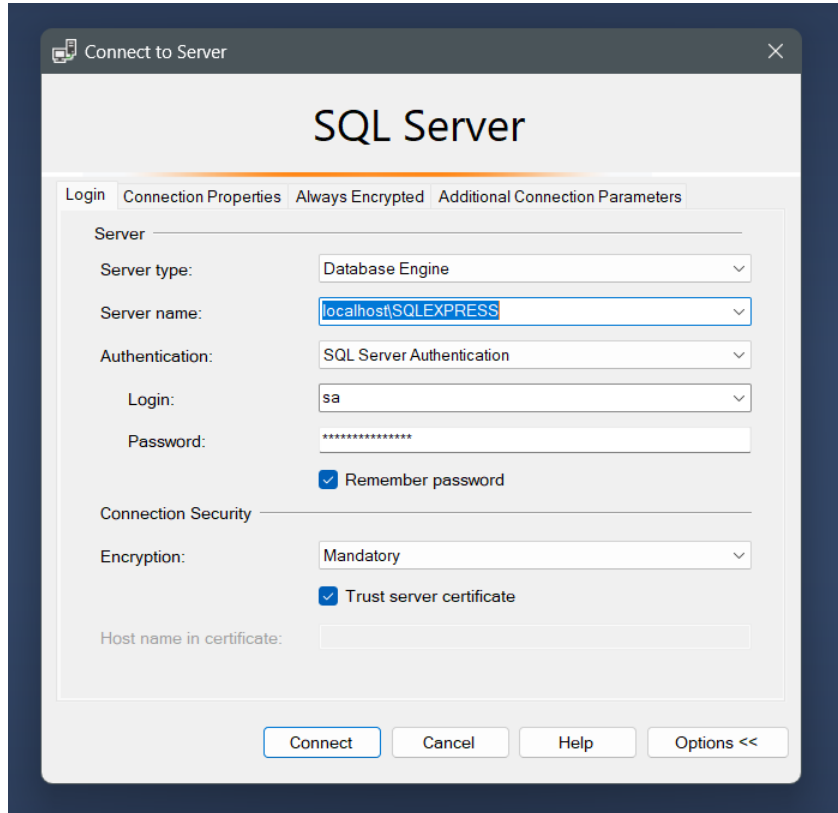
- Dropdown chọn phiên: hiển thị id phiên và thời gian tạo phiên
- Nhiệt độ
- Độ ẩm: biểu diễn theo dạng biểu đồ tròn
- Dữ liệu về nồng độ khí từ các cảm biến: hiển thị theo dạng biểu đồ đường (trục y đơn vị ppm, trục x đơn vị thời gian)



