

รายงานฉบับสมบูรณ์
Senior Project Final Report

เรื่อง
แพลตฟอร์มเปิดอินเทอร์เน็ตของสรรพสิ่งสำหรับระบบสุขภาพของประเทศไทยพร้อมระบบ
ปัญญาประดิษฐ์
Open Healthcare IoT Platform with AI

โดย

นายณัฐภัทร	จารุชัยสิทธิกุล	รหัสนิสิต 6230177821
นางสาวณิชกานต์	ชัยพจนา	รหัสนิสิต 6231322621
นางสาวมารีนญา	ตะโจปะรัง	รหัสนิสิต 6231352421
นางสาวศิวกาญจน์	จิตต์วโรดม	รหัสนิสิต 6231363321

อาจารย์ที่ปรึกษา
รศ.ดร. กุลธิดา โรจน์วิบูลย์ชัย

รายงานนี้เป็นส่วนหนึ่งของวิชา 2110489 โครงการรบบยอดวิศวกรรมคอมพิวเตอร์ 2
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ประจำปีการศึกษา 2565

บทคัดย่อ

ในปัจจุบันเทคโนโลยีถูกพัฒนาอย่างต่อเนื่องอย่างรวดเร็ว ทุกหน่วยงานมีการปรับใช้เทคโนโลยีเข้ากับระบบการทำงานของตัวเอง หน่วยงานทางการแพทย์ก็เช่นกัน มีการใช้อุปกรณ์ IoT ในการตรวจวัดข้อมูลสุขภาพของผู้ป่วย เช่น เซนเซอร์วัดความดัน เซนเซอร์วัดอัตราการเต้นของหัวใจ เซนเซอร์วัดปริมาณออกซิเจนในเลือด เป็นต้น แต่ยังมีขาดระบบกลางในการจัดเก็บและจัดการข้อมูลเหล่านั้น อีกทั้งอุปกรณ์แต่ละประเภทยังมีรูปแบบการส่งข้อมูลที่ต่างกัน ทำให้ต้องใช้ซอฟต์แวร์เฉพาะของแต่ละอุปกรณ์ในการจัดการข้อมูล ซึ่งมีค่าใช้จ่ายค่อนข้างสูง นอกจากนี้ยังส่งผลให้ผู้ใช้งานมีประสบการณ์การใช้งานที่ไม่น่าพอใจเท่าที่ควรอีกด้วย โครงการนี้นำเสนอการออกแบบพัฒนาแพลตฟอร์มสำหรับข้อมูลด้านสุขภาพจากอุปกรณ์ IoT เพื่อเป็นศูนย์กลางข้อมูลด้านสุขภาพ ลดค่าใช้จ่ายในการพัฒนาและดูแลระบบ IoT รวมทั้งอำนวยความสะดวกให้กับบุคลากรทางการแพทย์และผู้ป่วย โดยระบบแรก คือ ระบบการส่งข้อมูลจากอุปกรณ์เข้าสู่แพลตฟอร์มเพื่อเป็นสร้างศูนย์กลางที่รวมข้อมูลด้านสุขภาพที่รองรับอุปกรณ์ทุกประเภท ระบบที่สองคือการจัดเก็บข้อมูลในแพลตฟอร์มเพื่อจัดเก็บข้อมูลเป็นสัดส่วนเพื่อส่งข้อมูลย้อนหลังหรือข้อมูล real-time ให้กับผู้ใช้งานและนำข้อมูลไปวิเคราะห์เพื่อใช้ประโยชน์ต่อไป ระบบที่สามคือระบบติดต่อผู้ใช้งาน เพื่อตอบสนองต่อผู้ใช้งานในแต่ละประเภท และระบบที่สี่คือระบบแอดมินเพื่อดูแลจัดการอุปกรณ์และข้อมูลในระบบ โดยปัจจุบันได้ดำเนินการสร้างระบบที่รองรับการนำข้อมูลเข้าโดยผ่าน MQTT protocol Kafka server เพื่อบันทึกข้อมูลไปยัง database ระบบส่งข้อมูลย้อนหลังและข้อมูลแบบ real-time ให้แก่ผู้ใช้งาน และระบบแอดมินเพื่อจัดการกับอุปกรณ์การแพทย์และสิทธิของผู้ใช้งานในการเข้าถึงข้อมูลของอุปกรณ์แล้วเสร็จเรียบร้อยแล้ว

Abstract

Nowadays, technology is constantly evolving rapidly. Every department should adapt the technology to their own system. The medical departments do as well. IoT devices are used to measure health data such as pressure sensors, heart rate sensor, Blood Oxygen Sensor, etc. At present, it still lacks a central system to store and manage those data. Moreover, each type of device has a different data transmission protocol, so they require a specific software for each device to control. The software has a relatively high cost, and it also results in an unsatisfying user experience. This project proposes the design and development of a platform for IoT healthcare as a health information hub. It reduces the cost of developing and administering IoT systems, as well as facilitating both doctors and patients. The project divides the entire system into 4 subsystems, those are a data receiving system, a data management system, a user interaction system, and an admin system to manage devices and data in the system. Now, the data receiving system by using MQTT protocol Kafka server, sending data to users, and the admin system have already implemented.

สารบัญ

1. บทนำ.....	1
1.1. ที่มาและความสำคัญ.....	1
1.2. วัตถุประสงค์ของโครงการ.....	1
1.3. ขอบเขตของโครงการและผู้รับผิดชอบในแต่ละส่วน.....	2
1.4. แผนการดำเนินงาน.....	5
1.5. ประโยชน์ที่คาดว่าจะได้รับ.....	7
2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง.....	8
2.1. ทฤษฎีที่เกี่ยวข้อง.....	8
3. วิธีการนำเสนอ/แนวทางที่คิดว่าจะทำ.....	10
3.1. สถาปัตยกรรมของระบบ.....	10
3.2. ระบบภายใน.....	10
4. แผนการทำงาน.....	17
4.1. Developer Dashboard.....	17
4.2. Admin Dashboard.....	21
4.3. Webserver.....	24
4.4. Kafka Consumer.....	26
4.5. Kafka Producer.....	26
4.6. Logging.....	26
4.7. การ deploy web client.....	27
4.8. Web service cluster.....	28
4.9. Kafka cluster.....	29
4.10. การอัปโหลดข้อมูลจากอุปกรณ์การแพทย์ขึ้นสู่ google cloud storage.....	29
5. ผลการทดลอง/ผลการวิเคราะห์.....	32
5.1. เครื่องมือที่ใช้.....	32

5.2.	การเครื่องมือที่ใช้ในการทำ testing.....	34
5.3.	การทดสอบระบบ.....	35
6.	ผลกระทบทางสังคมของโครงการ	47
7.	บทสรุปและงานที่จะทำต่อไป.....	47
7.1.	บทสรุป	47
7.2.	อภิปรายผล.....	47
7.3.	ข้อจำกัด.....	50
7.4.	แนวทางในการต่อยอดในอนาคต.....	50
8.	เอกสารอ้างอิง	51

สารบัญรูปภาพ

รูปที่ 1 Gantt chart.....	5
รูปที่ 2 Gantt chart.....	6
รูปที่ 3 สถาปัตยกรรมของระบบ.....	10
รูปที่ 4 การส่งข้อมูลของอุปกรณ์ IoT.....	11
รูปที่ 5 ระบบการจัดการข้อมูล	12
รูปที่ 6 การส่งข้อมูลจาก GCS ไป BigQuery.....	13
รูปที่ 7 ระบบการให้บริการนักพัฒนา.....	14
รูปที่ 8 การขอข้อมูลย้อนหลัง	15
รูปที่ 9 การขอข้อมูลแบบ real-time	15
รูปที่ 10 Login Page	17
รูปที่ 11 การดู application ทั้งหมดของตนเอง.....	18
รูปที่ 12 การสร้าง application ใหม่.....	18
รูปที่ 13 การลบ application	19
รูปที่ 14 การแก้ไข application.....	19
รูปที่ 15 การดูและฟิลเตอร์ subscription ทั้งหมด.....	20
รูปที่ 16 การสร้าง subscription ใหม่.....	20
รูปที่ 17 หน้า My Account.....	21
รูปที่ 18 หน้า Login.....	21
รูปที่ 19 หน้าเว็บสำหรับการจัดการโมเดลอุปกรณ์.....	22
รูปที่ 20 หน้าเว็บสำหรับการเพิ่มโมเดลอุปกรณ์	22
รูปที่ 21 หน้าเว็บสำหรับจัดการบัญชีนักพัฒนา	23
รูปที่ 22 สำหรับการเพิ่มบัญชีนักพัฒนา.....	23
รูปที่ 23 สำหรับจัดการการขอ subscription ของนักพัฒนา	24
รูปที่ 24 API document.....	24
รูปที่ 25 log ของการอัปโหลดข้อมูลเข้า GCS	26
รูปที่ 26 Uptime.....	27
รูปที่ 27 CPU/Mem usage.....	27
รูปที่ 28 docker file ของ web client.....	28
รูปที่ 29 docker file ของระบบ	28

รูปที่ 30 folder structure ที่ไว้ build docker image.....	29
รูปที่ 31 nginx configuration file	29
รูปที่ 32 kafka cluster บน cloud	29
รูปที่ 33 รูปแบบของไฟล์ที่จัดเก็บใน GCS.....	30
รูปที่ 34 Flow การอัปโหลดข้อมูลเข้าสู่ GCS	31
รูปที่ 35 รูปแบบการเก็บ log ของการอัปโหลดข้อมูลขึ้น GCS.....	31
รูปที่ 36 GUI ของ BigQuery.....	33

สารบัญตาราง

ตารางที่ 1 ผู้รับผิดชอบงานในแต่ละส่วน.....	3
ตารางที่ 2 endpoint ต่าง ๆ ใน web service	25
ตารางที่ 3 ผลการทดสอบ Developer Dashboard.....	35
ตารางที่ 4 ผลการทดสอบ Admin Dashboard	37
ตารางที่ 5 ผลการทดสอบ application API.....	40
ตารางที่ 6 ผลการทดสอบ authentication และ authorization ของ kafka cluster.....	42
ตารางที่ 7 ผลการทดสอบ MQTT-Bridge	43
ตารางที่ 8 ผลการทำ load testing web service.....	46
ตารางที่ 9 ผลการทดสอบการขอข้อมูลแบบ real-time	46

1. บทนำ

1.1. ที่มาและความสำคัญ

Internet of Things (IoT) หรือ "อินเทอร์เน็ตในทุกสิ่ง" คือ การที่อุปกรณ์หรือสิ่งต่าง ๆ ได้ถูกเชื่อมต่อกับอินเทอร์เน็ต ทำให้มนุษย์สามารถรับข้อมูลและสั่งการควบคุมการใช้งานอุปกรณ์ต่างๆ ผ่านทางเครือข่ายอินเทอร์เน็ต เช่น การรับข้อมูลจากอุปกรณ์การแพทย์ของผู้ป่วยติดเตียง และส่งสัญญาณไปยังอุปกรณ์ผู้ดูแล

เทคโนโลยีอินเทอร์เน็ตประสานสรรพสิ่งจำนวนของอุปกรณ์ที่เชื่อมต่อระบบพื้นฐานสำหรับอินเทอร์เน็ตประสานสรรพสิ่ง (IoT platform) จะมีกว่าหลายพันล้านอุปกรณ์ อย่างไรก็ตามในปัจจุบัน การทำแพลตฟอร์ม IoT เพื่อรองรับอุปกรณ์การแพทย์นั้นมีค่าใช้จ่ายที่สูง โดยเซิร์ฟเวอร์ขนาดเล็กมีราคาเริ่มต้นที่ 10,000 บาท [1] ต่อเดือนซึ่งโรงพยาบาลในประเทศไทยมีมากกว่า 1,356 โรงพยาบาล [2] ทำให้มีค่าใช้จ่ายต่อเดือนประมาณ 13,560,000 บาท นอกจากนี้การใช้งานอาจจะต้องใช้งานกับซอฟต์แวร์เฉพาะจากผู้ผลิต ซึ่งมีค่าใช้จ่ายเพิ่ม และอาจสร้างประสบการณ์การใช้งานที่ไม่ดีแก่ผู้ใช้ อีกทั้งยังไม่มีกระบวนการรวบรวมข้อมูลที่ดีเพื่อนำเอาข้อมูลไปใช้ประโยชน์ เช่น การสร้างโมเดลปัญญาประดิษฐ์

จากปัญหาดังกล่าวข้างต้นจึงควรเตรียมระบบ IoT Platform ไว้ เพื่อช่วยโรงพยาบาลในประเทศไทยในการลดค่าใช้จ่ายและความเสี่ยงจากการพึ่งพาซอฟต์แวร์ดังกล่าวของต่างประเทศ และเพิ่มทางเลือกในประเทศให้เข้าถึงระบบ IoT Platform ได้มากขึ้น เพิ่มประสิทธิภาพการใช้ทรัพยากร โทรคมนาคม ในประเทศ และขยายโอกาสสร้างนวัตกรรมดิจิทัลได้มากขึ้น

1.2. วัตถุประสงค์ของโครงการ

- 1.2.1. เพื่อติดตามอาการของผู้ป่วยในห้องฉุกเฉิน ห้อง ICU และรพพยาบาลได้แบบอัตโนมัติ
- 1.2.2. เพื่อติดตามผู้ป่วยที่ได้รับการรักษาต่อเนื่องในรูปแบบของ Self Care หรือ Home Care ได้
- 1.2.3. เพื่อให้บริการผ่านทาง Telemedicine โดยที่แพทย์สามารถใช้ข้อมูลจากอุปกรณ์ IoT ประกอบการวินิจฉัย
- 1.2.4. เพื่อสร้างศูนย์รวมข้อมูลจากอุปกรณ์ทางการแพทย์ IoT จากทั่วประเทศ และเป็นแหล่งข้อมูลในการประมวลผลเชิงสถิติและพัฒนาระบบปัญญาประดิษฐ์ที่ช่วยในการวินิจฉัยโรค
- 1.2.5. เพื่อพัฒนาระบบสาธารณสุข โดยการแนะนำและให้การดูแลแบบ Personalization
- 1.2.6. เพื่อสร้างมาตรฐานการส่งข้อมูลที่เปิดให้นักพัฒนาสามารถนำไปใช้ในการพัฒนาอุปกรณ์ IoT ได้อย่างอิสระ
- 1.2.7. เพื่อพัฒนาส่วนเชื่อมต่อกับอุปกรณ์ทางการแพทย์ที่มีใช้อยู่เดิมที่อาจเป็น IoT หรือไม่เป็น IoT ให้สามารถใช้งานกับแพลตฟอร์มได้โดยตรง

- 1.2.8. เพื่อลดการพึ่งพาอุปกรณ์ทางการแพทย์และซอฟต์แวร์จากต่างประเทศที่มีราคาสูงและมีข้อจำกัดในการใช้งาน
- 1.2.9. เพื่อลดต้นทุนการพัฒนาและดูแลระบบ IoT ของผู้ให้บริการด้านสุขภาพโดยใช้การรวมศูนย์และเปิดให้ทุกฝ่ายสามารถใช้งานได้โดยอาจมีหรือไม่มีค่าใช้จ่าย
- 1.2.10. เพื่อส่งเสริมการใช้งาน Blockchain-based Personal Health Record เช่น HealthTAG [3] ในการให้สิทธิ์การเข้าถึงข้อมูลระหว่างโรงพยาบาลและหน่วยงานที่เกี่ยวข้องโดยมีการยินยอมจากเจ้าของข้อมูลซึ่งเป็นผู้รับบริการด้านสุขภาพ
- 1.2.11. เพื่อส่งเสริมโครงการ Smart ER, Smart EMS และ Smart ICU ให้สามารถใช้ประโยชน์จากข้อมูลจากอุปกรณ์ทางการแพทย์ให้มีประสิทธิภาพสูงสุด

1.3. ขอบเขตของโครงการและผู้รับผิดชอบในแต่ละส่วน

1.3.1. ขอบเขตของโครงการ

โครงการนี้พัฒนาระบบเพื่อส่งเสริมการทำแพลตฟอร์มรวบรวมข้อมูลจากอุปกรณ์ IoT ทางทางการแพทย์เพื่อรวบรวมข้อมูลให้อยู่ที่ส่วนกลางทำให้สามารถนำใช้ประโยชน์ได้ง่ายขึ้น ทั้งนี้ขอบเขตของแอปพลิเคชันที่จะพัฒนาขึ้นนั้นมีทั้งหมด 5 ระบบงาน ดังนี้

- 1.3.1.1. ระบบการส่งข้อมูลของอุปกรณ์การแพทย์
 - 1.3.1.1.1. รองรับข้อมูลที่ส่งผ่าน MQTT protocol
 - 1.3.1.1.2. รองรับการส่งข้อมูลของ Kafka
- 1.3.1.2. ระบบการจัดการข้อมูล
 - 1.3.1.2.1. การบันทึกข้อมูลลงฐานข้อมูล (MongoDB)
 - 1.3.1.2.2. การบันทึกข้อมูลลง Data Lake และการทำ Data Warehouse
 - 1.3.1.2.3. การทำ Dashboard แสดงข้อมูลที่ได้รับจากอุปกรณ์ใช้ Data Studio
- 1.3.1.3. ระบบการให้บริการนักพัฒนา
 - 1.3.1.3.1. การจัดการ Application ของตนเองในระบบ
 - 1.3.1.3.2. การจัดการ Subscription ของตนเองในระบบ
 - 1.3.1.3.3. การส่งข้อมูลอุปกรณ์ไปให้ผู้ใช้งานทั้งแบบ real-time และ historical
- 1.3.1.4. ระบบแอดมิน
 - 1.3.1.4.1. การจัดการผู้ใช้งาน (นักพัฒนา)
 - 1.3.1.4.2. การจัดการอุปกรณ์ IoT
 - 1.3.1.4.3. การจัดการสิทธิของผู้ใช้งานในการเข้าถึงข้อมูล
- 1.3.1.5. ระบบการจัดการ Log File

- 1.3.1.5.1. การบันทึกผลการอัปโหลดข้อมูลลง Data Lake
- 1.3.1.5.2. การบันทึกประวัติการเรียก API
- 1.3.1.5.3. การเก็บบันทึกสถานะของระบบ
- 1.3.1.5.4. การทำ Dashboard สำหรับดู Log

