

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH



Mạng máy tính (CO3003)

---

# Báo cáo bài tập lớn 1

---

Giảng viên: Nguyễn Hồng Nam  
Bùi Xuân Giang  
SV thực hiện: Đỗ Long Biên – 1510216  
Hoàng Công Nhật Nam – 1512061  
Phạm Văn Linh – 1511792  
Nguyễn Nam Quân – 1512683  
Nguyễn Thanh Nam – 1512091

Tp. Hồ Chí Minh, Tháng 10/2017



## Mục lục

<b>1</b>	<b>Đề bài</b>	<b>3</b>
<b>2</b>	<b>Giới thiệu ứng dụng</b>	<b>3</b>
2.1	Phân tích yêu cầu đặc điểm ứng dụng	3
2.2	Mô tả ứng dụng	4
<b>3</b>	<b>Những kiến thức cần chuẩn bị</b>	<b>4</b>
3.1	Giới thiệu về socket	4
3.2	Giới thiệu về xử lý đa luồng trong Java	5
3.3	Giới thiệu về CloudMQTT	7
<b>4</b>	<b>Thiết kế mô hình giao tiếp</b>	<b>8</b>
4.1	Thiết kế chi tiết kiến trúc hệ thống	8
4.2	Mô tả các bộ giao thức Data	9
4.2.1	Định dạng 1	9
4.2.2	Định dạng 2	9
4.2.3	Định dạng 3	9
4.2.4	Định dạng Error	10
<b>5</b>	<b>Thiết kế chương trình và chi tiết hiện thực</b>	<b>10</b>
5.1	Hiện thực Gateway	10
5.1.1	Class Diagram	11
5.1.2	UDPGateway	12
5.1.3	handlePacket	12
5.1.4	MQTTPublishData	13
5.2	Hiện thực Analytic Application	14
5.2.1	Class Diagram	16
5.2.2	ApplicationAnalytic	16
5.2.3	ServiceThread	17
5.2.4	MQTTReceive	18
5.2.5	Lưu trữ dữ liệu	19
5.2.6	Tổng quan truy xuất Database và kết quả xử lý	20
5.2.7	Chi tiết hàm xử lý ở Analytic Application	21
5.3	Hiện thực App Client	24
5.3.1	Class Diagram	24
5.3.2	Google Map	24
5.3.3	GPS	25
5.3.4	Client UDP	26
<b>6</b>	<b>Đánh giá sản phẩm</b>	<b>27</b>
6.1	Ưu điểm	27
6.2	Nhược điểm	27
6.3	Hướng phát triển	27



<b>7</b>	<b>Hướng dẫn sử dụng</b>	<b>28</b>
7.1	Khởi động Gateway và Application analysis . . . . .	28
7.2	Đăng nhập Ứng dụng . . . . .	29
7.3	Chức năng thức nhất: Thu thập thông tin . . . . .	30
7.4	Chức năng thứ hai: gửi yêu cầu và nhận kết quả . . . . .	32
7.5	Kết quả nhận dữ liệu ở Getway và Analytic Application . . . . .	32
<b>8</b>	<b>Chức năng mở rộng</b>	<b>34</b>
	<b>Tài liệu tham khảo</b>	<b>34</b>

# 1 Đề bài

Xây dựng ứng dụng Internet of Things (IoT) sử dụng thiết bị di động và các dịch vụ trên Internet.

## 2 Giới thiệu ứng dụng

### 2.1 Phân tích yêu cầu đặc điểm ứng dụng

Với chủ đề "Ứng dụng IoT", sử dụng giao thức MQTT thông qua cloudMQTT và những mục tiêu được nêu cụ thể trong file bài tập lớn gồm:

- Biết cách thiết kế một giao thức đơn giản cho mô hình giao tiếp Client-Server.
- Hiểu cơ chế hoạt động của Socket và có thể hiện thực một chương trình sử dụng các chức năng đơn giản của Socket.
- Có kỹ năng lập trình mạng cơ bản

Từ đó các thành viên của nhóm đã thống nhất những yêu cầu cho ứng dụng như sau:

- Ứng dụng sử dụng cảm biến GPS trên thiết bị di động.
- Sử dụng lập trình socket để hiện thực giao thức TCP - UDP.
- Sử dụng dịch vụ cloudMQTT.
- Thiết kế hệ thống theo đúng mô hình được mô tả gồm các cặp kết nối: Client, Sensor - Gateway, Gateway - CloudMQTT, CloudMQTT - Server, server-Database, Server - Client; cũng như các loại kết nối được sử dụng.

Và những đặc điểm sau:

- Đề tài được lựa chọn có thể sẽ không phù hợp với logic thiết kế của một hệ thống thực tế vì các hệ thống IoT khi sử dụng cảm biến của smartphone thường không cần đến một Gateway trung gian. Hầu hết smartphone hiện tại luôn đủ sức để kết nối với một cloud service hoặc một server trực tiếp.
- Trong thời gian một tháng, các thành viên trong nhóm bắt đầu làm quen với nhiều kiến thức gồm ngôn ngữ Java, kỹ thuật lập trình Android, lập trình socket, sử dụng cloudMQTT. Vì vậy các Exception trong quá trình lập trình không thể được handle hoàn toàn, lỗi trong sản phẩm hoàn thành là không tránh khỏi. Tuy nhiên những chức năng cơ bản của ứng dụng, khả năng test thực tế (hoặc trên máy ảo đối với GPS khoảng cách lớn) là phải thực hiện được.
- Các thiết bị client, gateway, server được thiết kế để giao tiếp trong cùng mạng một LAN nên không có điều kiện di chuyển nên chủ yếu dữ liệu được tự khởi tạo.

Từ những phân tích trên, nhóm đã thống nhất thiết kế ứng dụng "GPS Tracker".

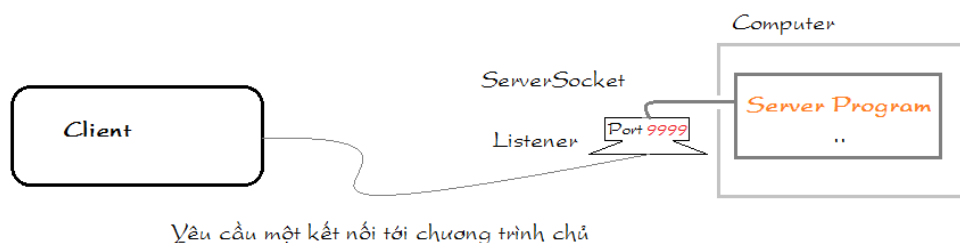
## 2.2 Mô tả ứng dụng

**GPS Tracker** là ứng dụng thi thập dữ liệu vị trí của người dùng. Dữ liệu này được sử dụng để thống kê những địa điểm mà người dùng thường xuyên đi qua. Mọi người dùng của hệ thống đều có thể gửi yêu cầu để truy xuất thông tin này( sẽ nhận được tối đa 3 vị trí có tần suất xuất hiện nhiều nhất)

## 3 Những kiến thức cần chuẩn bị

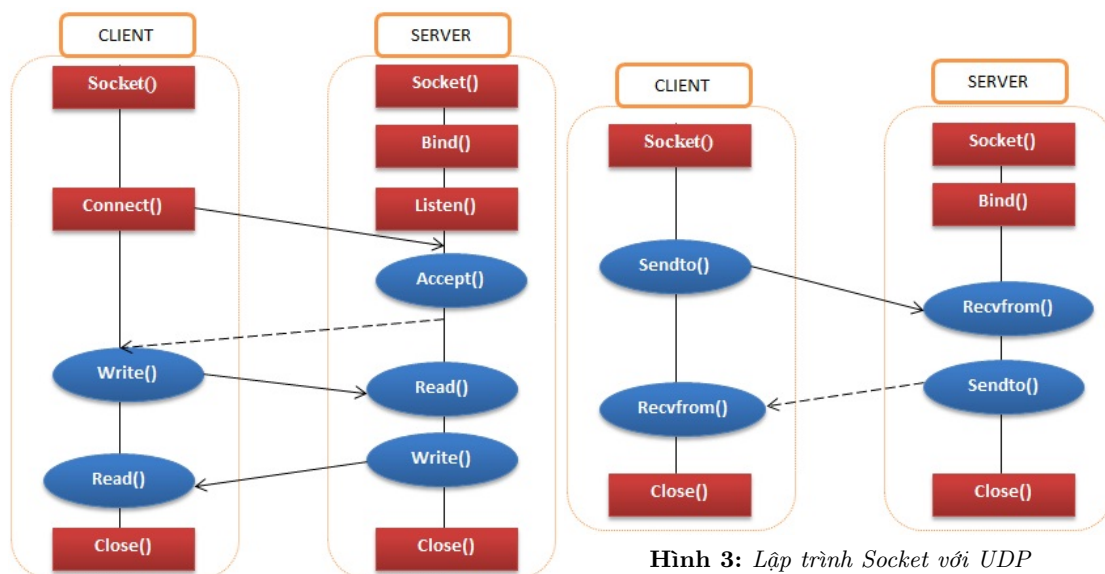
### 3.1 Giới thiệu về socket

- **Lập trình socket** là cách lập trình cho phép chúng ta kết nối các máy tính truyền tải và nhận dữ liệu từ máy tính thông qua mạng .



Hình 1: Socket

- **Socket** là một giao diện lập ứng dụng mạng . Thông qua giao diện này, chúng ta có thể lập trình điều khiển, truyền thông giữa 2 máy sử dụng giao thức TCP/IP và UDP.
- **Socket** là thiết bị truyền thông hai chiều gửi và nhận dữ liệu từ máy khác
- Các loại socket:
  - Socket có hướng kết nối (**TCP**)
    - \* Có một đường kết nối (địa chỉ IP) giữa 2 tiến trình
    - \* Một trong 2 tiến trình kia phải đợi tiến trình kia yêu cầu kết nối.
    - \* Có thể dùng để liên lạc theo mô hình client và sever
    - \* Mô hình client /sever thì sever lắng nghe và chấp nhận từ client
    - \* Mỗi thông điệp gửi phải có xác nhận trả về
    - \* Các gói tin chuyển đi tuần tự.
  - Socket không hướng kết nối (**UDP**)
    - \* 2 Tiến trình liên lạc với nhau không kết nối trực tiếp
    - \* Thông điệp gửi đi phải kèm theo thông điệp người nhận
    - \* Thông điệp có thể gửi nhiều lần
    - \* Người gửi không chắc chắn thông điệp đến tay người nhận.
    - \* Thông điệp gửi sau có thể đến trước và ngược lại.



Hình 3: Lập trình Socket với UDP

Hình 2: Lập trình Socket với TCP

### 3.2 Giới thiệu về xử lý đa luồng trong Java

Trong bài tập lớn này, để Gateway và Analytic Application hoạt động ổn định, không bị mất mát dữ liệu cũng như có thể đáp ứng được nhiều yêu cầu từ nhiều client cùng một lúc, việc áp dụng xử lý đa luồng ở đây là khá cần thiết và quan trọng. Sau đây là một số kiến thức cơ bản về lập trình đa luồng trong Java.

Đa luồng trong java sẽ đề cập đến hai khái niệm Multitasking và Multithreading.

- Multitasking: Là khả năng chạy đồng thời một hoặc nhiều chương trình cùng một lúc trên một hệ điều hành. Hệ điều hành quản lý việc này và sắp xếp lịch phù hợp cho các chương trình đó. Ví dụ, trên hệ điều hành Windows chúng ta có làm việc đồng thời với các chương trình khác nhau như: Microsoft Word, Google Chrome, ...
- Multithreading: Là khả năng thực hiện đồng thời nhiều phần khác nhau của một chương trình được gọi là thread. Ví dụ trong Microsoft Excel chúng ta có thể làm việc đồng thời với các sheet khác nhau.

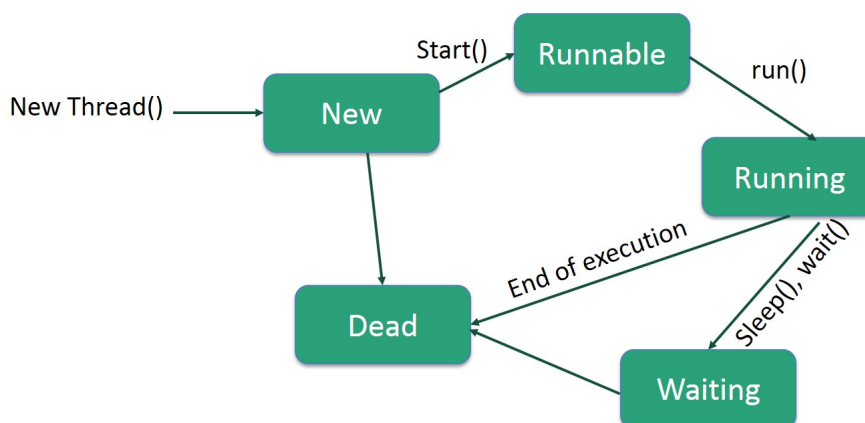
Tiếp theo ta sẽ đề cập đến khái niệm Thread:

- Thread là đơn vị nhỏ nhất của mã thực thi mà đoạn mã đó thực hiện một nhiệm vụ cụ thể.
- Một ứng dụng có thể được chia nhỏ thành nhiều nhiệm vụ và mỗi nhiệm vụ có thể được giao cho một thread.
- Nhiều thread cùng thực hiện đồng thời được gọi là đa luồng (multithread). Các quá trình đang chạy dường như là đồng thời, nhưng thực ra nó không phải là như vậy.