## CHƯƠNG V. TRIỂN KHAI VÀ KẾT QUẢ

### 5.1. Luồng hoạt động của toàn bộ hệ thống

Đầu tiên khởi chạy database



Hình 5.1: Khởi động database

Sau đó khởi chạy file main.py bằng cách khởi chạy môi trường ảo và cho phép server lắng nghe từ mọi IP thông qua host 0.0.0.0

```
PS D:\vscode\do an 1 (web)> venv\Scripts/activate
(venv) PS D:\vscode\do an 1 (web)> uvicorn main:app --host 0.0.0.0 --port 8000
D:\vscode\do an 1 (web)\venv\Lib\site-packages\pydantic\_internal\_config.py:373: UserWarning: Valid config keys have changed in V2:
* 'orm_mode' has been renamed to 'from_attributes'
  warnings.warn(message, UserWarning)
INFO: Started server process [3452]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

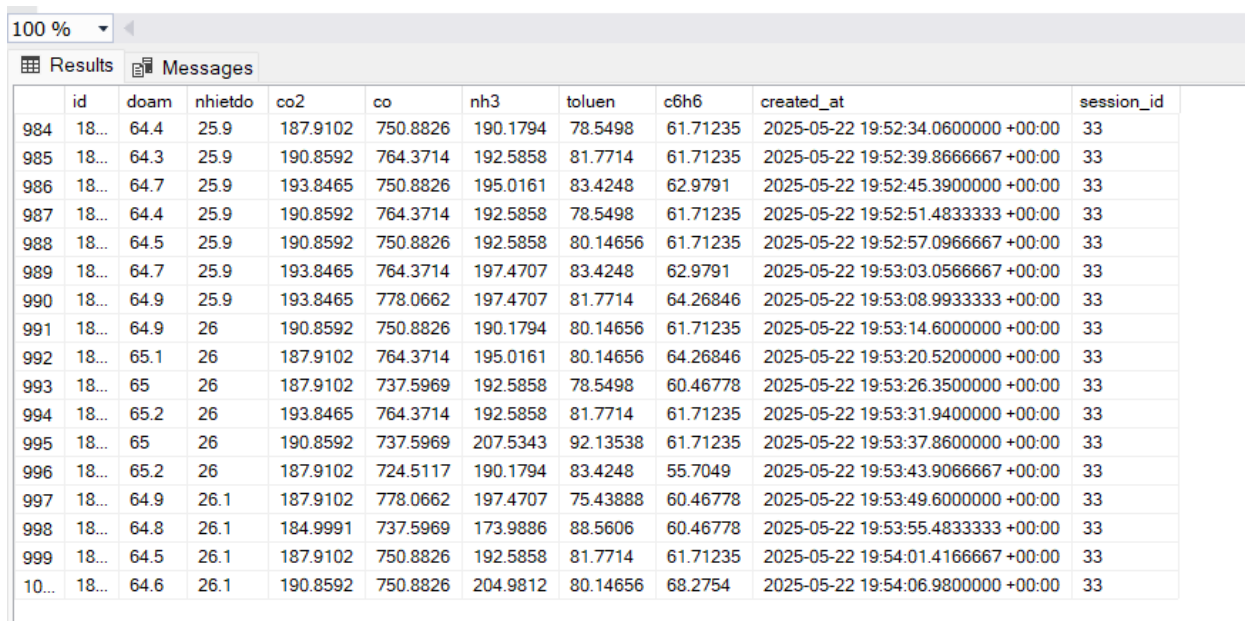
Hình 5.2: Khởi động backend

Tiếp theo khởi động phần cứng (trước khi khởi động phải thay đổi lại địa chỉ Ipv4 trong trường hợp thay đổi mạng wifi)



Hình 5.3: Phần cứng

Lúc này dữ liệu cảm biến đo được sẽ được gửi tới server sau đó gửi xuống database thông qua API

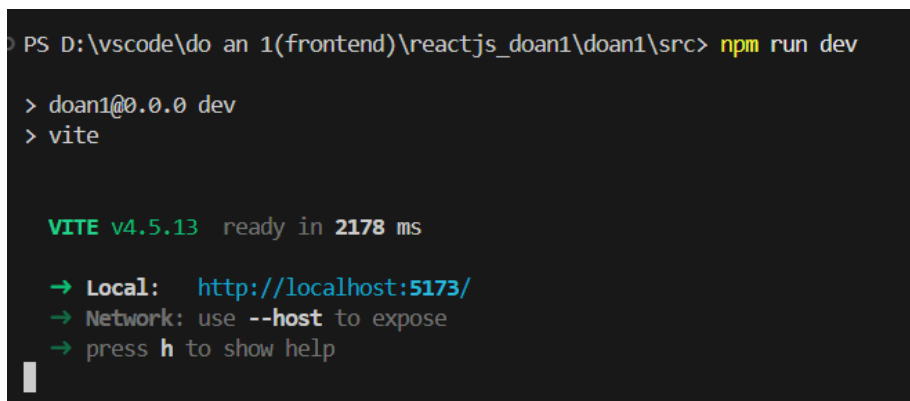


The screenshot shows a database interface with a 'Results' tab. It displays a table with 11 columns: id, doam, nhietdo, co2, co, nh3, toluen, c6h6, created\_at, and session\_id. The table contains 20 rows of data, with the first row starting at id 984 and the last row at id 1000. The data represents various sensor readings over time, all associated with session\_id 33.

	id	doam	nhietdo	co2	co	nh3	toluen	c6h6	created_at	session_id
984	18...	64.4	25.9	187.9102	750.8826	190.1794	78.5498	61.71235	2025-05-22 19:52:34.0600000 +00:00	33