1.3.2. ผู้รับผิดชอบในแต่ละส่วน

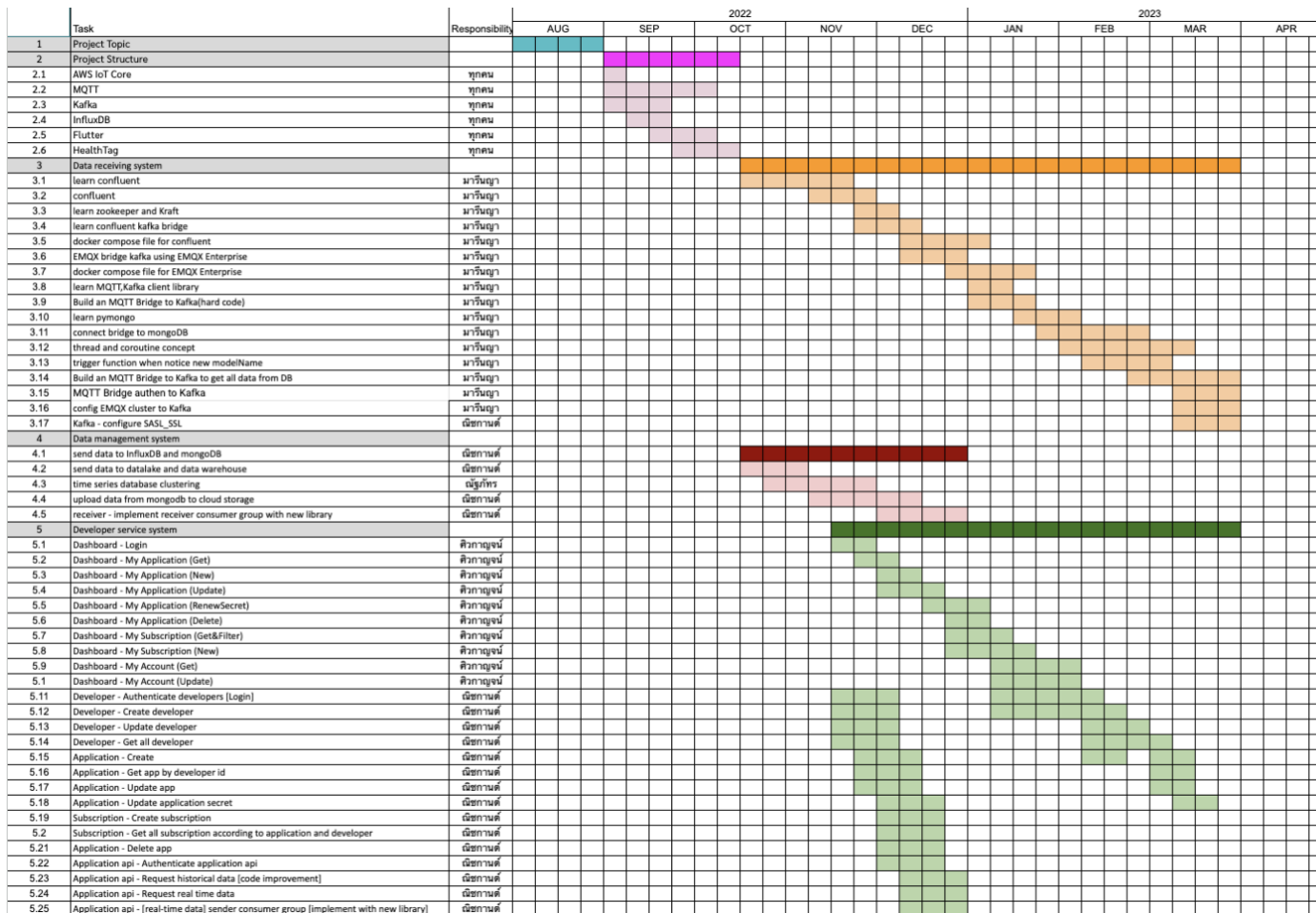
ตารางที่ 1 ผู้รับผิดชอบงานในแต่ละส่วน

ชื่อ - สกุล	งานที่รับผิดชอบ
นายณัฐภัทร จารุชัยสิทธิกุล	<ul style="list-style-type: none"> - การทำ Admin Dashboard - การจัดการผู้ใช้งาน (นักพัฒนา) - การจัดการอุปกรณ์ IoT - การจัดการสิทธิของผู้ใช้งานในการเข้าถึงข้อมูล
นางสาวณิกานต์ ชัยพวงนา	<ul style="list-style-type: none"> - รองรับการส่งข้อมูลของ Kafka server - การบันทึกข้อมูลลงฐานข้อมูล - การบันทึกข้อมูลลง Data Lake - การจัดการผู้ใช้งาน (นักพัฒนา) - การจัดการอุปกรณ์ IoT - การส่งข้อมูล IoT ไปให้ผู้ใช้งาน - การจัดการสิทธิของผู้ใช้งานในการเข้าถึงข้อมูล - การจัดการ Application ของตนเองในระบบ - การจัดการ Subscription ของตนเองในระบบ - การบันทึกผลการอัปโหลดข้อมูลลง Data Lake
นางสาวมารีนญา ตะโจประรัง	<ul style="list-style-type: none"> - รองรับข้อมูลที่ส่งผ่าน MQTT protocol
นางสาวศิวกาญจน์ จิตต์วโรดม	<ul style="list-style-type: none"> - การทำ Data Warehouse - การทำ Developer Dashboard - การจัดการ Application ของตนเองในระบบ

	<ul style="list-style-type: none">- การจัดการ Subscription ของตนเองในระบบ- การบันทึกประวัติการเรียก API- การเก็บบันทึกสถานะของระบบ- การทำ Dashboard สำหรับดู Log
--	---

1.4. แผนการดำเนินงาน

คณะผู้จัดทำได้วางแผนการทำงานตลอดระยะเวลา 9 เดือน คือ ตั้งแต่เดือนสิงหาคม 2565 ถึงเดือนเมษายน 2566 โดยแบ่งออกงานเป็น 10 ส่วนหลัก ๆ ดังรูปที่ 1 และ 2



รูปที่ 1 Gantt chart

	Task	Responsibility	2022				2023												
			AUG	SEP	OCT	NOV	DEC	JAN	FEB	MAR	APR								
6	Admin service system	นักบริหาร																	
6.1	Dashboard - Get Developer's Subscription	นักบริหาร																	
6.2	Dashboard - Update Developer's Subscription	นักบริหาร																	
6.3	Dashboard - Delete Developer's Subscription	นักบริหาร																	
6.4	Dashboard - Get Developer's Account	นักบริหาร																	
6.5	Dashboard - Create Developer's Account	นักบริหาร																	
6.6	Dashboard - Delete Developer's Account	นักบริหาร																	
6.7	Dashboard - Get Device	นักบริหาร																	
6.8	Dashboard - Create Device	นักบริหาร																	
6.9	Dashboard - Delete Device	นักบริหาร																	
6.10	Admin - Authentication [Login]	นักพัฒนา																	
6.11	Developer - Create developers	นักพัฒนา																	
6.12	Developer - Get all developers in the system	นักพัฒนา																	
6.13	Developer - Delete developers	นักพัฒนา																	
6.14	Medical model - Get all model in the systems [code improvement]	นักพัฒนา																	
6.15	Medical model - Get all model and device id list	นักพัฒนา																	
6.16	Medical model - Create medical model [code improvement]	นักพัฒนา																	
6.17	Medical model - Update medical model	นักพัฒนา																	
6.18	Medical model - Delete medical model	นักพัฒนา																	
6.19	Subscription - Update subscription status	นักพัฒนา																	
7	Log system	นักพัฒนา																	
7.1	Logging gcs upload status	นักพัฒนา																	
7.2	Logging all request	วิศวกร																	
7.3	Uptime monitoring	วิศวกร																	
7.4	CPU/Mem monitoring	วิศวกร																	
7.5	Kibana	วิศวกร																	
7.6	Improve log when file was uploaded on google cloud storage	นักพัฒนา																	
8	Deployment	นักพัฒนา																	
8.1	learned how to write nginx config file	นักพัฒนา																	
8.2	Write nginx config file and create docker image for application	นักพัฒนา																	
8.3	Write docker compose file for web service cluster	นักพัฒนา																	
8.4	Deploy on AWS [3 instance 1 for nginx server and others for web service]	นักพัฒนา																	
8.5	Create docker file for web service [need to have java runtime]	นักพัฒนา																	
8.6	Configure Kafka with SASL_SSL with 3 zookeeper and 3 broker using 3 instance	นักพัฒนา																	
9	Testing	นักพัฒนา																	
9.1	List data receiving test case	มาร์ตินดา																	
9.2	List admin dashboard test case	นักบริหาร																	
9.3	List developer dashboard test case	วิศวกร																	
9.4	Fundamental testing - Data receiving system	ทุกคน																	
9.5	Fundamental testing - Developer service system [not include application api]	ทุกคน																	
9.6	Fundamental testing - Application api	ทุกคน																	
9.7	Fundamental testing - Admin service system	ทุกคน																	
9.8	Web service high availability testing	ทุกคน																	

1.5. ประโยชน์ที่คาดว่าจะได้รับ

1.5.1. บุคลากรทางการแพทย์

1.5.1.1. ได้รับระบบที่มีความสะดวกในการตรวจสอบข้อมูลของคนไข้ สามารถดูข้อมูลของคนไข้ได้แบบ real-time จากที่ไหนก็ได้

1.5.1.2. ได้รับข้อมูลสุขภาพประกอบการวินิจฉัยโรค เนื่องจากการเก็บข้อมูลสุขภาพของผู้ป่วยย้อนหลัง

1.5.2. ผู้ป่วย

1.5.2.1. ได้รับความสะดวกในการไปพบแพทย์ เนื่องจากการเก็บข้อมูลสุขภาพย้อนหลัง

1.5.2.2. ได้รับความปลอดภัยในชีวิต เนื่องจากมีอุปกรณ์วัดข้อมูลสุขภาพตลอดเวลา เมื่อมีเหตุฉุกเฉินจะมีระบบแจ้งเตือนทำให้แพทย์สามารถรักษาได้อย่างทันท่วงที

1.5.3. นักวิจัย

1.5.3.1. ได้รับประโยชน์จากการเก็บข้อมูลใน Data Lake สามารถนำเอาข้อมูลไปใช้ในการศึกษาวิจัยต่อได้ในอนาคต

1.5.3.2. ได้รับประโยชน์จากการเก็บรวบรวมข้อมูลที่เป็นระเบียบมากขึ้น ทำให้สะดวกในการนำเอาข้อมูลไปใช้

1.5.3.3. ได้รับประโยชน์ในการทำ Data Dashboard ซึ่งสามารถช่วยให้เข้าใจข้อมูลได้ดียิ่งขึ้น

2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1. ทฤษฎีที่เกี่ยวข้อง

2.1.1. อุปกรณ์ทางการแพทย์

อุปกรณ์ที่ใช้ในโรงพยาบาลที่สามารถส่งต่อข้อมูลได้ [4]

2.1.2. Structured Data

Structured data ข้อมูลที่มีโครงสร้างชัดเจน หรือมีจำนวนและชื่อ attribute ที่แน่นอนซึ่งเป็นข้อมูลที่นักวิทยาศาสตร์สามารถข้อมูลสามารถนำไปวิเคราะห์ใช้งานต่อได้ง่าย ตัวอย่างเช่น ไฟล์ csv หรือไฟล์ excel [5]

2.1.3. Unstructured Data

Unstructured data ข้อมูลที่ไม่มีโครงสร้างหรือไม่สามารถระบุโครงสร้างที่ชัดเจนได้ เช่น ไฟล์เสียง รูปภาพ วิดีโอ [6]

2.1.4. Messaging Systems

เป็นระบบที่มีหน้าที่ส่งข้อมูลจากแอปพลิเคชันหนึ่งไปยังแอปพลิเคชันหนึ่งโดย แอปพลิเคชันที่ต้องการส่งข้อมูลไม่จำเป็นต้องรอการตอบกลับ ทำให้แอปพลิเคชันสามารถส่งข้อมูลได้ต่อเนื่องโดยไม่ต้องหยุดการทำงาน [7]

2.1.5. Kafka

Kafka คือ Messaging systems แบบหนึ่งในลักษณะ Publish/Subscribe ที่รองรับการ scale ในลักษณะ Horizontal Scale และด้วยคุณสมบัติของ Partition ของ Kafka ที่รองรับ MultiWrite ทำให้รองรับ events จำนวนมากได้ จึงเหมาะกับงานที่เป็น event streaming [8]

2.1.6. MQTT protocol

โพรโตคอลในการส่งข้อมูลระหว่างอุปกรณ์ที่พัฒนามาเพื่อใช้ในระบบ IoT มีทำงานแบบ Broker and Clients Network ซึ่งถูกออกแบบมาเพื่อใช้สื่อสารในระบบเครือข่ายที่มีทรัพยากรค่อนข้างจำกัด ใช้งาน Bandwidth ต่ำและสามารถ publish-subscribe ข้อมูลระหว่าง Device เพื่อสื่อสารกันระหว่างอุปกรณ์ [9]

2.1.7. Kafka Connector

ทำหน้าที่แปลงข้อมูลจาก source systems ไปยัง target systems โดยที่ไม่ต้องสร้าง consumer เพื่อมา subscribe MQTT broker เอง [10]

2.1.8. Web Service

ส่วนหนึ่งของซอฟต์แวร์ที่ให้บริการผ่านอินเทอร์เน็ตโดยมีการแลกเปลี่ยนข้อมูลด้วยวิธีที่เป็นมาตรฐานในรูปแบบที่เป็นมาตรฐาน [11]

2.1.9. REST API

REST ย่อมาจาก Representational State Transfer เป็นข้อกำหนดการส่งข้อมูลระหว่าง Server-Client รูปแบบหนึ่งซึ่งอยู่บนพื้นฐานของ HTTP Protocol เป็นการสร้าง Web Service เพื่อแลกเปลี่ยนข้อมูลกันผ่านแอปพลิเคชัน วิธีหนึ่ง ซึ่งส่งข้อมูลได้หลายชนิด ไม่ว่าจะเป็น Text, XML, JSON REST API คือ interface สำหรับติดต่อสื่อสารของแอปพลิเคชันโดยใช้ REST [12] [13]

2.1.10. Data Lake

Data Lake คือที่เก็บข้อมูลส่วนกลางซึ่งสามารถเก็บข้อมูลทุกรูปแบบ คือ structured data semi-structured data และ unstructured data ซึ่งนิยมใช้เก็บข้อมูลดิบเนื่องจากมีความยืดหยุ่นในการเก็บข้อมูล [14]

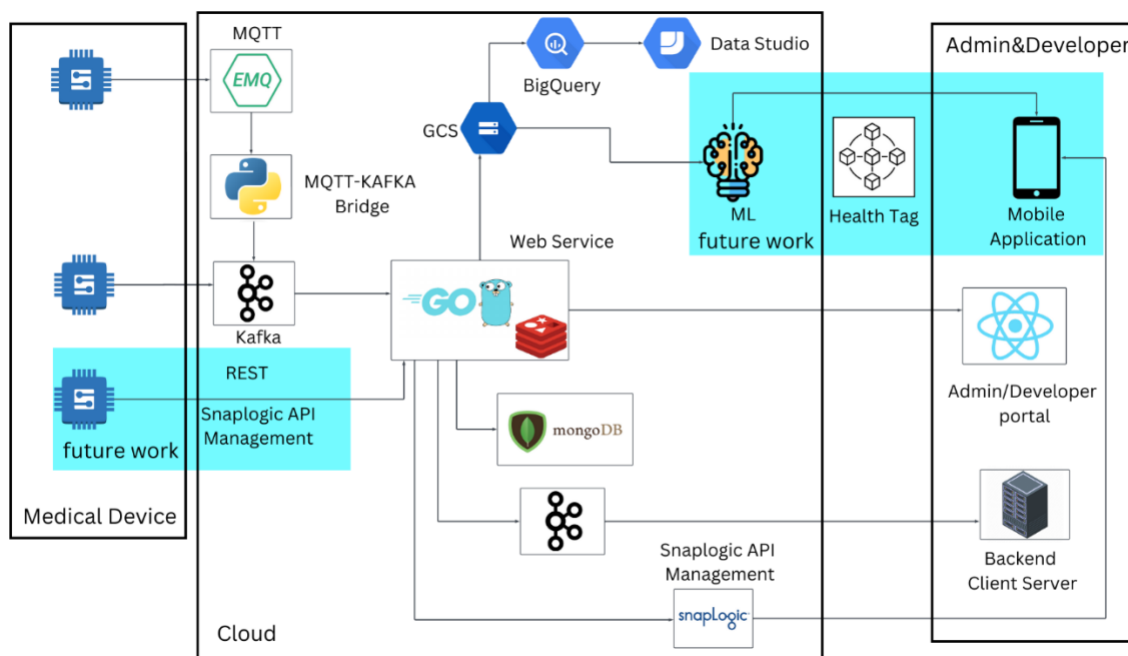
2.1.11. Data Warehouse

Data Warehouse คือ รูปแบบของระบบจัดการข้อมูลที่สามารถเก็บข้อมูลแบบ structured data ได้ดี และถูกออกแบบมาเพื่อสนับสนุนการสร้าง Business Intelligence (BI) และการวิเคราะห์ข้อมูลโดยเฉพาะ นอกจากนั้นยังมีความสามารถในการสืบค้นข้อมูล (Query) จำนวนมาก [15]

3. วิธีการนำเสนอ/แนวทางที่คิดว่าจะทำ

3.1. สถาปัตยกรรมของระบบ

คณะผู้จัดทำได้ทำการศึกษาและออกแบบสถาปัตยกรรมของระบบซึ่งประกอบไปด้วย ส่วนที่เป็นอุปกรณ์ทางการแพทย์ ส่วนระบบหลังบ้านและฐานข้อมูลซึ่งอยู่บน cloud และระบบติดต่อกับผู้ใช้งานและแอดมิน โดยนำเสนอสถาปัตยกรรมได้ดังรูปที่ 3



รูปที่ 3 สถาปัตยกรรมของระบบ

3.2. ระบบภายใน

ระบบภายในแบ่งออกเป็น 5 ส่วน

3.2.1. ระบบการส่งข้อมูลของอุปกรณ์การแพทย์

การส่งข้อมูลของอุปกรณ์ทางการแพทย์จะแบ่งออกเป็น 5 รูปแบบ

- อุปกรณ์ที่ไม่เป็น Time Series ซึ่งจะส่งข้อมูลโดยใช้ Bluetooth ส่งต่อให้สมาร์ทโฟน และ สมาร์ทโฟนจะส่งข้อมูลให้กับเซิร์ฟเวอร์โดยใช้ REST API
- อุปกรณ์ที่เชื่อมต่อผ่าน Mobile Application และใช้ MQTT protocol ในการส่งข้อมูลให้กับเซิร์ฟเวอร์
- อุปกรณ์ที่เชื่อมต่อผ่าน Arduino หรือ Raspberry Pi และใช้ MQTT protocol ในการส่งข้อมูลให้กับเซิร์ฟเวอร์

- อุปกรณ์ที่เชื่อมต่ออินเทอร์เน็ตและส่งข้อมูลโดยใช้ MQTT protocol ให้กับเซิร์ฟเวอร์โดยตรง
- อุปกรณ์ที่เชื่อมต่อกับระบบอื่นอยู่แล้ว จะมีการส่งข้อมูลในรูปแบบของ MQTT protocol, Kafka และ REST API

เพราะฉะนั้นจึงสรุปการส่งข้อมูลได้เป็น 3 ส่วนดังรูปที่ 4 ได้แก่

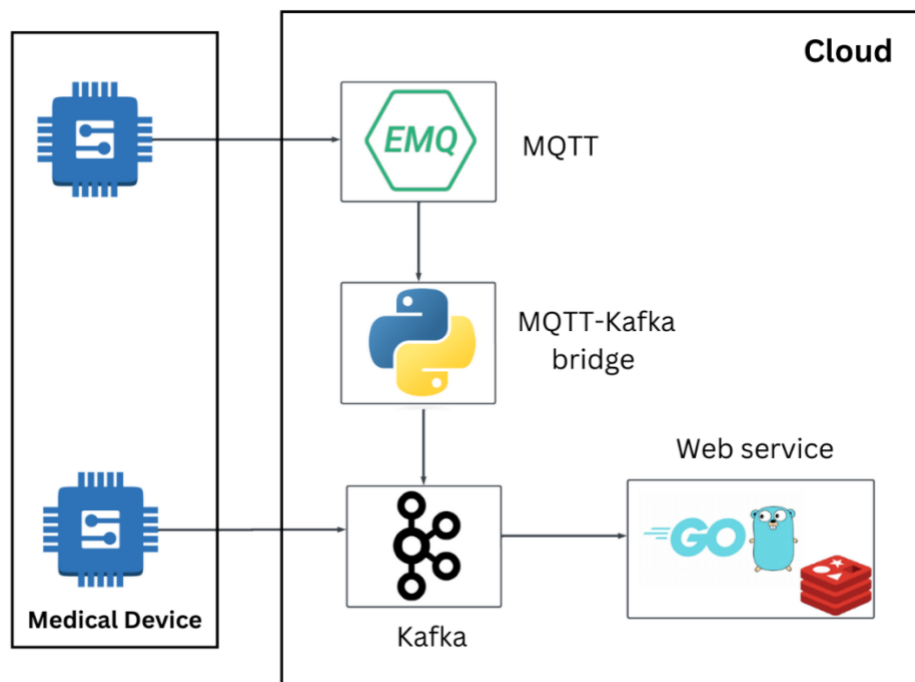
3.2.1.1. การส่งข้อมูลผ่าน REST API (อยู่นอกขอบเขตของโครงการ)

3.2.1.2. การส่งข้อมูลผ่าน Kafka server

จะมีการออกแบบ topic โดย 1 โมเดลอุปกรณ์การแพทย์จะส่งไปยัง topic เดียวกันกล่าวคือ อุปกรณ์ชนิดเดียวกันจะมีการส่งข้อมูลไปยัง topic เดียวกัน และส่งข้อมูลให้กับ web service เพื่อประมวลผลในลำดับถัดไป

