

Trường Đại Học Công Nghệ
ĐẠI HỌC QUỐC GIA HÀ NỘI



BÁO CÁO
MÔN: LẬP TRÌNH MẠNG
(AMQP)

Nhóm 4

Lớp: 2223II_INT3304_20

Giảng viên: TS. Nguyễn Ngọc Tân

Năm học 2022 - 2023

Mục lục

CHƯƠNG 1: GIỚI THIỆU	3
<i>YÊU CẦU</i>	3
<i>CÁC THÀNH PHẦN CHÍNH</i>	3
CHƯƠNG 2: CÀI ĐẶT CHƯƠNG TRÌNH	4
1. Hiển thị thông tin các cảm biến thực hiện kết nối với gateway	4
2. Sinh dữ liệu cho các cảm biến và gửi dữ liệu lên gateway	7
3. Hiển thị dữ liệu nhận được từ phía server.....	9
4. Hiển thị thông tin điều khiển tại các cảm biến	11
5. Đánh giá hiệu năng của giao thức	14

CHƯƠNG 1: GIỚI THIỆU

YÊU CẦU

- **Giao thức chính:** AMQP
- **Ngôn ngữ:** C++, Java, JS
- **Công nghệ:** RabbitMQ, NodeJS, ReactJS
- **Tài liệu tham khảo:**
 - JAVA: <https://www.rabbitmq.com/tutorials/tutorial-one-java.html>
 - JAVASCRIPT: <https://www.rabbitmq.com/tutorials/tutorial-one-javascript.html>
- **Server RabbitMQ:**
 - URL: g4amqp.freedomdns.org
 - user: g4amqp
 - password: g4amqp
 - port: 5672
- **RabbitMQ Management:**
 - URL: <https://rabbitmq.g4amqp.freedomdns.org>
 - user: g4amqp
 - password: g4amqp

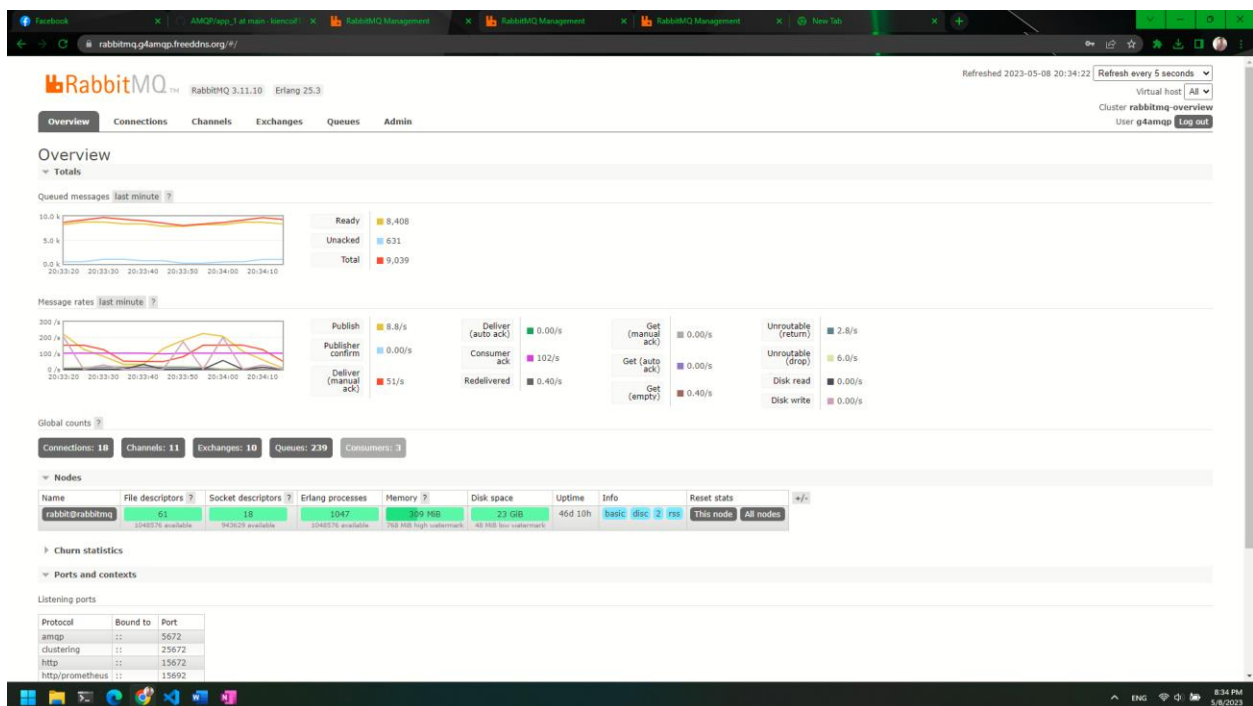
CÁC THÀNH PHẦN CHÍNH

- Chương trình hiển thị thông tin các cảm biến thực hiện kết nối với gateway.
- 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 từ gateway (sử dụng các biểu đồ line chart, bar chart, pie chart để hiển thị)
- Chương trình hiển thị thông tin điều khiển tại các cảm biến (được gửi từ gateway)
- Chương trình hiển thị đánh giá hiệu năng của giao thức: Throughput, delay, v.v. khi số lượng node cảm biến tăng lên.

