ระบบประมวลผลข้อความผ่านช่องทางเอพีไอ

Message Processing API

แคทลียา อเนกจินต์

Kathaleeya Anakjin

ธานี จรัสตระกูล

Thanee Charattrakool

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมสารสนเทศ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2563

Message Processing API

Kathaleeya Anakjin

Thanee Charattrakool

THIS THESIS IS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INFORMATION ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2020

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

.....

หัวข้อปริญญานิพนธ์	ระบบประมวลผลข้อความผ่านช่องทางเอพีไอ
Project Title	Message Processing API
นักศึกษา	แคทลียา อเนกจินต์รหัสนักศึกษา 60010123
	ธานี จรัสตระกูล รหัสนักศึกษา 60010465
ระดับปริญญา	วิศวกรรมศาสตรบัณฑิต
สาขาวิชา	วิศวกรรมสารสนเทศ
ภาควิชา	วิศวกรรมคอมพิวเตอร์
ปีการศึกษา	2563

ผศ. มยุรี เลิศเวชกุล

รศ.ดร.ชวลิต เบญจางคประเสริฐ

อาจารย์ที่ปรึกษาปริญญานิพนธ์

หัวข้อปริญญานิพนธ์	ระบบประมวลผลข้อความผ่านช่องทางเอพีไอ	
รายชื่อนักศึกษา	นางสาวแคทลียา อเนกจินต์	รหัสนักศึกษา 60010123
	นายธานี จรัสตระกูล	รหัสนักศึกษา 60010465
ปริญญา	วิศวกรรมศาสตรบัณฑิต	
สาขาวิชา	วิศวกรรมสารสนเทศ	
ปีการศึกษา	2563	
อาจารย์ที่ปรึกษาปริญญานิพนธ์	ผศ. มยุรี เลิศเวชกุล	
	รศ. ดร. ชวลิต เบญจางคประเสริฐ	

บทคัดย่อ

ปริญญานิพนธ์เรื่องระบบประมวลผลข้อความผ่านช่องทางเอพีไอ เกิดจากแนวคิดและปัญหาที่ว่านักพัฒนาส่วนใหญ่มักมีเวลาในการพัฒนาอย่างจำกัด ซึ่งการพัฒนาฟีเจอร์บางอย่างที่มีอยู่แล้วนั้นเป็นสิ่งที่ทำให้เสียเวลาเป็นอย่างมาก แล้วทำไมนักพัฒนาจะต้องพัฒนาสิ่งที่มีอยู่ขึ้นมาซ้ำ ในเมื่อนักพัฒนาสามารถนำสิ่งที่มีอยู่มาใช้งานได้เลยทันที ดังนั้นจึงเกิดแนวคิดที่โครงงานนี้ขึ้นมาเพื่อแก้ไขปัญหาที่กล่าวมาข้างต้น จึงต้องมี Open API คืออินเตอร์เฟสที่เผยแพร่ต่อสาธารณะ และมีฟีเจอร์ที่พร้อมใช้งาน ออกแบบโครงสร้างของอินเตอร์เฟสนี้ให้มีความยืดหยุ่นรองรับการทำงานทั้งแบบ Synchronous และ Asynchronous เพื่อลดระยะเวลาในการอิมพลีเมนท์ ไม่ต้องเสียเวลาให้ API Gateway ทำความรู้จักกับฟีเจอร์ที่พัฒนาขึ้นมาใหม่ทุกตัว

Thesis Title	Message Processing API		
Student	Kathaleeya Anakjin	Student ID	60010123
	Thanee Charattrakool	Student ID	60010465
Degree	Bachelor of Engineering		
Department	Computer Engineering		
Major	Information Engineering		
Academic Year	2020		
Thesis Advisor	Asst.Prof. Mayuree Lertwatechakul		
	Assoc.Prof.Dr. Chawalit Benjangkaprasert		

ABSTRACT

This project presents about Message Processing System via API Channel. It consist of concept and problem that mostly the developers have limited development time. It is waste time to develop some of the features that are already there. So what for developers have to re-develop the existing stuff. He should be able to implement immediately what he has already had. Therefore, the idea of this project came up was to solve the above mentioned problems, Open API is a publicly available interface. It has a features available, The structure of this interface is flexible, supporting both synchronous and asynchronous, to reduce the lead time. Don't waste time getting API Gateway to know all the newly develop features.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จได้ด้วยดีต้องขอขอบพระคุณอาจารที่ปรึกษา ผศ. มยุรี เลิศเวชกุล และรศ.ดร. ชวลิต เบญจางคประเสริฐ คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่มอบความกรุณาอย่างสูงในการสนับสนุนและให้ความช่วยเหลือทั้งคำปรึกษาและคำแนะนำทางด้านวิชาการ รวมทั้งปรับปรุงแก้ไขข้อบกพร่องต่าง ๆ อันเป็นประโยชน์อย่างยิ่งในการทำปริญญานิพนธ์ด้วยความตั้งใจจริงและความเอาใจใส่ รวมถึงคอยให้กำลังใจแก่ผู้ทำวิจัยเสมอ ผู้ทำปริญญานิพนธ์ตระหนักถึงความเมตตานั้นและกราบขอขอบพระคุณเป็นอย่างสูงที่ให้ความรู้และประสบการณ์ที่ดีแก่ข้าพเจ้า