Vòng đời của một Thread:

- New: một thread ở trạng thái "new" nếu bạn tạo ra một đối tượng thread nhưng chưa gọi phương thức start().

- Ready: Sau khi thread được tạo, nó sẽ ở trạng thái sẵn sàng (ready) chờ phương thức `start()` gọi nó.
- Running: Thread ở trạng thái chạy (đang làm việc).
- Sleeping: Phương thức `sleep()` sẽ đưa thread vào trạng thái sleeping – dừng lại tạm thời. Sau thời gian sleeping thread lại tiếp tục hoạt động trở lại.
- Waiting: Khi method `wait()` hoạt động, thread sẽ rơi vào trạng thái waiting – đợi. Method này được sử dụng khi hai hoặc nhiều thread cùng đồng thời hoạt động.
- Blocked: Thread sẽ rơi vào trạng thái “blocked” – bị chặn khi thread đó đang đợi một sự kiện nào đó của nó như sự kiện I/O.
- Dead: Thread rơi vào trạng thái dead – ngừng hoạt động sau khi thực hiện xong phương thức `run` hoặc gọi phương thức `stop()`.



**Hình 4:** Vòng đời của một thread

Trong lập trình Java, có 2 cách để khởi tạo Thread:

- Cách 1: Tạo thread bằng cách sử dụng interface Runnable:
  - Viết 1 class thực thi interface Runnable và viết lại phương thức ‘`public void run()`’
  - Tạo ra 1 object vừa thực thi interface Runnable.
  - Tạo ra 1 object của class Thread với tham số truyền vào là object thực thi interface Runnable.
  - Gọi phương thức `start()` để chạy thread.
- Cách 2: Tạo thread bằng cách kế thừa từ lớp Thread:
  - Tạo một lớp kế thừa từ lớp Thread và viết lại (override) phương thức `run()`
  - Tạo ra một đối tượng của lớp vừa kế thừa từ lớp Thread
  - Gọi phương thức `start()` để chạy thread

### 3.3 Giới thiệu về CloudMQTT

**CloudMQTT** được quản lý bởi các máy chủ Mosquitto trong đám mây. Mosquitto hiện thực MQ Telemetry Transport protocol, MQTT, để cung cấp những phương thức đơn giản cho việc thực hiện tin nhắn bằng cách sử dụng một mô hình hàng đợi tin nhắn thông báo publish/subscribe.

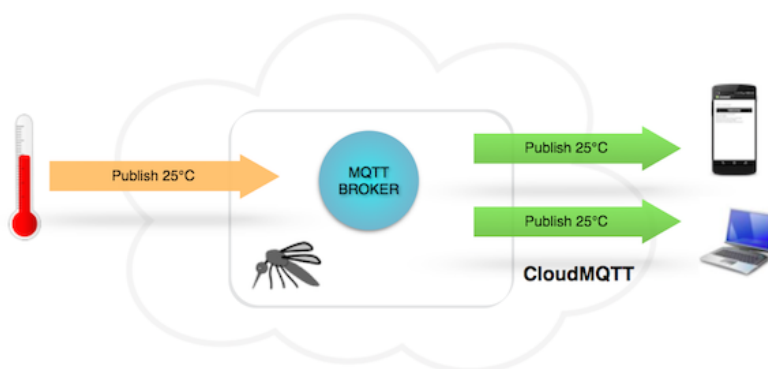


Hình 5: Mô hình cơ bản của cloudMQTT

**MQTT** là phương thức machine-to-machine (M2M) của tương lai. Nó là ý tưởng cho “Internet of Things” liên kết tất cả các thiết bị trên toàn thế giới. Với thiết kế đơn giản của nó, nó phù hợp với những hệ thống đã được xây dựng sẵn, với các thiết bị di động, các bộ nhớ khác và các ứng dụng có băng thông không ổn định.

Hàng đợi tin nhắn thông báo (Message queues) cung cấp một giao thức giao tiếp bất đồng bộ, người gửi và người nhận của tin nhắn không cần phải tương tác với hàng đợi tin nhắn cùng thời điểm. Các tin nhắn được đặt vào hàng đợi đã được lưu trữ mãi cho đến khi người nhận lấy nó ra hoặc là các tin nhắn đó hết thời gian lưu trữ. MQTT và Mosquitto được sử dụng tốt cho các thiết bị có băng thông nhạy cảm.

CloudMQTT cho bạn tập trung vào ứng dụng thay vì phải mất nhiều thời gian cho việc nhân rộng máy chủ hoặc là vá nền tảng.



Hình 6: Mô hình minh họa chi tiết hơn của cloudMQTT

Đầu tiên ta sẽ nói về 2 khía cạnh của MQTT:

- **Client:** MQTT client bao gồm người gửi và người nhận. Một MQTT client là bất kỳ thiết bị từ một vi điều khiển đến một máy chủ lớn đầy đủ chức năng, nó có một thư viện MQTT đang chạy và đang kết nối đến một MQTT Broker trên bất kỳ loại mạng nào. Các thư viện MQTT client là có sẵn cho các ngôn ngữ lập trình đang sử dụng phổ biến hiện nay như Android, Arduino, C, C++, Ruby, Go, iOS, Java, JavaScript, .NET.
- **Broker** ( còn được xem như Server): MQTT broker là trái tim của bất kỳ giao thức publish/subscribe nào đó. Một Broker có thể xử lý hàng ngàn MQTT clients kết nối đồng



thời tới nó. Broker chịu trách nhiệm chính trong việc nhận tất cả các tin nhắn, lọc chúng, quyết định ai là người quan tâm đến những tin nhắn này và sau đó nó sẽ gửi những tin nhắn này đến tất cả những khách hàng đã đăng ký nhận tin nhắn này. Một trách nhiệm khác của broker là xác nhận và ủy quyền cho các khách hàng.

Trong một hệ thống sử dụng giao thức MQTT, nhiều node trạm (gọi là MQTT client - gọi tắt là client) kết nối tới một MQTT server (gọi là Broker). Mỗi client sẽ đăng ký một vài kênh (topic), ví dụ như "channel1", "channel2". Quá trình đăng ký này gọi là "subscribe", giống như chúng ta đăng ký nhận tin trên một kênh Youtube vậy. Mỗi client sẽ nhận được dữ liệu khi bất kỳ trạm nào khác gửi dữ liệu và kênh đã đăng ký. Khi một client gửi dữ liệu tới kênh đó, gọi là "publish".

Cuối cùng ta sẽ nói về QoS. Ở đây có 3 tùy chọn \*QoS (Qualities of service) \* khi "publish" và "subscribe":

- QoS0 Broker/client sẽ gửi dữ liệu đúng 1 lần, quá trình gửi được xác nhận bởi chỉ giao thức TCP/IP, giống kiểu đem con bỏ chợ.
- QoS1 Broker/client sẽ gửi dữ liệu với ít nhất 1 lần xác nhận từ đầu kia, nghĩa là có thể có nhiều hơn 1 lần xác nhận đã nhận được dữ liệu.
- QoS2 Broker/client đảm bảo khi gửi dữ liệu thì phía nhận chỉ nhận được đúng 1 lần, quá trình này phải trải qua 4 bước bắt tay

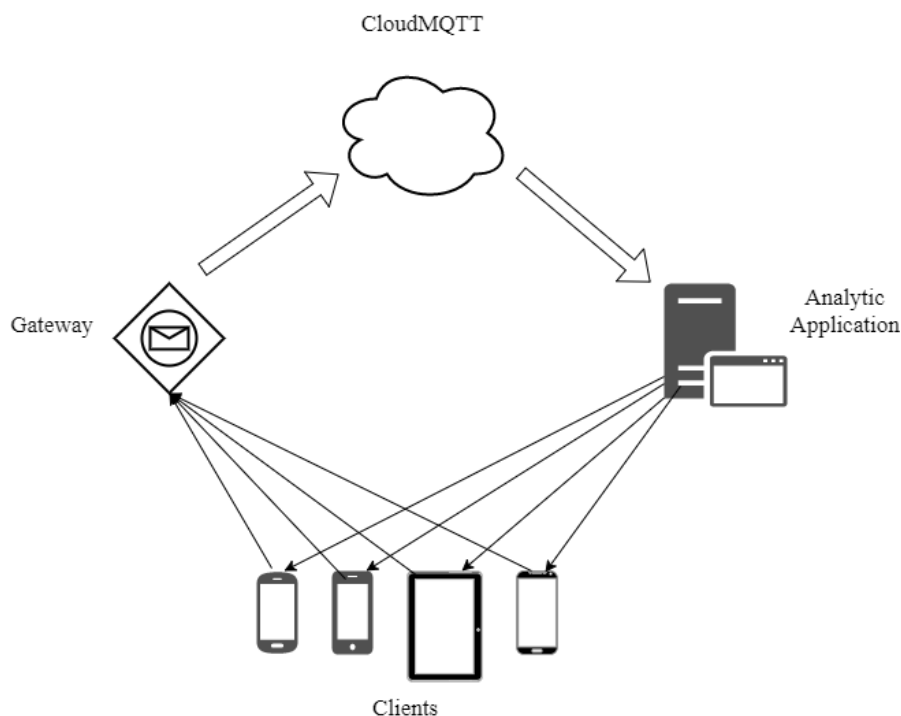
## 4 Thiết kế mô hình giao tiếp

### 4.1 Thiết kế chi tiết kiến trúc hệ thống

Để xây dựng được kiến trúc hệ thống, cần phải nắm rõ và khái quát được mô hình từng bộ phận chính của hệ thống ứng dụng IoT. Dựa vào thiết kế tổng thể kiến trúc theo yêu cầu của bài tập lớn, nhóm đã xây dựng cho mình mô hình như sau:

Nhìn tổng quan, hệ thống được chia thành 4 thành phần chính, bao gồm:

- Client: Là app ứng dụng được xây dựng trên các thiết bị di động Android. Ứng dụng sẽ sử dụng cảm biến vị trí (GPS thông qua mạng) trên smartphone, kết hợp với những Google Map vào ứng dụng để xác định vị trí của mình trên bản đồ. Sau khi thu thập được 1 vị trí, app sẽ chuyển ngay tọa độ vị trí về Gateway thông qua giao thức UDP.
- Gateway: Trong hệ thống, nhóm xây dựng 1 cổng Gateway đóng vai trò như cổng trung gian. Tất cả dữ liệu tọa độ vị trí từ Client sẽ được chuyển về đây ngay sau khi Client nhận được vị trí tọa độ hiện tại của mình. Từ đó, Gateway sẽ thu thập lại rồi chuyển lên CloudMQTT thông qua giao thức Message Queuing Telemetry Transport (MQTT).
- CloudMQTT: Đóng vai trò như trung tâm lưu trữ dữ liệu. Mọi dữ liệu vị trí của người dùng sẽ được Gateway chuyển về đây và lưu trữ để Server có thể lấy về qua giao thức MQTT nhưng vì MQTT dùng phiên bản miễn phí nên nó chỉ tiếp nhận thông tin và gửi dữ liệu về Analytic Application ngay lập tức chứ không lưu trữ.
- Analytic Application: Đóng vai trò như 1 Server nhận dữ liệu từ MQTT và lưu vào Database. Sau đó, nếu có người dùng gửi yêu cầu truy xuất thông tin của một người dùng nào đó thì Server sẽ gửi dữ liệu đã xử lý thông qua một protocol dựa trên nền tảng TCP/IP. Ngoài ra, giao thức này cũng được sử dụng để Client có thể gửi trực tiếp yêu cầu đến Server.



Hình 7: Kiến trúc hệ thống ứng dụng IoT

## 4.2 Mô tả các bộ giao thức Data

### 4.2.1 Định dạng 1

Sử dụng: Client(App) → CloudMQTT → Analytic Application → DataBase hoặc DataBase → Analytic Application

Định dạng: "Email|KinhDo|ViDo"

Trong đó **Email** là đại chỉ email của người dùng (được đăng kí ở màn hình Đăng nhập). **KinhDo**, **ViDo** lần lượt tương ứng với Kinh độ và vĩ độ mà App thu thập được. Cứ sau mỗi khoảng 20s Ứng dụng sẽ lấy vị trí tọa độ 1 lần và tạo thành 1 chuỗi với định dạng như trên và gửi lên Gateway.