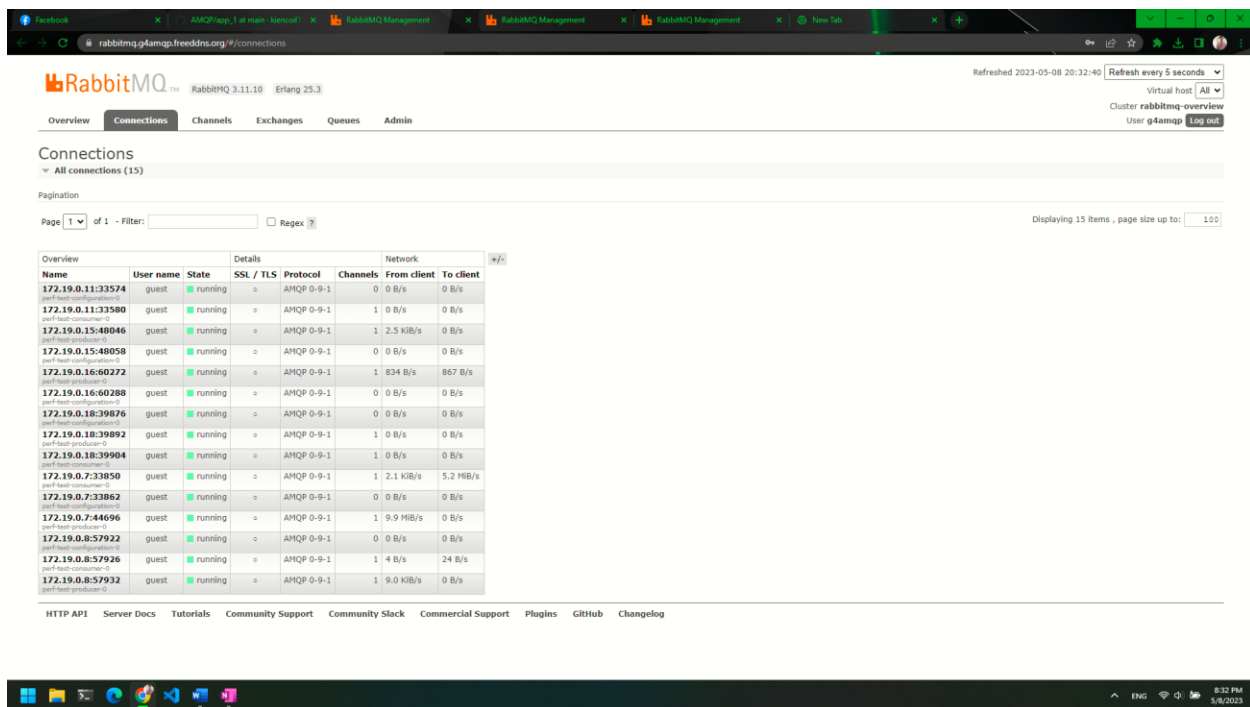
CHƯƠNG 2: CÀI ĐẶT CHƯƠNG TRÌNH

1. Hiển thị thông tin các cảm biến thực hiện kết nối với gateway

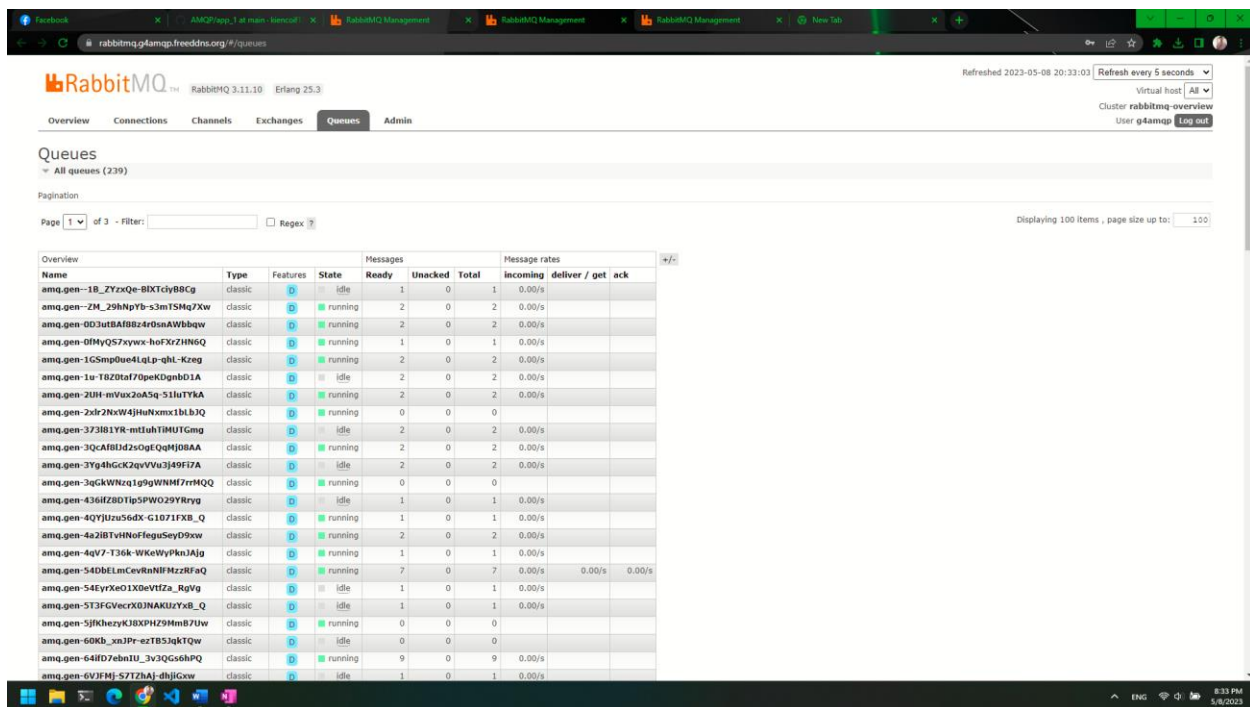
- Connections: <https://rabbitmq.g4amqp.freedomdns.org/#/connections>
- Queues: <https://rabbitmq.g4amqp.freedomdns.org/#/queues>
- Channels: <https://rabbitmq.g4amqp.freedomdns.org/#/channels>



Hình 1.1 Overview



Hình 1.2 Connections



Hình 1.3 Queues

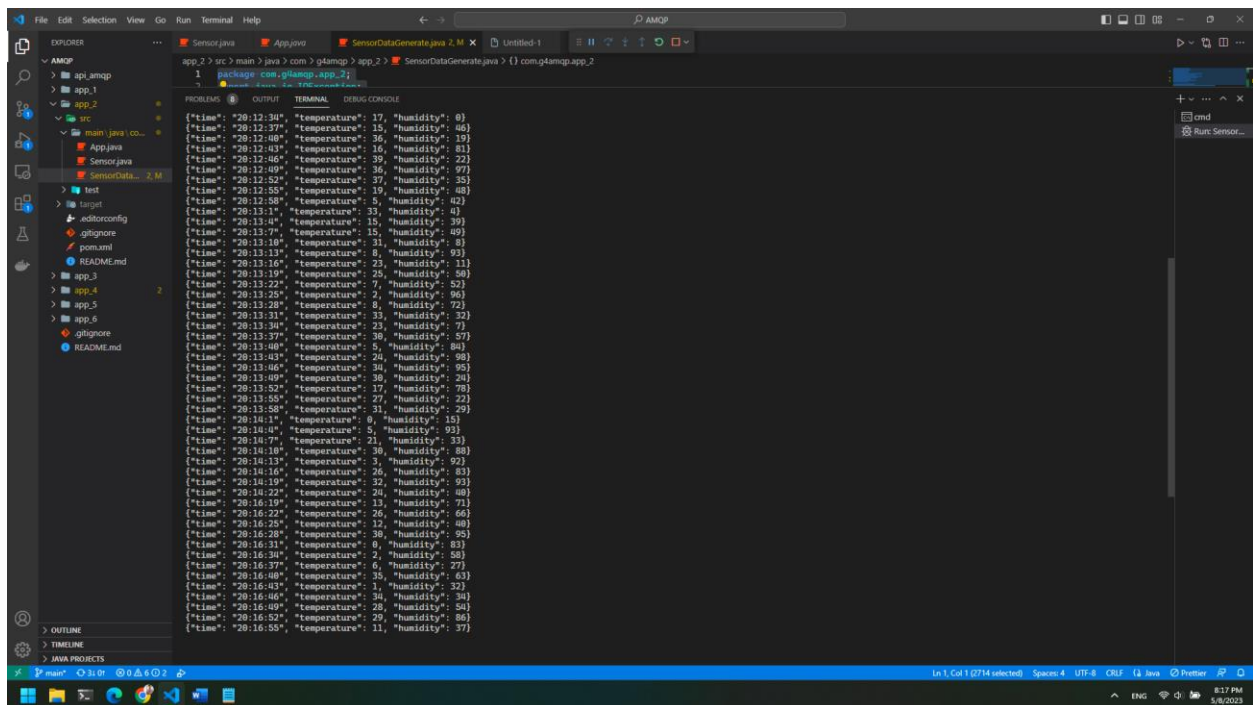
The screenshot displays the RabbitMQ Management interface at the URL `rabbitmq-g4mapg.freemdns.org/#/channels`. The top navigation bar includes links for Overview, Connections, Channels (active), Exchanges, Queues, and Admin. On the right, there's a refresh button set to "Refresh every 5 seconds" and a virtual host selector set to "All". Below the navigation bar, the main heading is "Channels" with a sub-link for "All channels (11)". A pagination control shows "Page 1 of 1 - Filter:". The central part of the screen contains a table with two tabs: "Overview" (selected) and "Details". The table lists 11 channels, each with columns for Channel name, User name, Mode, State, Unconfirmed, Prefetch, Unsacked, Message rates (publish, confirm, unrouteable/drop, deliver/get, ack). The bottom of the interface features a horizontal menu with links: HTTP API, Server Docs, Tutorials, Community Support, Community Slack, Commercial Support, Plugins, GitHub, and Changelog.

Channel	User name	Mode	State	Unconfirmed		Prefetch	Unsacked	Message rates				
								publish	confirm	unrouteable (drop)	deliver / get	ack
172.19.0.11:32580 (1)	guest		running	0	0						0.00/s	0.00/s
172.19.0.14:50696 (1)	guest		running	0	5	5					0.20/s	0.00/s
172.19.0.14:50702 (1)	guest		idle	0	0	0		1.8/s	0.00/s	0.00/s		
172.19.0.15:48046 (1)	guest		running	0	0	0		11/s	0.00/s	11/s		
172.19.0.16:60272 (1)	guest		running	0	0	0		4.8/s	0.00/s	0.00/s		
172.19.0.18:39892 (1)	guest		idle	0	0	0		0.00/s	0.00/s	0.00/s		
172.19.0.18:39904 (1)	guest		running	0	0	0					0.00/s	0.00/s
172.19.0.7:32580 (1)	guest		running	0	50	50					51/s	51/s
172.19.0.7:44696 (1)	guest		idle	0	0	0		101/s	0.00/s	0.00/s		
172.19.0.8:57926 (1)	guest		running	0	2000	259					101/s	51/s
172.19.0.8:57932 (1)	guest		running	0	0	0		101/s	0.00/s	0.00/s		