3.2.1.3. การส่งข้อมูลผ่าน MQTT protocol

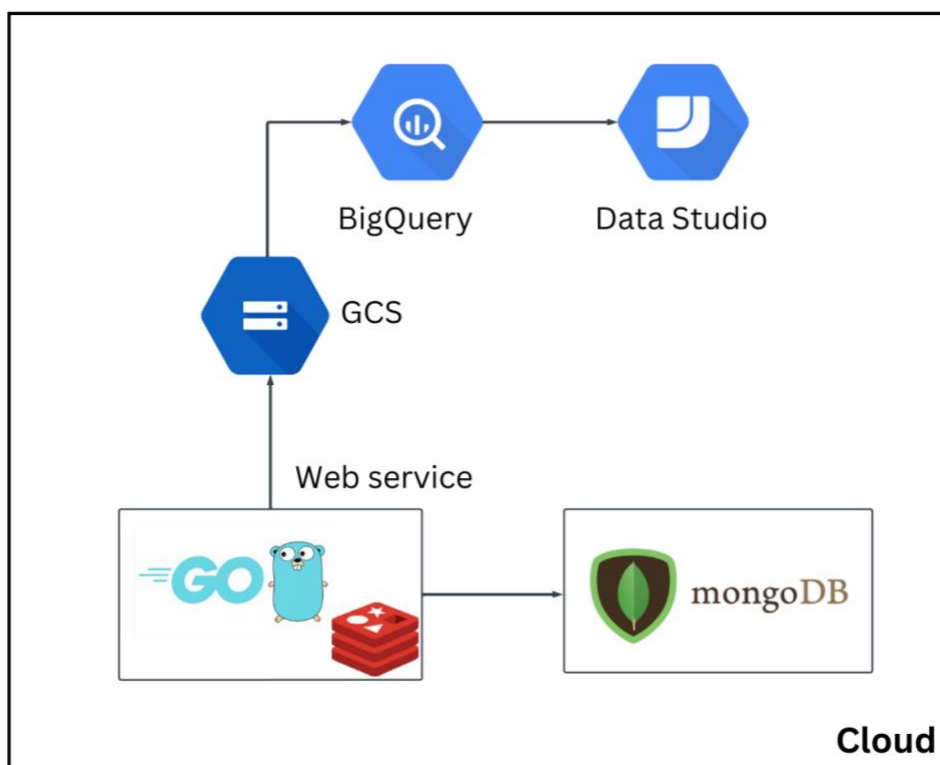
มีการทำ MQTT-Kafka Bridge โดยใช้ภาษา Python ในการพัฒนา เพื่อนำข้อมูลจาก EMQX ซึ่งเป็น MQTT Messaging Platform เชื่อมต่อไปยัง Kafka server และส่งประมวลผลที่ web service เดียวกัน



รูปที่ 4 การส่งข้อมูลของอุปกรณ์ IoT

3.2.2.ระบบการจัดการข้อมูล

ระบบนี้จะเป็นการรับข้อมูลจาก Kafka server เพื่อเก็บข้อมูลจากอุปกรณ์การแพทย์ลงสู่ mongoDB database และมีการส่งข้อมูลขึ้น Google Cloud Storage (Data Lake) ทุกๆ 15 นาที เพื่อจัดเก็บข้อมูลและทำ Data Pipeline ในลำดับถัดไปโดยมีการทำงานดังรูปที่ 5



รูปที่ 5 ระบบการจัดการข้อมูล

3.2.2.1. การออกแบบการจัดเก็บข้อมูลลง MongoDB database

ระบบของเราใช้ MongoDB database ในการเก็บข้อมูลจากอุปกรณ์การแพทย์ โดยมีการออกแบบการเก็บข้อมูลดังนี้คือ ใช้ 1 collection ในการเก็บข้อมูลของอุปกรณ์ โดยจะใช้ค่า deviceId และชื่ออุปกรณ์ในการแบ่งแยกข้อมูล

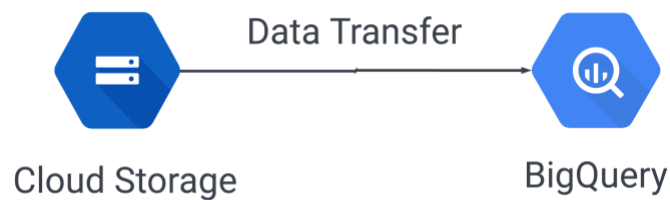
3.2.2.2. การอัปโหลดข้อมูลขึ้น Google Cloud Storage (GCS)

ระบบของเรามีการใช้ Job Scheduler ที่จะคอยทำงานทุกๆ 15 นาที เพื่อดึงข้อมูลจาก MongoDB และสร้าง csv file format เพื่ออัปโหลดข้อมูลเข้าสู่ GCS ซึ่งเป็น Data Lake ในส่วนของการออกแบบการจัดเก็บข้อมูลใน GCS มีการออกแบบดังนี้คือ model/deviceId/year/month/day/timestamp.csv และมีการเก็บผลลัพธ์การทำงานเข้า

Elasticsearch เพื่อสามารถติดตามได้ว่าการอัปเดตได้สำเร็จหรือไม่และใช้ค่า end-time ของการอัปเดตก่อนหน้านี้เป็นค่าอ้างอิงในการค้นหาข้อมูลเพื่ออัปเดตในครั้งถัดไป

3.2.2.3. การทำ Data Warehouse

ระบบจะดึงข้อมูลจาก Google Cloud Storage (GCS) ไปบันทึกลงในตารางของ Google Cloud BigQuery โดยใช้ Data Transfer ซึ่งเป็นบริการภายในของ BigQuery โดยกำหนดให้ทำงานทุก ๆ 15 นาที ตามรูปที่ 6



รูปที่ 6 การส่งข้อมูลจาก GCS ไป BigQuery

3.2.3.ระบบการให้บริการนักพัฒนา

ระบบการให้บริการนักพัฒนามีลักษณะการทำงานดังรูปที่ 7, 8 และ 9 โดยประกอบด้วย 3 ส่วนที่สำคัญ ดังนี้

3.2.3.1. การจัดการ Application ของตนเองในระบบ

ระบบมี Dashboard สำหรับนักพัฒนาในการสร้าง application ของตนเองได้ โดยระบุ application name และ description ซึ่งหลังจากสร้าง application เสร็จจะได้รับ ClientSecret เพื่อนำไปใช้ในการทำ authentication และ authorization ในการขอข้อมูลต่าง ๆ โดย ClientSecret จะมีวันหมดอายุ เมื่อ ClientSecret หมดอายุ นักพัฒนาสามารถกด renew เพื่อขอ ClientSecret ใหม่ได้ นอกจากนี้ นักพัฒนายังสามารถลบ application ของตนเองที่ไม่ต้องการ หรือแก้ไข application name และ description ใน application ของตนเองได้ด้วย

3.2.3.2. การจัดการ Subscription ของตนเองในระบบ

ระบบมี Dashboard สำหรับนักพัฒนาในการสร้าง subscription ของแต่ละ application ของตนเองได้เพื่อขอข้อมูลของอุปกรณ์การแพทย์ โดยระบุค่าต่าง ๆ ดังนี้

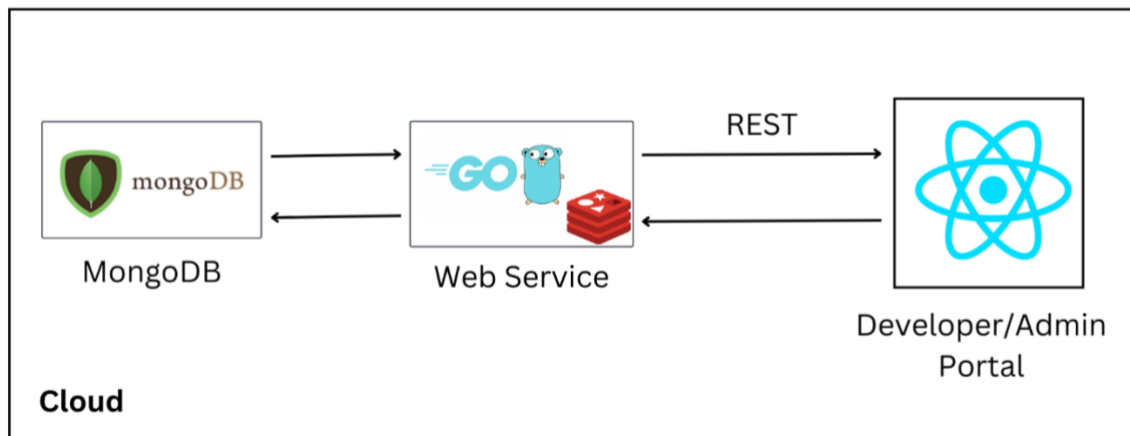
- application name: application ของ subscription นั้น ๆ
- model name: ชื่ออุปกรณ์
- device id: รหัสอุปกรณ์ที่ต้องการข้อมูล

- start-time และ end-time: ขอบเขตระยะเวลาของข้อมูลที่ต้องการ
- mode: มี 2 mode คือ REST (กรณีต้องการข้อมูลย้อนหลัง) และ Kafka (กรณีต้องการข้อมูลแบบ real-time)
- note: ระบุหรือไม่ก็ได้

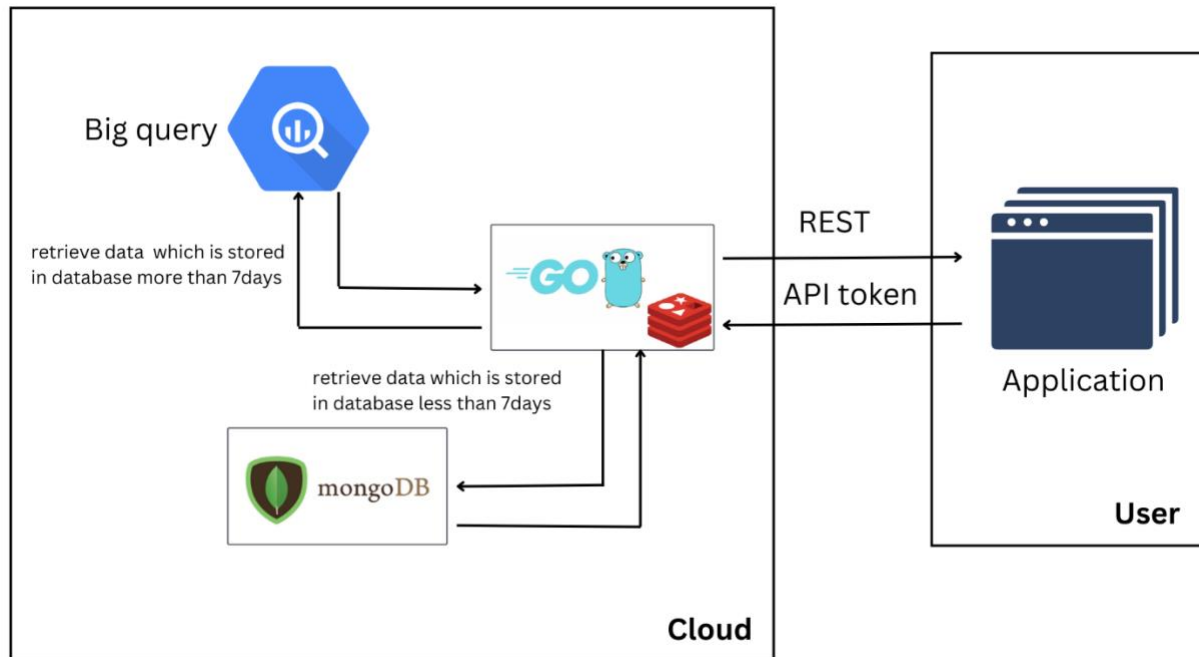
3.2.3.3. การส่งข้อมูลอุปกรณ์ไปให้ใช้งานทั้งแบบ real-time และ historical

ระบบของเราจะมีการใช้สิทธิ์ในการเข้าถึงข้อมูลของผู้ใช้ก่อนจะมีการส่งข้อมูล และการส่งข้อมูลของระบบแบ่งออกเป็น 2 ประเภท คือ

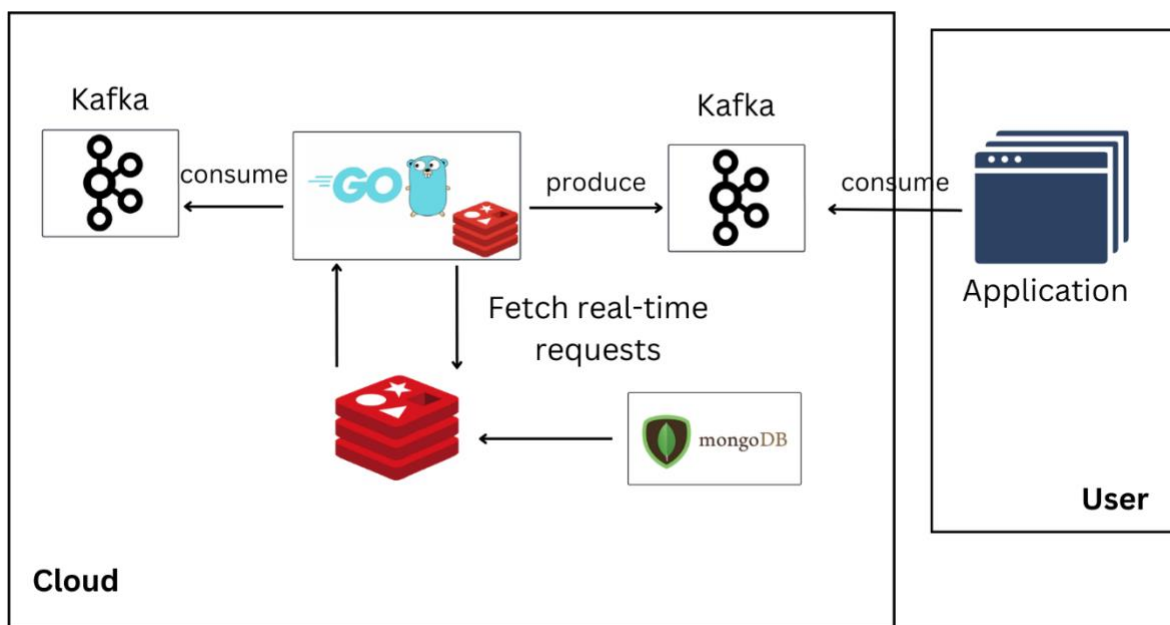
- Historical Data ซึ่งเป็นข้อมูลย้อนหลังของอุปกรณ์ต่างๆ ซึ่งทางระบบจะมี API ในการส่งข้อมูลย้อนหลังให้กับผู้ใช้งาน โดยถ้าผู้ใช้งานขอข้อมูลย้อนหลังน้อยกว่า 7 วัน จะดึงข้อมูลมาจาก MongoDB แต่ถ้าผู้ใช้งานขอข้อมูลย้อนหลังมากกว่า 7 วัน จะดึงข้อมูลมาจาก BigQuery
- Real-Time Data ระบบจะมี API ในการให้ผู้ใช้งานสามารถขอข้อมูลได้ หลังจากนั้นระบบจะมีการบันทึกข้อมูลการร้องขอลงสู่ MongoDB และระบบจะค่อยๆ ดึงข้อมูลร้องขอจาก MongoDB และสร้าง Kafka consumer เพื่ออ่านข้อมูลตามเงื่อนไขของผู้ใช้งาน และส่งข้อมูล ผ่าน Kafka server โดยมีการออกแบบ topic เป็นดังนี้คือ 1 topic ต่อ 1 ผู้ใช้งาน



รูปที่ 7 ระบบการให้บริการนักพัฒนา



รูปที่ 8 การขอข้อมูลย้อนหลัง



รูปที่ 9 การขอข้อมูลแบบ real-time

3.2.4.ระบบแอดมิน

3.2.4.1. การจัดการผู้ใช้งาน (นักพัฒนา)

ระบบมี Dashboard สำหรับแอดมินในการสร้าง username และ password ให้กับนักพัฒนา เพื่อให้ นักพัฒนาสามารถใช้งานระบบการให้บริการนักพัฒนาได้

3.2.4.2. การจัดการอุปกรณ์ IoT

ระบบมี Dashboard สำหรับแอดมินในการลงทะเบียนอุปกรณ์ IoT ซึ่งจะเป็นข้อมูลที่รายละเอียดของอุปกรณ์ว่าสามารถวัดค่าอะไรได้บ้าง และประเภทข้อมูลของค่านั้นๆ ซึ่งระบบจะมีการบันทึกค่าเหล่านั้นใน MongoDB เพื่อใช้ในการอ้างอิงในการทำงานของระบบและจะมีการสร้าง topic ใหม่สำหรับโมเดลอุปกรณ์การแพทย์ใหม่บน kafka cluster

3.2.4.3. การจัดการสิทธิของผู้ใช้งานในการเข้าถึงข้อมูล

ระบบมี Dashboard สำหรับแอดมินในการสร้าง แก้ไข และ ลบ subscription ต่าง ๆ ในระบบ

3.2.5.ระบบการจัดการ Log File

3.2.5.1. การบันทึกผลการอัปโหลดข้อมูลลง Data Lake

ระบบมีการเก็บบันทึกผลการอัปโหลดข้อมูลลงใน GCS โดยจะบันทึก log ใน elasticsearch

3.2.5.2. การบันทึกประวัติการเรียก API

ระบบจะมีการเก็บข้อมูล traffic ต่าง ๆ ที่เข้ามาในระบบ รวมถึง API response status code

3.2.5.3. การเก็บบันทึกสถานะของระบบ

ระบบมีการเก็บ uptime ของระบบ เพื่อเก็บสถิติการทำงานของ service นอกจากนี้ยังเก็บ memory/cpu usage ของระบบ เพื่อตรวจสอบติดตามการใช้ทรัพยากรในเครื่อง

3.2.5.4. การทำ Dashboard สำหรับดู Log

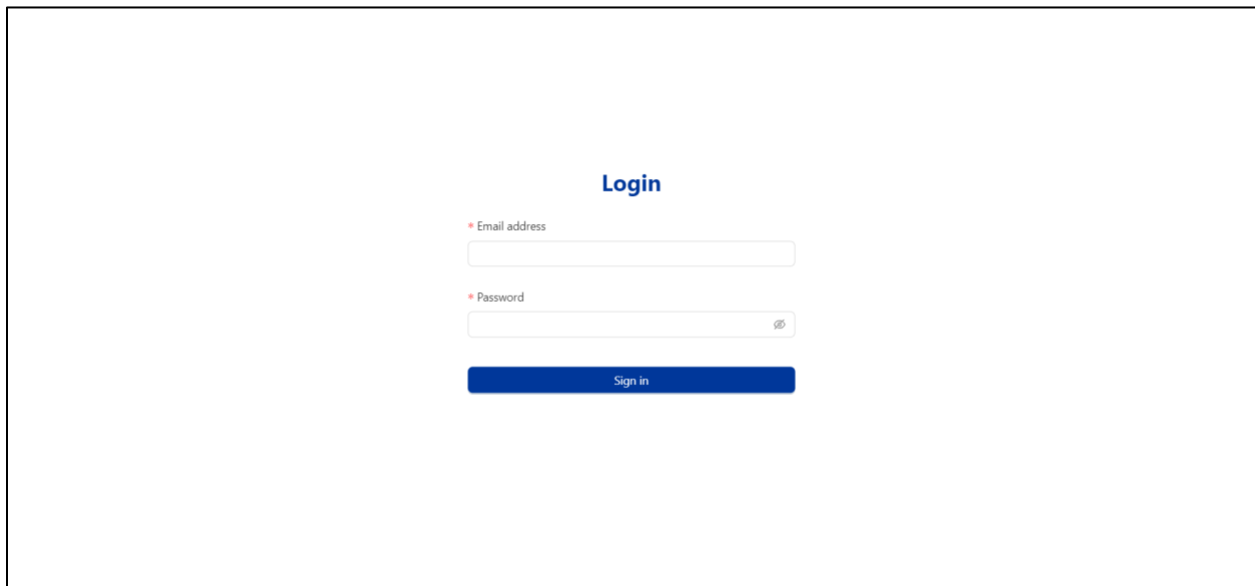
ระบบมี Dashboard สำหรับดูค่า upload log, api log, uptime, และ memory/cpu usage

4. แผนการทำงาน

4.1. Developer Dashboard

4.1.1. Log in

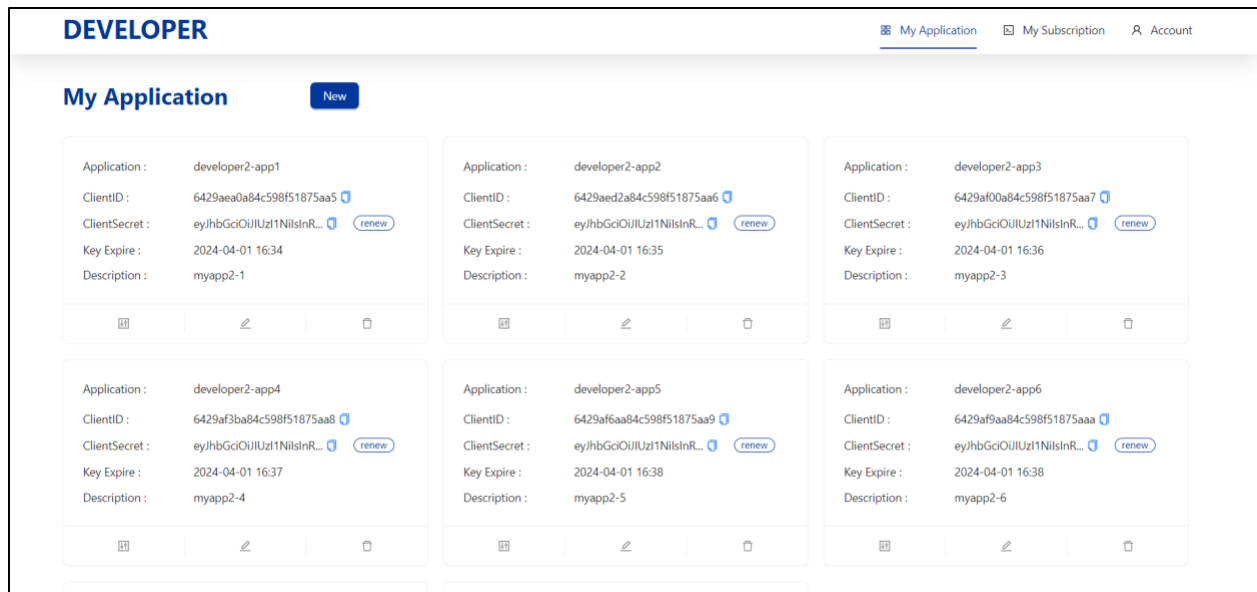
หน้า Login ให้ใส่อีเมล (E-mail) และรหัสผ่าน (Password) ตามรูปที่ 10 โดยหากไม่ได้เข้าสู่ระบบจะไม่สามารถเข้าถึงหน้าอื่น ๆ ในเว็บไซต์ได้

The image shows a login form titled "Login" in blue text. Below the title, there are two input fields. The first field is labeled "Email address" with a red asterisk and has a light gray border. The second field is labeled "Password" with a red asterisk and has a light gray border and a small eye icon on the right side. Below these fields is a blue button with the text "Sign in" in white.