### 4.2.2 Định dạng 2

Sử dụng: Analytic Application → Client

Định dạng: "KinhDo1|ViDo1-KinhDo2|ViDo2..."

Danh sách TỐI ĐA 3 cặp kinh độ vĩ độ (vì ở đây chỉ lấy tối đa 3 vị trí hay đến nhất). Kinh độ vĩ độ cách nhau bởi dấu "|" và giữa các cặp kinh độ vĩ độ cách nhau bởi dấu "-".

### 4.2.3 Định dạng 3

Sử dụng: Client → Analytic Application

Định dạng: "Email"

Chứa Email là thông tin cần để Analytic Application truy vấn cơ sở dữ liệu và trả về kết quả.

#### 4.2.4 Định dạng Error

Sử dụng: Client → Analytic Application

Định dạng: "9999|9999"

Nếu Email không tồn tại trong Database gửi chuỗi với định dạng này về cho Client.

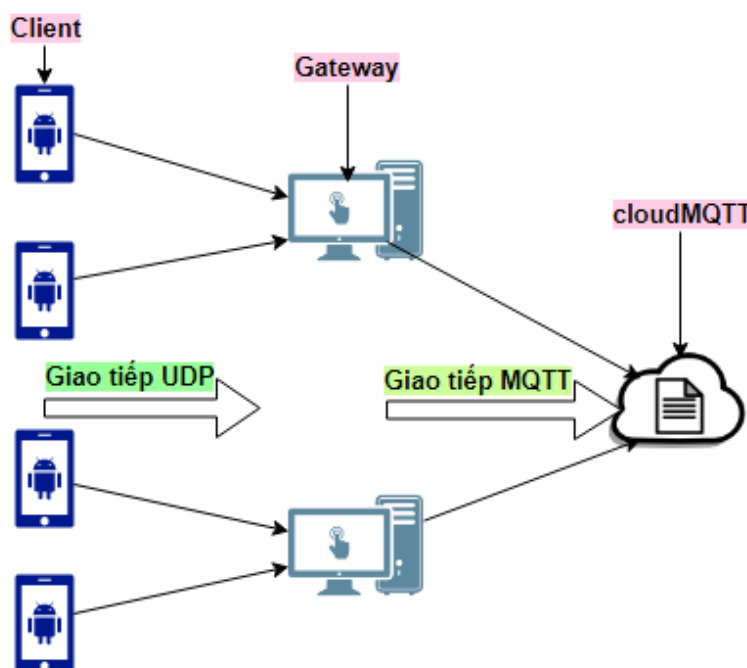
## 5 Thiết kế chương trình và chi tiết hiện thực

### 5.1 Hiện thực Gateway

Trong ứng dụng này, Gateway đóng vai trò trung gian làm cầu nối giao tiếp giữa client và cloudMQTT. Client sẽ gửi dữ liệu lên Gateway rồi sau đó dữ liệu từ Gateway được truyền lên cloudMQTT. Các phương thức giao tiếp được hiện thực ở đây bao gồm:

- Giao tiếp giữa client và Gateway được hiện thực theo phương thức giao tiếp UDP - User Datagram Protocol.
- Giao tiếp giữa Gateway và cloudMQTT được hiện thực theo phương thức giao tiếp MQTT - Message Queuing Telemetry Transport.

Mô hình giao tiếp được minh hoạ như bên dưới:



Hình 8: Mô hình giao tiếp của Gateway

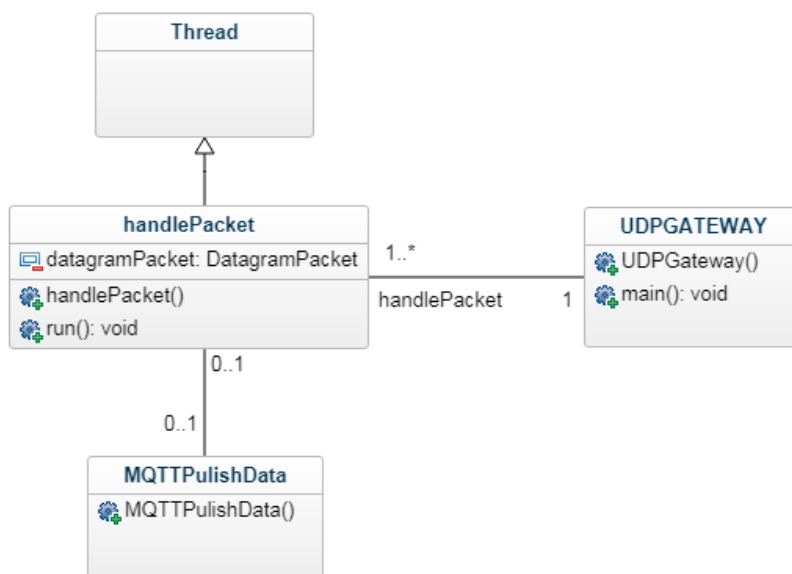
Gateway được cấu hình đầy đủ những thông tin của cloudMQTT để đảm bảo việc gửi dữ liệu lên cloudMQTT đúng vào Topic đã được quy định để Analytic Application có thể nhận được những dữ liệu này.

Gateway cần có một port để lắng nghe tất cả các gói tin từ client bất kỳ khi nó dữ liệu vào Gateway này. Port sẽ được nhập vào khi chạy Gateway. Những client phải biết được những Gateway nào đang mở và port tương ứng để gửi dữ liệu lên, đảm bảo được việc dữ liệu sẽ không bị mất mát. Phương thức lắng nghe kết nối của Gateway được đặt trong một vòng lặp vô tận đảm bảo việc nhận dữ liệu không bị gián đoạn.

Những gói tin này sau khi được Gateway nhận, Gateway sẽ tiến hành tạo ra một luồng thực thi để tiến hành xử lý những gói tin này. Và luồng thực thi này đơn giản chỉ là tiến hành gửi dữ liệu lên cho cloudMQTT, sau khi kết thúc việc gửi, luồng thực thi sẽ tự hủy. Cơ chế về đa luồng đã được mô tả rõ ở mục 3.2 bên trên. Sau khi tạo ra một luồng thực thi này, Gateway ngay lập tức trở về tình trạng chờ gói tin mới được gửi đến. Việc xử lý đa luồng ở đây giúp cho Gateway có thể đáp ứng được nhiều người dùng cùng một lúc.

### 5.1.1 Class Diagram

Gateway được hiện thực gồm 3 class là UDPGATEWAY, handlePacket và MQTTPushData. Trong đó class handlePacket là class dùng để xử lý đa luồng và với đối tượng truyền vào ở đây là một packet được nhận từ client gửi lên. Với cách thiết kế này, khi gateway đang lắng nghe các client gửi dữ liệu lên thì lúc nào có một gói tin được gửi lên thì một đối tượng handlePacket sẽ được khởi tạo và tạo thành một luồng thực thi mới. Trong luồng thực thi này thì class handlePacket sẽ tiến hành khởi tạo đối tượng MQTTPushData. Sau khi đối tượng này được khởi tạo thì nó sẽ tiến hành việc gửi dữ liệu lên cho cloudMQTT. Việc gửi hoàn tất thì đối tượng này bị hủy, nó sẽ luồng thực thi này cho đối tượng handlePacket. Đối tượng handlePacket sau khi nhận lại được luồng thực thi thì cũng bị hủy ngay khi đó. Quá trình chạy cứ như thế diễn ra.



Hình 9: Class Diagram của Gateway

### 5.1.2 UDPGateway

---

```
1 public static void main(String[] args) throws IOException {
2     DatagramSocket serverSocket = new DatagramSocket(GUI_GateWay.port);
3     while(true) {
4         byte[] receiveData = new byte[1024];
5         DatagramPacket receivePacket = new DatagramPacket(receiveData,
6             ↳ receiveData.length);
7         serverSocket.receive(receivePacket);
8         new handlePacket(receivePacket).start();
9     }
}
```

---

#### Chức năng:

- Đây là hàm main khởi chạy cho chương trình. Chức năng chính của class là lắng nghe kết nối và nhận dữ liệu từ các client.

#### Hiện thực:

- Class này là class chứa hàm main để khởi chạy cho chương trình. Trong hàm main ta sẽ khởi tạo một DatagramSocket với con số port được người chạy khi chạy chương trình giao diện GUI sẽ nhập vào. Thông số tương ứng của port sẽ được lưu trong GUI\_GateWay.port để đảm bảo tính trực quan cho ứng dụng.
- Sau khi khởi tạo cổng lắng nghe kết nối từ client, ta đặt một vòng lặp vô tận để đợi client gửi thông điệp lên. Sau khi dữ liệu được nhận và được lưu vào trong receivePacket. Ta bắt đầu tạo ra một đối tượng handlePacket và thực hiện phương thức start để nó có thể tạo ra một luồng thực thi việc xử lý dữ liệu. Sau đó Gateway quay trở lại đầu vòng lặp và chờ một gói tin mới gửi đến.

### 5.1.3 handlePacket

---

```
1 private static class handlePacket extends Thread {
2     private DatagramPacket datagramPacket;
3     private handlePacket(DatagramPacket datagramPacket) {
4         this.datagramPacket = datagramPacket;
5     }
6     @Override
7     public void run() {
8         byte[] receiveByte = datagramPacket.getData();
9         String dataPacket = new String(receiveByte, 0,
10             ↳ datagramPacket.getLength());
11         MQTTPublishData mqttPublishData = new MQTTPublishData(dataPacket);
12     }
}
```

---

#### Chức năng:

- Class này là class thực hiện quá trình đa luồng cho Gateway. Có hai cơ chế để hiện thực đa luồng và ở đây nhóm chọn hiện thực theo cơ chế kế thừa class Thread

#### Hiện thực:

- Class handlePacket có thuộc tính là datagramPacket, một constructor và một hàm run() để hiện thực việc kế thừa class Thread
- Trong hàm run(). Ta tiến hành phân giải dữ liệu của client gửi lên trở thành dạng String. Vì dữ liệu này đã được quy định từ trước nên việc phân giải thành String này đã được tính toán từ trước. Dữ liệu này sẽ được truyền vào như đối số để khởi tạo một đối tượng MQTTPulishData để tiến hành gửi dữ liệu lên cloudMQTT.

#### 5.1.4 MQTTPulishData

```
1 public class MQTTPulishData {
2     public MQTTPulishData(String content) {
3         String topic = "TestMQTTGateway";
4         int qos = 0;
5         String broker = "tcp://m12.cloudmqtt.com:16534";
6         String clientId = "ClientIdGateway";
7         MemoryPersistence persistence = new MemoryPersistence();
8         try {
9             MqttClient mqttClient = new MqttClient(broker, clientId,
10              ↳ persistence);
11             mqttClient.setCallback(new MqttCallback() {
12                 public void messageArrived(String topic, MqttMessage msg)
13                  ↳ throws Exception {
14                     System.out.println("Received: " + new
15                      ↳ String(msg.getPayload()));
16                 }
17             });
18             MqttConnectOptions connOpts = new MqttConnectOptions();
19             connOpts.setCleanSession(true);
20             connOpts.setUsername("phamvanlinh");
21             connOpts.setPassword(new char[] { '1', '4', '3', '1', '9', '9',
22              ↳ '7' });
23             mqttClient.connect(connOpts);
24             MqttMessage message = new MqttMessage(content.getBytes());
25             message.setQos(qos);
26             System.out.println("Publish message: " + message);
27             mqttClient.publish(topic, message);
28         } catch (MqttException me) {
29             }
30     }
31 }
```

#### Chức năng:

- Class có chức năng gửi dữ liệu lên cloudMQTT. Dữ liệu được gửi lên chính là String content được truyền vào ở constructor của class.

