

**ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**



Nhóm 5

XÂY DỰNG CHƯƠNG TRÌNH THỰC HIỆN GIAO THỨC COAP CHO MẠNG CẢM BIẾN KHÔNG DÂY

BÁO CÁO MÔN HỌC LẬP TRÌNH MẠNG
Ngành: Mạng máy tính và truyền thông dữ liệu

Cán bộ hướng dẫn: TS. Nguyễn Ngọc Tân

HÀ NỘI - 2023

MỤC LỤC

TÓM TẮT	4
Chương I. Giới thiệu	5
1. Mô tả bài toán	5
2. Mục tiêu	5
3. Phạm vi nghiên cứu	5
Chương II. Cơ sở lý thuyết	6
1. Giao thức coAP	6
1.1. Khái niệm	6
1.2. Các tính năng cơ bản	6
1.3. coAP hoạt động như thế nào	7
2. Mạng cảm biến không dây	8
2.1 Khái niệm	8
2.2. Cấu trúc của 1 nút mạng cảm biến không dây	8
2.3. Cấu trúc liên kết mạng cảm biến không dây	9
2.4. Các loại cảm biến không dây	10
Chương III. Thiết kế chương trình	12
1. Kiến trúc tổng quan	12
2. Giao thức và thuật toán sử dụng	14
1. Giao thức	14
2. Thuật toán	15
3. Thiết kế tổng quan	15
4. Thiết kế Gateway (Server)	16
1. <i>Hiển thị các kết nối từ các node cảm biến</i>	16
2. <i>Nhận dữ liệu từ các node cảm biến</i>	17
3. <i>Hiển thị dữ liệu thông qua biểu đồ</i>	18
5. Thiết kế Sensor (Client)	19

1. <i>Tạo Sensor</i>	19
2. <i>Tự động sinh dữ liệu</i>	19
3. <i>Tự động gửi dữ liệu lên Gateway qua giao thức coAP</i>	20
4. <i>Hiển thị các thông tin điều khiển (từ Gateway)</i>	21
5. <i>Hiển thị, đánh giá tối đa số node cảm biến có thể kết nối tới</i>	21
6. <i>Hiển thị, đánh giá hiệu năng của giao thức khi node cảm biến tăng lên</i>	22
Chương IV: Môi trường chạy chương trình	23
1. Hệ điều hành Linux	23
1.1. Giới thiệu	23
1.2. Lý do sử dụng chọn sử dụng Linux	24
2. Ngôn ngữ sử dụng	25
1. Java	25
2. HTML	25
3. CSS	25
Chương V: Demo	26
Chương VI: Kết luận	29
1. Tóm tắt kết quả	29
2. Hạn chế	29
3. Hướng phát triển trong tương lai	29
4. Quá trình làm việc của nhóm	29
5. Vai trò thành viên nhóm	30
Chương VII: Tài liệu tham khảo	31
LỜI KẾT	32

TÓM TẮT

Tóm tắt: Hiện nay, với sự phát triển nhanh chóng của các dịch vụ IoT và sự bùng nổ của Internet đã dẫn đến một loạt các vấn đề, một trong số đó là các vấn đề liên quan tới truyền dữ liệu trong mạng cảm biến không dây. CoAP được ra đời với mục đích cung cấp một giao thức đơn giản, hiệu quả và bảo mật, có thể hỗ trợ các vấn đề mà mạng cảm biến không dây đang gặp phải. Do vậy, trong bản báo cáo này, em xin trình bày về giao thức coAP và cùng tìm hiểu cách xây dựng một chương trình đơn giản sử dụng giao thức này để truyền tải dữ liệu trong mạng cảm biến không dây. Nội dung của báo cáo sẽ tập trung trình bày những kiến thức cơ bản về coAP cũng như cách triển khai xây dựng chương trình này.

Từ khoá: *IoT, mạng cảm biến không dây, coAP.*

Chương I. Giới thiệu

1. Mô tả bài toán

Đề bài yêu cầu xây dựng chương trình sử dụng giao thức CoAP cho mạng cảm biến không dây. Chương trình này bao gồm các thành phần:

- Gateway đóng vai trò server
- Chương trình sinh dữ liệu cho các cảm biến để gửi đến server.
- Chương trình có giao diện hiển thị phía server.
- Chương trình cho phép các cảm biến nhận dữ liệu
- Chương trình sinh dữ liệu cho các cảm biến: tự động sinh dữ liệu cảm biến và gửi dữ liệu lên gateway. Chương trình hiển thị dữ liệu nhận được phía gateway (sử dụng các biểu đồ line chart, bar chart, pie chart để hiển thị)

2. Mục tiêu

- Chương trình hiển thị thông tin các cảm biến thực hiện kết nối với gateway.
- Chương trình hiển thị thông tin điều khiển tại các cảm biến (được gửi từ gateway)
- Chương trình hiển thị đánh giá số lượng tối đa các node cảm biến có thể kết nối đến gateway.
- Chương trình hiển thị đánh giá hiệu năng của giao thức: Throughput, delay, v.v. khi số lượng node cảm biến tăng lên.

3. Phạm vi nghiên cứu

Do kiến thức và kỹ năng có phần hạn chế, nhóm chúng em xin phép được giới hạn phạm vi nghiên cứu ở mức cơ bản, có thể đọc hiểu và xây dựng được chương trình chương trình cơ bản ạ.

Chương II. Cơ sở lý thuyết

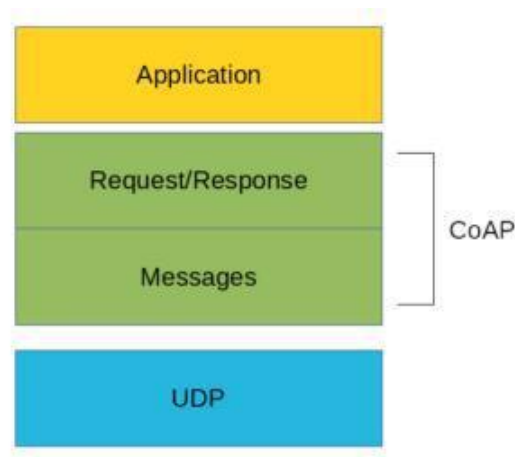
1. Giao thức coAP

1.1. Khái niệm

- Constrained Application Protocol (CoAP) là một giao thức truyền tải web chuyên dụng được sử dụng với constrained nodes và constrained networks trong Internet of Things (IoT). CoAP được thiết kế để cho phép các thiết bị đơn giản, bị ràng buộc, kết nối vào IoT ngay cả qua các mạng bị hạn chế với băng thông thấp và khả năng khả dụng thấp. Nó thường được sử dụng cho các ứng dụng máy-máy (M2M) như năng lượng thông minh và tự động hóa tòa nhà. Giao thức này được thiết kế bởi Internet Engineering Task Force (IETF) và được chỉ định trong IETF RFC 7252.

1.2. Các tính năng cơ bản

- Giao thức web được sử dụng trong M2M với các yêu cầu bị ràng buộc
 - Trao đổi tin nhắn không đồng bộ
 - Chi phí thấp và rất đơn giản để phân tích
 - Hỗ trợ URI, định danh tài nguyên đồng nhất
 - Khả năng proxy và Caching
- Từ lớp giao thức trừu tượng hóa, CoAP có thể được biểu diễn dưới dạng:



- Có hai lớp khác nhau tạo giao thức CoAP: Tin nhắn và Yêu cầu / Phản hồi. Lớp Tin nhắn liên quan đến UDP và với các tin nhắn không đồng bộ. Lớp Yêu cầu / Phản hồi quản lý tương tác yêu cầu / phản hồi dựa trên các thông báo yêu cầu / phản hồi.
- CoAP hỗ trợ bốn loại thông báo khác nhau:
 - Xác nhận
 - Không thể xác nhận
 - Nhìn nhận
 - Cài lại

1.3. coAP hoạt động như thế nào

CoAP hoạt động như một loại HTTP dành cho các thiết bị bị hạn chế, cho phép các thiết bị như cảm biến hoặc bộ kích hoạt truyền thông trên IoT. Các cảm biến và bộ kích hoạt này được điều khiển và đóng góp bằng cách truyền dữ liệu của họ như một phần của hệ thống. Giao thức này được thiết kế để đảm bảo tính tin cậy trong môi trường có băng thông thấp và tắc nghẽn cao thông qua việc tiêu thụ ít năng lượng và thiếu gánh nặng mạng. Trong một mạng có nhiều tắc nghẽn hoặc kết nối hạn chế, CoAP có thể tiếp tục hoạt động trong khi các giao thức dựa trên TCP như MQTT gặp khó khăn trong việc trao đổi thông tin và giao tiếp hiệu quả.