985	18...	64.3	25.9	190.8592	764.3714	192.5858	81.7714	61.71235	2025-05-22 19:52:39.8666667 +00:00	33
986	18...	64.7	25.9	193.8465	750.8826	195.0161	83.4248	62.9791	2025-05-22 19:52:45.3900000 +00:00	33
987	18...	64.4	25.9	190.8592	764.3714	192.5858	78.5498	61.71235	2025-05-22 19:52:51.4833333 +00:00	33
988	18...	64.5	25.9	190.8592	750.8826	192.5858	80.14656	61.71235	2025-05-22 19:52:57.0966667 +00:00	33
989	18...	64.7	25.9	193.8465	764.3714	197.4707	83.4248	62.9791	2025-05-22 19:53:03.0566667 +00:00	33
990	18...	64.9	25.9	193.8465	778.0662	197.4707	81.7714	64.26846	2025-05-22 19:53:08.9933333 +00:00	33
991	18...	64.9	26	190.8592	750.8826	190.1794	80.14656	61.71235	2025-05-22 19:53:14.6000000 +00:00	33
992	18...	65.1	26	187.9102	764.3714	195.0161	80.14656	64.26846	2025-05-22 19:53:20.5200000 +00:00	33
993	18...	65	26	187.9102	737.5969	192.5858	78.5498	60.46778	2025-05-22 19:53:26.3500000 +00:00	33
994	18...	65.2	26	193.8465	764.3714	192.5858	81.7714	61.71235	2025-05-22 19:53:31.9400000 +00:00	33
995	18...	65	26	190.8592	737.5969	207.5343	92.13538	61.71235	2025-05-22 19:53:37.8600000 +00:00	33
996	18...	65.2	26	187.9102	724.5117	190.1794	83.4248	55.7049	2025-05-22 19:53:43.9066667 +00:00	33
997	18...	64.9	26.1	187.9102	778.0662	197.4707	75.43888	60.46778	2025-05-22 19:53:49.6000000 +00:00	33
998	18...	64.8	26.1	184.9991	737.5969	173.9886	88.5606	60.46778	2025-05-22 19:53:55.4833333 +00:00	33
999	18...	64.5	26.1	187.9102	750.8826	192.5858	81.7714	61.71235	2025-05-22 19:54:01.4166667 +00:00	33
10...	18...	64.6	26.1	190.8592	750.8826	204.9812	80.14656	68.2754	2025-05-22 19:54:06.9800000 +00:00	33

Hình 5.4: Data thu được

Tiếp theo khởi chạy frontend