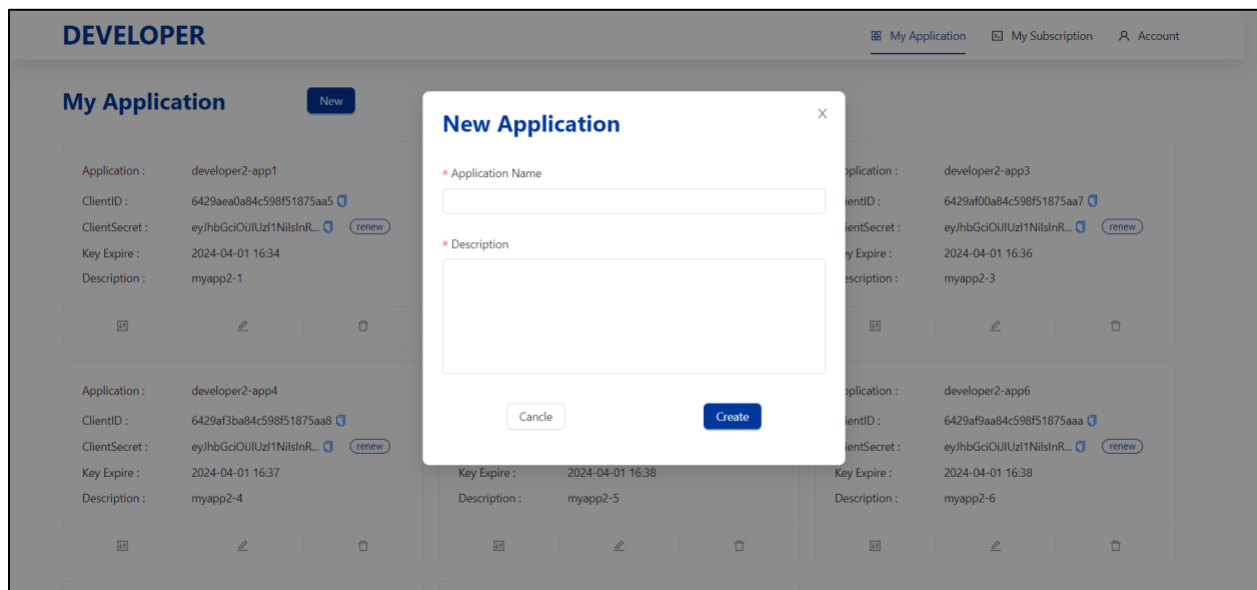
รูปที่ 10 Login Page

4.1.2. My Application

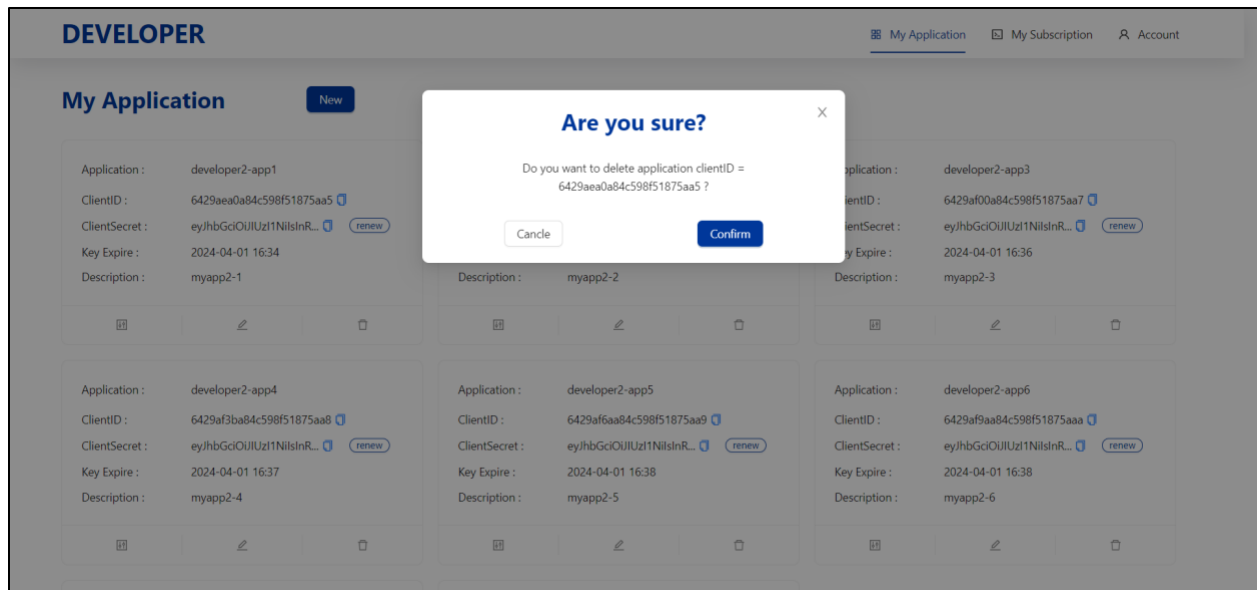
หน้า My Application สำหรับให้นักพัฒนาดู application ทั้งหมดของตนเองดังรูปที่ 11 สามารถสร้าง application ใหม่ได้โดยกดปุ่ม New จากนั้นกรอก application name และ description ในป๊อปอัพตามรูปที่ 12 นอกจากนี้ยังสามารถขอ clientSecret ใหม่ได้โดยการกดปุ่ม renew อีกทั้งยังสามารถลบและแก้ไขข้อมูลของ application ของตนเองได้ตามรูปที่ 13 และ 14 ตามลำดับ



รูปที่ 11 การดู application ทั้งหมดของตนเอง



รูปที่ 12 การสร้าง application ใหม่



รูปที่ 13 การลบ application

Application :

developer2-app1

ClientID :

6429aea0a84c598f51875aa5

ClientSecret :

eyJhbGciOiJIUzI1NiIsInR...

renew

Key Expire :

2024-04-01 16:34

Description :

myapp2-1

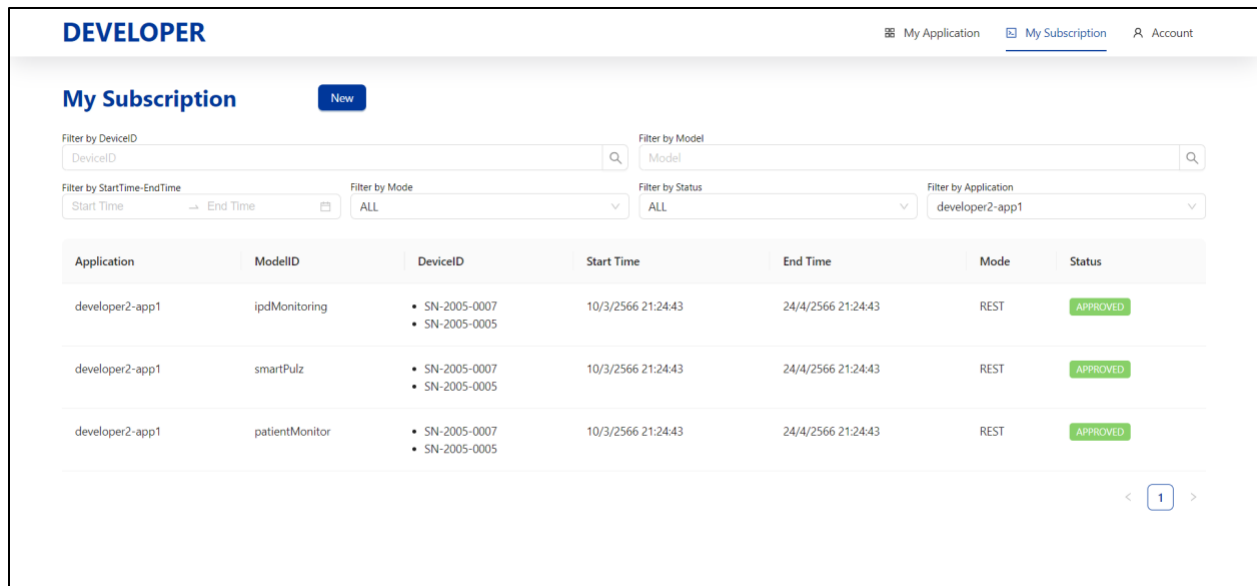
Cance

Save

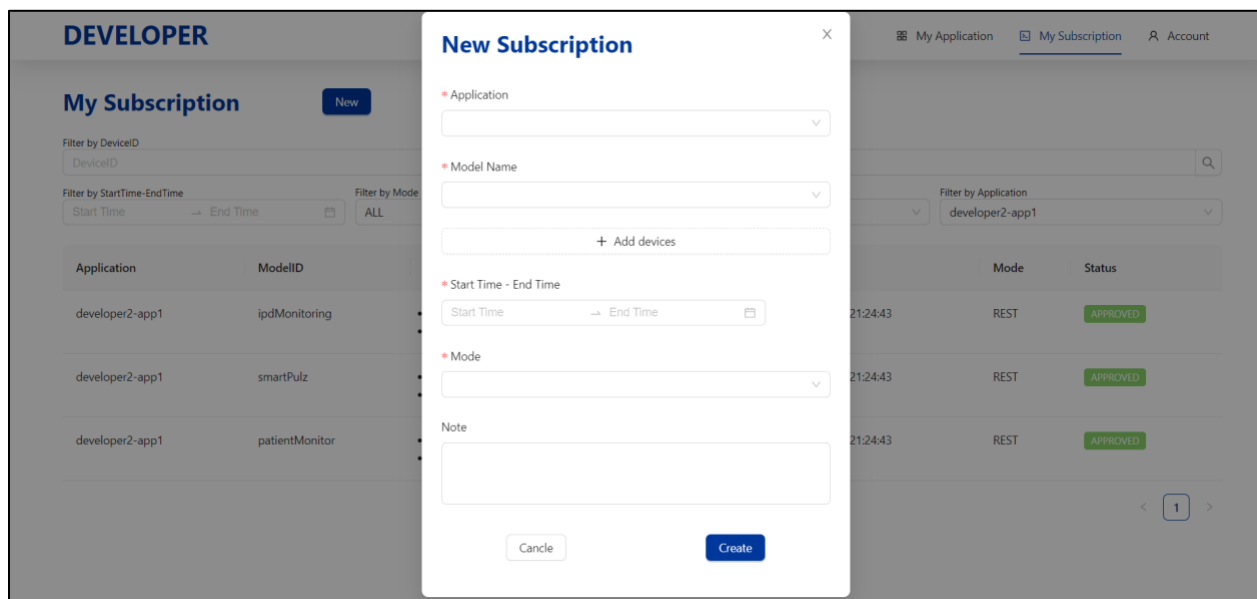
รูปที่ 14 การแก้ไข application

4.1.3. My Subscription

หน้า My Subscription สำหรับให้นักพัฒนาดู subscription ของตนเอง โดยสามารถฟิลเตอร์ subscription ได้ตาม application, device Id, model name, start time - end time, mode และ status ตามรูปที่ 15 และสามารถกดสร้าง subscription ใหม่ได้โดยการกดปุ่ม New จากนั้นระบุค่าใน field ต่าง ๆ ในป๊อปอัพ ตามรูปที่ 16 (note เป็น optional)



รูปที่ 15 การดูและฟิเตอร์ subscription ทั้งหมด



รูปที่ 16 การสร้าง subscription ใหม่

4.1.4.My Account

นักพัฒนาสามารถดูข้อมูลบัญชีได้ที่หน้า My Account และสามารถแก้ไขรหัสผ่านได้โดยการกรอกรหัสผ่านเดิมและรหัสผ่านใหม่ 2 ครั้ง จากนั้นกดปุ่ม Change Password ดังรูปที่ 17

DEVELOPER

My Application My Subscription Account

My Account

Email:
developer2@gmail.com

Name:
developer1

Associated name:
hospital1

* Old Password:

* New Password: * Re-New Password:

[Change Password](#)

รูปที่ 17 หน้า My Account

4.2. Admin Dashboard

4.2.1.Login

หน้า Login ให้ใส่อีเมล (E-mail) และรหัสผ่าน (Password) ตามรูปที่ 18 โดยหากไม่ได้เข้าสู่ระบบจะไม่สามารถเข้าถึงหน้าอื่น ๆ ในเว็บไซต์ได้

Login

* Email address

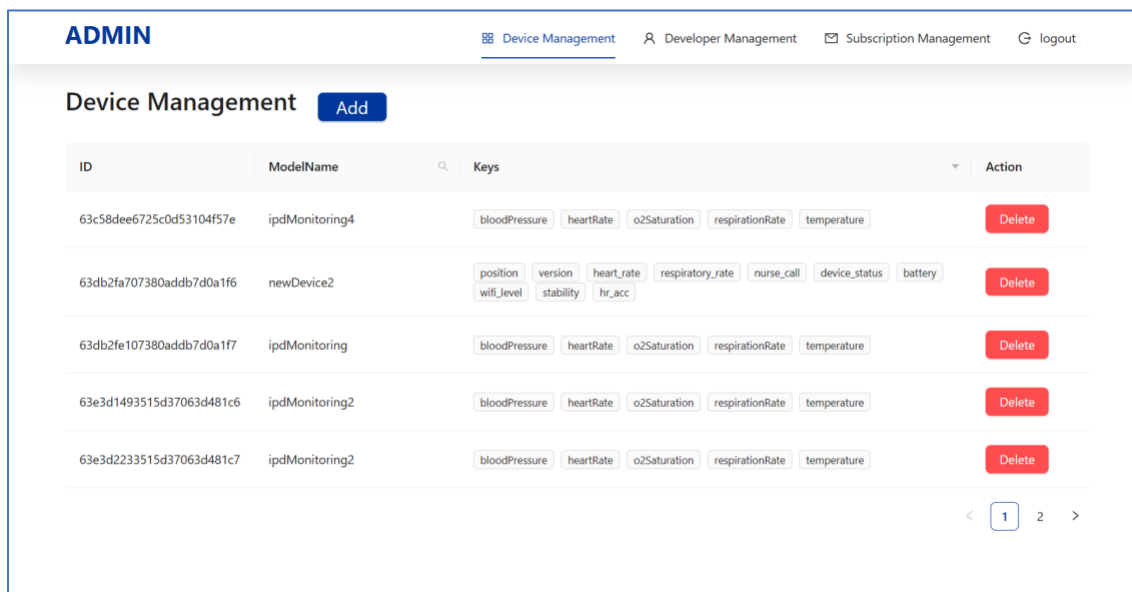
* Password

[Sign in](#)

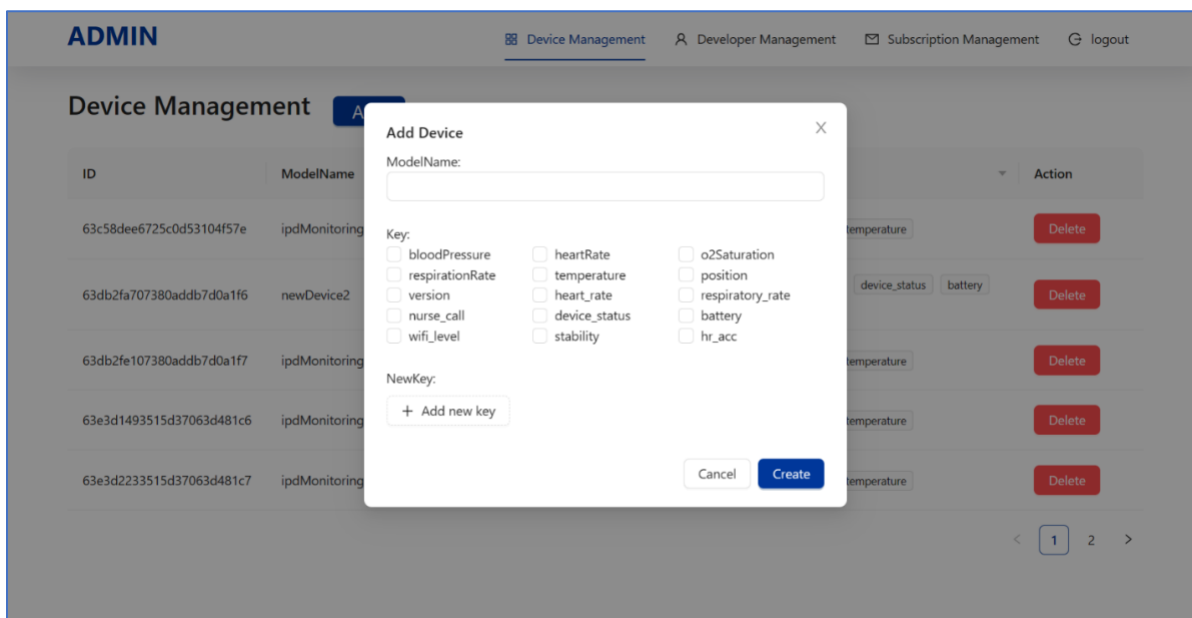
รูปที่ 18 หน้า Login

4.2.2.Device Management

หน้าเว็บสำหรับจัดการโมเดลอุปกรณ์ที่ส่งข้อมูลมายัง web service ดังรูปที่ 19 โดยสามารถเพิ่มโมเดลอุปกรณ์โดยการกดปุ่ม Add โดยจะมีป๊อปอัพสำหรับการเพิ่มโมเดลอุปกรณ์ขึ้นมาตามรูปที่ 20 สามารถที่จะกรอกข้อมูลโดย NewKeys ไม่จำเป็นต้องกรอกถ้าไม่ทำการกดเพิ่มด้วยปุ่ม Add new key