Ngoài ra, các tính năng hiệu quả và thông thường của CoAP cho phép các thiết bị hoạt động trong chất lượng tín hiệu kém gửi dữ liệu của họ một cách đáng tin cậy hoặc cho phép một vệ tinh quay quanh duy trì việc giao tiếp từ xa thành công. CoAP cũng hỗ trợ các mạng với hàng tỷ nút. Đối với bảo mật, các tham số DTLS được chọn mặc định tương đương với các khóa RSA 128 bit.

CoAP sử dụng UDP như giao thức mạng cơ bản. CoAP cơ bản là một giao thức IoT kiểu client-server, trong đó client gửi yêu cầu và server trả lại phản hồi, tương tự như trong HTTP. Các phương thức được sử dụng bởi CoAP tương tự như trong HTTP.

2. Mạng cảm biến không dây

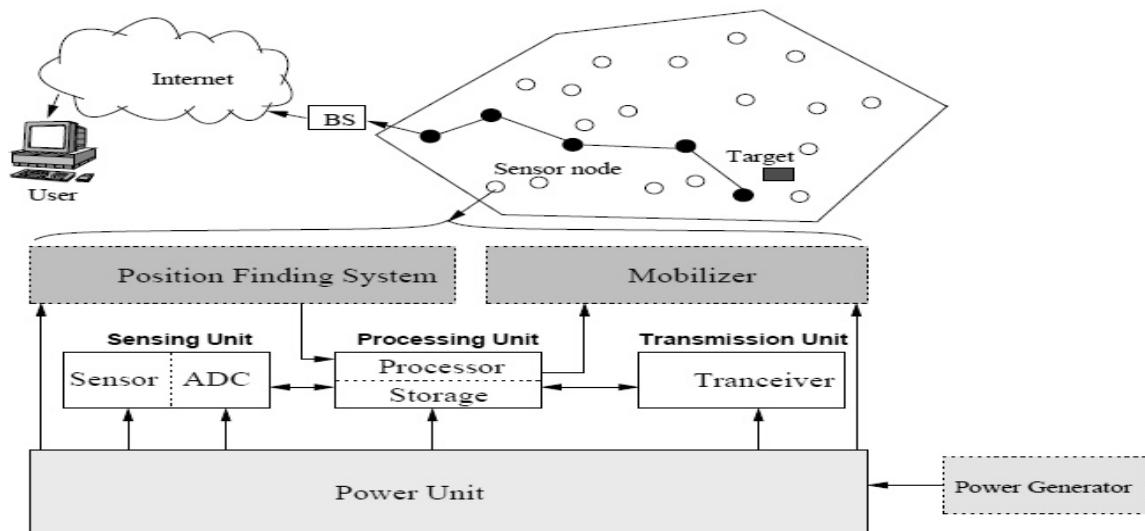
2.1 Khái niệm

Mạng cảm biến không dây (Wireless Sensor Networks - WSNs) là một hệ thống mạng được sử dụng để thu thập thông tin từ hiện trường thông qua các liên kết không dây và sóng vô tuyến.

WSN bao gồm hai thành phần chính là trạm gốc và các nút cảm biến. Các nút cảm biến thường có kích thước nhỏ gọn và được phân tán rải rác trên một khu vực rộng lớn. Chúng được sử dụng để giám sát các điều kiện vật lý hoặc môi trường như nhiệt độ, âm thanh, rung động, áp suất, chuyển động và cả các chất ô nhiễm. Dữ liệu từ các nút này được truyền qua mạng đến trạm thu phát (Sink) hoặc trạm gốc, nơi dữ liệu có thể được quan sát và phân tích.

Một trạm thu phát hoặc trạm gốc hoạt động như một giao diện giữa người dùng và mạng. Người dùng có thể truy xuất thông tin cần thiết từ mạng bằng cách gửi các truy vấn tới trạm thu phát và thu nhận kết quả từ nó.

2.2. Cấu trúc của 1 nút mạng cảm biến không dây



Cấu trúc cơ bản của nút mạng cảm biến không dây

Trong mạng cảm biến không dây, các nút cảm biến có khả năng giao tiếp với nhau thông qua tín hiệu vô tuyến. Mỗi nút cảm biến được hình thành từ bốn thành phần cơ bản gồm: bộ cảm biến, bộ xử lý, bộ truyền nhận vô tuyến và nguồn cung cấp năng lượng. Tùy thuộc vào ứng dụng cụ thể, các nút cảm biến còn có thể có các thành phần bổ sung như hệ thống định vị GPS, nguồn điện phụ và các thiết bị di động.

Thành phần cảm biến của mỗi nút thường bao gồm hai đơn vị con: cảm biến và bộ chuyển đổi tương tự-số (ADC). Các tín hiệu tương tự được tạo ra bởi cảm biến được chuyển đổi thành tín hiệu kỹ thuật số bởi ADC, sau đó được đưa vào bộ xử lý. Đơn vị xử lý thường kết hợp với một bộ nhớ nhỏ và có khả năng quản lý các quy trình để nút cảm biến có thể tương tác với các nút khác để thực hiện các nhiệm vụ cảm biến được giao.

Bộ truyền nhận vô tuyến có vai trò kết nối nút cảm biến với mạng. Một trong những thành phần quan trọng nhất của mỗi nút cảm biến là nguồn năng lượng. Các nguồn năng lượng có thể được hỗ trợ bằng các thiết bị thu gom năng lượng như pin mặt trời. Các thành phần khác của nút cảm biến phụ thuộc vào ứng dụng cụ thể.

2.3. Cấu trúc liên kết mạng cảm biến không dây

Hiện có 3 loại cấu trúc mạng cảm biến không dây:

- **Cấu trúc hình sao – star**

- Ưu điểm: Tính đơn giản, khả năng giữ mức tiêu thụ điện năng của nút từ xa ở mức tối thiểu. Nó cũng cho phép thông tin liên lạc có độ trễ thấp giữa nút từ xa và trạm gốc.
- Nhược điểm: Trạm gốc phải nằm trong phạm vi truyền dẫn vô tuyến của tất cả các nút riêng lẻ và khả năng mở rộng không bằng các mạng khác do sự phụ thuộc của nó vào một trạm gốc để quản lý mạng.

- **Cấu trúc cây - tree**

- Ưu điểm: Có thể dễ dàng mở rộng mạng và việc phát hiện lỗi cũng trở nên dễ dàng.
- Nhược điểm: phụ thuộc rất nhiều vào cáp bus, nếu nó bị hỏng, tất cả mạng sẽ sụp đổ.

- **Cấu trúc liên kết - mesh**

- Ưu điểm: Có lợi thế về khả năng dự phòng và khả năng mở rộng. Nếu một nút riêng lẻ bị lỗi, một nút từ xa vẫn có thể giao tiếp với bất kỳ nút nào khác trong phạm vi của nó, do đó, có thể chuyển tiếp thông điệp đến vị trí mong muốn. Ngoài ra, phạm vi của mạng không nhất thiết bị giới hạn bởi phạm vi giữa các nút đơn lẻ; nó chỉ có thể được mở rộng bằng cách thêm nhiều nút hơn vào hệ thống.

- Nhược điểm: Tiêu thụ năng lượng cho các nút triển khai truyền thông đa bước thường cao hơn so với các nút không có khả năng này, thường làm hạn chế tuổi thọ pin. Ngoài ra, khi số lượng các bước truyền thông tin đến đích tăng lên, thời gian để gửi thông điệp cũng tăng lên. Chi phí đầu tư mạng lưới lớn và đòi hỏi vốn đầu tư nhiều.

2.4. Các loại cảm biến không dây

- **WSN trên cạn (Terrestrial wireless sensor networks)**

- Mạng cảm biến không dây (WSN) trên mặt đất có khả năng giao tiếp hiệu quả với các trạm gốc và bao gồm từ hàng trăm đến hàng nghìn nút cảm biến không dây được triển khai theo cách phi cấu trúc hoặc có cấu trúc (được lên kế hoạch trước). Trong WSN này, nguồn pin bị hạn chế; tuy nhiên, pin được trang bị pin mặt trời để làm nguồn năng lượng phụ.

- **WSN ngầm (Underground wireless sensor networks)**

- Mạng cảm biến không dây ngầm đất hơn mạng WSN trên mặt đất về mặt triển khai, bảo trì. Mạng WSNs bao gồm một số nút cảm biến được ฝัง trong lòng đất để theo dõi các điều kiện dưới lòng đất. Để chuyển tiếp thông tin từ các nút cảm biến đến trạm gốc, các nút chìm bổ sung được đặt trên mặt đất. Các mạng cảm biến không dây ngầm được triển khai trong lòng đất rất khó để sạc pin lại. Thêm vào đó, môi

trường ngầm khiến giao tiếp không dây trở thành một thách thức do mức độ suy giảm và mất tín hiệu cao.