#### Hiện thực:

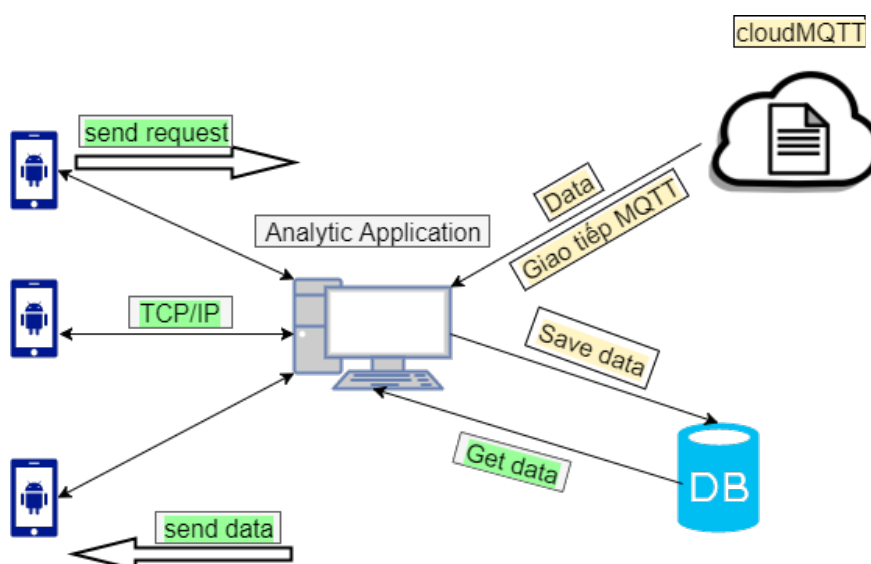
- Khi hàm run() của class handlePacket khởi tạo một đối tượng của class MQTTPublishData. Hàm này sẽ được khởi tạo rồi gửi dữ liệu lên cho cloud MQTT. Những thông tin về topic, qos, broker, clientID được khởi tạo cho phù hợp với thông tin đã đăng ký trên website của cloud MQTT. Những khái niệm về topic, qos, broker đã được trình bày trong mục 3.3 bên trên.
- Sau đó ta cần phần lấy dữ liệu mà handlePacket gửi qua để tiến hành publish lên cloudMQTT. Những câu lệnh bên dưới thực hiện quá trình gửi dữ liệu. Trong đó câu lệnh cuối cùng là thực hiện publish message lên đúng topic.

## 5.2 Hiện thực Analytic Application

Trong ứng dụng của nhóm, một Analytic Application được thiết kế để thực hiện các chức năng sau:

- Nhận dữ liệu từ cloudMQTT rồi sau đó lưu xuống Database đã được nhóm thiết kế sẵn. Phương thức giao tiếp giữa Analytic Application và cloudMQTT là phương thức MQTT - Message Queuing Telemetry Transport.
- Chờ người dùng gửi yêu cầu kết nối và giao tiếp với nó. Phương thức giao tiếp ở đây là phương thức TCP - Transmission Control Protocol.
- Khi nhận được yêu cầu lấy thông tin từ người dùng. Analytic Application tiến hành truy cập Database để tiến hành truy xuất dữ liệu theo yêu cầu sau đó tiến hành phân tích rồi gửi dữ liệu về cho người dùng.

Mô hình giao tiếp được minh họa như hình 15 bên dưới.



Hình 10: Mô hình giao tiếp của Analytic Application

Lưu ý ở đây cơ chế giao tiếp giữa client và Analytic Application là TCP/IP nên luồng kết nối phải được giữ trong suốt quá trình kết nối giao tiếp. Bên cạnh đó Analytic Application còn có chức năng thao tác dữ liệu nên nhóm đã tiến hành hiện thực Analytic Application theo cơ chế đa luồng. Chi tiết về lập trình đa luồng trong Java đã được trình bày ở mục 3.2.

Ở đây Analytic Application được thiết kế để đảm nhận rất nhiều chức năng. Analytic Application phải được cấu hình đầy đủ những thông tin cần thiết của cloudMQTT để đảm bảo việc nhận dữ liệu về chính xác. Các thông số được cấu hình tương tự như là các thông số của Gateway.

Analytic Application cần có một port để lắng nghe tất cả các kết nối yêu cầu lấy thông tin từ client. Port sẽ được nhập vào khi khởi chạy Analytic Application. Các client phải biết được chính xác số port cũng như IP của Analytic Application để yêu cầu lấy dữ liệu. Trong quá trình này thì Analytic Application phải luôn chạy để đảm bảo việc truy cập không gián đoạn. Phương thức lắng nghe kết nối của Analytic Application được đặt trong một vòng lặp vô tận để đảm bảo nhiều client có thể truy cập vào cùng một lúc. Sau khi có một client gửi yêu cầu lên, một luồng thực thi cho client này được thiết lập. Tiếp theo sau đó luồng thực thi sẽ làm gì thì các bạn có thể đọc ở đoạn sau. Sau khi một luồng thực thi được thiết lập, luồng thực thi chính trở lại đầu vòng lặp và lắng nghe các kết nối khác.

Bên cạnh đó, Analytic Application còn luôn phiên công việc nhận dữ liệu từ cloudMQTT gửi về. Dữ liệu này được các client gửi lên cho Gateway, nếu muốn biết thêm về quá trình gửi dữ liệu lên các bạn có thể tham khảo ở mục 5.1. Dữ liệu được nhận về cloudMQTT sẽ được Analytic Application tiến hành phân giải ra. Vì dữ liệu này đã được nhóm quy định trước nên việc phân giải cũng đã được hiện thực rõ ràng. Sau khi phân giải dữ liệu này xong, Analytic Application tiến hành lưu dữ liệu này xuống Database. Ở đây, việc thực thi quá trình nhận dữ liệu từ cloudMQTT đã được hiện thực trong một luồng thực thi riêng, khác với luồng thực thi trong giao tiếp với client.

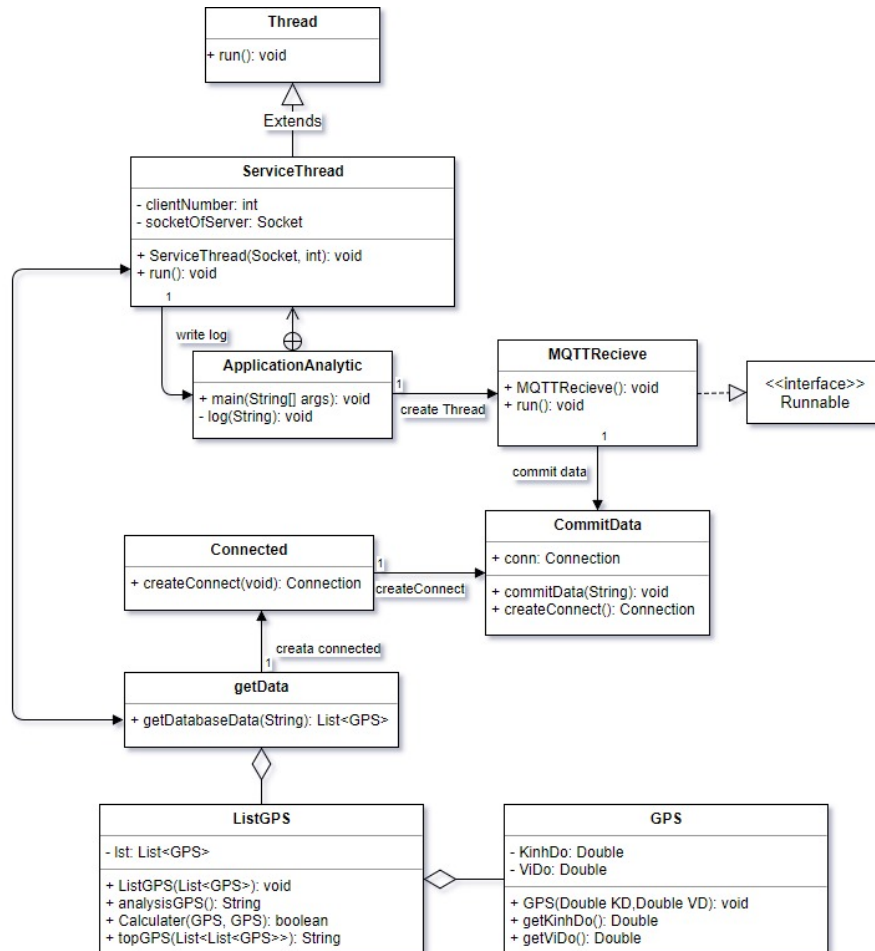
Trở về lại Analytic Application, sau khi nhận được yêu cầu từ client, cụ thể ở đây yêu cầu chính là lấy những vị trí hay xuất hiện nhất của đối tượng mà client yêu cầu, đối tượng này được xác định thông qua email của chính đối tượng đó. Lúc này Analytic Application tiến hành truy xuất Database để lấy lên chính xác yêu cầu của client. Dữ liệu sau khi được truy xuất từ Database sẽ được gửi trở lại client. Lúc này giao tiếp giữa client gửi yêu cầu và Analytic Application kết thúc. Luồng thực thi cho quá trình này cũng kết thúc. Luồng thực thi lúc này sẽ được chuyển qua cho các quá trình khác.

Mô hình của Analytic Application được xây dựng theo class diagram ở hình 11.



### 5.2.1 Class Diagram

Ta có hình 11 mô tả lược đồ lớp của Analytic Application.



Hình 11: Class Diagram của Analytic Application

### 5.2.2 ApplicationAnalytic

```

1 public static void main(String[] args) throws IOException {
2     Thread mThread = new Thread(new MQTTRecieve());
3     mThread.start();
4     ServerSocket listener = null;
5     int clientNumber = 0;
6     try {
7         listener = new ServerSocket(GUI.port);
8     } catch (IOException e) {
9     }
10    try {
11        while (true) {

```

```
12         Socket socketOfServer = listener.accept();
13         new ServiceThread(socketOfServer, clientNumber++).start();
14     }
15     } finally {
16         listener.close();
17     }
18 }
19 private static void log(String mes) {
20     System.out.println(mes);
21 }
```

---

#### Chức năng:

- Class ApplicationAnalytic có chức năng khởi tạo các luồng thực thi cho việc nhận dữ liệu từ cloudMQTT, khởi tạo vòng lặp cho việc giao tiếp giữa server và các client.

#### Hiện thực:

- Class được thiết kế có hàm main để khởi chạy cho chương trình. Khi bắt đầu hàm main. Ta tạo ra một luồng thực thi riêng cho việc nhận dữ liệu từ cloudMQTT. Cụ thể ở đây là các dòng lệnh 2 và 3.
- Sau khi tạo luồng thực thi trên. ApplicationAnalytic tiến hành mở một cổng kết nối với port được nhập vào từ giao diện đã thiết kế sẵn. Lúc này một cổng kết nối được thiết lập. Server bắt đầu việc lắng nghe kết nối từ các client.
- Vòng lặp while ở đây có thực hiện nhiệm vụ lắng nghe kết nối từ các client, khi một client kết nối, một luồng thực thi mới được tạo ra để giữ giao tiếp này.
- Hàm log ở đây được hiện thực với để xem thử hiện có những client đã và đang kết nối với Analytic Application.

#### 5.2.3 ServiceThread

---

```
1 private static class ServiceThread extends Thread {
2     private int clientNumber;
3     private Socket socketOfServer;
4     public ServiceThread(Socket socketOfServer, int clientNumber) {
5         this.clientNumber = clientNumber;
6         this.socketOfServer = socketOfServer;
7         log("New connection with client# " + this.clientNumber + " at " +
8             ↪ socketOfServer);
9     }
10    @Override
11    public void run() {
12        try{
13            String clientSentence, serverSentence;
14            BufferedReader inFromClient = new BufferedReader(new
15                ↪ InputStreamReader(socketOfServer.getInputStream()));
16            DataOutputStream outToClient = new
17                ↪ DataOutputStream(socketOfServer.getOutputStream());
```

```
15         clientSentence = inFromClient.readLine();
16         List<GPS> lst=getData.getDatabaseData(clientSentence);
17         ListGPS list=new ListGPS(lst);
18         serverSentence = list.analysisGPS()+ '\n';
19         outToClient.writeBytes(serverSentence);
20     } catch (IOException e) {
21     }
22 }
23 }
```

#### Chức năng:

- Class ServiceThread có chức năng lấy dữ liệu từ client gửi lên sau đó tiến hành mang dữ liệu này đi phân tích rồi sau đó gửi trả kết quả ngược lại cho client.

#### Hiện thực:

- Class được hiện thực theo cơ chế đa luồng bằng cách kế thừa class Thread trong hệ thống.
- Hàm khởi tạo của nó nhận vào hai thông tin là thông tin của client đang giữ luồng giao tiếp này với Analytic Application và thứ hai là thứ tự mà client này giao tiếp với Analytic Application.
- Hàm run() ở đây chính là hàm xử lý việc thao tác dữ liệu giữa Analytic Application và client. Đây là một trong những hàm quan trọng trong ứng dụng. Nó tiến hành việc phân tích gói tin mà client gửi lên, lấy data từ gói tin đó biến thành dạng String. Ở đây là String vì nhóm đã quy định trước tính chất của gói tin. Sau đó dữ liệu được gửi đến hàm xử lý dữ liệu thông qua lời gọi getData.getDatabaseData(clientSentence). Sau khi phân tích xong dữ liệu được gửi trở lại cho hàm. Nó bắt đầu tiến hành đóng gói dữ liệu lại và gửi ngược trở về client thông qua câu lệnh outToClient.writeBytes(serverSentence).