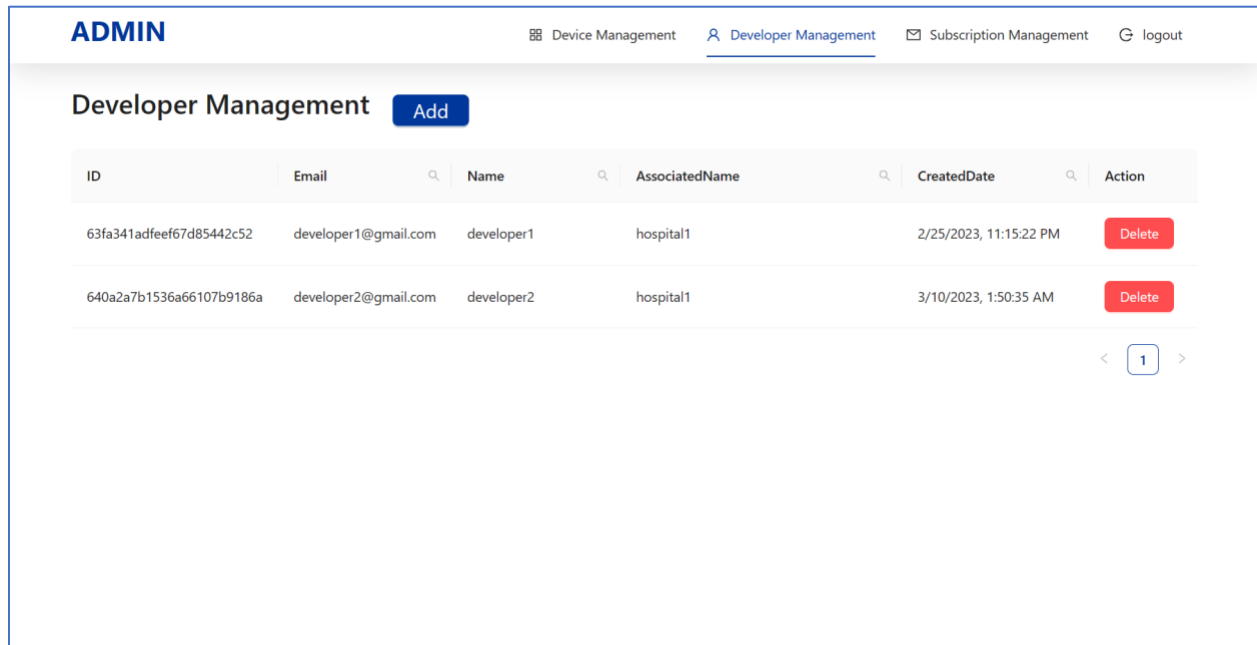
รูปที่ 19 หน้าเว็บสำหรับการจัดการโมเดลอุปกรณ์



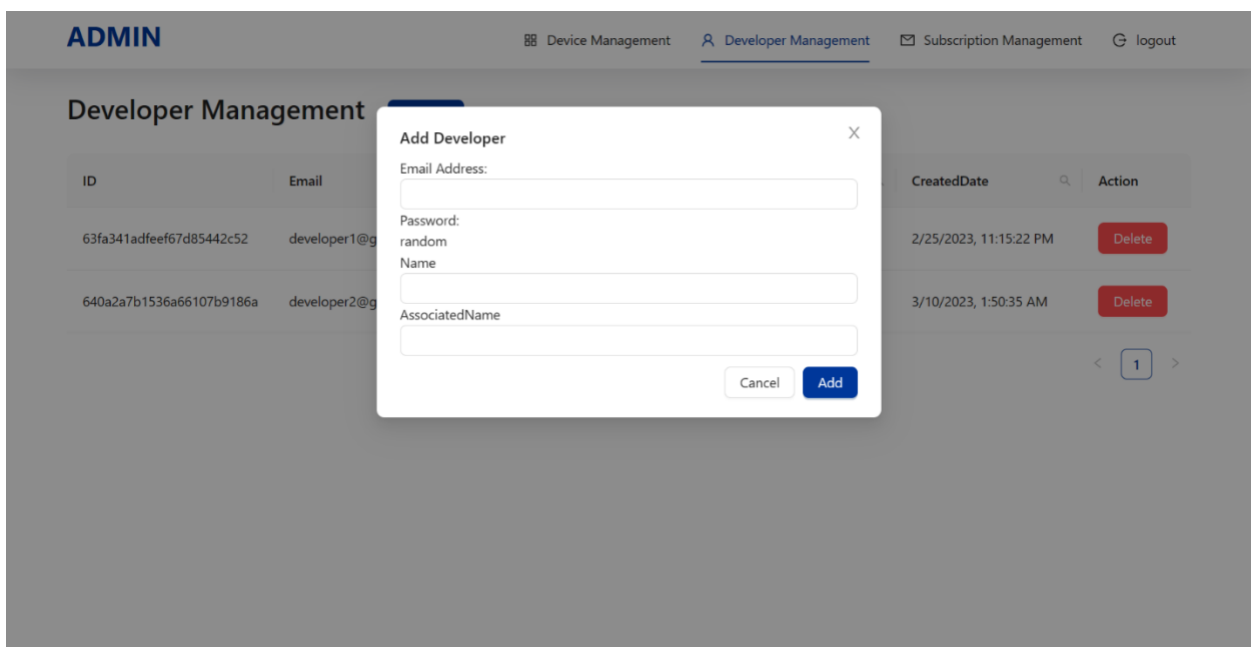
รูปที่ 20 หน้าเว็บสำหรับการเพิ่มโมเดลอุปกรณ์

4.2.3.Developer Management

หน้าเว็บสำหรับการจัดการบัญชีนักพัฒนาที่จะมาขอข้อมูลไปใช้งานดังรูปที่ 21 โดยสามารถที่จะเพิ่มบัญชีนักพัฒนาได้โดยการกดปุ่ม Add โดยจะมีป๊อปอัพสำหรับการเพิ่มบัญชีนักพัฒนาขึ้นมาตามรูปที่ 22 กรอกข้อมูลเพื่อเพิ่มบัญชีนักพัฒนา



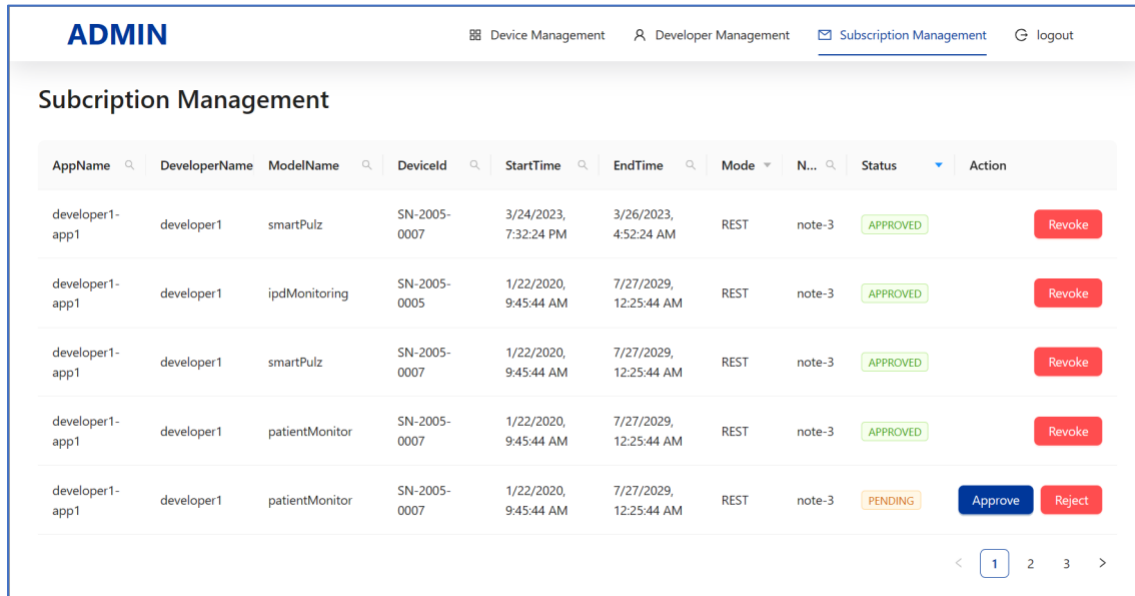
รูปที่ 21 หน้าเว็บสำหรับการจัดการบัญชีนักพัฒนา



รูปที่ 22 สำหรับการเพิ่มบัญชีนักพัฒนา

4.2.4.Subscription Management

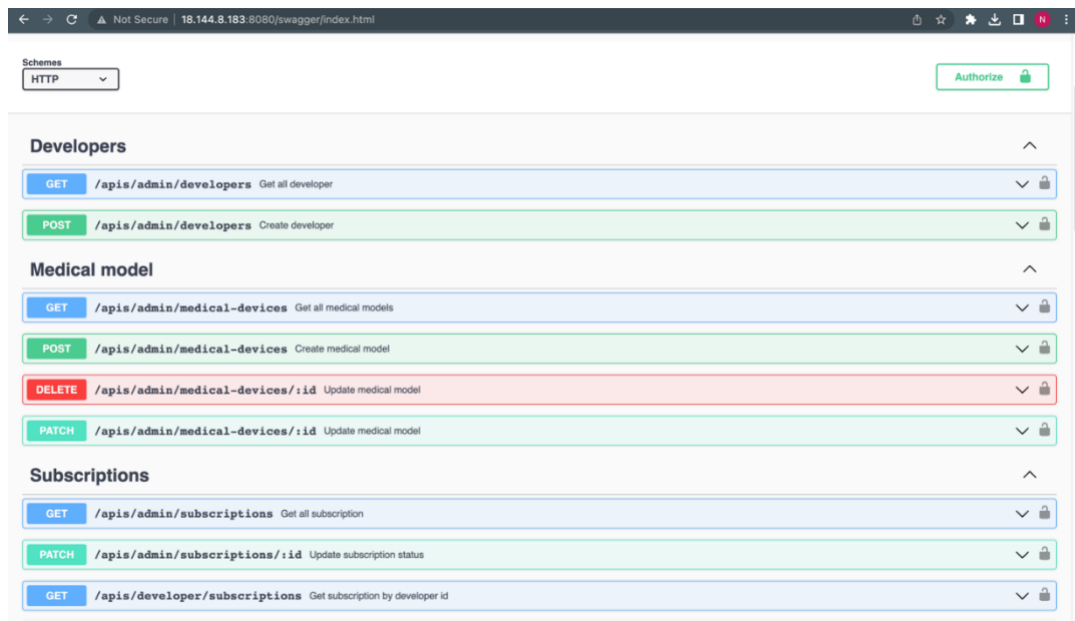
หน้าเว็บสำหรับจัดการ subscription ที่มีการขออนักพัฒนาดังรูปที่ 23 โดยสามารถที่จะทำการยอมรับด้วยการกดปุ่ม Accept หรือยกเลิกด้วยการกดปุ่ม Reject นอกจากนี้สามารถที่จะทำการถอนการยอมรับด้วยการกดปุ่ม Revoke



AppName	DeveloperName	ModelName	DeviceId	StartTime	EndTime	Mode	N...	Status	Action
developer1-app1	developer1	smartPulz	SN-2005-0007	3/24/2023, 7:32:24 PM	3/26/2023, 4:52:24 AM	REST	note-3	APPROVED	Revoke
developer1-app1	developer1	ipdMonitoring	SN-2005-0005	1/22/2020, 9:45:44 AM	7/27/2029, 12:25:44 AM	REST	note-3	APPROVED	Revoke
developer1-app1	developer1	smartPulz	SN-2005-0007	1/22/2020, 9:45:44 AM	7/27/2029, 12:25:44 AM	REST	note-3	APPROVED	Revoke
developer1-app1	developer1	patientMonitor	SN-2005-0007	1/22/2020, 9:45:44 AM	7/27/2029, 12:25:44 AM	REST	note-3	APPROVED	Revoke
developer1-app1	developer1	patientMonitor	SN-2005-0007	1/22/2020, 9:45:44 AM	7/27/2029, 12:25:44 AM	REST	note-3	PENDING	Approve Reject

รูปที่ 23 สำหรับจัดการการขอ subscription ของนักพัฒนา

4.3. Webserver



Section	Method	Endpoint	Description
Developers	GET	/apis/admin/developers	Get all developer
	POST	/apis/admin/developers	Create developer
Medical model	GET	/apis/admin/medical-devices	Get all medical models
	POST	/apis/admin/medical-devices	Create medical model
	DELETE	/apis/admin/medical-devices/:id	Update medical model
	PATCH	/apis/admin/medical-devices/:id	Update medical model
Subscriptions	GET	/apis/admin/subscriptions	Get all subscription
	PATCH	/apis/admin/subscriptions/:id	Update subscription status
	GET	/apis/developer/subscriptions	Get subscription by developer id

รูปที่ 24 API document

ตารางที่ 2 endpoint ต่าง ๆ ใน web service

Router	Method	Path	Description
/apis/auth	POST	/login	สำหรับการ login เข้าสู่ระบบ
/apis/admin	GET	/subscriptions	สำหรับการ list subscriptions ที่มีทั้งหมดในระบบให้กับ admin dashboard
/apis/admin	PATCH	/subscriptions/status/:id	สำหรับการอัปเดตสถานะของ subscription โดยใช้ ID
/apis/admin	GET	/medical-devices	สำหรับการ list medical models ทั้งหมดที่มีในระบบ
/apis/admin	GET	/medical-devices/list	สำหรับการ list medical models พร้อมกับ list ของ device's id ของอุปกรณ์นั้นๆ
/apis/admin	POST	/medical-devices	สำหรับการ register อุปกรณ์ทางการแพทย์ชนิดใหม่เข้าสู่ระบบ
/apis/admin	PATCH	/medical-devices/:id	สำหรับการอัปเดตอุปกรณ์ทางการแพทย์โดยใช้ ID
/apis/admin	DELETE	/medical-devices/:id	สำหรับการลบอุปกรณ์ทางการแพทย์โดยใช้ ID
/apis/admin	GET	/developers	สำหรับการ list developers ทั้งหมดที่มีอยู่ในระบบ
/apis/admin	POST	/developers	สำหรับการสร้าง developer เข้าสู่ระบบ
/apis/admin	DELETE	/developers/:id	สำหรับการลบ developer ออกจากระบบโดยใช้ ID
/apis/app	POST	/medical-data	สำหรับ application ที่ต้องการขอข้อมูลแบบ real-time data และ historical data
/apis/developer	GET	/info	สำหรับการขอข้อมูลของ developer คนนั้นๆ โดย identify developer ผ่าน API token
/apis/developer	PATCH	/password	สำหรับให้ developer เปลี่ยนรหัสผ่านในการเข้าสู่ระบบ
/apis/developer	GET	/subscriptions	สำหรับการ list subscription ของ developer คนนั้นๆ โดย identify ผ่าน API token
/apis/developer	POST	/subscriptions	สำหรับการสร้าง subscription เข้าสู่ระบบ
/apis/developer	GET	/apps	สำหรับการ list application ทั้งหมดของ developer คนนั้นๆ โดย identify ผ่าน API token
/apis/developer	POST	/apps	สำหรับการสร้าง application
/apis/developer	PATCH	/apps/:id	สำหรับการอัปเดต application โดยใช้ ID
/apis/developer	PATCH	/apps/secret/:id	สำหรับการอัปเดต application secret โดยใช้ ID
/apis/developer	DELETE	/apps/:id	สำหรับการลบ application โดยใช้ id
/apis/developer	GET	/medical-devices/list	สำหรับการ list medical models พร้อมกับ list ของ device's id ของอุปกรณ์นั้นๆ

4.4. Kafka Consumer

Web service ทำหน้าที่เป็น consumer มีหน้าที่รับ message จาก Kafka cluster ซึ่งเป็นข้อมูลจากอุปกรณ์การแพทย์ที่ produce message มาที่ Kafka cluster

4.5. Kafka Producer

Web service ทำหน้าที่เป็น producer มีหน้าที่ส่ง message ไปยัง Kafka cluster เพื่อให้ application ที่ขอข้อมูลแบบ real-time สามารถ consume ข้อมูลจาก Kafka cluster

4.6. Logging

4.6.1. ผลลัพธ์การอัปโหลดข้อมูลเข้า GCS

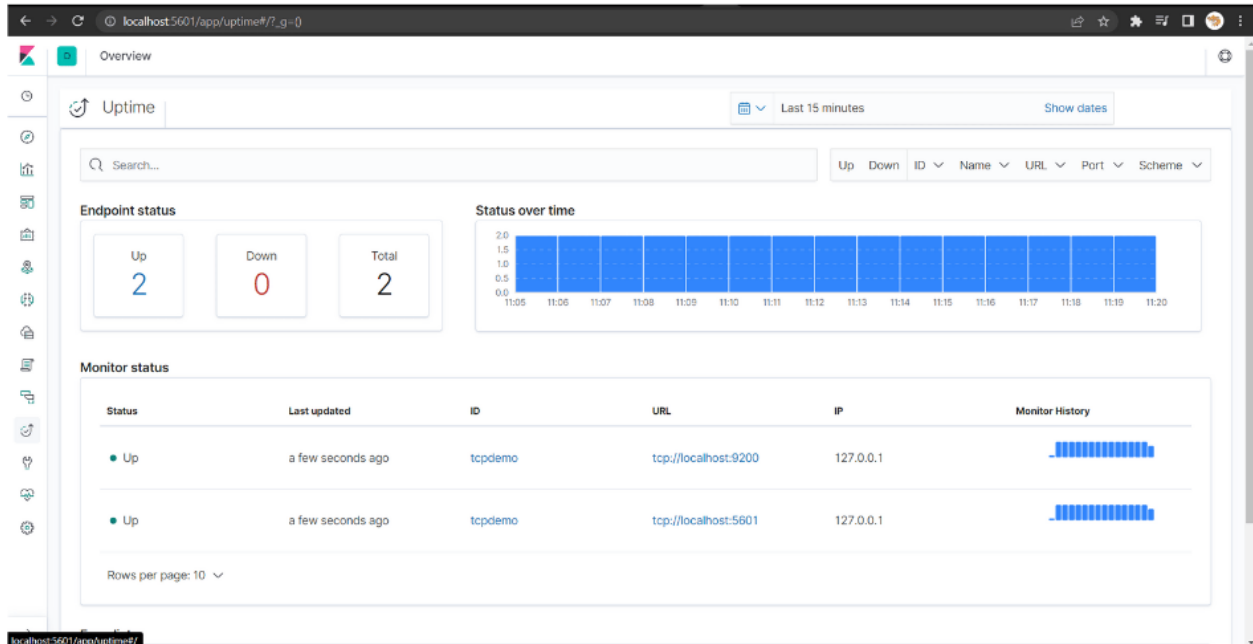
ระบบมีการเก็บ log ผลลัพธ์การอัปโหลดข้อมูลเข้าสู่ GCS ลงใน elasticsearch โดยลักษณะโครงสร้างของ log จะเป็นไปตามรูปที่ 25

```
"hits": [
  {
    "_index": "test-upload",
    "_id": "MxDGA4cBfea0yY0_CbxR",
    "_score": null,
    "_source": {
      "modelName": "ipdMonitoring",
      "deviceId": "SN-2005-0005",
      "startTime": 1679752002,
      "endTime": 1679806002,
      "createdDate": 1679395522,
      "isCompleted": true
    },
    "sort": [
      1679805950
    ]
  },
  {
    "_index": "test-upload",
    "_id": "KRCAAocBfea0yY0_17y7",
    "_score": null,
    "_source": {
      "modelName": "ipdMonitoring",
      "deviceId": "SN-2005-0005",
      "startTime": 1679698002,
      "endTime": 1679752002,
      "createdDate": 1679374211,
      "isCompleted": true
    },
    "sort": [
      1679752060
    ]
  }
]
```