- **WSN dưới nước (Under Water wireless sensor networks)**

- Hơn 70% diện tích trái đất là nước. Các mạng này bao gồm một số nút cảm biến và các phương tiện được triển khai dưới nước. Các phương tiện tự hành dưới nước được sử dụng để thu thập dữ liệu từ các nút cảm biến này. Một thách thức của liên lạc dưới nước là độ trễ truyền dài, băng thông và cảm biến bị lỗi. Ở dưới nước, WSN được trang bị một loại pin hạn chế không thể sạc lại hoặc thay thế. Vấn đề bảo tồn năng lượng cho các WSN dưới nước liên quan đến sự phát triển của các kỹ thuật mạng và truyền thông dưới nước.

- **WSN đa phương tiện (Multimedia wireless sensor networks)**

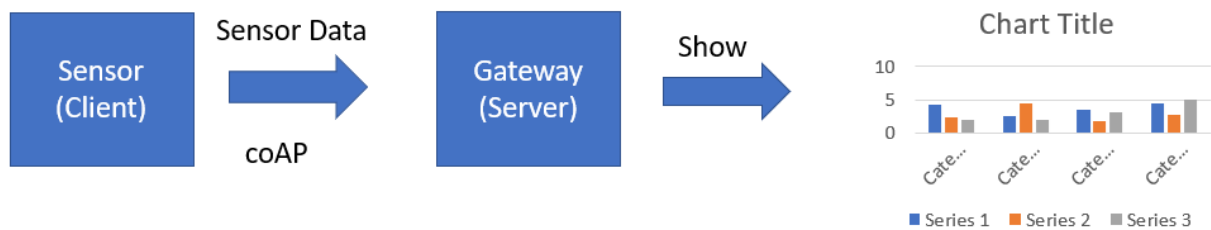
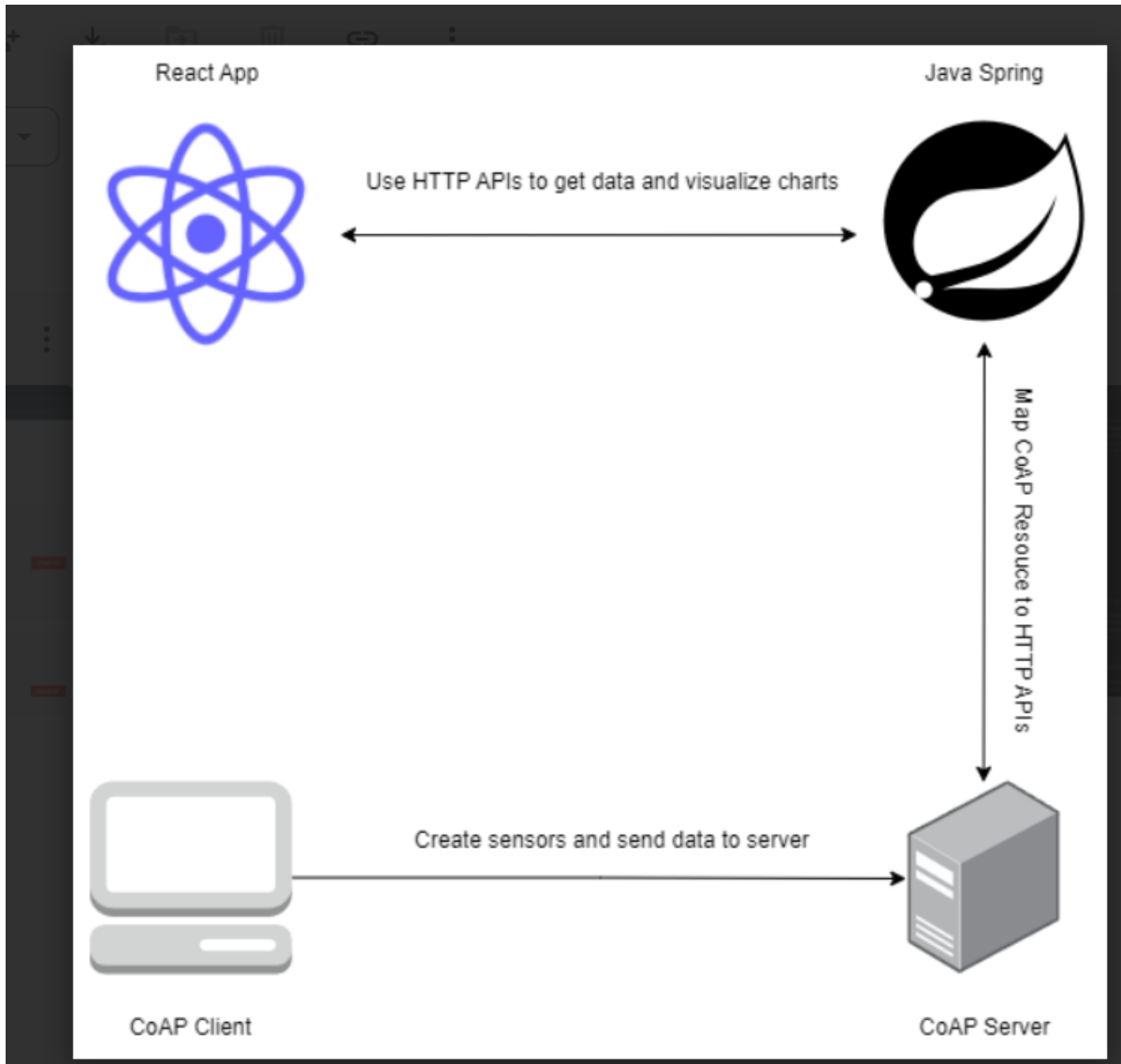
- Mạng cảm biến không dây đa phương tiện được đề xuất để giám sát và theo dõi các sự kiện đa phương tiện như hình ảnh, video và âm thanh. Mạng này sử dụng các nút cảm biến chi phí thấp trang bị micrô và máy ảnh, kết nối không dây để nén và truy xuất dữ liệu. Tuy nhiên, mạng đa phương tiện đối mặt với các thách thức như tiêu thụ năng lượng cao, yêu cầu băng thông lớn, xử lý dữ liệu và kỹ thuật nén

- **Mạng cảm biến không dây di động (Mobile Wireless Sensor Networks)**

- Mạng không dây đa phương tiện (MWSN) bao gồm một tập hợp các nút cảm biến có khả năng tự di chuyển và tương tác với môi trường vật lý. MWSN linh hoạt hơn so với mạng cảm biến không dây tĩnh, vì các nút cảm biến có thể triển khai trong bất kỳ tình huống nào và thích ứng với những thay đổi cấu trúc liên kết nhanh chóng. MWSN mang đến nhiều ưu điểm so với mạng cảm biến không dây tĩnh, bao gồm phạm vi phủ sóng tốt hơn và cải thiện, hiệu suất năng lượng tốt hơn, dung lượng kênh vượt trội và nhiều điểm mạnh khác.

Chương III. Thiết kế chương trình

1. Kiến trúc tổng quan



Hình: Mô hình hoạt động của chương trình

- Trong mô hình này, các cảm biến (Sensor) thu thập dữ liệu và gửi nó đến Gateway (Server).
- Gateway đóng vai trò như một server để nhận dữ liệu từ các cảm biến và hiển thị thông tin trên giao diện.
- Các cảm biến tự động sinh dữ liệu và gửi dữ liệu lên Gateway thông qua giao thức CoAP.
- Gateway sẽ nhận dữ liệu từ các cảm biến và hiển thị nó trên giao diện của mình, sử dụng các biểu đồ (line chart, bar chart, pie chart) để hiển thị thông tin cảm biến.



- Ngoài ra, Gateway cũng có khả năng gửi các thông điệp điều khiển đến các cảm biến. Điều khiển này có thể được hiển thị trên giao diện của cảm biến.
- Từ yêu cầu từ phía Gateway (Server), Sensor tiếp nhận và làm theo yêu cầu, sau đó hiển thị ra màn hình các thông tin mà Gateway yêu cầu (throughput, delay, etc...)