Hình 1.4 Channels

2. Sinh dữ liệu cho các cảm biến và gửi dữ liệu lên gateway

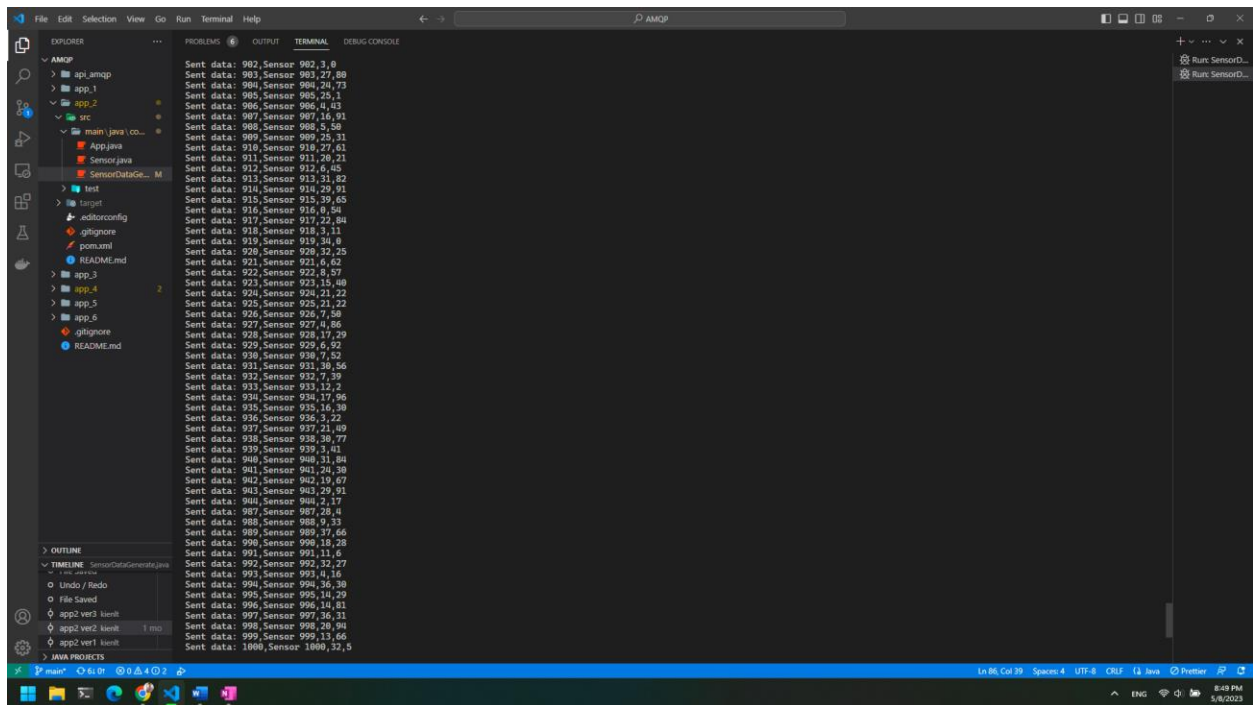
- Ngôn ngữ: Java.
- Chương trình random ra dữ liệu cảm biến: Nhiệt độ, Độ ẩm, ...
- Chương trình kết nối với server AMQP(RabbitMQ) và gửi dữ liệu đó lên server.
- Chương trình random dữ liệu cảm biến (nhiệt độ, độ ẩm) và gửi dữ liệu lên RabbitMQ



The screenshot shows an IDE with a project named 'api_amqp'. The 'SensorDataGenerate.java' file is open, displaying a loop that generates random temperature and humidity data and sends it to a RabbitMQ server. The terminal output shows the following data points:

Time	Temperature	Humidity
20:12:34	17	0
20:12:37	15	06
20:12:40	36	19
20:12:43	16	01
20:12:46	39	22
20:12:49	36	97
20:12:52	27	35
20:12:55	19	08
20:12:58	5	42
20:13:01	33	40
20:13:04	15	39
20:13:07	15	49
20:13:10	31	85
20:13:13	8	93
20:13:16	23	13
20:13:19	25	50
20:13:22	7	52
20:13:25	2	96
20:13:28	1	72
20:13:31	33	32
20:13:34	23	7
20:13:37	30	57
20:13:40	5	80
20:13:43	24	90
20:13:46	30	26
20:13:49	17	70
20:13:52	27	22
20:13:55	31	29
20:14:01	9	15
20:14:04	5	93
20:14:07	21	33
20:14:10	38	00
20:14:13	3	92
20:14:16	26	03
20:14:19	32	03
20:14:22	24	00
20:14:25	13	73
20:14:28	26	66
20:14:31	12	00
20:14:34	20	95
20:14:37	6	83
20:14:40	2	50
20:14:43	0	77
20:14:46	25	63
20:14:49	1	32
20:14:52	24	30
20:14:55	28	50
20:14:58	29	86
20:15:01	11	37

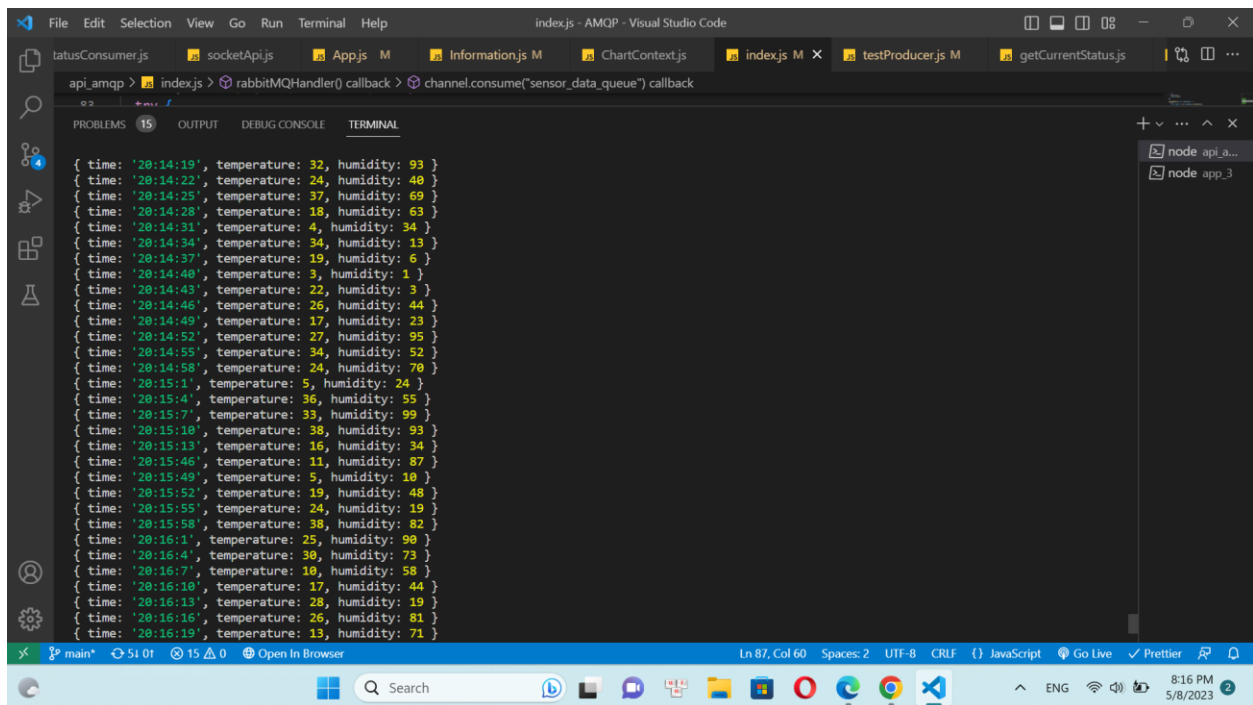
Hình 2.1 API gửi lên server AMQP(RabbitMQ)



Hình 2.2 Giải lập 1000 cảm biến cùng gửi lên server AMQP

3. Hiện thị dữ liệu nhận được từ phía server

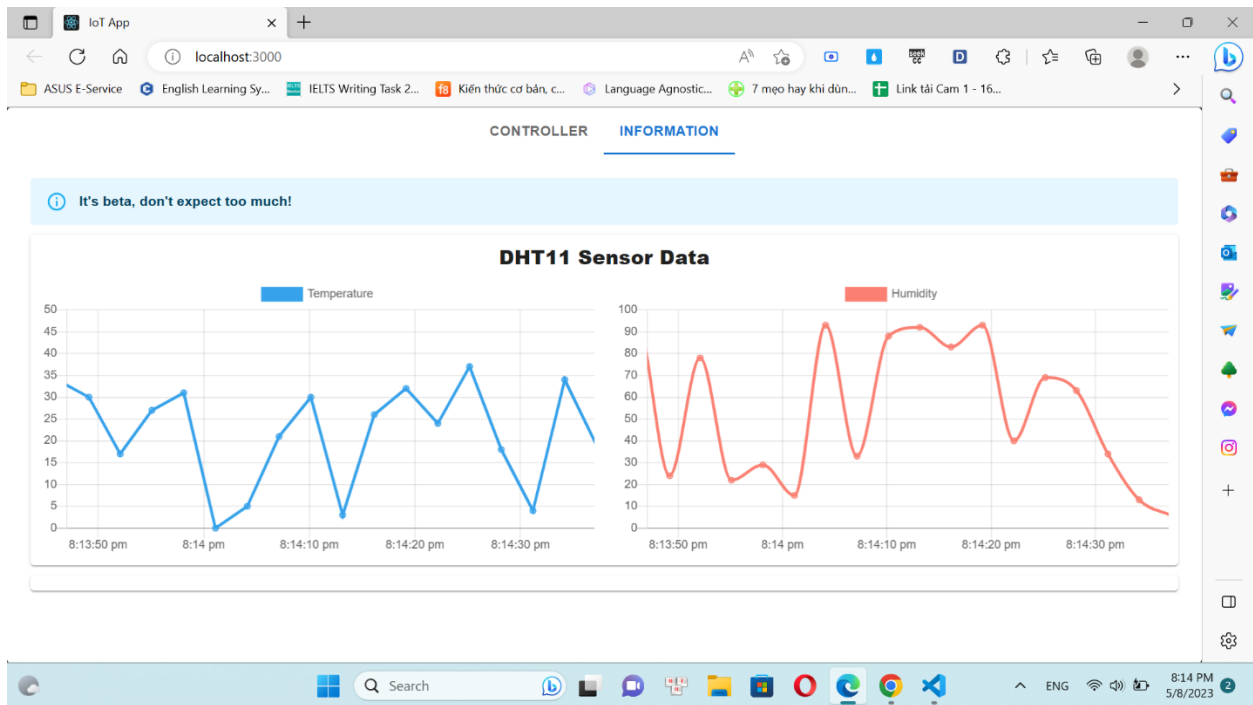
- Ngôn ngữ: JavaScript, ReactJS
- Kết nối với server AMQP(RabbitMQ) và nhận dữ liệu từ server.
- Lấy dữ liệu từ AMQP server (nhiệt độ, độ ẩm) và hiển thị bằng biểu đồ
- Sử dụng các biểu đồ line chart, bar chart, pie chart, ... để hiển thị
- Chương trình mô phỏng cảm biến (bật/ tắt đèn)