#### 5.2.4 MQTTRecieve

```
1 public class MQTTRecieve implements Runnable {
2     public MQTTRecieve() {
3     }
4     @Override
5     public void run() {
6         String topic = "Dell";
7         int qos = 1;
8         String broker = "tcp://m12.cloudmqtt.com:16534";
9         String clientId = "ClientIdAnalytic";
10        MemoryPersistence persistence = new MemoryPersistence();
11        try {
12            MqttClient mqttClient = new MqttClient(broker, clientId,
13                persistence);
14            mqttClient.setCallback(new MqttCallback() {
15                public void messageArrived(String topic, MqttMessage msg)
16                    throws Exception {
17                    String res = new String(msg.getPayload());
18                }
19            });
20        } catch (Exception e) {
21            e.printStackTrace();
22        }
23    }
24 }
```

```

16         System.out.println("Recived from MQTT: " + res + "\n");
17         CommitData cm = new CommitData();
18         cm.commitData(res);
19     }
20 });
21 MqttConnectOptions connOpts = new MqttConnectOptions();
22 connOpts.setCleanSession(true);
23 connOpts.setUserName("phamvanlinh");
24 connOpts.setPassword(new char[] { '1', '4', '3', '1', '9', '9',
    ↪ '7' });
25 mqttClient.connect(connOpts);
26 mqttClient.subscribe(topic, qos);
27 } catch (MqttException me) {
28 }
29 }
30 }

```

#### Chức năng:

- Class MQTTRecieve có chức năng nhận dữ liệu từ cloudMQTT gửi về sau đó mang dữ liệu đó gửi cho phương thức thuộc class CommitData để tiến hành lưu xuống cơ sở dữ liệu.

#### Hiện thực:

- Class được hiện thực theo cơ chế đa luồng bằng cách kế thừa interface Runnable trong hệ thống.
- Các thông tin được thiết lập trong class này tương tự như trong class MQTTPulishData đã được hiện thực ở mục 5.1.4. Nhóm xin phép không nhắc lại ở đây. Chỉ khác ở chỗ ở đây ta phải gọi phương thức subscribe để nhận dữ liệu được gửi về từ cloudMQTT.
- Sau khi dữ liệu được gửi về, sẽ được lưu vào một biến kiểu String sau đó truyền vào hàm đã được viết sẵn để tiến hành lưu dữ liệu xuống Database. Quá trình lưu dữ liệu và truy xuất dữ liệu sẽ được đề cập ở các mục tiếp theo.

#### 5.2.5 Lưu trữ dữ liệu

Để đảm bảo tính khách quan và chính xác cho ứng dụng, tất cả dữ liệu kinh độ, vĩ độ của người dùng (client) mỗi lần gửi lên Gateway sau khi đẩy lên cloudMQTT sẽ được đưa xuống Application Analysis và tiến hành lưu trữ vào database riêng biệt (cụ thể là SQL Server).



**Hình 12:** Lưu trữ dữ liệu từ cloud vào database.

Việc tạo kết nối (Connection) giữa SQL Server trong java được minh họa sơ lược trong Hình 16. Dữ liệu nhận được là một chuỗi ký tự bao gồm tên, kinh độ, vĩ độ với định dạng "ten|kinhdo|vido" sẽ được tách ra các trường dữ liệu tương ứng và ghi xuống database.

Việc tạo kết nối (Connection) giữa SQL Server trong java được minh họa sơ lược trong Hình 16. Dữ liệu nhận được là một chuỗi ký tự bao gồm tên, kinh độ, vĩ độ với định dạng "ten|kinhdo|vido" sẽ được tách ra các trường dữ liệu tương ứng và ghi xuống database.

```

1 public class CommitData {
2     Connection conn = createConnection.createConnect();

```

GPS
- KinhDo: Double - ViDo: Double
+ GPS(Double KD,Double VD): void + getKinhDo(): Double + getViDo(): Double

Hình 13: Class GPS

ListGPS
- lst: List<GPS>
+ ListGPS(List<GPS>): void + analysisGPS(): String + Calculater(GPS, GPS): boolean + topGPS(List<List<GPS>>): String

Hình 14: Class ListGPS

```

3      public void commitData(String str) {
4          // tách lấy từng nội dung trong chuỗi nhận được thành mảng
5          String[] splitString = str.split("\\|");
6          String ten = splitString[0];
7          String kinhdo = splitString[1];
8          String vido = splitString[2];
9          try {
10             // tạo kết nối và thực thi câu lệnh chèn dữ liệu vừa tách
11             Statement st = conn.createStatement();
12             st.executeUpdate("INSERT INTO DULIEU (ten,kinhdo,vido) VALUES
13                 ↳ (" + "\"" + ten + "\"" + "," + "\"" + kinhdo + "\"" + "," +
14                 ↳ + "\"" + vido + "\"" + ");");
15             System.out.println("Saved data to database successfully!");
16         } catch (SQLException e) {
17             e.printStackTrace();
18         } finally {
19             if (conn != null) try { conn.close(); } catch (Exception e) {}
20         }
21     }

```

Điều này có ý nghĩa lớn trong việc tính toán để giải quyết bài toán mà ứng dụng đặt ra. Để nhận dữ liệu từ cloudMQTT, chúng ta có thể theo xem lại mô tả ở mục 5.2.4.

### 5.2.6 Tổng quan truy xuất Database và kết quả xử lý

Khi một Client gửi yêu cầu lấy thông tin về địa điểm hay đến của 1 Client nào đó thuộc hệ thống thì Analytic Application sẽ tiếp nhận và gửi yêu cầu xuống Database để lấy lên một danh sách các kinh độ vĩ độ và được đưa vào 1 thành một List(GPS) với GPS là một class được khởi tạo với 2 thuộc tính KinhDo và ViDo. Danh sách này được giữ bởi 1 đối tượng thuộc lớp ListGPS trong lớp này sẽ có thuộc tính lst chứa List(GPS) được truyền vào ở Constructor và hàm xử lý chính ở đây là analysisGPS() sẽ trả về một String có định dạng ở 4.2.3 Định dạng 3 hoặc nếu không tìm thấy tọa độ nào thì gửi về theo định dạng 4.2.4 Định dạng lỗi. Các hàm Calculater và topGPS là các hàm xử lý được gọi bên trong hàm analysisGPS().

### 5.2.7 Chi tiết hàm xử lý ở Analytic Application

```
1 public String analysisGPS(){
2     List<List<GPS>> lstGPS=new ArrayList<List<GPS>>();
3
4     while (!lst.isEmpty()){
5         List<GPS> lsttemp =new ArrayList<GPS>();
6         lsttemp.add(lst.get(0));
7         lst.remove(0);
8         int i=0;
9         while (i<lst.size())
10             if (Calclater(lsttemp.get(0),lst.get(i))){
11                 lsttemp.add(lst.get(i));
12                 lst.remove(i);
13             }
14             else i+=1;
15         lstGPS.add(lsttemp);
16     }
17     return topGPS(lstGPS);
18 }
```

hàm analysisGPS() của class ListGPS duyệt danh sách các tọa độ chứa trong lst(thuộc tính của class ListGPS).Thực hiện gom nhóm các điểm gần nhau với bán kính 100m các được lưu vào lstGPS có kiểu List<List<GPS>.Việc gom nhóm thực hiện bằng cách lấy tọa độ đầu tiên trong lst làm tâm và toàn bộ những điểm có tọa độ nằm bán kính 100m bỏ vào 1 List và những điểm này khi được lấy ra thì sẽ xóa khỏi lst.Nhóm các điểm này sẽ được thêm vào lstGPS ở trên. Lst chứa các phần tử còn lại tiếp tục thực hiện chọn phần tử đầu tiên làm tâm và thực hiện tương tự cho đến khi lst không còn phần tử nào. Việc kiểm tra 1 điểm có nằm trong bán kính bán kính 100 tính từ 1 điểm được chọn làm tâm sẽ được tính toán trong hàm Calclater. Việc chọn ra các nhóm và định dạng để trả về nằm trong hàm topGPS.

- **Calclater** Nhận 2 tham số, tham số đầu tiên là tâm tham số thứ 2 là điểm cần kiểm tra. Sử dụng công thức kiểm tra một điểm nằm trong(hoặc trên) đường tròn:

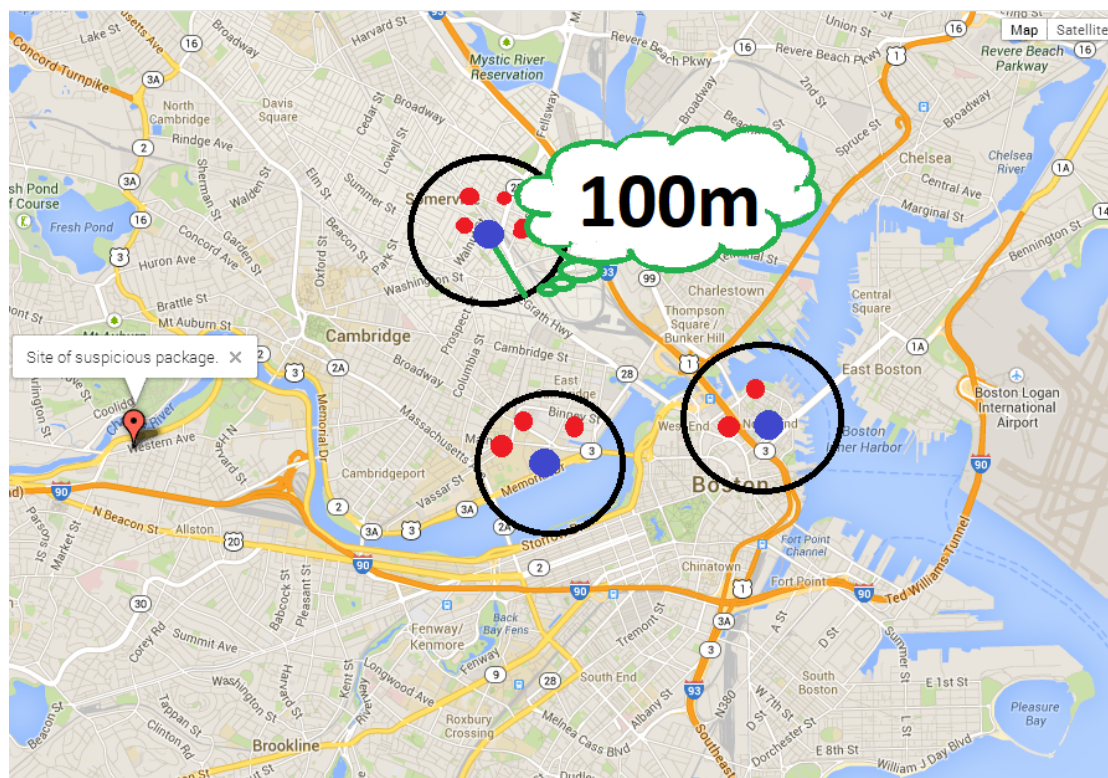
$$((x-a)^2 + (y-b)^2) \leq r^2$$

Ở đây bán kính được cho là 0.001(theo thực nghiệm của nhóm tính được thì sự chênh lệch của tọa độ giữa 2 điểm bằng 0.001 tương ứng với khoảng cách sấp xỉ 100m).Nếu điểm đó nằm trong bán kính 100m hàm sẽ trả về true ngược lại sẽ trả về false.

```
1 public boolean Calclater(GPS center,GPS pointOut){
2     return
3         ↪ (Math.pow((center.getKinhDo()-pointOut.getKinhDo()),2)+
4           Math.pow((center.getViDo()-pointOut.getViDo()),2))<=
5           (Math.pow(0.001,2));
6 }
```

- **topGPS** : hàm nhận vào một danh sách với phần tử là một nhóm các tọa độ được gom ở trên sử dụng hàm sort ở trong Java và override lại phương thức compare để có thể sắp