2. Giao thức và thuật toán sử dụng

1. Giao thức

- CoAP: Sử dụng giao thức CoAP để truyền dữ liệu từ phía Sensor (sau khi sinh dữ liệu) tới Gateway.

```
// sử dụng coAP
@GetMapping(value = "/observe", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
@CrossOrigin(origins = "*", allowedHeaders = "*", methods = {RequestMethod.GET})
public Flux<ServerSentEvent<String>> observeSensors() {
    // Tạo Flux<ServerSentEvent>
    return Flux.create(emitter -> {
        CoapClient client = new CoapClient(COAP_SERVER_URL);

        Request request = Request.newGet().setURI(COAP_SERVER_URL).setObserve();
        request.setConfirmable(con:true);

        CoapObserveRelation relation = client.observe(request, new CoapHandler() {
            @Override
            public void onLoad(CoapResponse coapResponse) {
                // Lấy dữ liệu sensors từ SensorResource
                String jsonString = sensorResource.sensorsToJsonObject().toString();

                // Tạo đối tượng ServerSentEvent
                ServerSentEvent<String> event = ServerSentEvent.builder(jsonString)
                    .id(Long.toString(System.currentTimeMillis()))
                    .build();

                // Trả về ServerSentEvent
                emitter.next(event);
            }
        });
    });
}
```

- HTTP: Sử dụng giao thức HTTP để truyền dữ liệu từ Gateway tới giao diện front-end.

```
// gửi dl từ server sang react để hiển thị dl sử dụng HTTP
@PutMapping(value = "/toggleSensor")
@CrossOrigin(origins = "*", allowedHeaders = "*", methods = {RequestMethod.PUT})
public String toggleSensor(@RequestBody Sensor sensor){
    CoapClient client = new CoapClient(COAP_SERVER_URL + "/" + sensor.getId());

    System.out.println(sensor);

    Request request = new Request(CoAP.Code.PUT);

    request.setConfirmable(con:true);

    request.setPayload(sensor.sensorToJsonObject().toString().getBytes());

    CoapResponse coapResp = null;
    try {
        coapResp = client.advanced(request);
    } catch (ConnectorException | IOException e) {
        e.printStackTrace();
    }

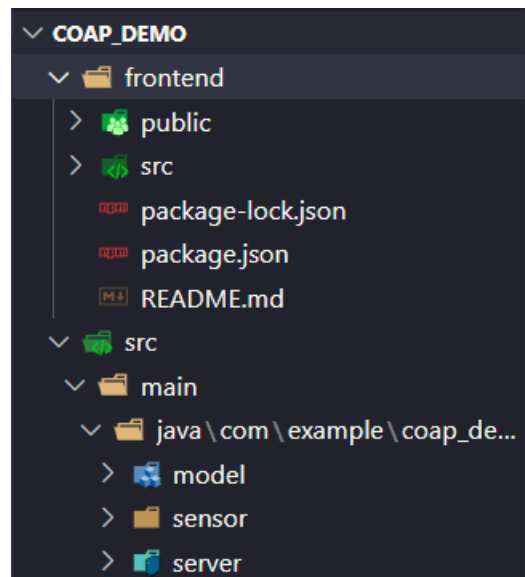
    return "toggle sensor succesfull!";
}
```

2. Thuật toán

- Chương trình chúng em tạo ra không sử dụng thuật toán
- Thay vào đó, ở phần V, trong phần yêu cầu tính delay khi số lượng node tăng lên, chúng em sử dụng công thức sau:
 - $\text{Delay} = \text{time gửi (client)} - \text{time nhận (server)}$

3. Thiết kế tổng quan

- Cấu trúc thư mục file như sau:



- Chương trình gồm 2 thư mục chính:
 1. Thư mục src chứa các thiết kế client-sever
 - Thư mục sensor chứa chương trình phía Sensor
 - Thư mục model chứa các thiết kế Sensor của client
 - Thư mục server chứa chương trình phía Gateway
 2. Thư mục “frontend” để hiển thị giao diện

4. Thiết kế Gateway (Server)

1. *Hiển thị các kết nối từ các node cảm biến*

```
eventSource.onmessage = (event) => {  
  const data = JSON.parse(event.data)  
  
  // create new sensors by list  
  const newSensors = []  
  
  // loop for creating attributes for sensors  
  for (var i = 0; i < data.result; i++) {  
    const sensor = data.data[i]  
  
    const existingSensor = sensors.filter(  
      (el) => el.id == data.data[i].id  
    )  
  
    // set value and time  
    const newData = {  
      value: sensor.value,  
      time: sensor.lastUpdate,  
    }  
  
    // add attribute for newSensor  
    const newSensorData = {  
      id: sensor.id,  
      lastValue: sensor.value,  
      isRunning: sensor.isRunning,  
    }  
  
    // create sensor data  
    if (existingSensor.length > 0) {  
      existingSensor[0].data.push(newData)  
      newSensorData.data = existingSensor[0].data  
    } else {  
      newSensorData.data = [newData]  
    }  
  
    newSensors.push(newSensorData)  
  }  
}
```

Hình: *Hiển thị thông tin về các node kết nối*

- Để hiển thị các node cảm biến kết nối tới server, ta sử dụng file Sensor.js và SensorCard.js trong thư mục coap_frontend
- Khi các sensor kết nối tới Gateway, gateway sẽ trả về các thông tin như id, thời gian, có kết nối hay không và giá trị là bao nhiêu

2. Nhận dữ liệu từ các node cảm biến

```
// nhận data
@Override
public void handlePUT(CoapExchange exchange) {
    System.out.println("test");

    // Nhận payload từ yêu cầu CoAP
    byte[] payload = exchange.getRequestPayload();

    // Chuyển đổi chuỗi payload thành đối tượng JSONObject
    String payloadStr = new String(payload);

    JSONObject jsonObject = new JSONObject(payloadStr);

    System.out.println(jsonObject);

    // Lấy giá trị từ đối tượng JSONObject
    String id = jsonObject.getString(name:"id");
    Double value = jsonObject.getDouble(name:"value");
    Boolean isRunning = jsonObject.getBoolean(name:"isRunning");

    Sensor newSensor = new Sensor(id, value);
    newSensor.setIsRunning(isRunning);
    this.sensor = newSensor;

    changed();

    SensorResource parentResource = (SensorResource) this.getParent();
    parentResource.updateSensorById(sensor);

    // Gửi phản hồi về cho yêu cầu CoAP
    Response response = new Response(CoAP.ResponseCode.CHANGED);
    response.setPayload("Đã nhận và update thành công".getBytes());
    exchange.respond(response);
}
```

- Hàm này được tạo với mục đích nhận các dữ liệu từ phía Sensor gửi lên

3. Hiển thị dữ liệu thông qua biểu đồ

- Sau khi đã nhận dữ liệu từ phía Client thành công, Gateway sẽ tiến hành hiển thị luồng dữ liệu đó thông qua biểu đồ bằng hàm sau đây:

```
coap_frontend > src > components > JS SensorCard.js > [e] SensorCard
44     <div className="card">
45       <h2>{sensor.id}</h2>
46       <p>
47         Value:{' '}
48         <span id="sensor1-value">{sensor.lastValue.toFixed(2)}</span>
49       </p>
50       {showChart && (
51         <LineChart width={400} height={300} data={data}>
52           <XAxis
53             dataKey="time"
54             tickCount={3}
55             tickFormatter={(time) =>
56               moment(time).format('mm:ss.SSS')
57             }
58           />
59           <YAxis />
60           <CartesianGrid strokeDasharray="3 3" />
61           <Tooltip />
62           <Legend />
63           <Line type="monotone" dataKey="value" stroke="#8884d8" />
64         </LineChart>
65       )}
66
67       <button
68         className={`button ${showChart ? 'off' : 'on'}`}
69         id="sensor1-button"
70         onClick={handleToggleChart}
71       >
72         {showChart ? 'Chart Off' : 'Chart On'}
73       </button>
74       <button
75         className={`button ${sensor.isRunning ? 'off' : ''}`}
76         id="sensor1-button"
77         onClick={handleToggle}
78       >
79         {sensor.isRunning ? 'Turn Off' : 'Turn On'}
80       </button>
```

- Dữ liệu nhận được hiển thị dưới dạng Line chart

5. Thiết kế Sensor (Client)

1. Tạo Sensor

```
8
9 // tạo sensors
10
11 public class CoapClientManager {
12     private final Logger logger = LoggerFactory.getLogger(clazz:CoapClientManager.class);
13     private static int NUMBER_SENSOR = 0;
14     private static final String COAP_SERVER_ADD_URL = "coap://localhost:5683/addSensor"; // Địa chỉ URI của server CoAP
15     private CoapPostSensorProcess coapPostSensorProcess;
16
17     Run | Debug
18     public static void main(String[] args) {
19         CoapClient clientManager = new CoapClient(COAP_SERVER_ADD_URL);
20
21         logger.info(format:"OBSERVING ... {}", COAP_SERVER_ADD_URL);
22
23         Request request = Request.newGet().setURI(COAP_SERVER_ADD_URL).setObserve();
24         request.setConfirmable(con:true);
25
26         CoapObserveRelation relation = clientManager.observe(request, new CoapHandler() {
27             public void onLoad(CoapResponse response) {
28                 String content = response.getResponseText();
29
30                 JSONObject jsonObject = stringToJsonObject(content);
31
32                 // Lấy giá trị từ đối tượng JSONObject
33                 int newNumber = jsonObject.getInt(name:"numberSensorCurrently");
34
35                 if (newNumber != NUMBER_SENSOR) {
36                     int numberCreateSensor = newNumber - NUMBER_SENSOR;
37                     NUMBER_SENSOR = newNumber;
38
39                     CoapPostSensorProcess coapPostSensorProcess = new CoapPostSensorProcess();
40                     if (numberCreateSensor > 1) {
```

2. Tự động sinh dữ liệu

```
// tự động sinh data
public void randomTemperature() {
    Random random = new Random();
    Double minTemperature = -10.0;
    Double maxTemperature = 40.0;
    Double temperatureRange = maxTemperature - minTemperature;
    Double generatedTemperature = random.nextDouble() * temperatureRange + minTemperature;
    this.setValue(generatedTemperature);
}
}
```

- Hàm sinh dữ liệu nằm trong file Sensor.java, loại dữ liệu được chọn ở đây là nhiệt độ ngoài trời, nằm trong khoảng từ âm 10 độ tới max là 40 độ C.