รูปที่ 25 Log ของการอัปโหลดข้อมูลเข้า GCS

4.6.2. Uptime

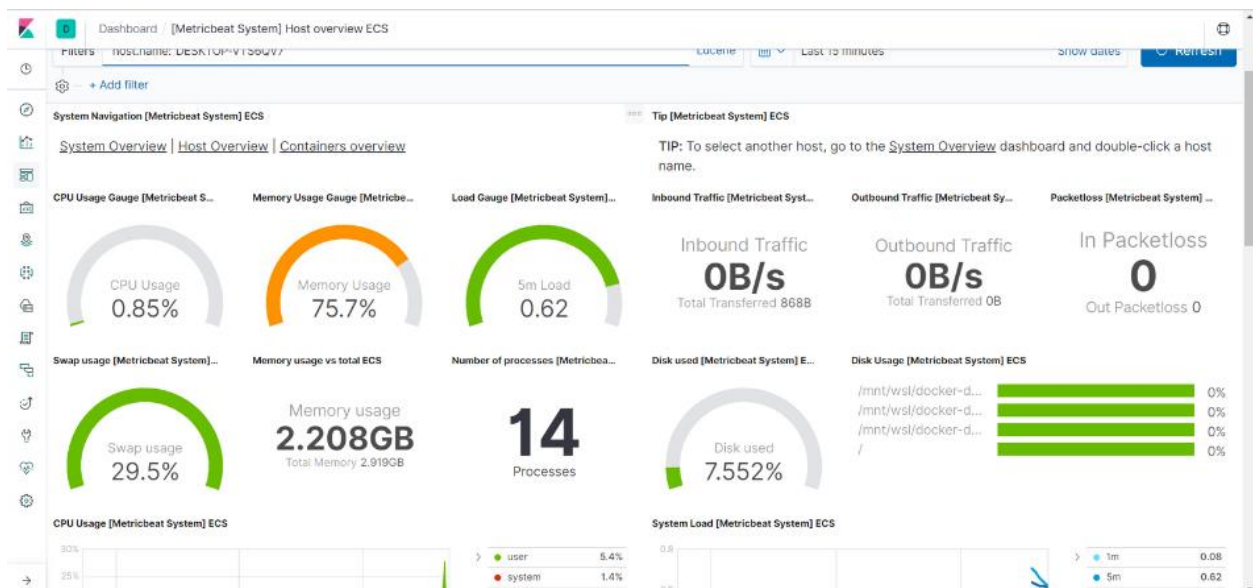
ระบบมีการเก็บประวัติและสถิติการทำงานของ web server โดยจะแสดงผลเป็น dashboard ที่มีลักษณะตามรูปที่ 26



รูปที่ 26 Uptime

4.6.3.CPU/Mem usage

ระบบมีการเก็บประวัติและสถิติการใช้งานทรัพยากรต่าง ๆ ของเครื่อง เช่น CPU usage และ memory usage เป็นต้น โดยจะแสดงผลเป็น dashboard ที่มีลักษณะตามรูปที่ 27



รูปที่ 27 CPU/Mem usage

4.7. การ deploy web client

ขณะผู้จัดทำได้มีการ deploy web client โดยทำการสร้าง Docker file และนำไป deploy ที่ EC2 instance 1 เครื่อง โดย docker file จะมีการเขียนดังรูปที่ 28

```
FROM node:16-alpine AS builder
WORKDIR /app

ARG REACT_APP_API_URL
ENV REACT_APP_API_URL=$REACT_APP_API_URL

COPY package.json ./
COPY . .

RUN yarn install --frozen-lockfile
RUN yarn run build

FROM node:16-alpine AS production
WORKDIR /app
RUN yarn global add serve
COPY --from=builder /app/build ./build

EXPOSE 3000
CMD ["serve", "-s", "build", "-l", "3000"]
```

รูปที่ 28 docker file ของ web client

4.8. Web service cluster

ขณะผู้จัดทำได้มีการ deploy web service โดยทำการสร้าง Docker file ดังรูปที่ 29 และมี folder structure ตามรูปที่ 30 จากนั้นนำไป deploy ที่ EC2 instance โดยใช้ instance ทั้งหมด 3 เครื่อง และขณะผู้จัดทำได้ใช้ nginx เพื่อเป็น load balancer และได้ deploy nginx server โดยใช้ EC2 instance 1 เครื่องโดยมีการเขียน nginx configuration file ดังภาพที่ 31

```
FROM amd64/eclipse-temurin:17-jre-alpine
WORKDIR /home

RUN apk add --update bash && rm -rf /var/cache/apk/*
COPY . .

CMD [ "./iot-healthcare" ]
```

รูปที่ 29 docker file ของระบบ

```

  ✓ bash-script
    $ authentication.sh
    $ authorization.sh
  > kafka
    ! config.yaml
  Dockerfile

```

รูปที่ 30 folder structure ที่ไว้ build docker image

```

events {}

http {
    upstream backend {
        server backend1:8080;
        server backend2:8080;
        server backend3:8080;
    }

    server {
        listen 80;
        listen [::]:80;
        server_name _;
        location / {
            proxy_pass http://backend;
        }
    }
}

```

รูปที่ 31 nginx configuration file

4.9. Kafka cluster

คณะผู้จัดทำได้มีการ configure Kafka cluster โดยใช้ EC2 instance 3 instances ดังรูปที่ 32 โดย instance แต่ละเครื่องจะมี zookeeper และ broker รันอยู่เครื่องละ 1 process และมีการ configure security โดยใช้ SASL_SSL/ SCRAM (SCRAM-SHA-512)

<input type="checkbox"/>	kafka-2	i-0bd4702d021ca205e	Running	🔍	t3.small	2/2 checks passed	No alarms	+	us-west-1b
<input type="checkbox"/>	kafka-1	i-002ceb9edf345eacc	Running	🔍	t3.small	2/2 checks passed	No alarms	+	us-west-1b
<input type="checkbox"/>	kafka-3	i-09f828f1a9450a10d	Running	🔍	t3.small	2/2 checks passed	No alarms	+	us-west-1b

รูปที่ 32 kafka cluster บน cloud

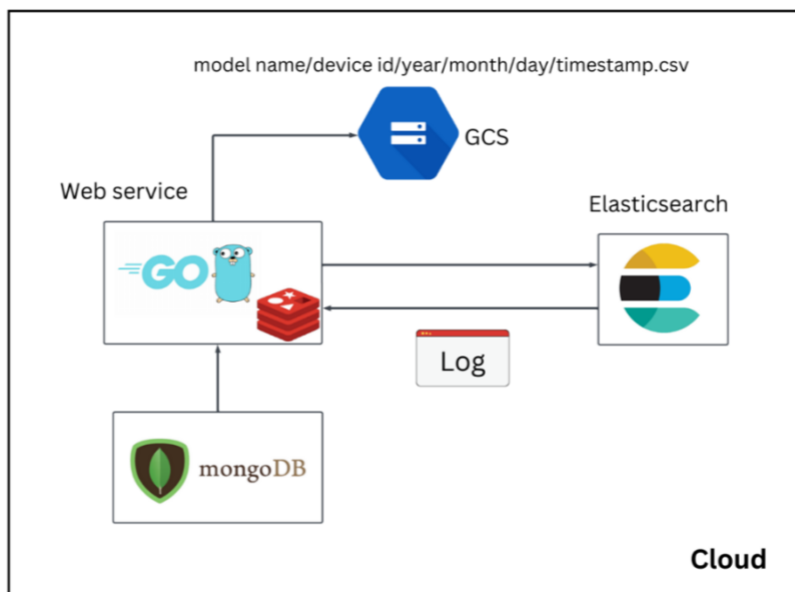
4.10. การอัปโหลดข้อมูลจากอุปกรณ์การแพทย์ขึ้นสู่ google cloud storage

คณะผู้จัดทำได้มีสร้างระบบที่ทำการอัปโหลดข้อมูลในรูปแบบของไฟล์ CSV ดังรูปที่ 33 ขึ้น google cloud storage ทุกๆ 15 นาที จากภาพที่ 34 ระบบนี้ได้ทำการ query โมเดลอุปกรณ์การแพทย์ และ

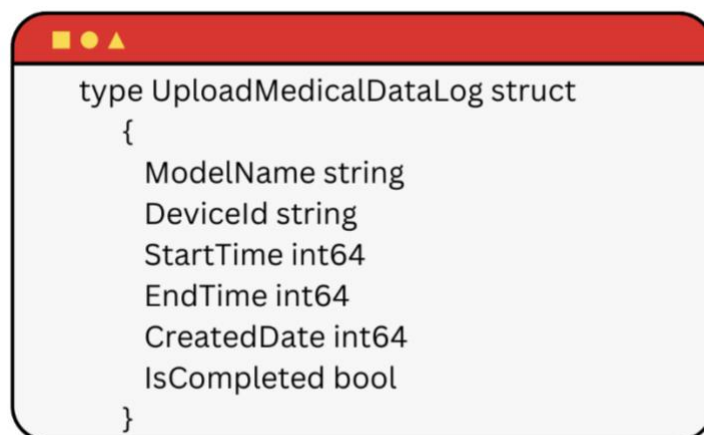
อุปกรณ์การแพทย์ชนิดนั้นๆ ทั้งหมดเพื่อใช้อ้างอิงใน query เพื่อดึงข้อมูลของอุปกรณ์นั้นๆ นอกจากนี้ระบบจะไปดึง log ดังรูปที่ 35 บน elasticsearch เพื่อใช้อ้างอิงเวลานั้นคือ จะใช้ end-time ของการอัปโหลด ข้อมูลที่สำเร็จล่าสุดเป็น start-time ของการอัปโหลดในครั้งถัดไป โดย folder structure ของการเก็บข้อมูล นั้นจะอยู่ในรูปแบบ model name/device id/year/month/day/timestamp.csv

	A	B	C	D	E	F	G	
	ipdMonitoring_SN-2005-0005_2023_4_20_2023_03_26_19_46_42							
1	timestamp	deviceId	temperature	heartRate	o2Saturation	respirationRate	bloodPressure	
2	2023-03-26 04:46:51 +0000 UTC	SN-2005-0005	2	3	6	0	0	
3	2023-03-26 04:47:01 +0000 UTC	SN-2005-0005	1	2	1	5	4	
4	2023-03-26 04:47:11 +0000 UTC	SN-2005-0005	6	1	7	9	7	
5	2023-03-26 04:47:21 +0000 UTC	SN-2005-0005	4	5	6	0	2	
6	2023-03-26 04:47:31 +0000 UTC	SN-2005-0005	8	4	8	6	5	
7	2023-03-26 04:47:41 +0000 UTC	SN-2005-0005	6	6	4	5	7	
8	2023-03-26 04:47:51 +0000 UTC	SN-2005-0005	3	6	5	4	5	
9	2023-03-26 04:48:01 +0000 UTC	SN-2005-0005	3	3	1	5	0	
10	2023-03-26 04:48:11 +0000 UTC	SN-2005-0005	5	7	2	2	8	
11	2023-03-26 04:48:21 +0000 UTC	SN-2005-0005	7	1	7	7	3	
12	2023-03-26 04:48:31 +0000 UTC	SN-2005-0005	5	9	9	1	5	
13	2023-03-26 04:48:41 +0000 UTC	SN-2005-0005	5	1	9	4	6	
14	2023-03-26 04:48:51 +0000 UTC	SN-2005-0005	7	8	4	8	8	
15	2023-03-26 04:49:01 +0000 UTC	SN-2005-0005	4	5	2	0	4	
16	2023-03-26 04:49:11 +0000 UTC	SN-2005-0005	7	1	2	0	0	
17	2023-03-26 04:49:21 +0000 UTC	SN-2005-0005	0	9	0	5	9	
18	2023-03-26 04:49:31 +0000 UTC	SN-2005-0005	6	6	1	6	7	
19	2023-03-26 04:49:41 +0000 UTC	SN-2005-0005	8	5	6	1	9	
20	2023-03-26 04:49:51 +0000 UTC	SN-2005-0005	9	4	9	4	0	
21	2023-03-26 04:50:01 +0000 UTC	SN-2005-0005	0	7	8	5	8	
22	2023-03-26 04:50:11 +0000 UTC	SN-2005-0005	9	4	9	8	0	
23	2023-03-26 04:50:21 +0000 UTC	SN-2005-0005	2	5	8	9	9	
24	2023-03-26 04:50:31 +0000 UTC	SN-2005-0005	7	5	1	7	4	
25	2023-03-26 04:50:41 +0000 UTC	SN-2005-0005	1	1	7	2	8	
26	2023-03-26 04:50:51 +0000 UTC	SN-2005-0005	0	8	5	9	5	
27	2023-03-26 04:51:01 +0000 UTC	SN-2005-0005	5	3	9	8	1	

รูปที่ 33 รูปแบบของไฟล์ที่จัดเก็บใน GCS



รูปที่ 34 Flow การอัปโหลดข้อมูลเข้าสู่ GCS



รูปที่ 35 รูปแบบการเก็บ log ของการอัปโหลดข้อมูลขึ้น GCS

5. ผลการทดลอง/ผลการวิเคราะห์

5.1. เครื่องมือที่ใช้

5.1.1. ภาษาที่เลือกใช้

ระบบ Web Service คณะผู้จัดทำเลือกใช้ภาษา GO ในการพัฒนาเนื่องจากเป็นภาษาที่ทำงานได้เร็ว อีกทั้งยังรองรับการทำ Concurrent Programming และ Multithreading เพราะฉะนั้นจึงเหมาะกับการรองรับ request เป็นจำนวนมาก นอกจากนี้ยังมี standard library รองรับการทำงานที่หลากหลาย [16]

5.1.2. ระบบฐานข้อมูลที่ใช้

คณะผู้จัดทำเลือกใช้ MongoDB เนื่องจาก MongoDB เป็น NoSQL ทำให้ง่ายต่อการเปลี่ยนแปลงโครงสร้างของข้อมูล อีกทั้งยังมี driver ที่ช่วยในการติดต่อกับฐานข้อมูลที่รองรับภาษา Go ที่ผู้จัดทำเลือกใช้ และยังสามารถ scale ฐานข้อมูลได้ง่าย [17]

5.1.3. Library ในการให้ Web Service ติดต่อกับ Kafka server

คณะผู้จัดทำเลือกใช้ kafka-go library เนื่องจากมี MIT license อีกทั้งยังรองรับการ authentication แบบ SASL_SSL [18]

5.1.4. เครื่องมือในการจัดการ Logs ของ web service

คณะผู้จัดทำเลือกใช้ Elasticsearch ร่วมกับ Kibana เนื่องจากเครื่องมือทั้งสองมีการพัฒนาให้สามารถทำงานร่วมกันได้ดี อีกทั้ง Elasticsearch สามารถทำงานได้รวดเร็วเนื่องจากการออกแบบการจัดเก็บข้อมูลโดย BKD tree นอกจากนี้ยังมี API ให้ Web Service สามารถติดต่อได้ง่าย [19] [20]

5.1.5. การทำ Data Lake

ระบบนี้จะทำการเก็บข้อมูลจากอุปกรณ์ IoT ทางการแพทย์ซึ่งข้อมูลเหล่านั้นมีลักษณะแบบ semi-structure เพราะอุปกรณ์ IoT แต่ละประเภทอาจเก็บข้อมูลลักษณะแตกต่างกัน เช่น อุปกรณ์ประเภทที่ 1 สามารถเก็บค่าอุณหภูมิร่างกายและอัตราการเต้นของหัวใจได้ ส่วนอุปกรณ์ประเภทที่ 2 เก็บค่าอัตราการเต้นของหัวใจได้อย่างเดียว เป็นต้น

ใช้ Google Cloud Storage (GCS) ในการทำ Data Lake เนื่องจากมี Data Retention ทำให้สามารถเก็บข้อมูลที่มีอยู่ไว้ได้หากมีการเปลี่ยนแปลงกฎการรักษาข้อมูล (Retention rule) ซึ่งข้อดีนี้เป็นคุณสมบัติที่ดีในการเก็บข้อมูลด้านสุขภาพ อีกทั้งยังมี interface ที่ทำให้ผู้ใช้งานใช้งานได้ง่าย [21]

5.1.6.การทำ Data Warehouse

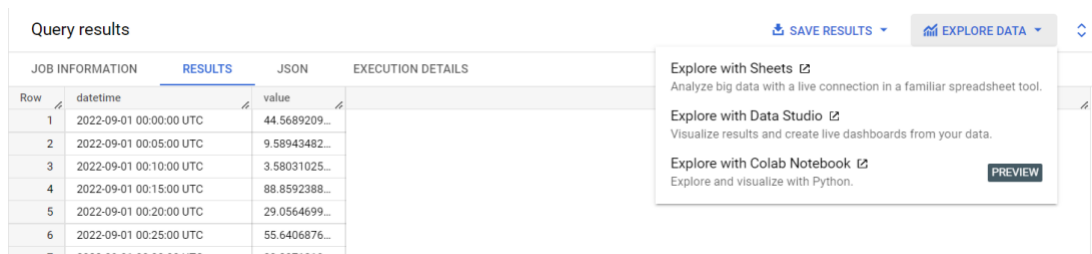
ในการทำ Data Warehouse เลือกใช้ Google Cloud BigQuery เนื่องจากเป็นบริการของ Google เช่นเดียวกันกับ GCS ทำให้สามารถเชื่อมต่อกันได้ง่าย มี API หรือ บริการภายในในการเชื่อมต่อกับ GCS

ระบบจะต้องนำข้อมูลที่อยู่ใน GCS ไปใส่ในตารางใน BigQuery ในส่วนนี้เลือกใช้ Data Transfer ซึ่งเป็นบริการใน BigQuery ที่จะเช็คข้อมูลใน GCS ทุก ๆ 15 นาทีว่ามีข้อมูลเพิ่มหรือไม่ ถ้ามีจะนำข้อมูลที่เพิ่มมานั้นไปต่อท้ายในตารางของ BigQuery เหตุผลที่ใช้ 15 นาที เพราะว่าเป็นค่าที่น้อยที่สุดที่สามารถตั้งค่าได้ใน Data Transfer [22]

5.1.7.การทำ Dashboard สำหรับดูข้อมูลจากอุปกรณ์

Dashboard สำหรับดูข้อมูลจากอุปกรณ์ เป็นสิ่งที่ช่วยอำนวยความสะดวกให้กับนักวิจัยในการวิเคราะห์ข้อมูล เพื่อดูแนวโน้ม ความเป็นไปได้ต่าง ๆ และนำข้อมูลไปวิจัย ต่อยอด พัฒนาองค์ความรู้ทางสาธารณสุขต่อไปในอนาคต

การทำ Dashboard สำหรับนักวิจัย ใช้ Google Data Studio ซึ่งเป็นบริการของ Google เช่นเดียวกันกับ GCS และ BigQuery เนื่องจากใน BigQuery สามารถ Explore Data โดยใช้ Data Studio ได้ ทำให้ไม่ต้องใช้ API หรือ เครื่องมืออื่น ๆ เพิ่มเติม นอกจากนี้เมื่อมีการเปลี่ยนแปลงของตารางใน BigQuery จะทำการเปลี่ยนแปลงข้อมูลใน Data Studio โดยอัตโนมัติดังรูปที่ 36



JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	datetime	value		
1	2022-09-01 00:00:00 UTC	44.5689209...		
2	2022-09-01 00:05:00 UTC	9.58943482...		
3	2022-09-01 00:10:00 UTC	3.58031025...		
4	2022-09-01 00:15:00 UTC	88.8592388...		
5	2022-09-01 00:20:00 UTC	29.0564699...		
6	2022-09-01 00:25:00 UTC	55.6406876...		
7	2022-09-01 00:30:00 UTC	33.3071219...		