```

PS D:\vscode\do an 1(frontend)\reactjs_doan1\doan1\src> npm run dev

> doan1@0.0.0 dev
> vite

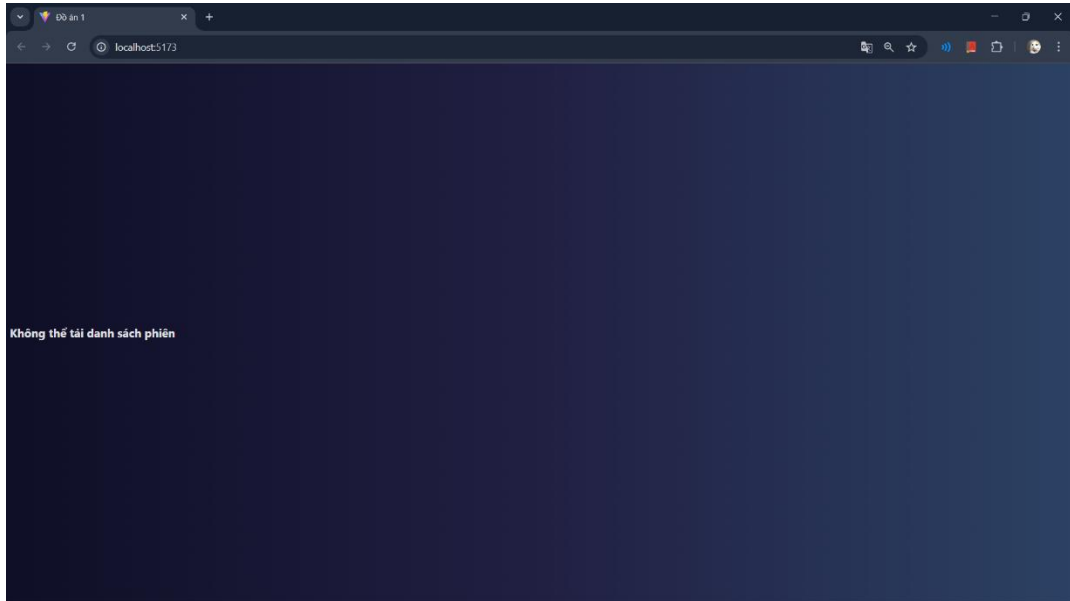
VITE v4.5.13 ready in 2178 ms

→ Local:   http://localhost:5173/
→ Network:  use --host to expose
→ press h  to show help
  
```

Hình 5.5: Khởi động frontend

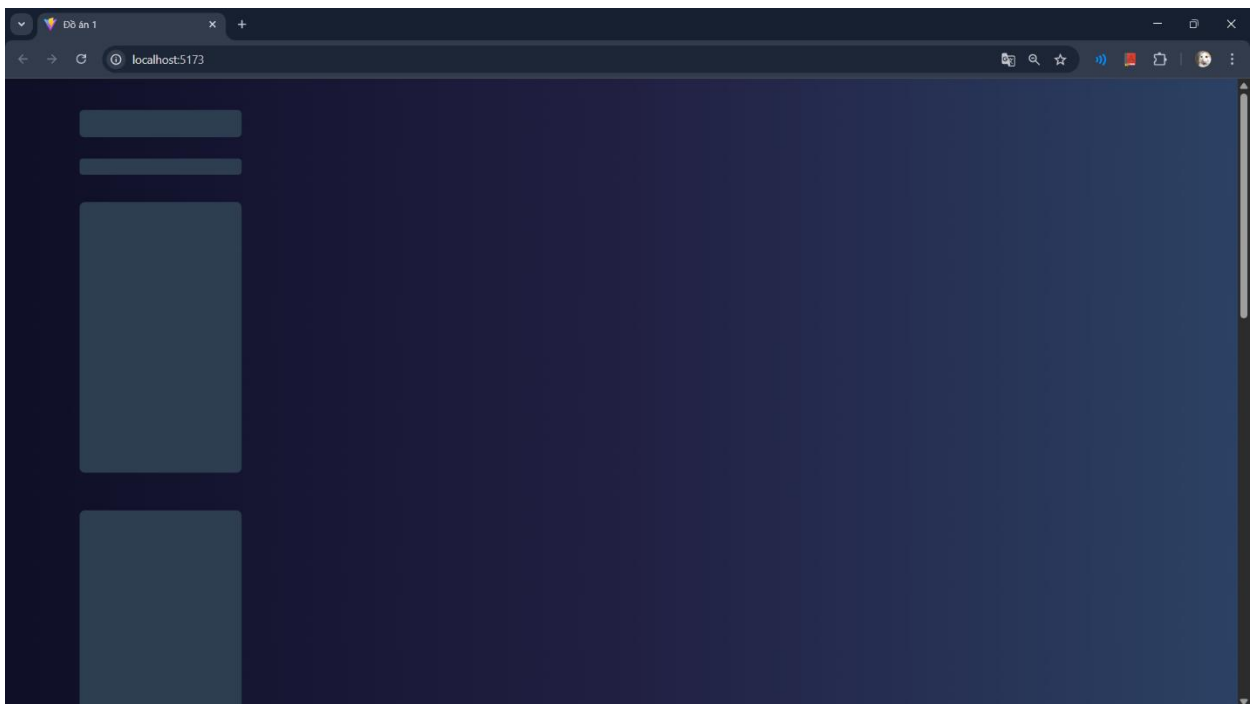
Truy cập thông qua đường dẫn <http://localhost:5173/>

Khi truy cập vào đường dẫn, nếu server backend lỗi thì sẽ hiện thông báo



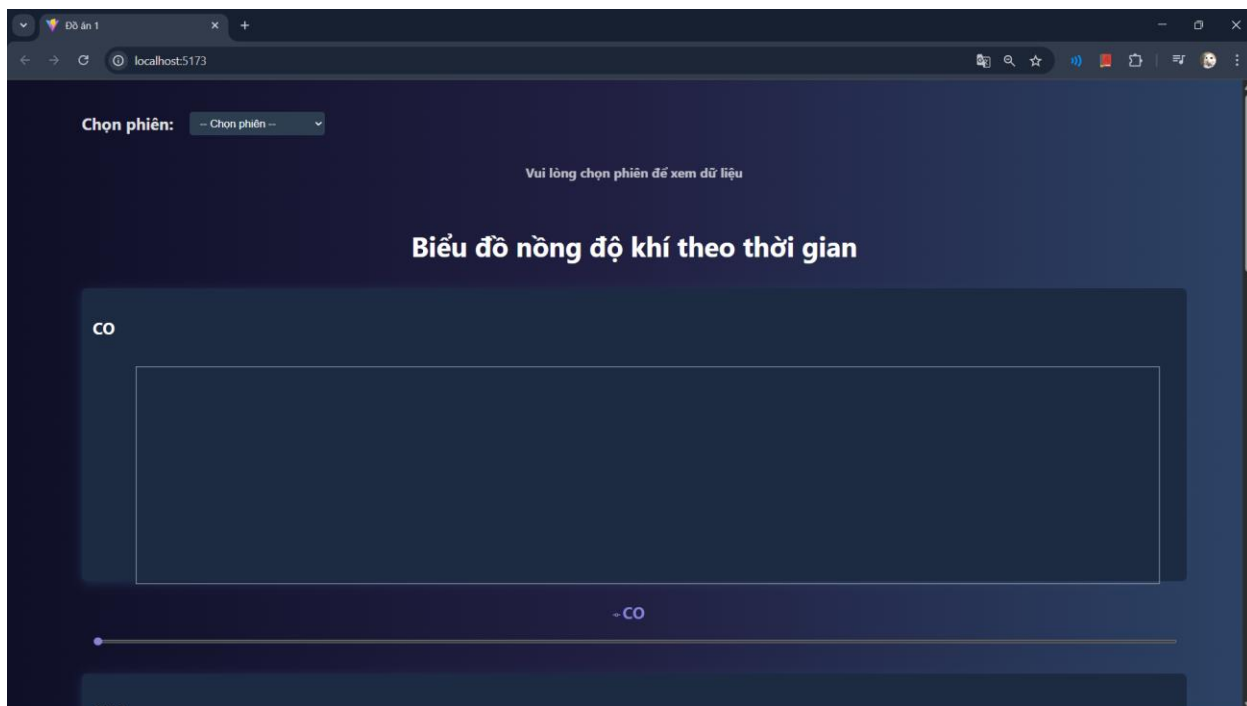
*Hình 5.6: Màn hình hiển thị lỗi*

Nếu không có lỗi sẽ hiện màn hình loading



*Hình 5.7: Màn hình loading*

Khi loading xong sẽ hiện



Hình 5.8: Load thành công

Chọn phiên muốn xem

Chọn phiên:

-- Chọn phiên --

-- Chọn phiên --

Phiên 16 - 3/5/2025

Phiên 17 - 3/5/2025

Phiên 18 - 3/5/2025

Phiên 19 - 4/5/2025

Phiên 20 - 4/5/2025

Phiên 21 - 4/5/2025

Phiên 22 - 4/5/2025

Phiên 23 - 4/5/2025