3. Tự động gửi dữ liệu lên Gateway qua giao thức coAP

```
// gửi sensor liên tục
private void continuousPutToServer() {
    // Tạo đối tượng Timer
    Timer timer = new Timer();

    // Tạo đối tượng TimerTask để định nghĩa công việc cần thực hiện
    TimerTask task = new TimerTask() {
        @Override
        public void run() {
            Request request = new Request(CoAP.Code.PUT);

            request.setConfirmable(con:true);

            request.setPayload(sensor.sensorToJsonObject().toString().getBytes());

            logger.info(format:"Request Pretty Print:\n{}", Utils.prettyPrint(request));
            CoapResponse coapResp = null;

            Boolean isRunning = sensor.getIsRunning();

            if(isRunning){
                try {
                    coapResp = sensorClient.advanced(request);

                    if(coapResp != null){
                        //Pretty print for the received response
                        logger.info(format:"Response Pretty Print: \n{}", Utils.prettyPrint(coapResp));

                        //The "CoapResponse" message contains the response.
                        String text = coapResp.getResponseText();
                        logger.info(format:"Payload: {}", text);
                        logger.info("Message ID: " + coapResp.advanced().getMID());
                        logger.info("Token: " + coapResp.advanced().getTokenString());

                        sensor.randomTemperature();
                    }
                }
            }
        }
    };
    timer.schedule(task, 0, 1000);
} catch (ConnectorException | IOException e) {
```

- Hàm này sử dụng giao thức coAP để truyền dữ liệu từ Sensor lên Gateway
- Dữ liệu được truyền liên tục sau mỗi 1s

4. *Hiển thị các thông tin điều khiển (từ Gateway)*

```
// nhận thông tin điều khiển từ server
private void getObserve() {
    Request request = Request.newGet().setURI(this.SENSOR_URL).setObserve();
    request.setConfirmable(true);

    CoapObserveRelation relation = sensorClient.observe(request, new CoapHandler() {
        public void onLoad(CoapResponse response) {
            String content = response.getResponseText();

            JSONObject jsonObject = stringToJsonObject(content);

            // Lấy giá trị từ đối tượng JSONObject
            Boolean isRunning = jsonObject.getBoolean("isRunning");

            sensor.setIsRunning(isRunning);

            logger.info(format:"Notification Response Pretty Print: \n{}", Utils.prettyPrint(response));
            logger.info("NOTIFICATION Body: " + jsonObject);
        }

        public void onError() {
            logger.error(msg:"OBSERVING FAILED");
        }
    });
};
```

- Client nhận luồng điều khiển được gửi từ Gateway thông qua hàm này

5. *Hiển thị, đánh giá tối đa số node cảm biến có thể kết nối tới*

- Với yêu cầu này, chúng em thêm vào các sensor liên tục cho tới khi server sập, thời điểm sập số node là bao nhiêu chính là tối đa số node có thể kết nối tới

6. *Hiển thị, đánh giá hiệu năng của giao thức khi node cảm biến tăng lên*

```
PerformanceResource.java X
src > main > java > com > example > coap_demo > server > resource > PerformanceResource.java > PerformanceResource > extractCPU()
77
78
79 public JSONObject performanceToJsonObject() {
80     // Tạo đối tượng JSON
81     JSONObject jsonObject = new JSONObject();
82
83     // Thiết lập thuộc tính "success" với giá trị true
84     jsonObject.put(name:"success", value:true);
85
86     jsonObject.put(name:"delay", DELAY);
87
88     jsonObject.put(name:"cpu", CPU);
89
90     jsonObject.put(name:"ram", RAM);
91
92     return jsonObject;
93 }
94
95 public double extractCPU() throws IOException {
96     ProcessBuilder processBuilder = new ProcessBuilder("top", "-b", "-n", "1");
97     Process process = processBuilder.start();
98
99     BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
100     double cpuUsage = 0;
101     String line;
102     while ((line = reader.readLine()) != null) {
103         if (line.startsWith("Cpu(s)")) {
104             // Xử lý thông tin CPU
105             String[] cpuInfo = line.split(",");
106             for (String info : cpuInfo) {
107                 if (info.contains("id")) {
108                     cpuUsage = 100 - Double.parseDouble(info.replaceAll("[^\\d.]", ""));
109                     System.out.println("Phần trăm CPU đang sử dụng: " + cpuUsage + "%");
110                 }
111             }
112         }
113     }
114 }
```

- Hàm này chúng em tính delay, ram và CPU, tuy nhiên, do bị xung đột với hệ thống nên chúng em chỉ làm được delay thôi ạ.

Chương IV: Môi trường chạy chương trình

1. Hệ điều hành Linux

1.1. Giới thiệu

Tương tự như Windows, iOS và Mac OS, Linux là một hệ điều hành. Thực tế, một trong những nền tảng phổ biến nhất trên hành tinh này, Android, được hoạt động bởi hệ điều hành Linux. Hệ điều hành là phần mềm quản lý tất cả các tài nguyên phần cứng liên quan đến máy tính để bàn hoặc laptop. Đơn giản mà nói, hệ điều hành quản lý việc giao tiếp giữa phần mềm và phần cứng của bạn. Nếu thiếu hệ điều hành (OS), phần mềm sẽ không hoạt động.

Hệ điều hành Linux bao gồm một số thành phần khác nhau:

Bootloader - Phần mềm quản lý quá trình khởi động của máy tính. Đối với hầu hết người dùng, đây chỉ là một màn hình khởi động xuất hiện và sau đó biến mất để khởi động vào hệ điều hành.

Kernel - Đây là một phần duy nhất của hệ điều hành được gọi là "Linux". Kernel là trái tim của hệ thống và quản lý CPU, bộ nhớ và các thiết bị ngoại vi. Kernel là mức thấp nhất của hệ điều hành.

Init system - Đây là một hệ thống con khởi động không gian người dùng và có nhiệm vụ điều khiển các dịch vụ (daemon). Một trong những hệ thống khởi động phổ biến nhất là systemd, cũng là một trong những hệ thống khởi động gây tranh cãi nhất. Hệ thống khởi động quản lý quá trình khởi động, sau khi quá trình khởi động ban đầu được chuyển giao từ bootloader (ví dụ như GRUB hoặc GRand Unified Bootloader).

Deamon - Đây là các dịch vụ nền (in ấn, âm thanh, lập lịch, v.v.) khởi động trong quá trình khởi động hoặc sau khi bạn đăng nhập vào môi trường làm việc.

Graphical server - Đây là hệ thống con hiển thị đồ họa trên màn hình. Thông thường, nó được gọi là máy chủ X hoặc chỉ đơn giản là X.

Desktop environment - Đây là phần mà người dùng thực sự tương tác. Có nhiều môi trường làm việc để lựa chọn (GNOME, Cinnamon, Mate, Pantheon, Enlightenment, KDE, Xfce, v.v.). Mỗi môi trường làm việc bao gồm các ứng dụng tích hợp sẵn (như quản lý tệp, công cụ cấu hình, trình duyệt web và trò chơi).

Application - Môi trường làm việc không cung cấp đầy đủ ứng dụng. Giống như Windows và macOS, Linux cung cấp hàng nghìn phần mềm chất lượng cao mà dễ dàng tìm thấy và cài đặt.

1.2. Lý do sử dụng Linux

- Khả năng tùy chỉnh: Linux là một hệ điều hành mã nguồn mở, cho phép người dùng tùy chỉnh và thay đổi mã nguồn để đáp ứng nhu cầu cụ thể của hệ thống IoT. Điều này cho phép các nhà phát triển điều chỉnh và tối ưu hóa hệ thống theo yêu cầu của từng ứng dụng IoT.
- Cộng đồng lớn : Linux có một cộng đồng lớn và tích cực, với nhiều nhà phát triển và người dùng trên toàn thế giới. Điều này đảm bảo rằng luôn có sự hỗ trợ và tài liệu phong phú để giúp giải quyết các vấn đề và phát triển các ứng dụng IoT.
- Bảo mật: Linux có một hệ thống bảo mật mạnh mẽ, với các tính năng như quản lý người dùng và quyền hạn, cơ chế kiểm soát truy cập, và hỗ trợ mã hóa. Điều này quan trọng đối với các hệ thống IoT, vì chúng thường liên quan đến việc thu thập và xử lý dữ liệu nhạy cảm.
- Hiệu suất cao: Linux được thiết kế để hoạt động trên nhiều loại thiết bị và có khả năng tối ưu hóa hiệu suất. Điều này đặc biệt quan trọng trong các ứng dụng IoT, nơi tài nguyên có thể giới hạn và hệ thống cần phải hoạt động một cách hiệu quả.
- Đa nền tảng: Linux hỗ trợ nhiều kiến trúc và nền tảng phần cứng, từ những thiết bị nhúng nhỏ gọn đến các máy chủ mạnh mẽ. Điều này cho phép

phát triển hệ thống IoT trên nhiều loại thiết bị và tích hợp các thành phần khác nhau một cách linh hoạt.