รูปที่ 36 GUI ของ BigQuery

5.1.8.การเชื่อมต่อ MQTT กับ Kafka

การเชื่อมต่อของ MQTT กับ Kafka สำคัญเป็นอย่างมากเนื่องจาก เพราะการรับข้อมูลโดยใช้ MQTT เหมาะกับการต่อกับอุปกรณ์จำพวก sensor แต่ไม่เหมาะกับการเก็บข้อมูลจำนวนมากจึงต้องมีการเชื่อมต่อกับ Kafka เพื่อเก็บข้อมูลให้มีประสิทธิภาพและรวบรวมของอุปกรณ์ทั้งหมดไว้ในที่เดียว เพื่อให้ง่ายต่อการส่งต่อ

การเชื่อมต่อ MQTT กับ Kafka ใช้ MQTT-TO-KAFKA-BRIDGE ในการเชื่อมต่อโดยเลือกใช้ python ในการเขียนเนื่องจากมี paho MQTT เป็นตัวกลางสำหรับการติดต่อกับ EMQX และมี

pymongo สำหรับติดต่อกับ database เพื่อขอข้อมูล modelName ที่ใช้เป็น topic ในการ subscribe [23]

5.1.9.การทำ Web Client

ผู้พัฒนาเลือกใช้ React ในการทำ Developer Dashboard และ Admin Dashboard ซึ่งเป็น Web Client ของระบบ และเลือกใช้ภาษา TypeScript ในการพัฒนา เนื่องจาก react มี library และ component ให้เลือกใช้มากมาย และมีฟังก์ชันการใช้งานเพียงพอตาม functional requirement และมี community ขนาดใหญ่ เมื่อเจอปัญหาสามารถหาวิธีแก้ไขจากแหล่งข้อมูลที่หลากหลาย และเลือกใช้ภาษา TypeScript เพราะ TypeScript รองรับ static typing ทำให้ลดความผิดพลาดในการพัฒนา [24][25]

5.2. การเครื่องมือที่ใช้ในการทำ testing

5.2.1.Jest

ผู้พัฒนาเลือกใช้ Jest ในการทดสอบการทำงานในส่วนของการทดสอบ Application API เนื่องจาก Jest นั้นง่ายต่อการ set up environment ในการทดสอบ และยังรองรับการทำ unit testing นอกจากนี้ยังรองรับ exception หลากหลายประเภท [26]

5.2.2.K6

ผู้พัฒนาเลือกใช้ K6 ในการทดสอบการตอบสนองของ web server เนื่องจาก K6 นั้นง่ายต่อการ set up environment และสามารถควบคุม traffic ในการทดสอบระบบได้ง่าย นอกจากนี้ยังรองรับการทำงานร่วมกับ InfluxDB และ Grafana เพื่อสามารถดูผลการทดสอบ performance ของ web server ผ่าน dashboard [27]

5.2.3.Selenium

ผู้พัฒนาเลือกใช้ Selenium ในการทดสอบระบบ dashboard เนื่องจาก Selenium รองรับการใช้งานเพื่อทดสอบเว็บไซต์ โดยสามารถทดสอบการทำงานโดยรวมของทั้งเว็บไซต์แบบอัตโนมัติได้โดยการเขียนโค้ด ช่วยลดระยะเวลาในการทดสอบเมื่อมีการแก้ไขโปรแกรมบางส่วน [28]

5.3. การทดสอบระบบ

5.3.1.การทดสอบ Developer Dashboard

ตารางที่ 3 ผลการทดสอบ Developer Dashboard

ID	Page	Description	Input	Expected output	Pass/Fail
1	login page	email or password (required) is empty	Email:" Password: "	ask user to enter email and password	Pass
2	login page	email format is wrong and password is not empty	Email: 'abc' Password: 'password'	warning email is wrong format	Pass
3	login page	email format is correct and password is correct	Email: 'abc@def.g' Password: 'password'	successfully login	Pass
4	login page	email format is correct but not exists	Email: 'a@b.com' Password: 'password'	invalid email or password	Pass
5	login page	email is correct and password is not correct	Email: 'a@b.com' Password: 'password'	invalid email or password	Pass
6	-	not login	-	Warning and redirect to login page	Pass
7	app page	get all of my applications	-	list all of my applications	Pass
8	app page	new application required fields(appname) are empty	AppName: " Description: "	ask user to enter app name	Pass
10	app page	new application app name is repetitive	AppName: 'my-app' Description: "	warning app name must be unique	Pass
11	app page	new application all required fields are complete	AppName: 'my-app-2' Description: 'this is my wonderful app'	successfully create a new app	Pass
12	app page	edit app info edit app name to be the name that is already used	AppName: 'my-app'	warning app name must be unique	Pass
13	app page	edit app info edit app name to be the name that isn't already used	AppName: 'my-app-2'	successfully edit app info	Pass
14	app page	edit app info edit description	Description: 'this is my wonderful app'	successfully edit app info	Pass
15	app page	renew client secret	-	client secret updated	Pass
16	app page	delete an app	-	successfully delete app	Pass
17	app page	list all subscription of this app	-	redirect to subscription page and set app name filter to this app	Pass

18	subscription page	get all of my subscriptions	-	list all of my subscriptions	Pass
19	subscription page	new subscription some of required fields are empty	application = " model = " device= [] starttime = " endtime = " mode = "	ask user to input those fields	Pass
20	subscription page	new subscription all required fields are not empty	application = 'my-app' model = 'ipdMonitoring' device= ['SN-2005-0005'] starttime = '10/4/2023 23:00' endtime = '11/4/2023 23:00' mode = 'REST'	successfully create new subscription	Pass
21	subscription page	filter subscriptions by device	device= 'SN-2005-0005'	show all subscriptions that request data of the device	Pass
22	subscription page	filter subscriptions by model	model = 'ipdMonitoring'	show all subscriptions that request data of the model type	Pass
23	subscription page	filter subscriptions by time	starttime = '10/4/2023 23:00' endtime = '11/4/2023 23:00'	show all subscriptions that request data in the time range	Pass
24	subscription page	filter subscriptions by mode	mode = 'REST'	show all subscriptions that are the mode	Pass
25	subscription page	filter subscriptions by status	status = 'pending'	show all subscriptions that are in the status	Pass
26	subscription page	filter subscriptions by application	application = 'my-app'	show all subscriptions that belong to the app	Pass
27	account page	get account info	-	show this account info	Pass
28	account page	change password old password is empty	OldPassword: ""	ask user to enter old password	Pass
29	account page	change password new password is empty	NewPassword: ""	ask user to enter new password	Pass

30	account page	change password re-new password is empty	ReNewPassword: ""	ask user to enter re-new password	Pass
31	account page	change password new password and re-new password aren't match	NewPassword: "newpass" ReNewPassword: "newpassword"	warning the passwords aren't match	Pass
32	account page	change password wrong password	OldPassword: "wrong" NewPassword: "newpassword" ReNewPassword: "newpassword"	warning old password is invalid	Pass
33	account page	change password correct password	OldPassword: "password" NewPassword: "newpassword" ReNewPassword: "newpassword"	successfully update password	Pass
34	-	logout	-	clear token and redirect to login page	Pass

จากการทดสอบการทำงานของ Developer Dashboard ได้ผลลัพธ์ดังตารางที่ 3 โดยสามารถสรุปได้ว่าระบบทำงานได้อย่างถูกต้องตรงตาม functional requirement

5.3.2.การทดสอบ Admin Dashboard

ตารางที่ 4 ผลการทดสอบ Admin Dashboard

ID	Page	Description	Input	Expected output	Pass/Fail
1	model management page	get all medical models		list all of medical models	Pass
2	model management page	delete a medical model		successfully delete medical model	Pass
3	model management page	Create new medical model. All fields are empty.	ModelName: "" ModelKey: [] NewKeys: []	ask user to enter model name and model key	Pass
4	model management page	Create new medical model. Model Name are empty. Model Keys are chosen	ModelName: "" ModelKey: ["heartRate", "bloodPressure"] NewKeys: []	ask user to enter model name	Pass

		New Keys are not added			
5	model management page	Create new medical model. Model Name are correct Model Keys are empty New Keys are not added	ModelName: "SN-1005-2" ModelKey: [] NewKeys: []	ask user to enter model key	Pass
6	model management page	Create new medical model. Model Name are correct Model Keys are chosen New Keys are not added	ModelName: "SN-1005-2" ModelKey: ["heartRate", "bloodPressure"] NewKeys: []	successfully create new medical model	Pass
7	model management page	Create new medical model. Model Name are correct Model Keys are chosen. New Keys are added and empty.	ModelName: "SN-1005-2" ModelKey: ["heartRate", "bloodPressure"] NewKeys: [""]	ask user to enter new model key	Pass
8	model management page	Create new medical model. Model Name are correct Model Keys are chosen New Keys are added and correct	ModelName: "SN-1005-2" ModelKey: ["heartRate", "bloodPressure"] NewKeys: ["o2"]	successfully create new medical model	Pass
9	developer management page	get all developers		list all of developers	Pass
10	developer management page	delete a developer		successfully delete developer	Pass

11	developer management page	Create new developer Email is empty	Email: "" Name: "developer" AssociatedName: "hospital"	ask user to enter Email	Pass
12	developer management page	Create new developer Email has incorrect format	Email: "developer" Name: "developer" AssociatedName: "hospital"	ask user to enter Email in format	Pass
13	developer management page	Create new developer Email has correct format and is duplicate	Email: "developer@gmail.com" Name: "developer" AssociatedName: "hospital"	ask user to enter Email that not already create	Pass
14	developer management page	Create new developer Name are empty	Email: "developer001@gmail.com" Name: "developer" AssociatedName: "hospital"	ask user to enter Name	Pass
15	developer management page	Create new developer AssociatedName are empty	Email: "developer001@gmail.com" Name: "developer" AssociatedName: "hospital"	ask user to enter AssociatedName	Pass
16	developer management page	Create new developer All field are correct	Email: "developer001@gmail.com" Name: "developer" AssociatedName: "hospital"	successfully create new developer	Pass
17	subscription management page	list all subscriptions that are pending and accepted		list all subscriptions that are pending and accepted	Pass
18	subscription management page	Accept subscription		successfully accept subscription	Pass
19	subscription management page	Reject subscription		successfully reject subscription	Pass
20	subscription management page	Revoke subscription		successfully revoke subscription	Pass
21	subscription management page	Filter by status (Revoked)	status = "REVOKED"	list all subscriptions that are revoked	Pass

จากการทดสอบการทำงานของ Admin Dashboard ได้ผลลัพธ์ดังตารางที่ 4 โดยสามารถสรุปได้ว่าระบบทำงานได้อย่างถูกต้องตรงตาม functional requirement

5.3.3.การทดสอบ application API

ตารางที่ 5 ผลการทดสอบ application API

ID	Mode	Input	Expected output	Pass/Fail	Note
1	REST	Valid access token Payload { modelName: "ipdMonitoring", deviceId: "SN-2005-0005", startTime: 0, endTime: 1680541934, mode: "REST", }	{ "data": { "ErrorCode": 403, "Message": "Forbidden" } }	Pass	Forbidden since start-time or end-time is not allow
2	REST	Valid access token Payload { modelName: "patientMonitor", deviceId: "SN-2005-0007", startTime: 1580511934, endTime: 1580541934, mode: "REST", }	{ "data": { "ErrorCode": 403, "Message": "Forbidden" } }	Pass	Forbidden since start-time or end-time is not allow
3	REST	Valid access token Payload { modelName: "xxx", deviceId: "SN-2005-0007", startTime: 1680511934, endTime: 1680541934, mode: "REST", }	{ "data": [] }	Fail	Not found since model name is not exist in the database return null
4	REST	Valid access token { modelName: "patientMonitor",	{ "data": [] # length == 60 }	Pass	successfully request data

		deviceId: "SN-2005-0007", startTime: 1680585127, endTime: 1680586127, mode: "REST" }			
5	REST	Invalid access token	{ "data": { "ErrorCode": 401, "Message": "Token was canceled" } }	Pass	Unauthorized since access token is invalid
6	KAFKA	Invalid access token	{ "data": { "ErrorCode": 401, "Message": "Token was canceled" } }	Pass	Unauthorized since access token is invalid
7	KAFKA	{ modelName: "ipdMonitoring", deviceId: "SN-2005-0005", startTime: 1680541921, endTime: 1680542921, mode: "KAFKA", }	{ "data": { "ErrorCode": 403, "Message": "Forbidden" } }	Pass	start-time or end-time is not allow
8	KAFKA	{ modelName: "xxx", deviceId: "SN-2005-0005", startTime: 1680541921, endTime: 1680542921, mode: "KAFKA", }	{ "data": { "ErrorCode": 403, "Message": "Forbidden" } }	Pass	model name or device id is not allow
9	KAFKA	{ modelName: "ipdMonitoring", deviceId: "SN-2005-0005", startTime: 1680585127, endTime: 1680585527, }	server return 201 http status	Pass	request with permission

		mode: "KAFKA" }			
--	--	--------------------	--	--	--

จากการทดสอบการทำงานของ application API ได้ผลลัพธ์ดังตารางที่ 5 โดยสามารถสรุปได้ว่าระบบทำงานได้อย่างถูกต้องตรงตาม functional requirement

5.3.4.authentication และ authorization ของ kafka cluster

ตารางที่ 6 ผลการทดสอบ authentication และ authorization ของ kafka cluster

ID	Role	Operation	Topic	Payload	Expected output	Note	Pass/Fail
1	application	Read	6429b8c5a84c598f51875aec	{ "username": "6429b8c5a84c598f51875aec", "password"invalid" }	Not allow	Invalid password	Pass
2	application	Read	6429b8c5a84c598f51875aec	{ "username": "6429b8c5a84c598f51875aec", "password"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2NDI5YjhjNWE4NGM1OTNmNTE4NzVhZWMiLCJ1c2VyVHlwZSI6WyJhcHAiXSwiZXhwIjoxNzExOTkxODc3fQ.kPCfAssYWjW_L0t7sgMJBfM0JonBEgY53fcklQnJVAK" }	allow		Pass
3	application	Write	6429b8c5a84c598f51875aec	{ "username": "6429b8c5a84c598f51875aec", "password"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2NDI5YjhjNWE4NGM1OTNmNTE4NzVhZWMiLCJ1c2VyVHlwZSI6WyJhcHAiXSwiZXhwIjoxNzExOTkxODc3fQ.kPCfAssYWjW_L0t7sgMJBfM0JonBEgY53fcklQnJVAK" }	Not allow	Forbidden [write its topic]	Pass
4	application	Write	smartPulz	{ "username":	Not allow	Forbidden	Pass

				"6429b8c5a84c598f51875aec", "password"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2NDI5YjhjNWE4NGM1OThmNTE4NzVhZWMiLCJ1c2VyVHlwZSI6WyJhcHAiXSwiZXhwIjoxNzExOTkxODc3fQ.kPCfAssYWjW_L0t7sgMJBfm0JonBEgY53fcklQnJVAk" }			
--	--	--	--	--	--	--	--

จากการทดสอบการ authentication และ authorization ของ kafka cluster ได้ผลลัพธ์ดังตารางที่ 6 โดยสามารถสรุปได้ว่าระบบทำงานได้อย่างถูกต้องตรงตาม functional requirement

5.3.5.การทดสอบ MQTT-Bridge

ตารางที่ 7 ผลการทดสอบ MQTT-Bridge

ID	use case	Input	Expected output	Pass/Fail
1	send 10 data	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	-	Pass
2	send 100 data	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	-	Pass
3	send 10 data	Topic: newDevice2 { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	Topic: newDevice2 { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	Pass
4	send 100 data	Topic: newDevice2 { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	Topic: newDevice2 { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	Pass
5	send 10 data + trigger 2	Topic: wrongTopic { "device_id": "SN-2005-0009",	-	Pass

		"temperature": 10 } * 10 differently		
6	send 10 data + trigger 8	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	-	Pass
7	send 100 data + trigger 5	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	-	Pass
8	send 100 data + trigger 10	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	-	Pass
9	send 100 data + trigger 15	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	-	Pass
5	send 10 data + trigger 2	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	
6	send 10 data + trigger 8	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	
7	send 100 data + trigger 5	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	
8	send 100 data + trigger 10	Topic: wrongTopic { "device_id": "SN-2005-0009",	Topic: wrongTopic { "device_id": "SN-2005-0009",	

		"temperature": 10 } * 100 differently	"temperature": 10 } * 100 differently	
9	send 100 data + trigger 15	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 100 differently	
10	10 device not authen SASL to kafka	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	-	
11	1 device not authen SASL to kafka + 9 device authen SASL to kafka	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 10 differently	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 9 differently	
11	5 device not authen SASL to kafka + 45 device authen SASL to kafka	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 50 differently	Topic: wrongTopic { "device_id": "SN-2005-0009", "temperature": 10 } * 50 differently	

จากการทดสอบการทำงานของ MQTT-Bridge ได้ผลลัพธ์ดังตารางที่ 7 โดยสามารถสรุปได้ว่าระบบทำงานได้อย่างถูกต้องตรงตาม functional requirement

5.3.6.ผลการ load testing web service

คณะผู้จัดทำได้ทำการ load testing ในส่วนของ endpoint POST /apis/app/medical-data โดย generate request throughput ที่ 100 requests/sec และขนาดของ response body อยู่ที่ 60 data points เนื่องจากการใช้งานจริงผู้ใช้ไม่ควรขอข้อมูลย้อนหลังครั้งละมากๆ แต่ควรทำ paging และขอข้อมูลหลาย ๆ ครั้งแทน เพื่องานต่อการจัดการข้อมูล

จากการ deploy web service cluster จะมี instance ที่รัน web service ทั้งหมด 3 เครื่อง เพราะฉะนั้น throughput ของแต่ละเครื่องจะเฉลี่ยอยู่ที่ $100/3$ คือประมาณ 33.34 requests/sec และผลการทดสอบของระบบนั้นเมื่อ generate throughput โดยเฉลี่ยที่ 36.4 requests/sec ได้ผลดังตารางที่ 8

ตารางที่ 8 ผลการทำ load testing web service

measurement	mean	med	min	max	P95
http request duration (ms)	774.25	808.25	515.48	1170	1000
http waiting time (ms)	773.37	807.51	514.81	1170	1000

5.3.7.ผลการทดสอบการให้ข้อมูลแบบ real-time

ในโรงพยาบาลจะมีจำนวนห้อง CPU ประมาณ 40 ห้อง และในแต่ละห้องจะมีอุปกรณ์ประมาณ 5 เครื่อง แสดงว่าใน 1 โรงพยาบาลจะมีอุปกรณ์ 200 เครื่อง โดยจะ assume ว่า 1 อุปกรณ์คือ 1 application จะได้ว่ามีทั้งหมด 200 requests จากนั้น bound up โดยการคูณ 2 เป็น 400 requests ดังนั้น ในการทดสอบคณะผู้จัดทำจึงได้ load testing ระบบโดยให้มี requests จำนวน 400 requests ซึ่งได้ผลลัพธ์ดังตารางที่ 9

ตารางที่ 9 ผลการทดสอบการขอข้อมูลแบบ real-time

ครั้งที่	Average waiting time
1	1.051 s
2	1.048 s
3	1.028 s
4	1.051 s
5	1.048 s
6	1.045 s
7	1.026 s
8	1.048 s
9	1.026 s
10	1.026 s

6. ผลกระทบทางสังคมของโครงการ

ทางผู้จัดทำได้ทำการพิจารณาผลกระทบทางสังคมที่จะเกิดจากโครงการที่กำลังพัฒนาขึ้นนี้ โดยจากการวิเคราะห์ได้ผลกระทบทางบวกออกมาดังนี้

- 6.1. เกิดแหล่งข้อมูลความรู้ทางด้านสุขภาพของประเทศไทยในอนาคต สามารถนำข้อมูลมาวิเคราะห์แนวโน้มสุขภาพของคนไทย
- 6.2. ช่วยให้การรักษาทางไกลสามารถทำได้อย่างมีประสิทธิภาพมากขึ้น ทำให้ช่วยเพิ่มพื้นที่สำหรับผู้ป่วยฉุกเฉินในโรงพยาบาล ลดความแออัดของผู้ป่วย
- 6.3. ลดงบประมาณในการดูแลระบบฐานของข้อมูลของแต่ละโรงพยาบาลรัฐบาล เนื่องจากแพลตฟอร์มจะเป็นศูนย์กลางในการรวบรวมข้อมูล

7. บทสรุปและงานที่จะทำต่อไป

7.1. บทสรุป

ระบบที่สร้างขึ้นเป็นระบบต้นแบบที่แสดงให้เห็นว่าข้อมูลจากอุปกรณ์ทางการแพทย์สามารถส่งไปยังระบบคลาวด์ทั้งผ่าน MQTT และ Kafka จากนั้นจะมีการจัดเก็บข้อมูล และส่งไปยัง application ต่าง ๆ ที่มา subscribe เพื่อขอข้อมูลทั้งแบบย้อนหลังและ real-time อีกทั้งยังมี dashboard สำหรับนักพัฒนาและแอดมินเพื่อจัดการข้อมูลต่าง ๆ นอกจากนี้ยังมีระบบ logging สำหรับการตรวจติดตามการทำงาน