Phiên 24 - 16/5/2025

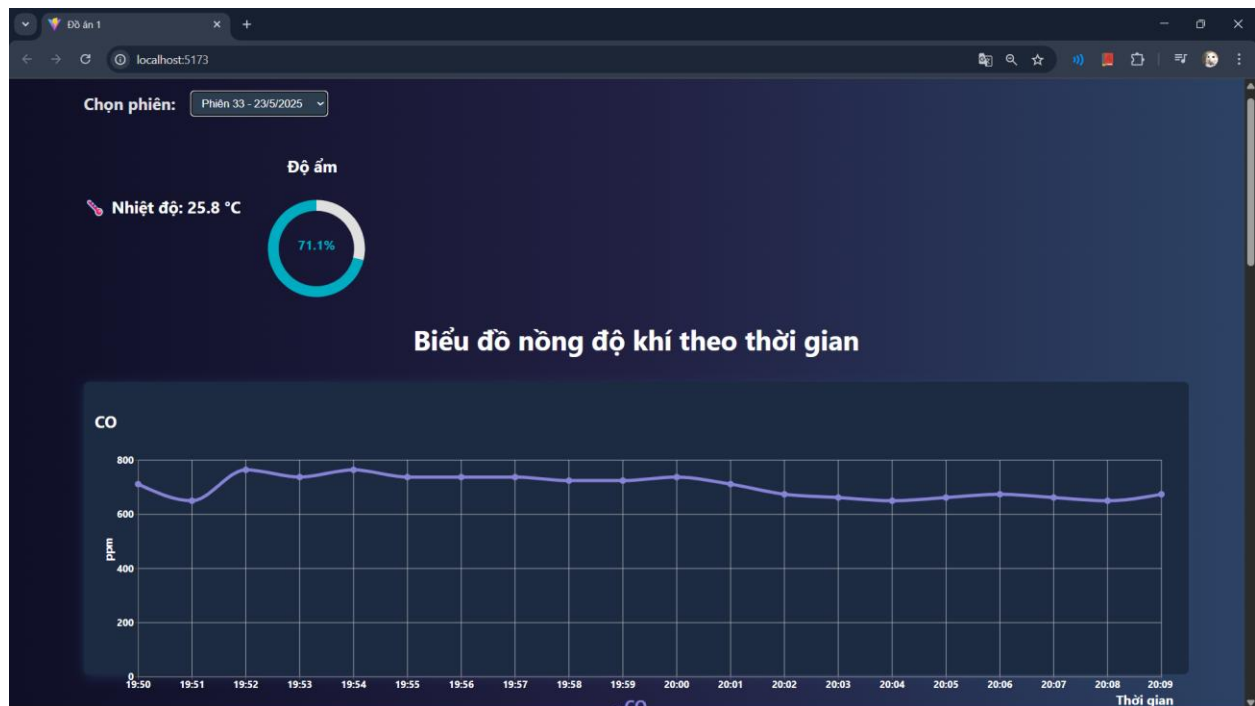
Phiên 25 - 16/5/2025

Phiên 26 - 16/5/2025

CO

Hình 5.9: Bảng chọn phiên

Khi chọn xong sẽ hiển thị



Hình 5.10: Web hiển thị dữ liệu của phiên 33

Trở vào biểu đồ để xem được dữ liệu, có thể kéo thanh bên dưới để xem được dữ liệu phía sau



Hình 5.11: Thông số CO<sub>2</sub> của phiên 33

## **CHƯƠNG VI: KẾT LUẬN**

### **6.1. Kết quả đạt được**

Xây dựng được hệ thống đo nhiệt độ, độ ẩm và nồng độ các chất khí gây nguy hại.

Xây dựng được database và giao diện người dùng.

### **6.2. Nhận xét**

#### **Ưu điểm**

Hệ thống hoạt động qua kết nối wifi, mang lại cấu trúc nhỏ gọn với khả năng truyền dữ liệu nhanh.

Đo được nhiều loại khí khác nhau, giúp tránh được nhiều nguy cơ ảnh hưởng sức khỏe do khí độc.

Server backend sử dụng websocket mang lại hiệu quả trong việc cập nhật dữ liệu nhanh chóng, đảm bảo tính real-time.