- Tích hợp công nghệ mới: Linux có khả năng tích hợp và hỗ trợ các công nghệ mới nhất trong lĩnh vực IoT như giao thức mạng, giao tiếp không dây, và quản lý thiết bị. Điều này cho phép người dùng tận dụng các tiến bộ công nghệ

2. Ngôn ngữ sử dụng

1. Java

- Là một ngôn ngữ lập trình mạnh mẽ và phổ biến, được sử dụng rộng rãi trong phát triển ứng dụng IoT. Java có khả năng chạy trên nhiều nền tảng và cung cấp các thư viện và framework hỗ trợ mạnh mẽ cho việc xây dựng các ứng dụng IoT phức tạp.

2. HTML

- Là ngôn ngữ đánh dấu chính được sử dụng để xây dựng các giao diện người dùng cho ứng dụng web và ứng dụng IoT. HTML định nghĩa cấu trúc và các phần tử trên trang web, cho phép hiển thị thông tin và tương tác với người dùng. Với sự phát triển của IoT, HTML cũng được sử dụng để tạo các giao diện đơn giản cho các thiết bị IoT.

3. CSS

- Là ngôn ngữ định dạng dùng để điều chỉnh giao diện và trình bày cho các trang web và ứng dụng IoT. CSS cho phép xác định các thuộc tính như màu sắc, kích thước, định dạng văn bản và bố cục, tạo ra các giao diện hấp dẫn và dễ đọc cho người dùng.

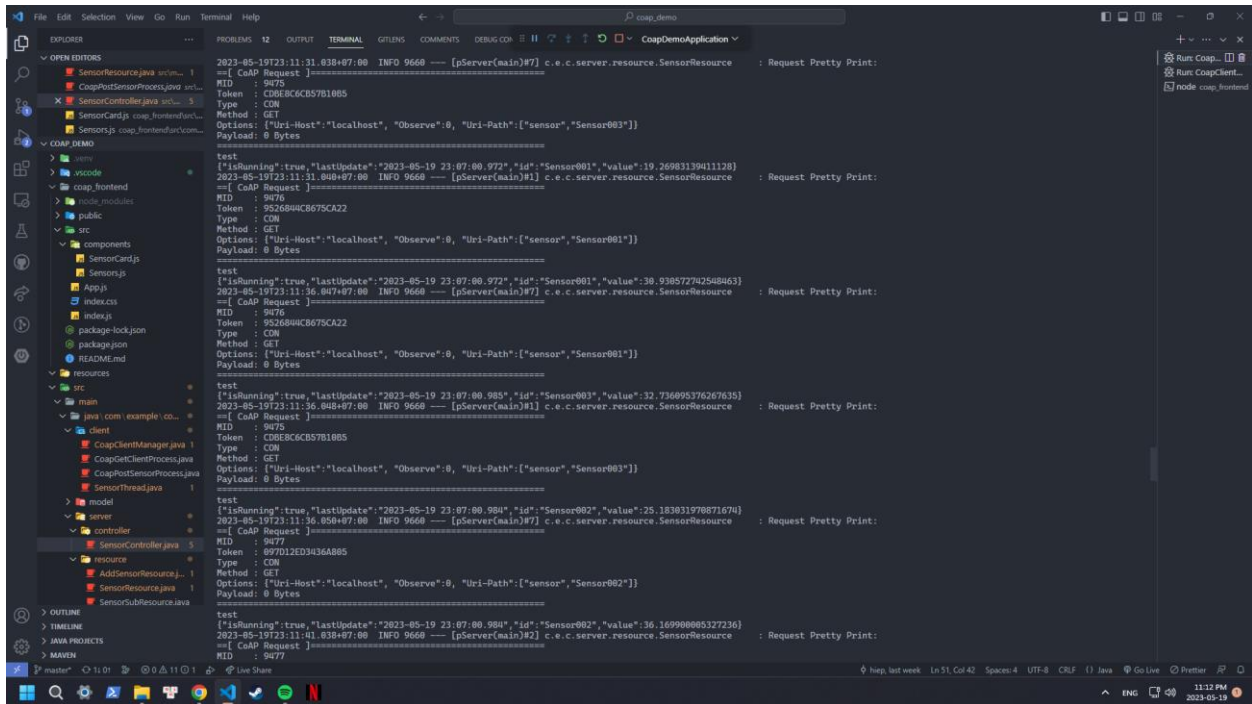
4. Javascript

- Là một ngôn ngữ lập trình phía máy khách phổ biến, thường được sử dụng để thêm tính năng tương tác động vào trang web và ứng dụng IoT. JavaScript cung cấp khả năng điều khiển các yếu tố trên trang, xử lý sự kiện, gửi và nhận dữ liệu từ máy chủ và thực hiện các hoạt động phức tạp như kiểm tra và xử lý dữ liệu từ các thiết bị IoT.