Hình 15: Mô phỏng xử lý Analytic Application

xếp theo kích thước của phần tử. Sau đó nếu danh sách có 3 phần tử trở lên thì trả về ba phần tử có kích thước lớn nhất. Có số phần tử từ 1 hoặc 2 thì trả về các phần tử đó. Nếu danh sách rỗng (User không tồn tại trong Database) thì gửi về chuỗi lỗi 4.2.4 Định dạng lỗi. Nếu danh sách khác rỗng thì tính trung bình tọa độ của từng nhóm, trung bình kinh độ bằng cách lấy tổng kinh độ của các phần tử trong nhóm chia cho kích thước của nhóm tương tự cho cách tính trung bình vĩ độ. Rồi gửi kết quả về Client theo 4.2.4 Định dạng lỗi.

```

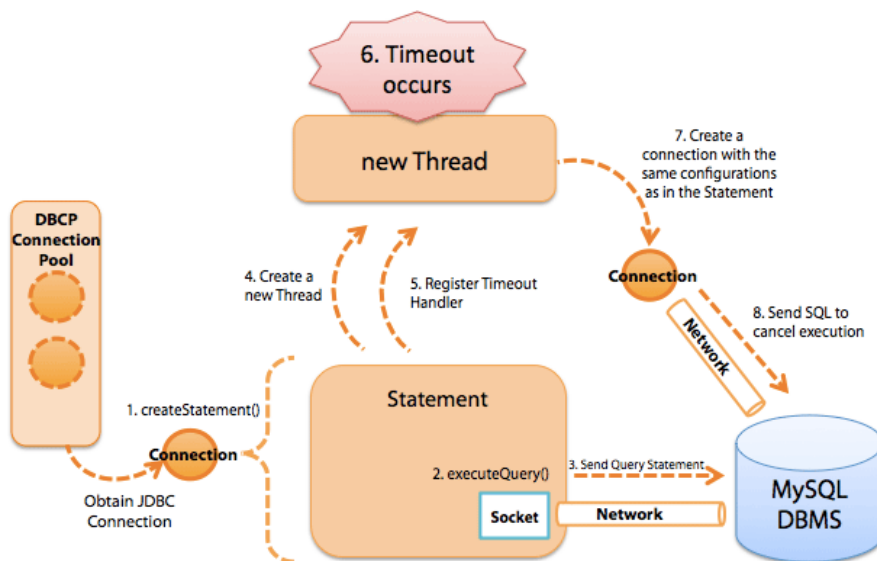
1      public String topGPS(List<List<GPS>> lst){
2
3          if(lst.size()!=0)
4          {
5
6              lst.sort(new Comparator<List<GPS>>() {
7                  @Override
8                  public int compare(List<GPS> o1, List<GPS> o2) {
9                      return Integer.compare(o2.size(), o1.size());
10                 }
11             });
12             for (List<GPS> item:lst)
13                 System.out.println(item.size());;
14             String result="";
15             for(int i=0;i<3;i++) {

```

```

15         double KinhDo=0.0,ViDo=0.0;
16         for (GPS item:lst.get(i)){
17             KinhDo+=item.getKinhDo();
18             ViDo+=item.getViDo();
19         }
20
21
22         ↵ result+=KinhDo/lst.get(0).size()+"|"+ViDo/lst.get(0).size();
23         if(i>=lst.size()-1||i==2)break;
24         else result+="-";
25     }
26
27     return result;
28 }
29 else {
30     return "9999|9999";
31 }
32 }

```

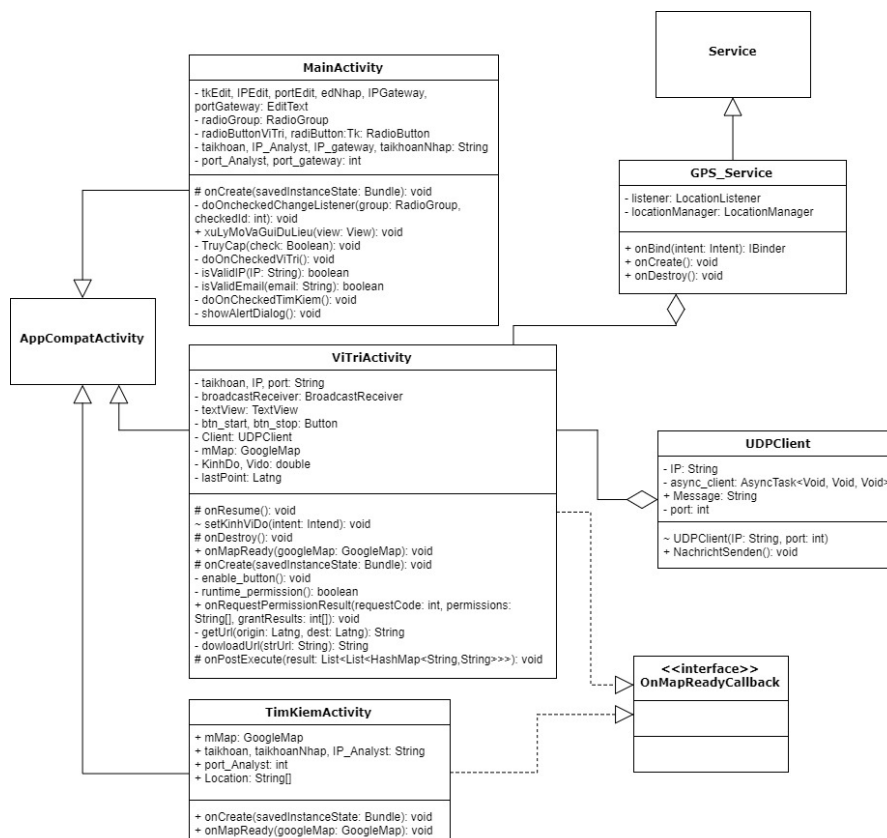


Hình 16: Lưu dữ liệu vào SQL Server database bằng JDBC.



## 5.3 Hiện thực App Client

### 5.3.1 Class Diagram



Hình 17: Mô hình của ứng dụng

### 5.3.2 Google Map

Để nhúng Google Map vào trong ứng dụng, trước hết ta phải đăng kí lấy key để được phép sử dụng API của Google. Sau đó ta cấp quyền trong AndroidManifest.xml:

```

1 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
2 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
3 <uses-permission android:name="android.permission.INTERNET"/>

```

Chức năng:

- Cấp các quyền liên quan đến vị trí và Internet để có thể sử dụng Google Map.

```

1 public void onMapReady(GoogleMap googleMap) {
2     mMap = googleMap;
3     if(KinhDo!=9999&&ViDo!=9999) {

```

```
4         LatLng LocationNew = new LatLng(KinhDo, ViDo);
5         mMap.addMarker(new
            ↳ MarkerOptions().position(LocationNew).title("Vị trí
            ↳ mới").anchor(0.5f, 0.5f));
6         mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(LocationNew,
            ↳ 19));
7
8         String url = getUrl(LocationNew, lastPoint);
9         FetchUrl FetchUrl = new FetchUrl();
10        FetchUrl.execute(url);
11    }
12 }
```

---

#### Chức năng:

- addMaker(...): vẽ lên trên Google Map 1 mốc để đánh dấu vị trí.
- moveCamera(...): hiển thị điểm cần quan sát ở chính giữa màn hình điện thoại.
- getUrl(LocationNew, lastPoint): trả về 1 string chứa đường dẫn đến trang web chứa đường đi từ điểm LocationNew đến điểm lastPoint.
- FetchUrl.execute(url): vẽ lên Google Map đường đi giữa 2 điểm được tích hợp trong biến url.

#### 5.3.3 GPS

---

```
1 public void onCreate() {
2     listener = new LocationListener() {
3         @Override
4         public void onLocationChanged(Location location) {
5             Intent i = new Intent("location_update");
6             ↳ i.putExtra("coordinates",location.getLatitude()+"| "
7             ↳ +location.getLongitude());
8             sendBroadcast(i);
9         }
10        @Override
11        public void onProviderDisabled(String s) {
12            Intent i = new
13            ↳ Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
14            i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
15            startActivity(i);
16        }
17    };
18
19    locationManager = (LocationManager)
20    getSystemService(Context.LOCATION_SERVICE);
21
22    //noinspection MissingPermission
```

```
21     locationManager.requestLocationUpdates(  
22         locationManager.NETWORK_PROVIDER,1000,0,listener);  
23     }  
24     public void onDestroy() {  
25         super.onDestroy();  
26         if(locationManager != null){  
27             //noinspection MissingPermission  
28             locationManager.removeUpdates(listener);  
29         }  
30     }
```

---

#### Chức năng:

- onCreate: khởi tạo và bắt sự kiện thay đổi tọa độ.
- onDestroy: dừng việc bắt sự kiện thay đổi tọa độ.

#### Hiện thực:

- onCreate:location.getLatitude() lấy vị trí kinh độ và location.getLongitude() lấy vị trí vĩ độ sau đó đẩy vào biến thuộc lớp Intent với tên trường là coordinates và sẽ được lấy ra ở class ViTriActivity.locationManager là biến quản lý thiết bị định vị trên Android được đặt với kiểu lấy tọa độ sử dụng mạng(*NETWORK\_PROVIDER*) với thời gian 1000(16s).
- onDestroy: hủy sự kiện thay đổi tọa độ bằng hàm removeUpdates với sự kiện listener khi Client thực hiện dừng việc thu thập vị trí.

#### 5.3.4 Client UDP

---

```
1     public void NachrichtSenden(){  
2         async_cient = new AsyncTask<Void, Void, Void>() {  
3             @Override  
4             protected Void doInBackground(Void... params) {  
5                 DatagramSocket ds = null;  
6                 try {  
7                     InetAddress addr = InetAddress.getByName(IP);  
8                     ds = new DatagramSocket(port);  
9                     DatagramPacket dp;  
10                    dp = new DatagramPacket(Message.getBytes(),  
11                        ↳ Message.getBytes().length, addr, port);  
12                    ds.setBroadcast(true);  
13                    ds.send(dp);  
14                } catch (Exception e) {  
15                    e.printStackTrace();  
16                } finally {  
17                    if (ds != null) {  
18                        ds.close();  
19                    }  
20                }  
21                return null;  
22            }  
23        }  
24    }
```



```
21     }  
22     }  
23 }
```

---

#### Chức năng:

- Chuỗi Message nhận được lên Getway với địa chỉ IP và Port được cung cấp.

#### Hiện thực:

- Sử dụng biến thuộc lớp InetAddress để lưu giữ địa chỉ IP, biến thuộc lớp DatagramPacket để lưu dữ liệu gửi đi, độ dài chuỗi gửi đi, địa chỉ được lưu vào biến của InetAddress ở trên và port. Sau đó gửi đi bằng phương thức send của biến thuộc lớp DatagramSocket (biến này được khởi tạo với tham số truyền vào là port của server).

## 6 Đánh giá sản phẩm

### 6.1 Ưu điểm

- Xây dựng được ứng dụng có đầy đủ các thành phần và chức năng theo kiến trúc hệ thống ứng dụng IoT như yêu cầu của bài tập lớn.
- Các chức năng cũng như giao thức chuyển đổi dữ liệu được xây dựng thỏa mãn yêu cầu đề bài.
- Ứng dụng đơn giản dễ sử dụng, đáp ứng nhiều người dùng tại một thời điểm.
- Tùy biến việc nhập cổng (port) thay vì cố định và có giao diện thân thiện với người dùng.

### 6.2 Nhược điểm

- Giao diện của ứng dụng chưa bắt mắt.
- Thời gian xử lý để thu thập và gửi dữ liệu có độ trễ tương đối lớn.
- Xác định vị trí người dùng có sai số nhỏ.
- Trong việc xử lý ở Analytic Application chọn vị trí tâm một cách khá ngẫu nhiên dẫn đến việc gom nhóm có phần mang tính ngẫu nhiên.

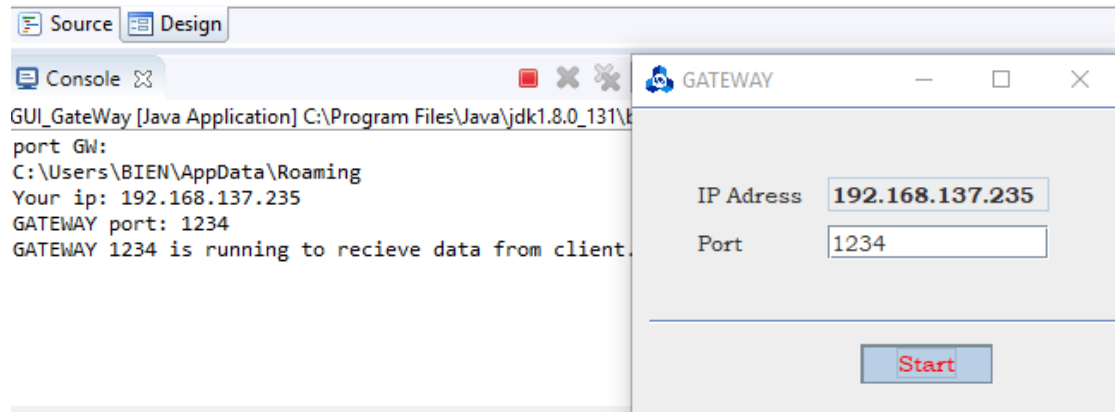
### 6.3 Hướng phát triển

- Cải thiện tốc độ xử lý ở Client cũng như ở các Server.
- Xây dựng ứng dụng bắt mắt và có tính tương tác cao hơn với người dùng.
- Phát triển thêm các tính năng nhờ dữ liệu thu thập được.