7.2. อภิปรายผล

7.2.1. อภิปรายเรื่องกรณีระบบเครือข่ายของโรงพยาบาลขัดข้อง

เมื่อเครือข่ายของโรงพยาบาลขัดข้อง ข้อมูลจากอุปกรณ์ในโรงพยาบาลจะไม่สามารถถูกส่งมาที่ระบบกลางของเราได้ แต่ในแต่ละอุปกรณ์จะมีหน่วยความจำที่ใช้เก็บสำรองข้อมูล เมื่อเครือข่ายกลับมาปกติจะมีการส่งข้อมูลที่สำรองไว้เข้าสู่ระบบ โดยในการใช้จริงจะมีการติดตั้ง private cloud ที่โรงพยาบาล ซึ่งหากเครือข่ายขัดข้องผู้ให้บริการ cloud จะเป็นคนรับผิดชอบในส่วนนี้

7.2.2. อภิปรายเรื่องการติดต่อขอข้อมูล

กรณีขอข้อมูลจากภายในโรงพยาบาล (อยู่ในเครือข่ายเดียวกัน) ระบบภายในจะติดต่อกับ web service ผ่าน load balancer โดย IP ของ load balancer จะเป็น private IP

กรณีขอข้อมูลจากภายนอกโรงพยาบาล (อยู่ในคนละเครือข่าย) ระบบภายในจะติดต่อกับ web service ผ่าน load balancer โดย IP ของ load balancer จะเป็น public IP

7.2.3. อภิปรายเรื่องฐานข้อมูล

ระบบปัจจุบันมีการใช้ MongoDB เป็นฐานข้อมูล เพราะเป็น NoSQL ทำให้ง่ายต่อการเปลี่ยนแปลงโครงสร้างของข้อมูล รองรับ use case การใช้งานในปัจจุบัน สามารถ deploy และ scale ได้ง่าย โดยหากในอนาคตมีขนาด traffic เพิ่มขึ้น และต้องจัดการกับข้อมูลขนาดใหญ่ อาจมีการทำ indexing และ sharding เพื่อเพิ่มประสิทธิภาพการทำงาน หรือหากมีกรณีการใช้งานที่ต้องการใช้ประโยชน์จาก query engine ของ time series database อาจเพิ่ม database ชนิดนี้ เช่น influxDB เข้ามาในระบบ

7.2.4.อภิปรายผลเรื่องการเพิ่มประสิทธิภาพในการอ่านและเขียนของ MongoDB

จาก requirement ระบบจะต้องบันทึกข้อมูลจากอุปกรณ์ทางการแพทย์เป็นจำนวนมาก และในส่วนของ การส่งข้อมูลให้กับแอปพลิเคชันที่เข้ามาลงทะเบียนนั้น แอปพลิเคชันส่วนใหญ่จะขอข้อมูลภายในภูมิภาคของตนเอง จากปัญหาที่กล่าวมาจึงเสนอการเพิ่มประสิทธิภาพของฐานข้อมูลดังนี้

การเพิ่มประสิทธิภาพของเขียนข้อมูลลงฐานข้อมูลควรทำ write replica คือการติดตั้งฐานข้อมูลมากกว่า 1 เครื่องในการจัดการการบันทึกข้อมูล และมีฐานข้อมูล 1 เครื่องในการอ่านข้อมูล

การเพิ่มประสิทธิภาพของการอ่านข้อมูลควรทำ sharding by location ซึ่งในส่วนนี้ควรให้ข้อมูลกระจายข้อมูลตามภูมิภาค

7.2.5.อภิปรายผลเรื่องการจัดการ topic และ partition ของ Kafka

ระบบปัจจุบันได้ออกแบบ topic กับ partition ดังนี้คือจำนวน topic ขึ้นกับจำนวนโมเดล อุปกรณ์การแพทย์ และจำนวน partition คือ 1 ซึ่งการออกแบบปัจจุบันนั้นไม่ได้ใช้ความสามารถของ kafka ได้เต็มที่ และความสัมพันธ์ในการส่งข้อมูลของอุปกรณ์การแพทย์นั้นแตกต่างกัน นอกจากนี้บางรุ่นอุปกรณ์ไม่ได้ถูกนำมาใช้งานกับผู้ป่วยจำนวนมาก จากที่กล่าวมานั้นจะต้องมีบาง topic ที่รับความถี่ของข้อมูลที่สูงมากกว่า topic อื่นๆ ซึ่งในส่วนนี้จะทำให้เกิดปัญหา message imbalance ดังนั้นจึงเสนอวิธีการพัฒนาประสิทธิภาพการทำงานของระบบ ดังนี้ ใช้ 1 topic ในการรับข้อมูลจากอุปกรณ์การแพทย์ โดยจะมีการตั้งค่าจำนวน default partitions เป็น 5 ซึ่งสามารถเพิ่มจำนวน partition ได้ในอนาคตโดยไม่ต้องปิดระบบ ผู้ดูแลสามารถตั้งค่าอุปกรณ์ทางการแพทย์และ partition ที่ต้องการข้อมูลผ่าน key โดยจะมี dashboard สำหรับผู้ดูแลในส่วนของการขอ key เพื่อใช้งาน partition นั้นๆ โดยผู้ดูแลสามารถเลือก partition ได้ตามความเหมาะสมจาก dashboard ที่แสดงค่าเฉลี่ยความถี่ของข้อความของ partition นั้นๆ และการลงทะเบียนเพื่อรับ key นั้นจำเป็นต้องใส่ค่าความถี่ของข้อความของอุปกรณ์นั้นๆ ด้วย จากนั้นระบบจะมีการอัปเดตค่าความถี่ของข้อความให้หลังจากลงทะเบียนอุปกรณ์นั้นๆ นอกจากนี้ในส่วนนี้จำเป็นต้อง scale จำนวน web service ตามจำนวนข้อมูลขาเข้าที่เพิ่มขึ้น

7.2.6.อภิปรายผลเรื่องการตั้งค่า replication factor

จากการศึกษาพบว่าควรตั้งค่า replication factor ไม่มากเกินไปจนเกินจำนวน broker ใน cluster และไม่ควรมากเกินไป เนื่องจากถ้าทุก instance มี copy partition จำนวนมาก จะทำให้ instance ทุกเครื่องนั้นต้องบันทึก message ลง partition และทำให้เก็บ message ที่ซ้ำซ้อนมากเกินไปจนความจำเป็น นอกจากนี้การกระจาย message จาก partition ที่เป็น leader ไปยัง partition ที่เป็น follower ทั้งหมดต้องใช้เวลานานซึ่งในส่วนนี้จะกระทบต่อเวลาในการส่ง acknowledge ไปยัง producer จากการศึกษาศา Apache Kafka เสนอให้ใช้ค่า replication factor เป็น 3 และมีจำนวน broker ไม่ต่ำกว่า 3 เครื่อง ซึ่งการกระจาย replicated partition นั้น zookeeper จะกระจาย partition ไปยัง broker ต่างๆให้โดยอัตโนมัติ ซึ่งในส่วนนี้จะมั่นใจได้ว่า replicated partition จะไม่อยู่ใน broker เดียว[29]

7.2.7.อภิปรายผลเรื่องการตั้งค่า log retention ของ Kafka

จาก business requirement ระบบจะมีการเก็บข้อมูลย้อนหลังให้เป็นเวลา 7 วัน และความถี่ในการส่งข้อมูลคือ 80 messages/วินาที จากการศึกษาศา Kafka จะมีการเก็บข้อมูลโดยใช้ไฟล์ 3 ไฟล์ นั้นคือ ไฟล์ที่เก็บ message, ไฟล์ที่เก็บ index และ ไฟล์ที่เก็บความสัมพันธ์ระหว่าง timestamp กับ index โดยไฟล์ที่เก็บ message นั้น Kafka จะใช้พื้นที่ 93 bytes ในการเก็บ message header และขนาดข้อมูลจากอุปกรณ์ทางการแพทย์จะมีขนาด 48 bytes ซึ่งได้เผื่อขนาดเป็น 2 เท่าจากขนาดข้อมูลโดยเฉลี่ย ดังนั้น message จะมีขนาดเป็น 141 bytes เพราะฉะนั้นการเก็บข้อมูลย้อนหลัง 7 วัน และความถี่ของการส่งข้อมูลคือ 80 messages/วินาที นั้นจะต้องใช้พื้นที่ในการเก็บเป็น 48384000 bytes ซึ่งมีขนาดเท่ากับ 48.384 MB นอกจากนี้ไฟล์ที่เก็บ index และไฟล์ที่เก็บความสัมพันธ์ระหว่าง timestamp กับ index ขนาดจะมีความสัมพันธ์กับไฟล์ที่เก็บ messages โดยมีความสัมพันธ์เป็นดังนี้ ถ้าไฟล์ที่เก็บ messages มีขนาด 1 GB นั้นไฟล์ที่เก็บ index จะมีขนาด 2 MB และไฟล์ที่เก็บความสัมพันธ์ระหว่าง timestamp กับ index จะมีขนาด 3 MB จากการคำนวณจะเห็นได้ว่าการเก็บข้อมูลย้อนหลัง 7 วัน โดยความถี่ของการส่งข้อมูลอยู่ที่ 80 messages/วินาที นั้นใช้พื้นที่เก็บ message น้อยมากเมื่อเปรียบเทียบกับ instance ที่มีพื้นที่จัดเก็บข้อมูลเป็น 100 GB ซึ่งเป็นขนาดพื้นที่ที่ใช้ในการ deploy kafka cluster ดังนั้นเรื่องการตั้งค่าตัวแปร log.retention.hours เป็น 168 hours ซึ่งเป็นเวลาเท่ากับ 7 วัน นั้น จะมั่นใจได้ว่า Kafka cluster จะมีพื้นที่ในการเก็บ log เพียงพอนั่นเอง

7.2.8.อภิปรายผลเรื่องการประสานการทำงานระหว่าง Kafka กับระบบภายนอก

นอกเหนือจากการที่ให้ผู้เข้ามา subscribe ระบบ ในอนาคตอาจนำ Kafka ในระบบปัจจุบันไปประสานโดยตรงกับระบบของผู้ใช้โดยใช้ Kafka connector ซึ่งสามารถเชื่อมต่อกับระบบปัจจุบันกับระบบภายนอกได้หลากหลาย เช่น database, data lake, messaging systems เป็นต้น ซึ่งสามารถจัดการสิทธิการเข้าถึงข้อมูลของระบบภายนอกแต่ละระบบได้โดยใช้ Apache Ranger ที่มีรูปแบบการจัดการสิทธิการเข้าถึงข้อมูลแบบเข้าสู่ศูนย์กลาง และเป็นแบบ role-base คือ สามารถให้ role กับผู้ใช้แต่ละคน

และให้สิทธิการเข้าถึงข้อมูลต่างกัน นอกจากสามารถลบหรือแก้ไขสิทธิต่าง ๆ ได้ในอนาคตผ่านทาง web interface ที่ช่วยอำนวยความสะดวกในการใช้งาน

7.3. ข้อจำกัด

7.3.1.ระบบปัจจุบันสามารถจัดการกับการส่งข้อมูลแบบ real-time ได้ด้วยข้อมูลขนาดมากกว่าความต้องการในปัจจุบัน 2 เท่า โดยถ้าในอนาคตมีความต้องการเพิ่มขึ้นจนเกินกว่าที่ตั้งไว้จะต้องมีการปรับ algorithm ในการส่งข้อมูลแบบ real time

7.3.2.ระบบปัจจุบันนักพัฒนาไม่สามารถสมัครบัญชีเพื่อใช้งานด้วยตนเองได้ ต้องติดต่อแอดมินเพื่อสร้างบัญชี

7.4. แนวทางในการต่อยอดในอนาคต

7.4.1.ระบบปัจจุบันรองรับการรับข้อมูลแบบ MQTT และ Kafka ในอนาคตควรพัฒนา REST api เพื่อรองรับอุปกรณ์ที่หลากหลายมากขึ้น

7.4.2.ระบบปัจจุบันมีการทำ docker file เพื่อนำไป deploy บน cloud ในอนาคตควรทำระบบ CI/CD ในการส่งมอบระบบ เพื่อให้ง่ายต่อการ deploy มากขึ้น

7.4.3.ระบบปัจจุบันจะมีการ monitor webservice โดยในอนาคตควรมีการ monitor kafka cluster

7.4.4.ระบบปัจจุบันมีการใช้ instance 1 เครื่องในการ run MQTT server โดยในอนาคตควรทำ MQTT cluster เพื่อเพิ่ม availability และ reliability

7.4.5.ระบบปัจจุบันสามารถวิเคราะห์ข้อมูลจาก dashboard เพียงอย่างเดียว ในอนาคตควรนำข้อมูลไปทำ Machine learning model

7.4.6.ระบบปัจจุบันยังไม่มี integrate health tag เข้าในระบบ ในอนาคตควรเพิ่มเพื่อจัดการกับ user credential

8. เอกสารอ้างอิง

- [1] Google Cloud. (ม.ป.ป.). Google Cloud Pricing Calculator. สืบค้นเมื่อ 3 พฤศจิกายน 2565. จาก. <https://cloud.google.com/products/calculator/#id=05c923fb-db28-416f-afdd-527421f6cc5f>
- [2] Hansa Manakitsomboon. (2565). Number of hospitals in Thailand from 2011 to 2020. สืบค้นเมื่อ 3 พฤศจิกายน 2565. จาก. <https://www.statista.com/statistics/1114483/thailand-number-of-hospitals/>
- [3] HealthTAG. (2565). HealthTAG. สืบค้นเมื่อ 30 ตุลาคม 2565. จาก. <https://healthtag.io/th>
- [4] อภิรักษ์ คงคาเพชร. (2560). ความหมาย ความเป็นมา แนวคิด ทฤษฎี ในการคุ้มครองผู้บริโภคที่เกี่ยวข้องกับเครื่องมือแพทย์ชนิดต้านมะเร็งชนิดใช้ฝังในร่างกาย. สืบค้นเมื่อ 3 พฤศจิกายน 2565. จาก. <http://dspace.spu.ac.th/bitstream/123456789/5397/10/10.%E0%B8%9A%E0%B8%97%E0%B8%97%E0%B8%B5%E0%B9%88%202.pdf>
- [5] TIBCO. (ม.ป.ป.). What is Structured Data? สืบค้นเมื่อ 29 ตุลาคม 2565. จาก. <https://www.tibco.com/reference-center/what-is-structured-data>
- [6] Mongo. (ม.ป.ป.). Unstructured Data. สืบค้นเมื่อ 29 ตุลาคม 2565. จาก. <https://www.mongodb.com/unstructured-data>
- [7] Pethuru Raj. (2561). A Deep Dive into NoSQL Databases: The Use Cases and Applications. สืบค้นเมื่อ 3 พฤศจิกายน 2565. จาก. <https://www.sciencedirect.com/topics/computer-science/messaging-system>
- [8] Kafka. (ม.ป.ป.). Introduction. สืบค้นเมื่อ 28 ตุลาคม 2565. จาก. <https://kafka.apache.org/intro>
- [9] Sonic Automation. (2563). MQTT กับงาน Industrial Internet of Things (IoT). สืบค้นเมื่อ 28 ตุลาคม 2565. จาก. [https://sonicautomation.co.th/mqtt-for-iiot-application/#:~:text=MQTT%20\(Message%20Queueing%20Telemetry%20Transport,MQTT%20%E0%B8%96%E0%B8%B9%E0%B8%81%E0%B8%9E%E0%B8%B1%E0%B8%92%E0%B8%99%E0%B8%B2%E0%B8%82%E0%B8%B6%E0%B9%89%E0%B8%99%E0%B8%A1%E0%B8%B2](https://sonicautomation.co.th/mqtt-for-iiot-application/#:~:text=MQTT%20(Message%20Queueing%20Telemetry%20Transport,MQTT%20%E0%B8%96%E0%B8%B9%E0%B8%81%E0%B8%9E%E0%B8%B1%E0%B8%92%E0%B8%99%E0%B8%B2%E0%B8%82%E0%B8%B6%E0%B9%89%E0%B8%99%E0%B8%A1%E0%B8%B2)
- [10] Confluent. (ม.ป.ป.). Kafka Connect. สืบค้นเมื่อ 28 ตุลาคม 2565. จาก. <https://docs.confluent.io/platform/current/connect/index.html#kafka-connect>
- [11] Team Cleo. (ม.ป.ป.). What Are Web Services? Easy-to-Learn Concepts with Examples. สืบค้นเมื่อ 3 พฤศจิกายน 2565. จาก. <https://www.cleo.com/blog/knowledge-base-web-services>
- [12] <https://medium.com/@Biewz/ความหมายแตกต่างระหว่าง-web-application-และ-web-service-aa12b992d8ae>

- [13] RedHat. (2563). What is a REST API? สืบค้นเมื่อ 2 พฤศจิกายน 2565. จาก.
<https://www.redhat.com/en/topics/api/what-is-a-rest-api>
- [14] Google Cloud. (ม.ป.ป.). What is a Data Lake? สืบค้นเมื่อ 2 พฤศจิกายน 2565. จาก.
<https://cloud.google.com/learn/what-is-a-data-lake>
- [15] Oracle. (ม.ป.ป.). What Is a Data Warehouse? สืบค้นเมื่อ 2 พฤศจิกายน 2565. จาก.
<https://www.oracle.com/database/what-is-a-data-warehouse/>
- [16] GO. (ม.ป.ป.). Case studies. สืบค้นเมื่อ 3 พฤศจิกายน 2565. จาก. <https://go.dev/solutions/#>
- [17] MongoDB. (ม.ป.ป.). Why Use MongoDB and When to Use It? สืบค้นเมื่อ 30 ตุลาคม 2565. จาก.
<https://www.mongodb.com/why-use-mongodb>
- [18] Segmentio. (2566). Kafka-go. สืบค้นเมื่อ 19 มกราคม 2566. จาก.
<https://pkg.go.dev/github.com/segmentio/kafka-go>
- [19] Elastic. (ม.ป.ป.). What is Kibana? สืบค้นเมื่อ 25 ตุลาคม 2565. จาก. <https://www.elastic.co/what-is/kibana>
- [20] Elastic. (ม.ป.ป.). Elasticsearch. สืบค้นเมื่อ 25 ตุลาคม 2565. จาก.
<https://www.elastic.co/elasticsearch/>
- [21] Olga Weis. (2564). Differences between Google Cloud Storage and Amazon S3. สืบค้นเมื่อ 1 พฤศจิกายน 2565. จาก. <https://cloudmounter.net/amazon-s3-vs-google-cloud-storage/>
- [22] Google Cloud. (ม.ป.ป.). BigQuery Data Transfer Service API Client Libraries. สืบค้นเมื่อ 26 ตุลาคม 2565. จาก.
<https://cloud.google.com/bigquery/docs/reference/datatransfer/libraries#client-libraries-install-go>
- [23] Techletters. (2564). MQTT and Kafka. สืบค้นเมื่อ 15 มกราคม 2566. จาก.
<https://medium.com/python-point/mqtt-and-kafka-8e470eff606b>
- [24] Suraj Surve. (2564). Why You Should Use React.js For Web Development. สืบค้นเมื่อ 20 มกราคม 2566. จาก. <https://www.freecodecamp.org/news/why-use-react-for-web-development/>
- [25] Codemotion. (2565). Why You Should Use Typescript for Your Next Project. สืบค้นเมื่อ 20 มกราคม 2566. จาก. <https://www.codemotion.com/magazine/backend/why-you-should-use-typescript-for-your-next-project>

[26] JEST. (ม.ป.ป.). JEST. สืบค้นเมื่อ 25 มีนาคม 2566. จาก.

<https://jestjs.io/#:~:text=Jest%20is%20a%20JavaScript%20testing,extended%20to%20match%20your%20requirements>

[27] Grafana Labs. (ม.ป.ป.). What is K6?. สืบค้นเมื่อ 25 มีนาคม 2566. จาก.

<https://k6.io/docs/#:~:text=Using%20k6%2C%20you%20can%20test,Grafana%20Labs%20and%20the%20community>

[28] Selenium. (ม.ป.ป.). Selenium automates browsers. สืบค้นเมื่อ 25 มีนาคม 2566. จาก.

<https://www.selenium.dev/>

[29] Main concepts and Terminology สืบค้นเมื่อ 25 มีนาคม 2566. จาก.

https://kafka.apache.org/documentation/#intro_concepts_and_terms