Chương V: Demo

1. Gateway (Server)

- Hiện thị các kết nối từ các node cảm biến



```
2023-05-19T23:11:31.838+07:00 INFO 9660 --- [pServer(main)37] c.e.c.server.resource.SensorResource : Request Pretty Print:
MID : 9475
Token : CDBEC6C85781085
Type : CON
Method : GET
Options: ["Uri-Host":"localhost", "Observe":0, "Uri-Path":["sensor", "Sensor003"]]
Payload: 0 Bytes

test
{"isRunning":true,"lastUpdate":"2023-05-19 23:07:00.972","id":"Sensor001","value":19.26983139411128}
2023-05-19T23:11:31.840+07:00 INFO 9660 --- [pServer(main)31] c.e.c.server.resource.SensorResource : Request Pretty Print:
MID : 9476
Token : 952686C8678CA22
Type : CON
Method : GET
Options: ["Uri-Host":"localhost", "Observe":0, "Uri-Path":["sensor", "Sensor001"]]
Payload: 0 Bytes

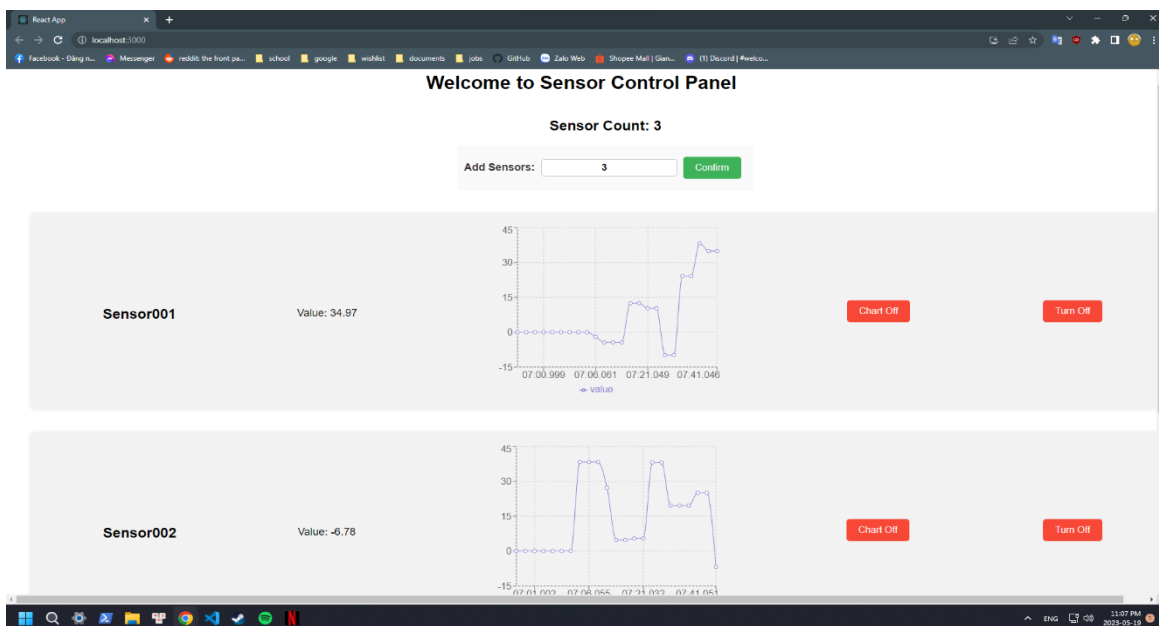
test
{"isRunning":true,"lastUpdate":"2023-05-19 23:07:00.972","id":"Sensor001","value":36.938577742548653}
2023-05-19T23:11:36.484+07:00 INFO 9660 --- [pServer(main)37] c.e.c.server.resource.SensorResource : Request Pretty Print:
MID : 9476
Token : 952686C8678CA22
Type : CON
Method : GET
Options: ["Uri-Host":"localhost", "Observe":0, "Uri-Path":["sensor", "Sensor001"]]
Payload: 0 Bytes

test
{"isRunning":true,"lastUpdate":"2023-05-19 23:07:00.983","id":"Sensor003","value":32.736895376267635}
2023-05-19T23:11:36.484+07:00 INFO 9660 --- [pServer(main)31] c.e.c.server.resource.SensorResource : Request Pretty Print:
MID : 9475
Token : CDBEC6C85781085
Type : CON
Method : GET
Options: ["Uri-Host":"localhost", "Observe":0, "Uri-Path":["sensor", "Sensor003"]]
Payload: 0 Bytes

test
{"isRunning":true,"lastUpdate":"2023-05-19 23:07:00.984","id":"Sensor002","value":25.183831978871674}
2023-05-19T23:11:36.484+07:00 INFO 9660 --- [pServer(main)37] c.e.c.server.resource.SensorResource : Request Pretty Print:
MID : 9477
Token : 897012ED3436A885
Type : CON
Method : GET
Options: ["Uri-Host":"localhost", "Observe":0, "Uri-Path":["sensor", "Sensor002"]]
Payload: 0 Bytes

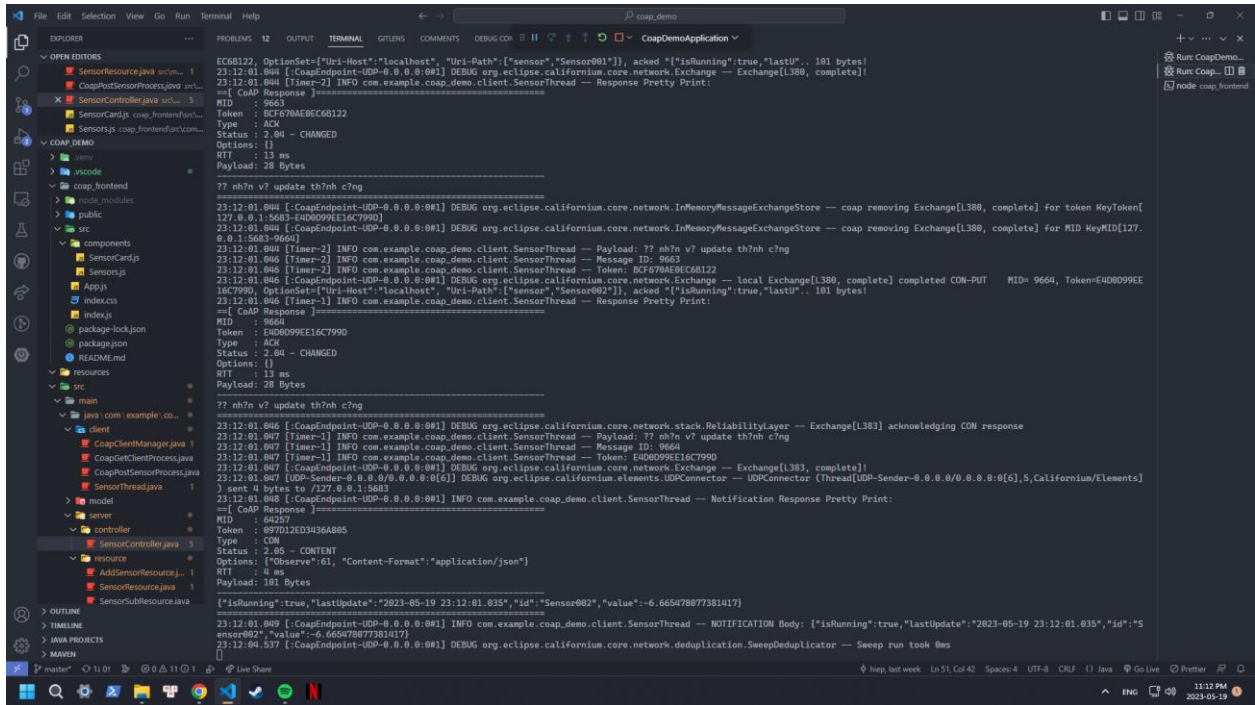
test
{"isRunning":true,"lastUpdate":"2023-05-19 23:07:00.984","id":"Sensor002","value":36.169980865327236}
```

- Nhận và hiển thị dữ liệu thông qua biểu đồ



2. Sensor (Client)

- Sensor tự động sinh dữ liệu rồi gửi lên Gateway



```
EC68122, OptionSet=[\"Uri-Host\":\"localhost\", \"Uri-Path\"=[\"sensor\", \"Sensor001\"]], acked \"[\"isRunning\":true, \"lastU\".. 101 bytes]
23:12:01.004 [CoapEndpoint-UDP-0.0.0.0:5683] DEBUG org.eclipse.californium.core.network.Exchange -- Exchange[L380, complete]
23:12:01.004 [Timer-2] INFO com.example.coap_demo.client.SensorThread -- Response Pretty Print:
=====
CoAP Response
=====
MID : 9663
Token : BCF678AEBC68122
Type : ACK
Status : 2.04 - CHANGED
Options : []
RTT : 13 ms
Payload: 28 Bytes

?? nh7n v? update th7nh c7ng
=====
23:12:01.004 [CoapEndpoint-UDP-0.0.0.0:5683] DEBUG org.eclipse.californium.core.network.InMemoryMessageExchangeStore -- coap removing Exchange[L380, complete] for token KeyToken[127.0.0.1:5683-4E4D099E16C7990]
23:12:01.004 [CoapEndpoint-UDP-0.0.0.0:5683] DEBUG org.eclipse.californium.core.network.InMemoryMessageExchangeStore -- coap removing Exchange[L380, complete] for MID KeyMID[127.0.0.1:5683-9663]
23:12:01.004 [Timer-2] INFO com.example.coap_demo.client.SensorThread -- Payload: ?? nh7n v? update th7nh c7ng
23:12:01.004 [Timer-2] INFO com.example.coap_demo.client.SensorThread -- Message ID: 9663
23:12:01.004 [Timer-2] INFO com.example.coap_demo.client.SensorThread -- Token: BCF678AEBC68122
23:12:01.004 [CoapEndpoint-UDP-0.0.0.0:5683] DEBUG org.eclipse.californium.core.network.Exchange -- local Exchange[L380, complete] completed CON-PUT MID= 9664, Token=4E4D099E16C7990, OptionSet=[\"Uri-Host\":\"localhost\", \"Uri-Path\"=[\"sensor\", \"Sensor002\"]], acked \"[\"isRunning\":true, \"lastU\".. 101 bytes]
23:12:01.004 [Timer-2] INFO com.example.coap_demo.client.SensorThread -- Response Pretty Print:
=====
CoAP Response
=====
MID : 9664
Token : 4E4D099E16C7990
Type : ACK
Status : 2.04 - CHANGED
Options : []
RTT : 13 ms
Payload: 28 Bytes

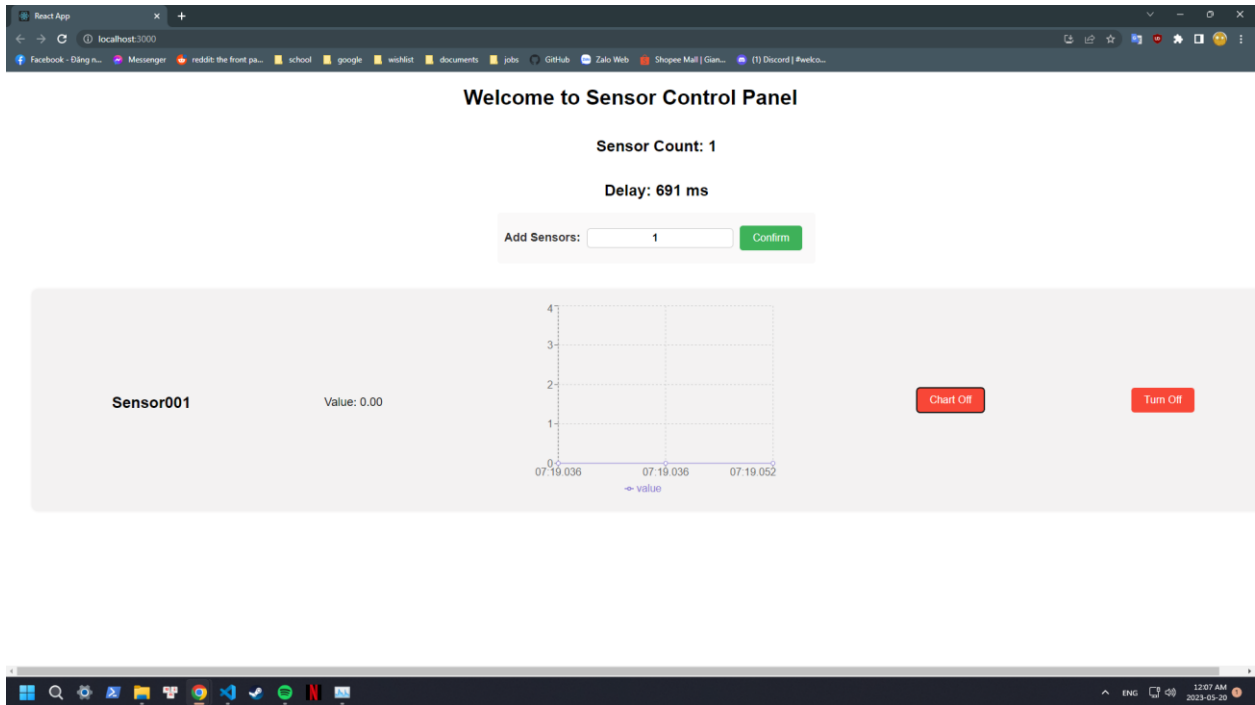
?? nh7n v? update th7nh c7ng
=====
23:12:01.004 [CoapEndpoint-UDP-0.0.0.0:5683] DEBUG org.eclipse.californium.core.network.stack.ReliabilityLayer -- Exchange[L383] acknowledging CON response
23:12:01.007 [Timer-1] INFO com.example.coap_demo.client.SensorThread -- Payload: ?? nh7n v? update th7nh c7ng
23:12:01.007 [Timer-1] INFO com.example.coap_demo.client.SensorThread -- Message ID: 9664
23:12:01.007 [Timer-1] INFO com.example.coap_demo.client.SensorThread -- Token: 4E4D099E16C7990
23:12:01.007 [CoapEndpoint-UDP-0.0.0.0:5683] DEBUG org.eclipse.californium.core.network.Exchange -- Exchange[L383, complete]
23:12:01.007 [UDP-Sender-0.0.0.0:5683] DEBUG org.eclipse.californium.elements.UDPConnector -- UDPConnector [Thread[UDP-Sender-0.0.0.0:5683(6), Californium/Elements]
] sent 4 bytes to /127.0.0.1:5683
23:12:01.008 [CoapEndpoint-UDP-0.0.0.0:5683] INFO com.example.coap_demo.client.SensorThread -- Notification Response Pretty Print:
=====
CoAP Response
=====
MID : 96257
Token : 697D12ED3436A8B5
Type : CON
Status : 2.05 - CONTENT
Options : [\"Observe\":61, \"Content-Format\":\"application/json\"]
RTT : 8 ms
Payload: 181 Bytes

[\"isRunning\":true, \"lastUpdate\":\"2023-05-19 23:12:01.035\", \"id\":\"Sensor002\", \"value\":-6.665478077381417]
23:12:01.009 [CoapEndpoint-UDP-0.0.0.0:5683] INFO com.example.coap_demo.client.SensorThread -- NOTIFICATION Body: [\"isRunning\":true, \"lastUpdate\":\"2023-05-19 23:12:01.035\", \"id\":\"Sensor002\", \"value\":-6.665478077381417]
23:12:04.537 [CoapEndpoint-UDP-0.0.0.0:5683] DEBUG org.eclipse.californium.core.network.duplication.SweepDuplicator -- Sweep run took 0ms
```