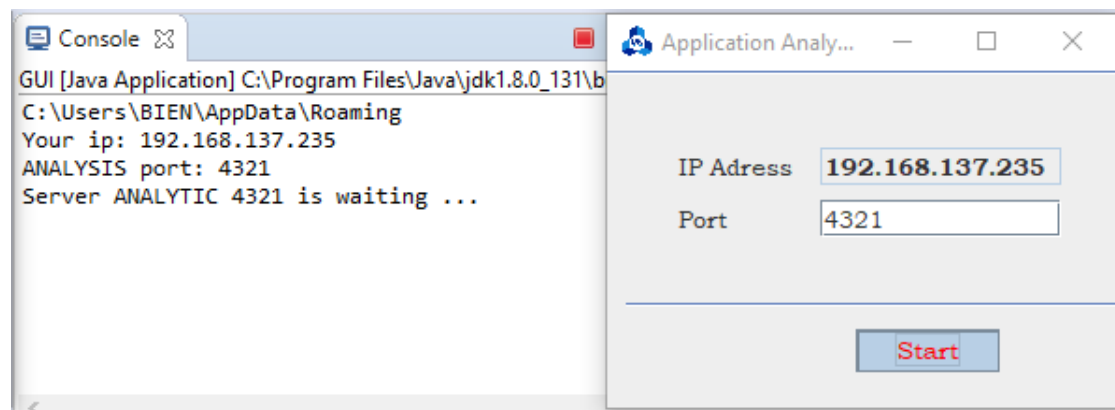
## 7 Hướng dẫn sử dụng

### 7.1 Khởi động Gateway và Application analysis

Ứng dụng sẽ tự động lấy IP và hiển thị. Port được yêu cầu nhập và sau đó có thể khởi động Gateway và Application Analytic(Server) bằng cách Click vào Start.

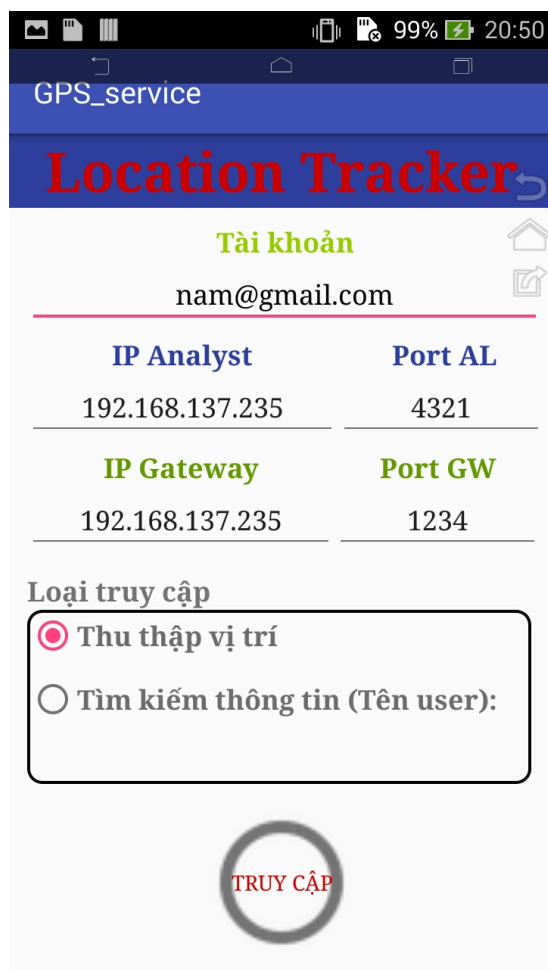


Hình 18: Ứng dụng hiển thị IP và nhập port cho Gateway

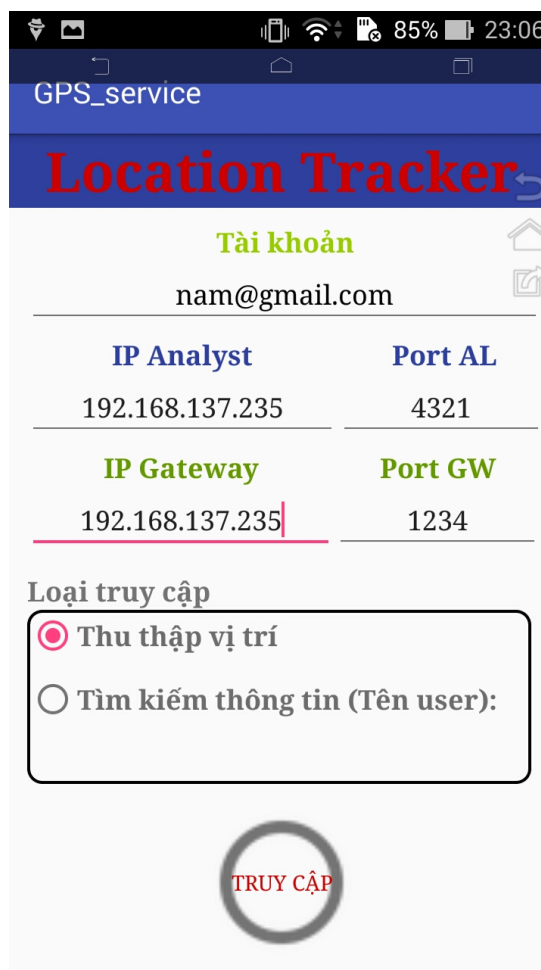


Hình 19: Hiển thị IP và nhập port cho Analytic Aplicationnn

## 7.2 Đăng nhập Ứng dụng



Hình 20: Màn hình đăng nhập



Hình 21: Nhập thông tin

Ứng dụng yêu cầu nhập đầy đủ thông tin ở màn hình đăng nhập bao gồm:

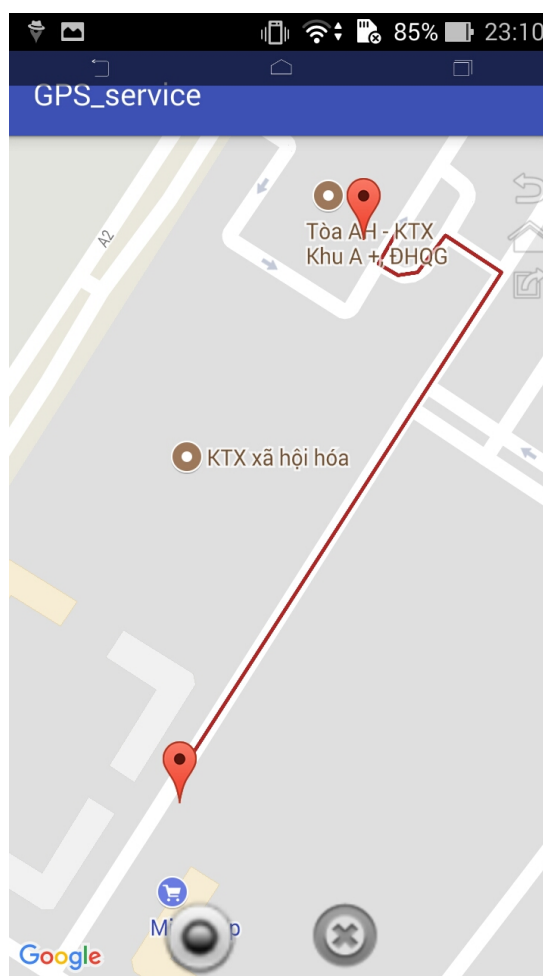
- Tài khoản: Cần nhập Email của người dùng (có kiểm tra tính hợp lệ của Email), cần nhập chính xác để phân biệt và ngoài ra để có thể phục vụ chức năng gửi kết quả về Email (chức năng sẽ phát triển thêm).
- IP Analyst và Port AL: lần lượt nhập địa chỉ của IP của Analytic Application (Server) và Port của Analytic Application (IP và Port này lấy từ Ứng dụng ở Hình 19)
- IP Gateway và Port GW: lần lượt nhập địa chỉ của IP của Gateway và Port của Gateway (IP và Port này lấy từ Ứng dụng ở Hình 18)
- Loại truy cập: có 2 lựa chọn:
  - Thu thập vị trí: Là chức năng thứ nhất giúp thu thập vị trí của người dùng

- Tìm kiếm thông tin( Tên user): Là chức năng thứ hai.Ở chức năng này cần nhập thông tin(Email) của người dùng muốn lấy dữ liệu.
- Truy cập : Sau khi hoàn tất việc điền thông tin ấn vào Truy cập sẽ chuyển đến màn hình chức năng tiếp theo.

### 7.3 Chức năng thứ nhất: Thu thập thông tin



Hình 22: Màn hình khởi động của chức năng 1

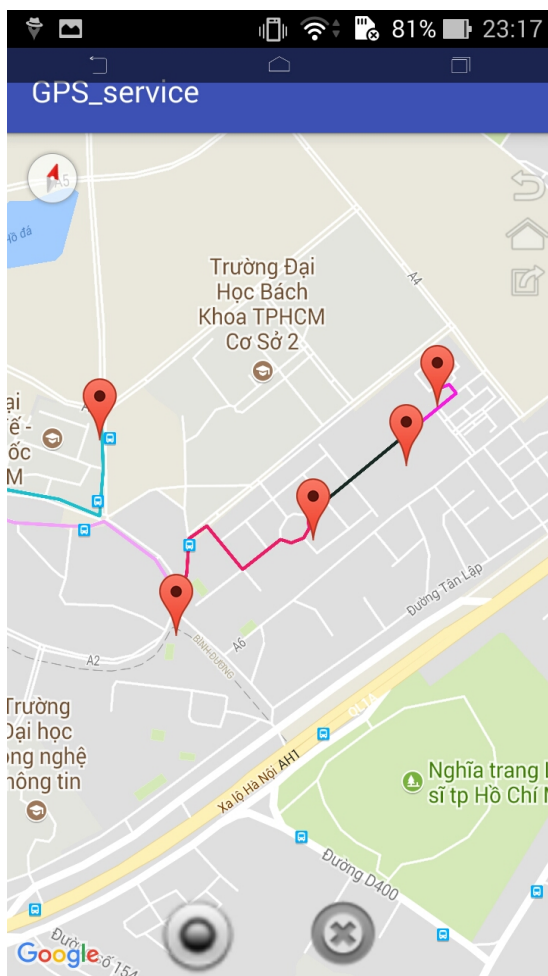


Hình 23: Bắt đầu thu thập tọa độ

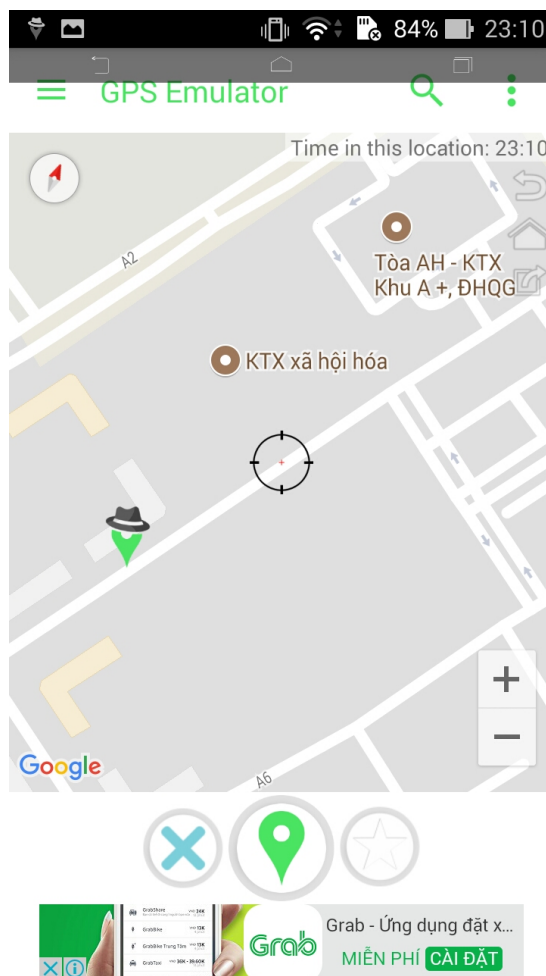
Ở màn hình chức năng này có 2 nút tùy chọn:

- Nút ở góc dưới bên trái(dấu tròn): có chức năng bắt đầu thu thập vị trí.
- Nút ở góc dưới bên phải(dấu nhân): có chức năng dừng việc thu thập vị trí.

Ứng dụng sẽ tự động vẽ các điểm thu thập được lên màn hình và nối các vị trí liên tiếp lại.