#### **Nhược điểm**

Cảm biến khí chưa mang lại độ chính xác cao, có thể do 2 nguyên nhân

1. Do cảm biến không chuyên trách đo một loại khí, dẫn đến còn sai số với một số loại khí
2. Do quá trình lập trình còn sai sót, sai sót có thể nằm ở thư viện hoặc nằm ở các hệ số khí do các hệ số này được tính thủ công từ đồ thị cung cấp bởi datasheet.

Chưa hiển thị được cảnh báo nguy hiểm khi nồng độ khí tăng cao.

## CHƯƠNG VII: TÀI LIỆU THAM KHẢO

1. Datasheet ESP32-DevKitC-32D Module WiFi Bluetooth 2.4GHz, [espressif.com](https://www.espressif.com)
2. Datasheet MQ135, [alldatasheet.com](https://www.alldatasheet.com)
3. Datasheet DHT22, [mouser.com](https://www.mouser.com)
4. Datasheet LCD 20x4, [uk.beta-layout.com](https://uk.beta-layout.com)
5. Wikipedia, “Log – log plot”, [en.wikipedia.org](https://en.wikipedia.org)



## CHƯƠNG VIII: PHỤ LỤC

### PHỤ LỤC I.

#### MÃ NGUỒN CỦA PHẦN CỨNG

Đoạn code để tính toán  $R_0$  trong môi trường bình thường