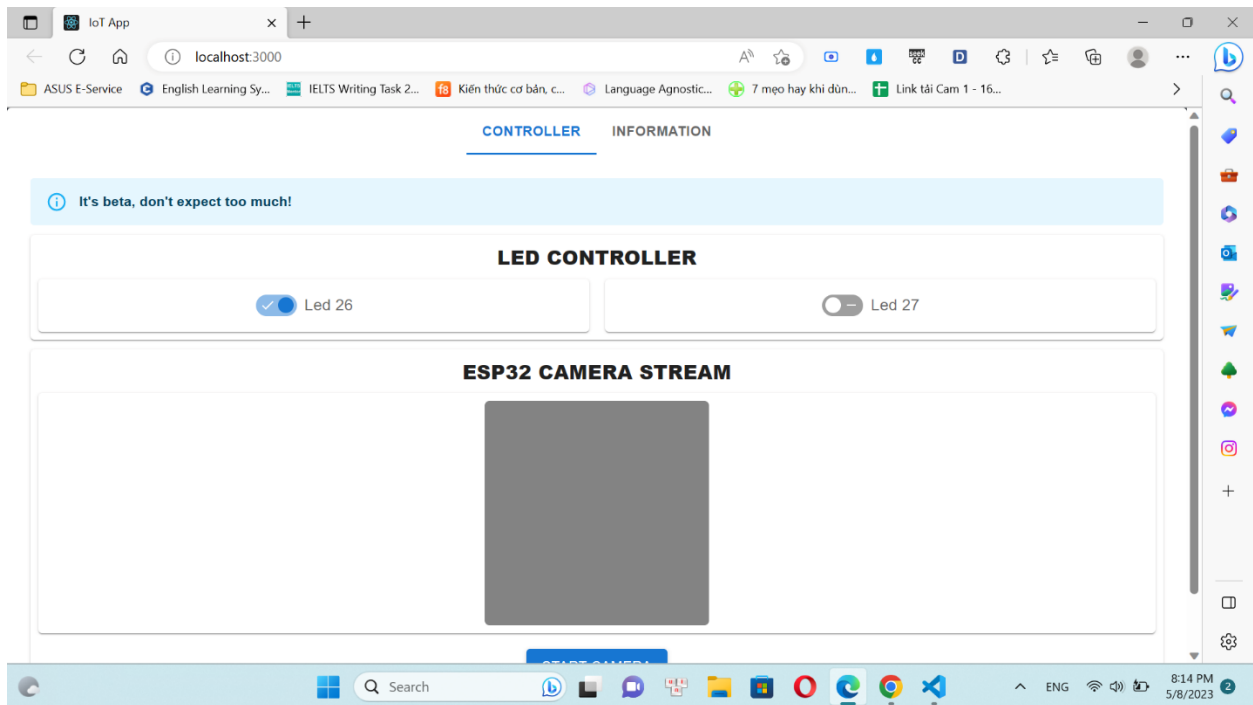
```
api_amqp > index.js > rabbitMQHandler() callback > channel.consume("sensor_data_queue") callback
{ time: '20:14:19', temperature: 32, humidity: 93 }
{ time: '20:14:22', temperature: 24, humidity: 40 }
{ time: '20:14:25', temperature: 37, humidity: 69 }
{ time: '20:14:28', temperature: 18, humidity: 63 }
{ time: '20:14:31', temperature: 4, humidity: 34 }
{ time: '20:14:34', temperature: 34, humidity: 13 }
{ time: '20:14:37', temperature: 19, humidity: 6 }
{ time: '20:14:40', temperature: 3, humidity: 1 }
{ time: '20:14:43', temperature: 22, humidity: 3 }
{ time: '20:14:46', temperature: 26, humidity: 44 }
{ time: '20:14:49', temperature: 17, humidity: 23 }
{ time: '20:14:52', temperature: 27, humidity: 95 }
{ time: '20:14:55', temperature: 34, humidity: 52 }
{ time: '20:14:58', temperature: 24, humidity: 70 }
{ time: '20:15:1', temperature: 5, humidity: 24 }
{ time: '20:15:4', temperature: 36, humidity: 55 }
{ time: '20:15:7', temperature: 33, humidity: 99 }
{ time: '20:15:10', temperature: 38, humidity: 93 }
{ time: '20:15:13', temperature: 16, humidity: 34 }
{ time: '20:15:46', temperature: 11, humidity: 87 }
{ time: '20:15:49', temperature: 5, humidity: 10 }
{ time: '20:15:52', temperature: 19, humidity: 48 }
{ time: '20:15:55', temperature: 24, humidity: 19 }
{ time: '20:15:58', temperature: 38, humidity: 82 }
{ time: '20:16:1', temperature: 25, humidity: 90 }
{ time: '20:16:4', temperature: 30, humidity: 73 }
{ time: '20:16:7', temperature: 10, humidity: 58 }
{ time: '20:16:10', temperature: 17, humidity: 44 }
{ time: '20:16:13', temperature: 28, humidity: 19 }
{ time: '20:16:16', temperature: 26, humidity: 81 }
{ time: '20:16:19', temperature: 13, humidity: 71 }
```

Hình 3.1 Lấy dữ liệu từ App 2 cung cấp thông qua server AMQP

Queue: "sensor_data_queue"



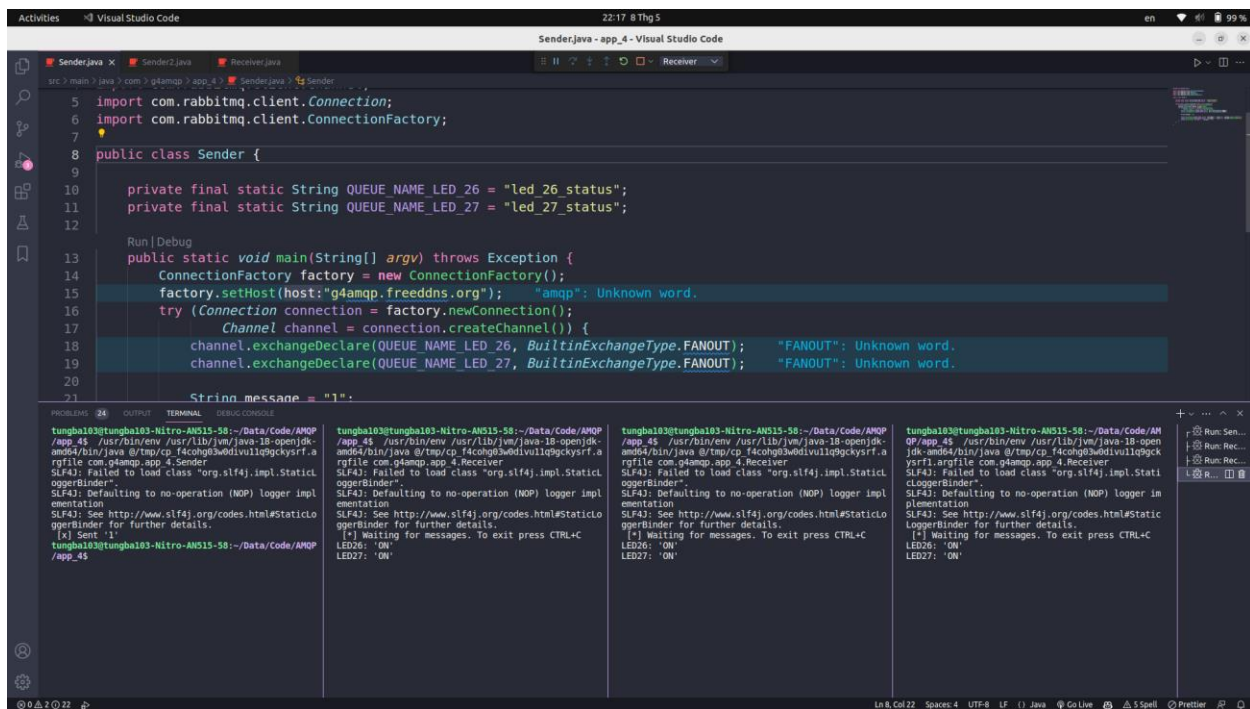
Hình 3.2 Vẽ biểu đồ dựa vào dữ liệu đã nhận



Hình 3.3 Mô phỏng cảm biến (tắt/ bật đèn) và gửi dữ liệu tới server AMQP

4. Hiện thị thông tin điều khiển tại các cảm biến

- Ngôn ngữ: Java.
- Chương trình này cũng như chương trình của phần 2, ...
- Chương trình sẽ kết nối với AMQP Server (RabbitMQ) và lắng nghe, nhận dữ liệu từ Server
- VD: Có 2 đèn LED 26 và LED 28 sẽ có 2 trạng thái là bật (1) và tắt(0). Chương trình sẽ lắng nghe xem trạng thái của các đèn có thay đổi hay không nếu thay đổi thì sẽ tắt hoặc bật đèn.