- Hiện thị các thông tin điều khiển (từ Gateway)



- Hiển thị, đánh giá số node tối đa (thông qua Sensor Count)
- Hiển thị, đánh giá delay của giao thức khi số node tăng lên (Thông qua Delay)



Chương VI: Kết luận

1. Tóm tắt kết quả

- Toàn bộ yêu cầu về chương trình đều được hoàn thành (trừ throughput)
- Giao diện tuy sơ sài nhưng vẫn hiển thị đủ thông tin cần thiết, dễ dùng
- Trình bày đầy đủ giao thức và thiết kế chương trình
- Không sử dụng thuật toán
- Ghi lại đầy đủ quá trình hoạt động của nhóm và vai trò từng thành viên
- Sử dụng github để lưu toàn bộ quá trình làm

2. Hạn chế

- Chương trình chưa hoàn thiện được phần Throughput
- Giao diện còn sơ sài
- Các yêu cầu mặc dù hoàn thành nhưng vẫn ở mức cơ bản

3. Hướng phát triển trong tương lai

- Hoàn thiện nốt Throughput
- Lập trình để kết nối được nhiều client tới 1 server
- Lập trình để có thể vẽ các loại biểu đồ khác
- Chỉnh sửa giao diện sao cho đẹp và tối ưu hơn
- Lập trình sử dụng các giao thức khác ngoài coAP
- Thêm tính năng đo nhiệt độ ở một nơi bất kì mình muốn

4. Quá trình làm việc của nhóm

- 21h thứ 5 (16/3): Tìm hiểu lý thuyết và các kiến thức liên quan
- 21h thứ 5 (23/3): Đưa ra thiết kế tổng quan về chương trình
- 21h thứ 5 (30/3): Tiến hành phân chia công việc
- 21h thứ 5 (6/4): Báo cáo tiến độ công việc (chưa hoàn thành, yêu cầu tìm hiểu thêm)

- 21h thứ 5 (13/4): Báo cáo tiến độ công việc (tiến hành code phần Gateway, tuy nhiên chưa hoàn thành)
- 21h thứ 5 (20/4): Báo cáo tiến độ công việc (hoàn thành phần Gateway, tiến hành hoàn thành phần Sensor)
- 21h thứ 5 (27/4): Không họp, nghỉ lễ
- 21h thứ 5 (4/5): Báo cáo tiến độ công việc (hoàn thành phần Sensor, tiến hành viết báo cáo)
- 21h thứ 5 (11/5): Báo cáo tiến độ công việc (hoàn thành 90%), tiến hành kiểm tra lỗi
- 21h thứ 5(18/5): Họp tổng kết, review lại code, testing, ...

5. Vai trò thành viên nhóm

Họ và tên (MSSV)	Vai trò	Đóng góp
Đỗ Lê Mạnh Hùng (20020322)	Front-end, báo cáo (I,III,V,VI)	20%
Nguyễn Ngọc Hiệp (20021349)	Front-end, back-end (Gateway)	20%
Nguyễn Tuấn Nam (20021398)	Back-end (Gateway, Sensor)	20%
Trần Mạnh Sơn (20021431)	Back-end (Sensor, Gateway)	20%
Bùi Minh Sơn (20021427)	Back-end (Sensor), báo cáo (II,IV)	20%

Chương VII: Tài liệu tham khảo

[What is Linux? - Linux.com](#)

[CoAP - Constrained Application Protocol - Radiocrafts](#)

[Tìm hiểu về Wireless Sensor Network \(WSN\) - QuanTriMang.com](#)

[Wireless Sensor Network \(WSN\) - GeeksforGeeks](#)

[Wireless sensor network - Wikipedia](#)

[WSN là gì? Đặc điểm và ứng dụng của WSN | BKHOST](#)

[Những điều cần biết về giao thức CoAP, sự khác biệt giữa CoAP và MQTT \(bkaii.com.vn\)](#)

[Constrained Application Protocol - Wikipedia](#)

[CoAP - Constrained Application Protocol - Radiocrafts](#)

[CoAP Protocol: What is Meaning, Architecture and Function ? \(wallarm.com\)](#)

[Tìm hiểu về HTTP \(HyperText Transfer Protocol\) \(viblo.asia\)](#)

[Hypertext Transfer Protocol – Wikipedia tiếng Việt](#)

[Học Spring Boot bắt đầu từ đâu? \(viblo.asia\)](#)

[Giới thiệu về Spring Boot. Spring Boot là gì? | TopDev](#)

[LineChart \(JavaFX 8\) \(oracle.com\)](#)

[Biểu đồ đường \(Line Chart\) trong JavaFX \(teamvietdev.com\)](#)

[JavaFX - Line Chart \(tutorialspoint.com\)](#)

[Wireless Sensors Gateway Manufacturer and Wholesaler \(pressac.com\)](#)

[Sensor Network Gateway - Sensor Development International \(sdinternational.nl\)](#)

[californium.tools/cf-browser at main · eclipse-californium/californium.tools · GitHub](#)

LỜI KẾT

*Chúng em xin chân thành cảm ơn thầy **TS. Nguyễn Ngọc Tân** đã tận tình chỉ bảo để chúng em có thể hoàn thành sản phẩm đúng hạn ạ. Do kiến thức và kỹ năng còn nhiều hạn chế, trong quá trình làm không thể tránh khỏi sai sót, mong thầy bỏ qua cho chúng em ạ.*

Chúc thầy một ngày làm việc tràn đầy năng lượng!