```
#include <Arduino.h>
#include <MQ135.h>

#define MQ135_PIN 34 //
float Rload = 36; //

MQ135 mq135_sensor(MQ135_PIN, 1.0, Rload); // Khởi tạo tạm với R0=1.0

void setup() {
  Serial.begin(9600);
  analogReadResolution(10);
  delay(2000); //

  Serial.println("Calibrating R0...");
  float rs = mq135_sensor.getResistance();

  // Theo datasheet MQ135, tỉ lệ RS/R0 trong không khí sạch là khoảng
  float R0 = rs / 0.6179480415;

  Serial.print("RS = ");
  Serial.println(rs);
  Serial.print("R0 (calibrated) = ");
  Serial.println(R0);
}

void loop() {
  \
```

## PHỤ LỤC II.

### MÃ NGUỒN CỦA SERVER BACKEND

- database.py

```
engine = create_engine(  
    SQLALCHEMY_DATABASE_URL,  
    pool_pre_ping=True,  
    pool_recycle=3600  
)
```

**pool\_pre\_ping:** Tự động kiểm tra kết nối trước khi dùng (tránh lỗi ngắt kết nối)

**pool\_recycle:** Giải phóng kết nối sau 1 giờ (3600s) để tránh timeout

- models.py

```
1  from sqlalchemy import Column, Integer, String, Double, DateTime, ForeignKey  
2  from sqlalchemy.orm import relationship  
3  from sqlalchemy.sql import func  
4  from database import Base  
5  
6  class Session(Base):  
7      __tablename__ = "session"  
8  
9      id = Column(Integer, primary_key=True, index=True)  
10     created_at = Column(DateTime(timezone=True), server_default=func.now())  
11  
12     data_entries = relationship("Data", back_populates="session")  
13  
14  
15     class Data(Base):  
16         __tablename__ = "data"  
17  
18         id = Column(Integer, primary_key=True, index=True)  
19         doam = Column(Double, index=True)  
20         nhietdo = Column(Double)  
21         co2 = Column(Double)  
22         co = Column(Double)  
23         nh3 = Column(Double)  
24         toluen = Column(Double)  
25         c6h6 = Column(Double)  
26         created_at = Column(DateTime(timezone=True), server_default=func.now())  
27  
28  
29         session_id = Column(Integer, ForeignKey("session.id"))  
30  
31  
32         session = relationship("Session", back_populates="data_entries")  
33
```

- Các class trong file schemas.py

```
class DataCreate(BaseModel):  
    doam: float  
    nhietdo: float  
    ppm_co2: float  
    ppm_co: float  
    ppm_nh3: float  
    ppm_toluen: float
```

```
ppm_c6h6: float
session_id: int
```

```
class Data(DataCreate):
    id: int
    created_at: datetime
```

```
class EmptyPayload(BaseModel):
    temp: int
class Session(BaseModel):
    session_id: int
    created_at: datetime
```

- Các hàm trong crud.py

Hàm get\_product

```
def get_product(session_id: int, db: Session):
    db_data = db.query(Data).where(Data.session_id == session_id)

    return [{"id": data.id,
            "doam": data.doam,
            "nhietdo": data.nhietdo,
            "ppm_co2": data.co2,
            "ppm_co": data.co,
            "ppm_nh3": data.nh3,
            "ppm_toluen": data.toluen,
            "ppm_c6h6": data.c6h6,
            "session_id": data.session_id,
            "created_at": data.created_at} for data in db_data]
```

Hàm get\_latest\_temperature

```
def get_latest_temperature(db: Session):
    latest = db.query(Data).order_by(Data.created_at.desc()).first()
    return {"id": latest.id,
            "doam": latest.doam,
            "nhietdo": latest.nhietdo,
            "ppm_co2": latest.co2,
            "ppm_co": latest.co,
            "ppm_nh3": latest.nh3,
            "ppm_toluen": latest.toluen,
            "ppm_c6h6": latest.c6h6,
            "session_id": latest.session_id,
```

```
"created_at": latest.created_at}
```

Hàm get\_all\_session

```
def get_all_session(db: Session):  
    db_data = db.query(Session).all()  
  
    return [{  
        "session_id": data.id,  
        "created_at": data.created_at,  
    } for data in db_data]
```

Hàm get\_latest\_session

```
def get_latest_session(db: Session):  
    latest = db.query(Session).order_by(Session.created_at.desc()).first()  
    return latest.id
```

Hàm lọc dữ liệu để hiển thị

```
const processData = (data) => {  
    const processedData = [];  
    for (let i = 0; i < data.length; i += 10) {  
        processedData.push(data[i]);  
    }  
    if (data.length % 10 !== 0) {  
        processedData.push(data[data.length - 1]);  
    }  
    return processedData;  
};  
const processedData = processData(Data);
```

## PHỤ LỤC III

### MÃ NGUỒN CỦA GIAO DIỆN NGƯỜI DÙNG (FRONTEND)

- index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>Đồ án 1</title>
8   </head>
9   <body>
10    <div id="root"></div>
11    <script type="module" src="/src/main.jsx"></script>
12  </body>
13 </html>
```