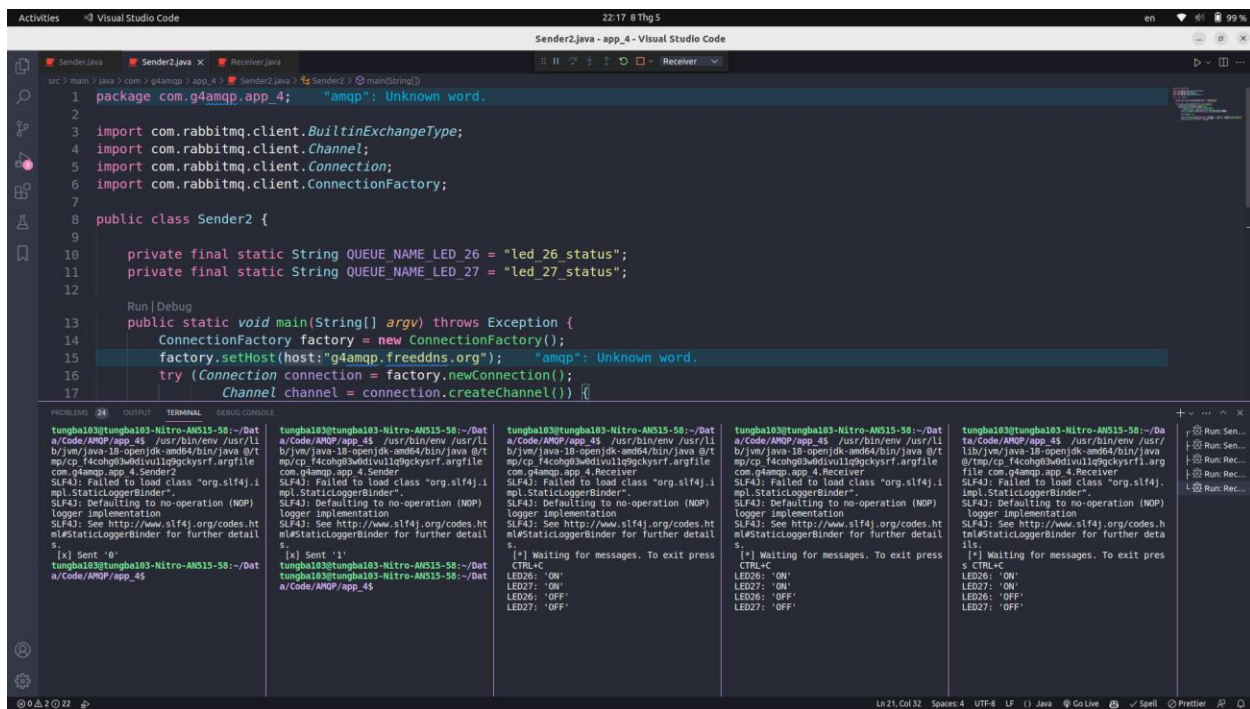
The screenshot displays the Visual Studio Code interface with the 'Sender.java' file open. The code defines a 'Sender' class that connects to an AMQP server and declares two queues, 'led_26_status' and 'led_27_status'. The main method sets up the connection and declares the queues. The terminal output shows the execution of the program, indicating that the queues were successfully declared and the program is waiting for messages.

```
Sender.java - app_4 - Visual Studio Code
Sender.java
5 import com.rabbitmq.client.Connection;
6 import com.rabbitmq.client.ConnectionFactory;
7
8 public class Sender {
9
10     private final static String QUEUE_NAME_LED_26 = "led_26_status";
11     private final static String QUEUE_NAME_LED_27 = "led_27_status";
12
13     public static void main(String[] argv) throws Exception {
14         ConnectionFactory factory = new ConnectionFactory();
15         factory.setHost(host:"g4amqp.freedomdns.org");
16         try (Connection connection = factory.newConnection();
17             Channel channel = connection.createChannel()) {
18             channel.exchangeDeclare(QUEUE_NAME_LED_26, BuiltinExchangeType.FANOUT);
19             channel.exchangeDeclare(QUEUE_NAME_LED_27, BuiltinExchangeType.FANOUT);
20
21             String message = "1";
22         }
23     }
24 }
```

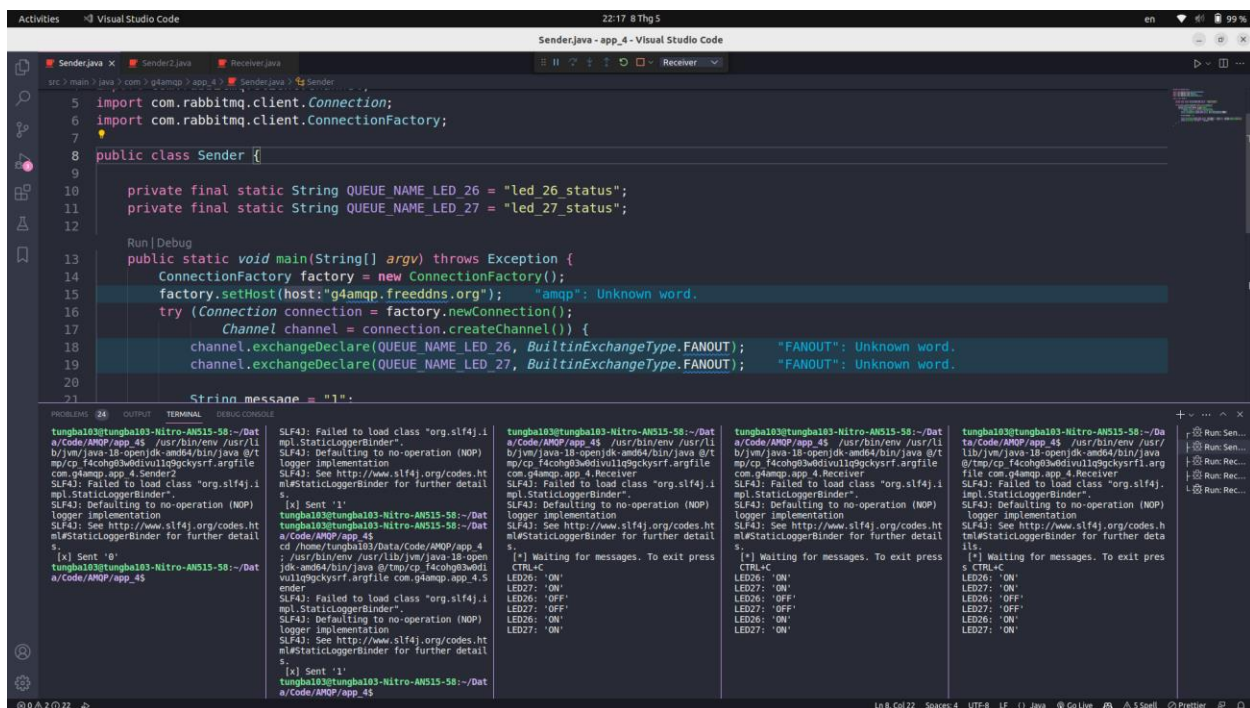
Terminal Output:

```
tungba183@tungba183-Nitro-AN515-S8:~/Data/Code/AMQP/app_4$ ./src/bin/env ./src/lib/jvm/java-18-openjdk-amd64/bin/java @/tmp/cp.f4c0hg3wddivullgqckysrf.a rgfile com.g4amqp.app.4.Sender
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
[+] Sent: '1'
tungba183@tungba183-Nitro-AN515-S8:~/Data/Code/AMQP/app_4$
```