Hình 24: Thu thập nhiều vị trí

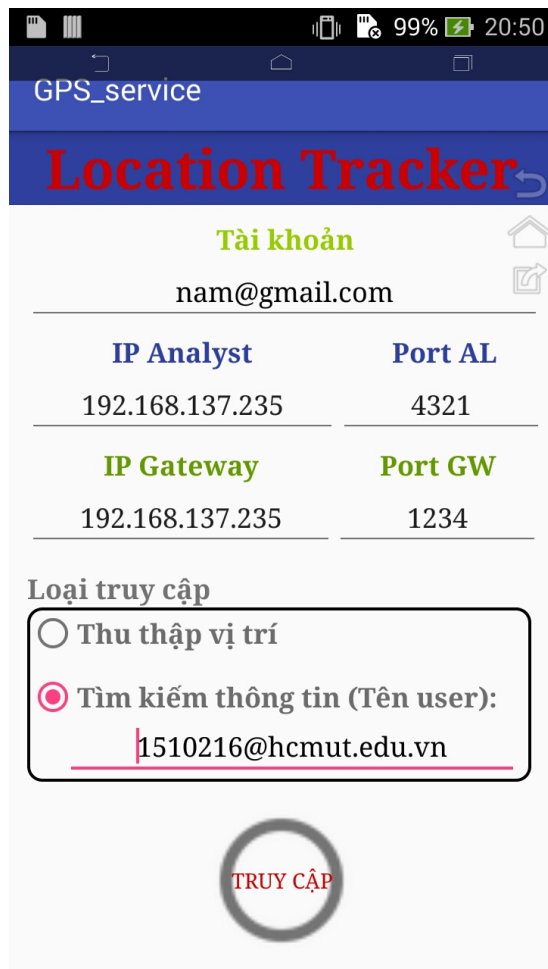


Hình 25: GPS Emulator giả lập tọa độ

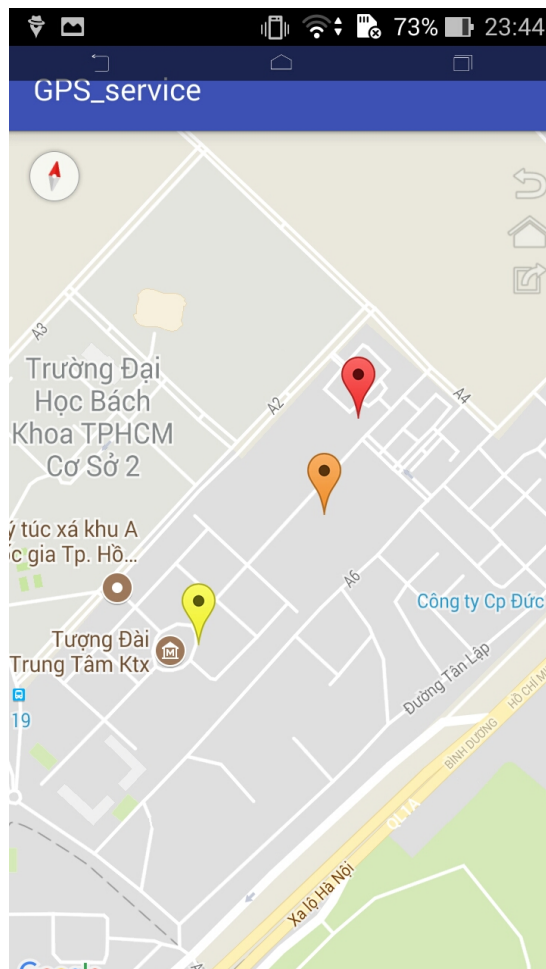
Các vị trí thu thập được sẽ hiển thị lên ứng dụng với các đường nối được phân biệt các màu. Ứng dụng GPS Emulator để giả lập tọa độ trên điện thoại di động.



## 7.4 Chức năng thứ hai: gửi yêu cầu và nhận kết quả



Hình 26: Chọn chức năng thu thập dữ liệu



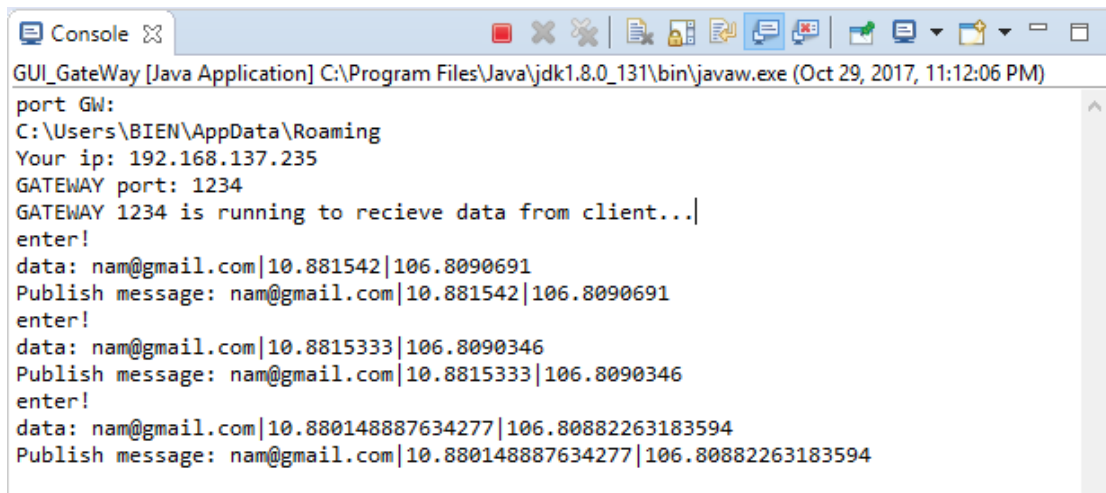
Hình 27: Kết quả của việc gửi yêu cầu

Nếu người dùng lựa chọn loại truy cập là Tìm kiếm thông tin(Tên user) và nhập thông tin người cần tìm sau đó chọn truy cập. Ứng dụng sẽ chuyển sang chức năng số hai là gửi yêu cầu và nhận kết quả trả về bởi Analytic Application(Server).

Màn hình sẽ hiển thị tối đa 3 vị trí với:

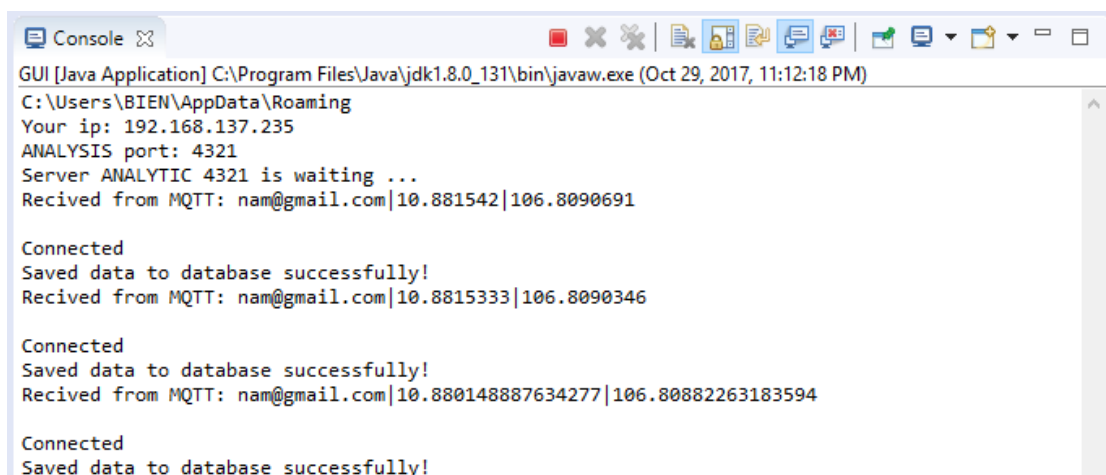
- Màu đỏ: Vị trí với số lần đến nhiều nhất.
- Màu cam: Vị trí với số đến nhiều thứ 2(nếu có).
- Màu vàng: Vị trí với số lần đến nhiều thứ 3(nếu có).

## 7.5 Kết quả nhận dữ liệu ở Getway và Analytic Application



```
GUI_GateWay [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (Oct 29, 2017, 11:12:06 PM)
port GW:
C:\Users\BIEN\AppData\Roaming
Your ip: 192.168.137.235
GATEWAY port: 1234
GATEWAY 1234 is running to recieve data from client...|
enter!
data: nam@gmail.com|10.881542|106.8090691
Publish message: nam@gmail.com|10.881542|106.8090691
enter!
data: nam@gmail.com|10.8815333|106.8090346
Publish message: nam@gmail.com|10.8815333|106.8090346
enter!
data: nam@gmail.com|10.880148887634277|106.80882263183594
Publish message: nam@gmail.com|10.880148887634277|106.80882263183594
```

Hình 28: Dữ liệu nhận từ Client gửi lên Gateway ở quá trình 7.3



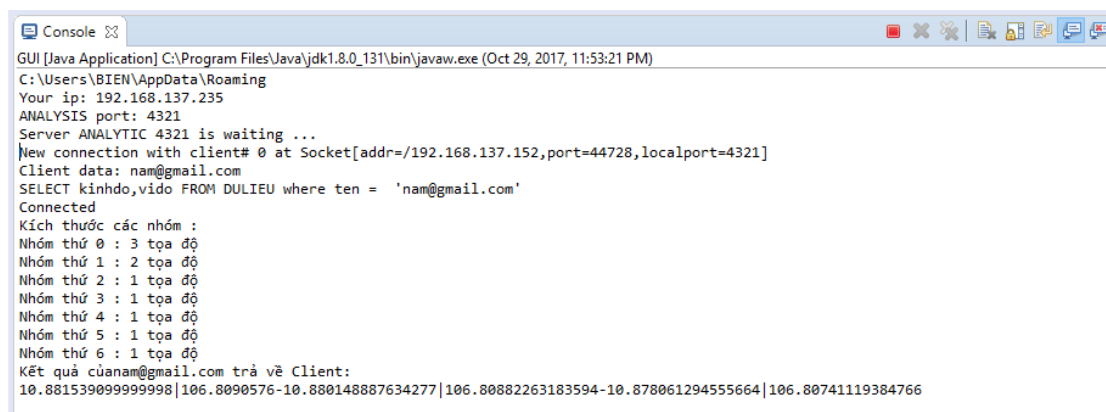
```
GUI [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (Oct 29, 2017, 11:12:18 PM)
C:\Users\BIEN\AppData\Roaming
Your ip: 192.168.137.235
ANALYSIS port: 4321
Server ANALYTIC 4321 is waiting ...
Recived from MQTT: nam@gmail.com|10.881542|106.8090691

Connected
Saved data to database successfully!
Recived from MQTT: nam@gmail.com|10.8815333|106.8090346

Connected
Saved data to database successfully!
Recived from MQTT: nam@gmail.com|10.880148887634277|106.80882263183594

Connected
Saved data to database successfully!
```

Hình 29: Dữ liệu nhận từ MQTT sau khi Gateway gửi lên và thông báo lưu xuống Database thành công



```
GUI [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (Oct 29, 2017, 11:53:21 PM)
C:\Users\BIEN\AppData\Roaming
Your ip: 192.168.137.235
ANALYSIS port: 4321
Server ANALYTIC 4321 is waiting ...
New connection with client# 0 at Socket[addr=/192.168.137.152,port=44728,localport=4321]
Client data: nam@gmail.com
SELECT kinhdo,vido FROM DULIEU where ten = 'nam@gmail.com'
Connected
Kích thước các nhóm :
Nhóm thứ 0 : 3 tọa độ
Nhóm thứ 1 : 2 tọa độ
Nhóm thứ 2 : 1 tọa độ
Nhóm thứ 3 : 1 tọa độ
Nhóm thứ 4 : 1 tọa độ
Nhóm thứ 5 : 1 tọa độ
Nhóm thứ 6 : 1 tọa độ
Kết quả củanam@gmail.com trả về Client:
10.881539099999998|106.8090576-10.880148887634277|106.80882263183594-10.878061294555664|106.80741119384766
```

Hình 30: Kết quả sau quá trình xử lý dữ liệu ở Analytic Application và gửi về Client ở quá trình 7.4



## 8 Chức năng mở rộng

- Tích hợp Google Map vào ứng dụng.
- Hệ thống sử dụng multi thread hỗ trợ nhiều người dùng cùng lúc.
- Sử dụng SQL Server để lưu trữ số lượng lớn dữ liệu, mà ở đây là vị trí của người dùng đã check-in.

## Tài liệu

- [1] Filip Vujovic. "**Android - Get GPS location via service.**" YouTube, Published on Jun 17, 2016  
<https://www.youtube.com/watch?v=lvcGh2ZgHeA>
- [2] Gia sư Tin học (12/04/2016), "**Lập trình đa luồng trong Java**".  
<http://giasutinhoc.vn/lap-trinh-java-co-ban/da-luong-trong-java-java-multithreading-bai-7/>
- [3] CloudMQTT - A Globally distributed MQTT broker, "**Introduction to CloudMQTT**"  
<https://www.cloudmqtt.com/docs.html>
- [4] Google Map  
<https://developers.google.com/maps/documentation/android-api/start?hl=vi>
- [5] Jim Kurose, Keith Ross, Addison-Wesley **Computer Networking – A top-down approach**, 5th edition, 2010