Hình 4.1 Bật đèn với Sender 1



Hình 4.2 Tắt đèn với Sender2

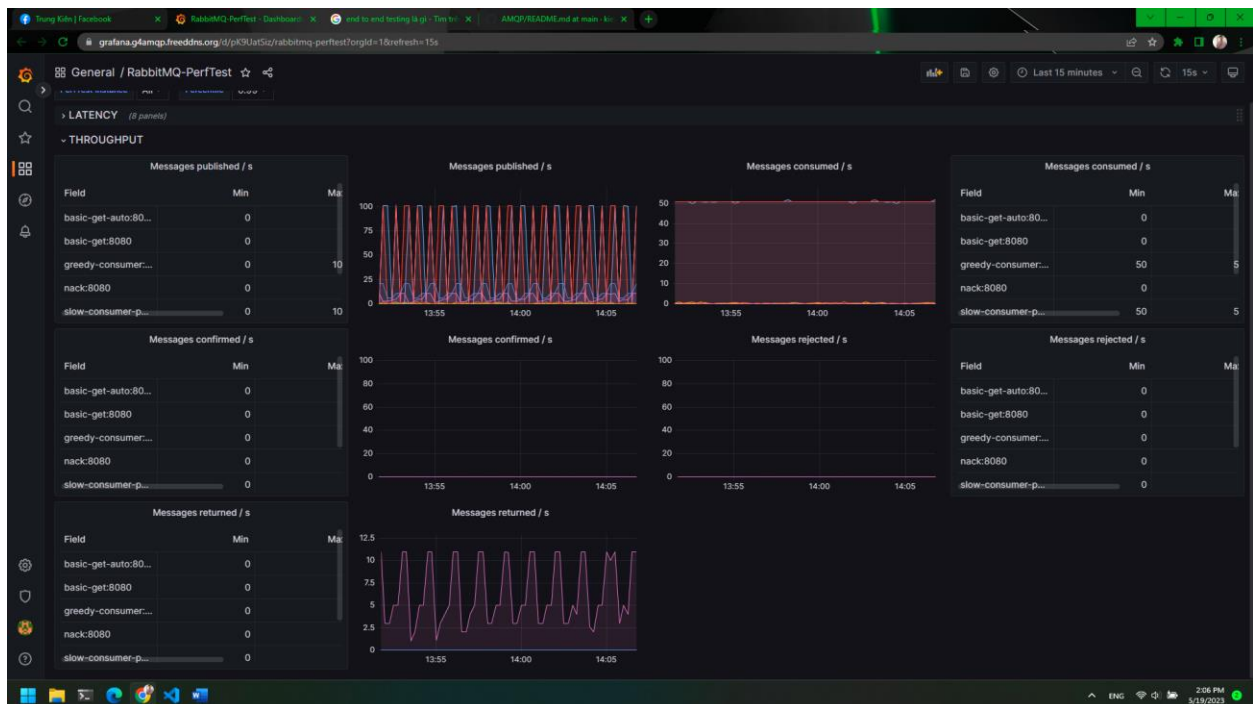


Hình 4.3 Bật đèn lại với Sender 1

5. Đánh giá hiệu năng của giao thức



Hình 5.0 Overview



Hình 5.0 Overview

1. Basic Get Test

Phương pháp Basic Get Test trong AMQP thường được sử dụng để kiểm tra tính đúng đắn và hiệu quả của hệ thống AMQP trong việc lấy các tin nhắn từ hàng đợi. Khi sử dụng phương pháp này, một số lượng lớn các tin nhắn sẽ được đưa vào hàng đợi, sau đó các nhà phát triển sẽ sử dụng tính năng Basic Get để lấy các tin nhắn từ hàng đợi.

Config Test:

```
environment:
  URI: "amqp://guest:guest@rabbitmq:5672/%2f"
  QUEUE: basic-get
  ROUTING_KEY: basic-get
  VARIABLE_RATE: "1:1,0:30"
  POLLING: "true"
  POLLING_INTERVAL: 5000
  AUTOACK: "false"
  SERVERS_STARTUP_TIMEOUT: &startup_timeout 60
  METRICS_PROMETHEUS: "true"
```

2. Basic-get-auto Test

Là một phương thức tương tự với basic-get, nhưng consumer sẽ tự động yêu cầu tin nhắn tiếp theo sau khi lấy tin nhắn trước đó.

Config Test:

```
environment:
  URI: "amqp://guest:guest@rabbitmq:5672/%2f"
  QUEUE: basic-get
  ROUTING_KEY: basic-get
  PRODUCERS: 0
  POLLING: "true"
  POLLING_INTERVAL: 5000
  AUTOACK: "true"
  SERVERS_STARTUP_TIMEOUT: *startup_timeout
  METRICS_PROMETHEUS: "true"
```

3. Greedy consumer Test

Ý tưởng của phương pháp này là sử dụng một consumer (người tiêu dùng) tham lam để lấy các tin nhắn từ một hàng đợi (queue) trong hệ thống. Consumer này sẽ luôn lấy tin nhắn đầu tiên trong hàng đợi và xử lý nó mà không quan tâm đến các tin nhắn tiếp theo. Khi tin nhắn đầu tiên đã được xử lý, consumer sẽ tiếp tục lấy tin nhắn tiếp theo.

Config Test:

```
environment:
  URI: "amqp://guest:guest@rabbitmq:5672/%2F"
  QUEUE: greedy-consumer
  ROUTING_KEY: greedy-consumer
  VARIABLE_RATE: "100:20,0:20"
  CONSUMER_RATE: 50
  QOS: 2000
  AUTOACK: "false"
  SERVERS_STARTUP_TIMEOUT: *startup_timeout
  METRICS_PROMETHEUS: "true"
```

4. Publisher-confirms Test

Publisher Confirms Test là một phương pháp kiểm tra tính năng Publisher Confirms của hệ thống AMQP. Phương pháp này thường được sử dụng để đảm bảo tính đúng đắn và tin cậy của hệ thống.

Config Test:

```
environment:
  URI: "amqp://guest:guest@rabbitmq:5672/%2f"
  QUEUE: publisher-confirms
  ROUTING_KEY: publisher-confirms
  AUTOACK: "true"
  VARIABLE_RATE: "12:30,25:30,50:30,100:30"
  CONFIRM: 1
  CONFIRM_TIMEOUT: 1
  SERVERS_STARTUP_TIMEOUT: *startup_timeout
  METRICS_PROMETHEUS: "true"
```


5. Slow-consumer-persistent Test

SCP test được sử dụng để đánh giá khả năng xử lý thông điệp của một consumer khi tốc độ nhận thông điệp chậm hơn tốc độ sản xuất của producer. Trong trường hợp này, consumer phải lưu trữ các thông điệp trong hàng đợi và xử lý chúng khi tốc độ nhận được tăng lên.

Config Test:

```
environment:
  URI: "amqp://guest:guest@rabbitmq:5672/%2f"
  QUEUE: ha3-slow-consumer-persistent
  ROUTING_KEY: slow-consumer-persistent
  QUEUE_ARGS: x-max-length=10000
  FLAG: persistent
  AUTO_DELETE: "false"
  SIZE: 51200
  VARIABLE_RATE: "100:20,0:20"
  CONSUMER_RATE: 50
  QOS: 50
  AUTOACK: "false"
  SERVERS_STARTUP_TIMEOUT: *startup_timeout
  METRICS_PROMETHEUS: "true"
```

6. Nack Test

NACK test được sử dụng để kiểm tra khả năng chịu lỗi của hệ thống trong trường hợp một consumer không thể xử lý một hoặc nhiều thông điệp. Kiểm tra này được thực hiện bằng cách đưa vào hệ thống một lượng lớn các thông điệp, sau đó cho phép các consumer xử lý các thông điệp đó. Nếu một consumer không thể xử lý một thông điệp, nó sẽ gửi một NACK tới exchange để đánh dấu thông điệp đó là không thể xử lý.

Config Test:

```
command: /bin/bash -c "while true; do bin/runjava com.rabbitmq.perf.PerfTest; sleep 10; done"
environment:
  TIME: 60
  URI: "amqp://guest:guest@rabbitmq:5672/%2f"
  VARIABLE_RATE: "1:10,0:20"
  QUEUE: nack
  QUEUE_ARGS: x-max-length=100
  ROUTING_KEY: nack
  AUTOACK: "false"
  NACK: "true"
  QOS: 5
  CONSUMER_LATENCY: 3000000
  SERVERS_STARTUP_TIMEOUT: *startup_timeout
  METRICS_PROMETHEUS: "true"
```

7. Unroutable-Return Test

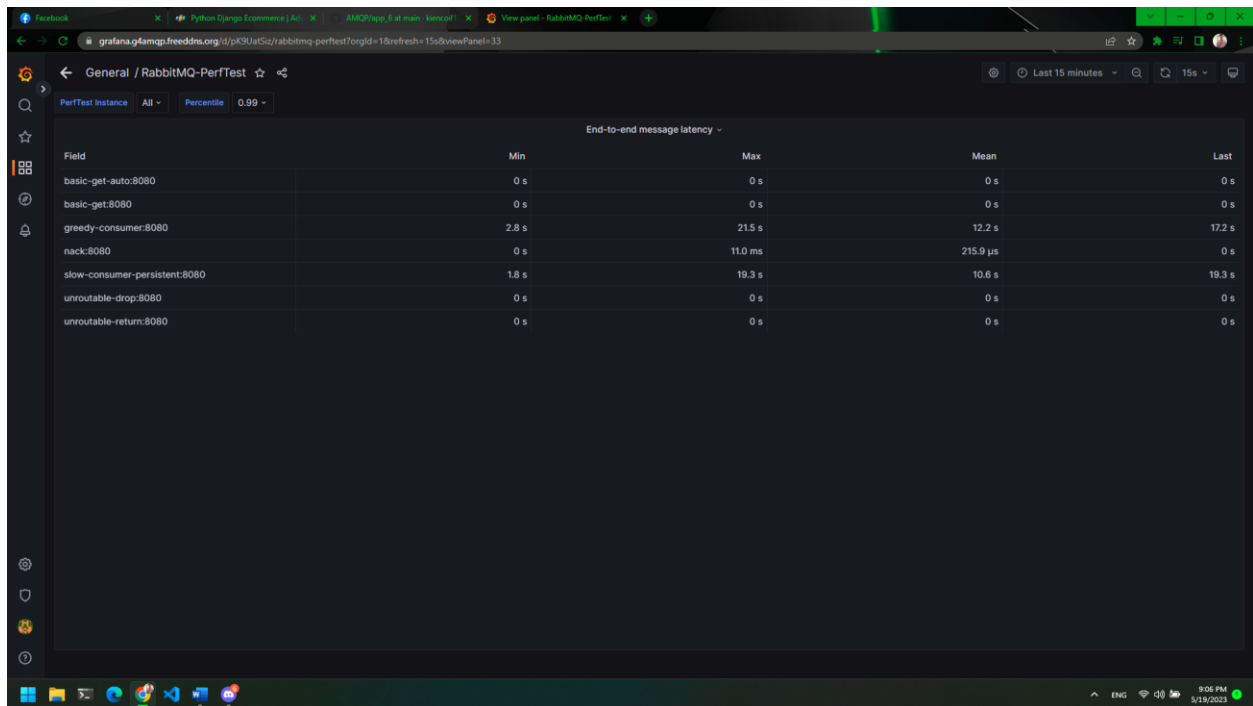
Unroutable-Return test được sử dụng để kiểm tra khả năng xử lý các thông điệp không thể định tuyến trong hệ thống AMQP. Kiểm tra này được thực hiện bằng cách đưa vào hệ thống một lượng lớn các thông điệp không thể định tuyến. Hệ thống sẽ tiếp nhận các thông điệp này từ producer và cố gắng định tuyến chúng tới queue phù hợp.

Config Test:

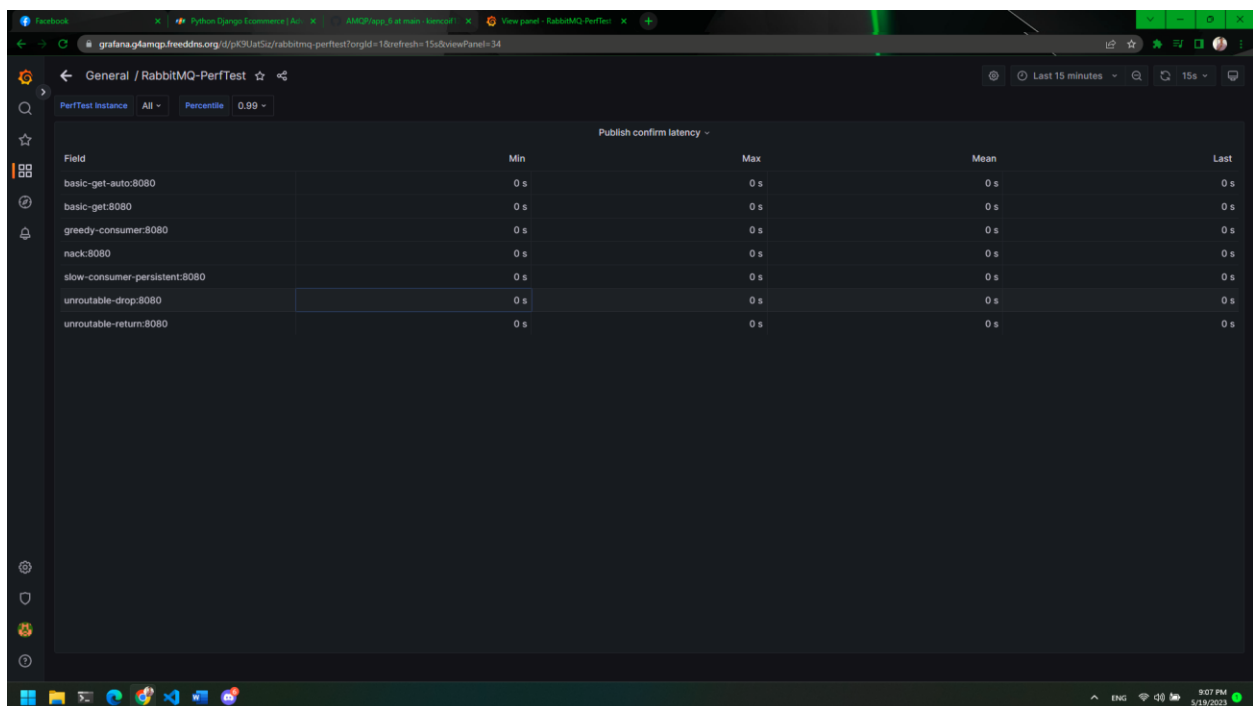
```
environment:
  URI: "amqp://guest:guest@rabbitmq:5672/%2f"
  VARIABLE_RATE: "2:30,4:30,10:30"
  VARIABLE_SIZE: "100:30,200:30"
  CONSUMERS: 0
  FLAG: mandatory
  SERVERS_STARTUP_TIMEOUT: *startup_timeout
  METRICS_PROMETHEUS: "true"
```

BẢNG ĐÁNH GIÁ

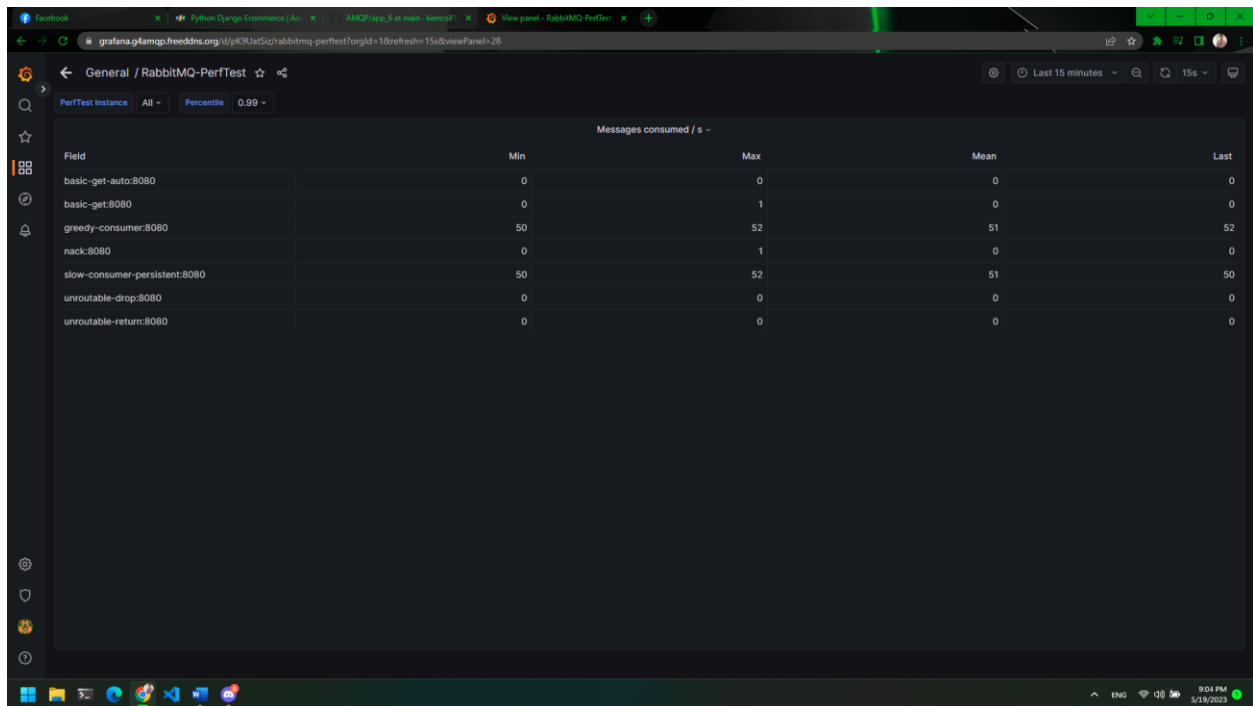
TEST	ĐÁNH GIÁ
Basic Get Test	Tốt
Basic-get-auto Test	Tốt
Greedy consumer Test	Tốt
Publisher-confirms Test	Tốt
Slow-consumer-persistent Test	Tốt
Nack Test	Tốt
Unroutable-Return Test	Tốt



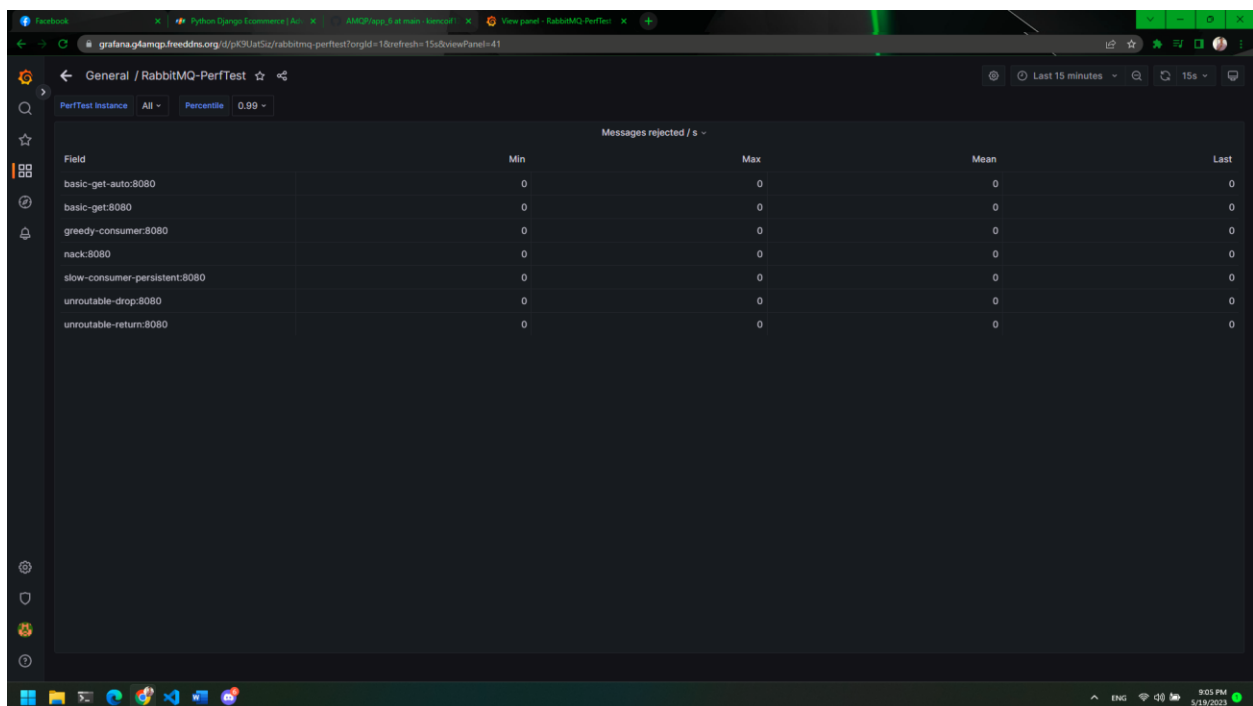
Hình 5.1 End-to-end message latency



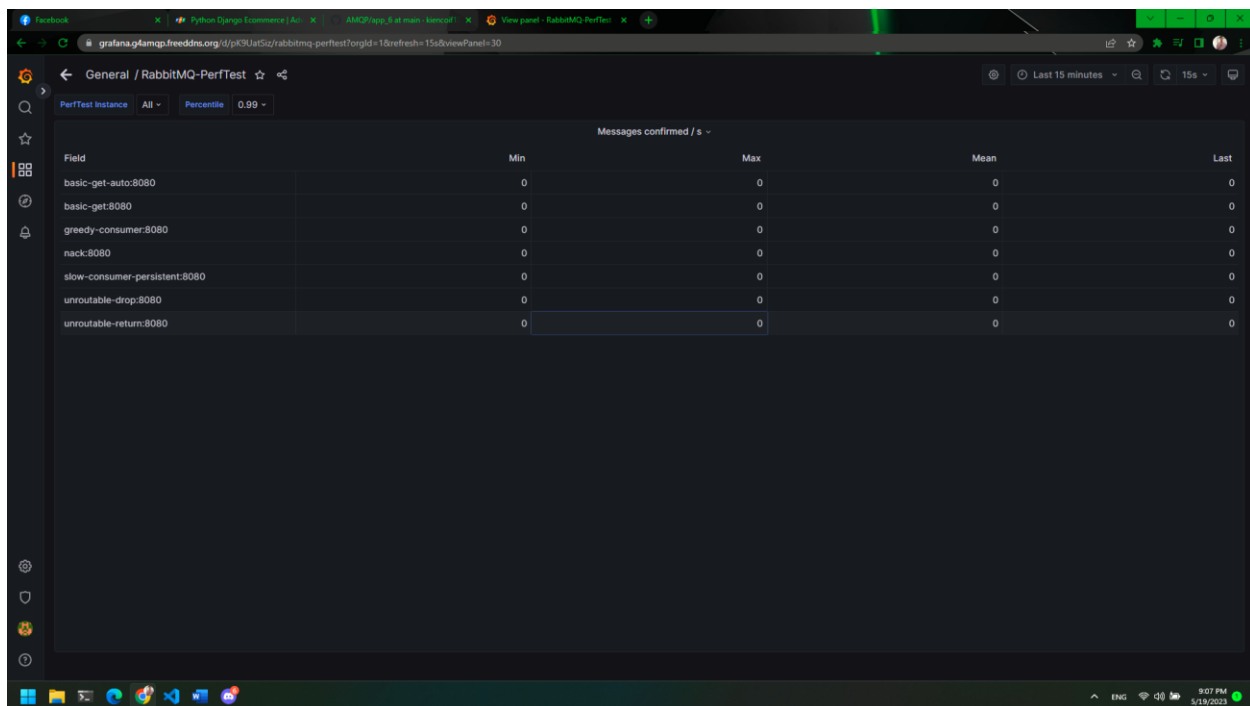
Hình 5.2 Publish confirm latency



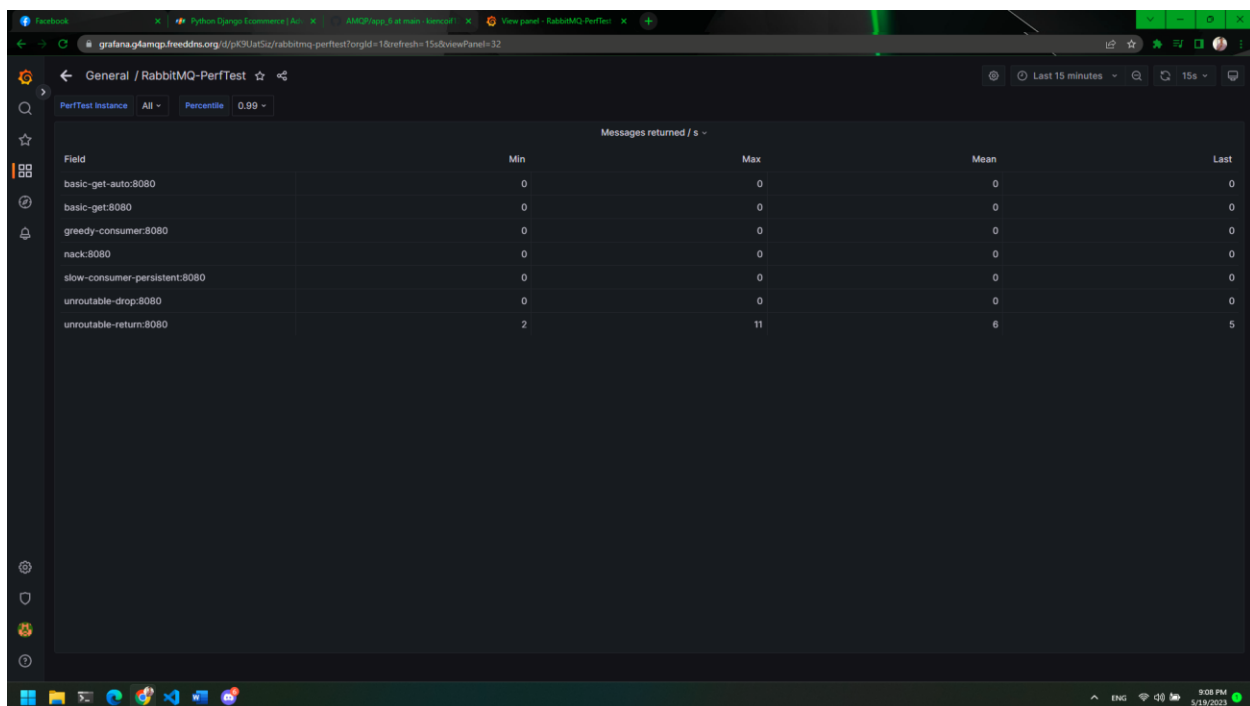
Hình 5.3 Messages consumed/s



Hình 5.4 Messages rejected / s



Hình 5.5 Messages confirmed / s



Hình 5.6 Messages returned / s

ĐÓNG GÓP CỦA CÁC THÀNH VIÊN

THÀNH VIÊN	ĐÓNG GÓP
Trần Tuấn Anh 20021297	BackEnd – Chart Demo
Nguyễn Đức Dũng 20020181	FrontEnd - Chart Test
Nguyễn Đình Hoàng 20021361	FrontEnd Dựng Server Test
Lương Trung Kiên 20021378	BackEnd - Gửi dữ liệu Docs
Bá Thanh Tùng 20021467	BackEnd - Nhận dữ liệu Thuyết trình

CẢM ƠN THẦY ĐÃ ĐỌC BÀI BÁO CÁO CỦA NHÓM CHÚNG EM Ạ

<3 <3 <3

THE END !!!