ขอขอบพระคุณอาจารย์ รุ่นพี่และเพื่อน ๆ ท่านอื่น ๆ ที่ไม่ได้กล่าวนาม ที่มีส่วนร่วมเกี่ยวข้องในการช่วยเหลือ คอยให้คำปรึกษาในเรื่องการทำงาน การแก้ไขปัญหาต่าง ๆ ทั้งทางตรงและทางอ้อมตั้งแต่เริ่มทำจนสำเร็จปริญญานิพนธ์ฉบับนี้ไปได้ด้วยดี ขอบพระคุณไว้ ณ ที่นี้

สำหรับคุณงามความดีอันใดที่เกิดจากปริญญานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดามารดา ซึ่งเป็นที่รักและเคารพยิ่งตลอดจนถึงครูอาจารย์ที่เคารพทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้และถ่ายทอดประสบการณ์ที่ดีให้แก่ข้าพเจ้า

แคทลียา อเนกจินต์
ธานี จรัสตระกูล
วิศวกรรมสารสนเทศ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

หน้า

สารบัญรูป (ต่อ)

หน้า

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

1.1 แนวคิดและที่มาของปัญหา

ในปัจจุบันนักพัฒนานั้นเป็นที่ต้องการของตลาดแต่ผู้พัฒนายังคงต้องประสบปัญหามากมายในการพัฒนาแอปพลิเคชัน (Application) ขึ้นมาคือ เวลาในการพัฒนาที่มีอย่างจำกัด ซึ่งในบางเซอร์วิส (Service) หรือฟีเจอร์ (feature) บางอย่างเป็นสิ่งที่มีอยู่แล้วหากนักพัฒนาจะต้องสร้างขึ้นมาใหม่ก็จะทำให้เสียเวลาอย่างมาก ซึ่งเซอร์วิสหรือฟีเจอร์บางอย่างจึงควรที่จะมีให้นักพัฒนาสามารถเข้าถึงและนำไปใช้งานได้เลย ดังนั้นปัญหาเหล่านี้จึงทำให้เกิดสิ่งที่เราเรียกว่า Open API ขึ้นมา Open API คือ อินเตอร์เฟส (Interface) ที่เผยแพร่ต่อสาธารณะซึ่งมีฟีเจอร์ที่พร้อมใช้งาน ช่วยให้นักพัฒนาซอฟต์แวร์ (Software) สามารถเข้าถึงและนำไปใช้งานได้ และยังรวมไปถึงปัญหาเวลาอิมพลีเมนท์ (Implement) เซอร์วิสขึ้นมาใหม่ต้องทำให้ API Gateway รู้จักกับกับเซอร์วิสที่เพิ่มเข้ามาใหม่จึงเป็นการเสียเวลาแก้ไข เวลาพัฒนาฟีเจอร์ต้องดูทำให้ Open API รองรับการทำงานแบบ Synchronous และ Asynchronous ดังนั้นเราจึงพยายามออกแบบโครงสร้าง (Architecture) ของระบบนี้ให้มีความยืดหยุ่น และไม่ยึดติดอยู่กับสิ่งใดในการใช้งาน ซึ่งทำให้เวลาพัฒนาฟีเจอร์ใหม่ ๆ ไม่จำเป็นต้องเข้าไปแก้ไขหรืออัปเดต (Update) เซอร์วิสเก่าที่เคยทำไว้

1.2 วัตถุประสงค์

1.2.1 เพื่อลดระยะเวลาในการพัฒนาแชตบอต (Chatbot)

1.2.2 เพื่อลดระยะเวลาในการพัฒนา Open API

1.2.3 เพื่อทำให้ Open API รองรับการทำงานทั้งรูปแบบ Synchronous และ Asynchronous

1.2.4 เพื่อเพิ่มความสะดวกสบายในการพัฒนา Open API

1.2.5 เพื่อทำให้ซอฟต์แวร์ที่ได้มีการใช้งานทรัพยากรที่ต่ำ

1.3 ขอบเขตของปริญญานิพนธ์

1.3.1 มีระบบแชตบอต

1.3.2 มีระบบตรวจสอบผลสลากกินแบ่งรัฐบาล

1.3.3 มี Open API รองรับการใช้งานทุกฟีเจอร์ผ่านเอพีไอ (API)

1.3.4 ระบบสามารถรองรับการต่อเติมฟีเจอร์ภายหลังได้อย่างง่ายดายและมีความยืดหยุ่นสูง

1.4 ผลที่คาดว่าจะได้รับ

1.4.1 ระบบที่พัฒนาขึ้นช่วยทำให้นักพัฒนาแชตบอตคนอื่น ๆ สามารถพัฒนาได้อย่างสะดวกสบายและรวดเร็วขึ้น

1.4.2 สามารถนำการออกแบบนี้ไปพัฒนา Open API ที่มีความง่ายต่อการพัฒนาและมีความยืดหยุ่นสูง

1.5 อุปกรณ์ที่ต้องใช้

1.5.1 ฮาร์ดแวร์ (Hardware)

- เครื่องคอมพิวเตอร์สำหรับพัฒนาโปรแกรมที่มีการเชื่อมต่อเน็ตเวิร์ก จำนวน 1 เครื่อง
- เครื่องคอมพิวเตอร์สำหรับเป็นเซิร์ฟเวอร์ (Server) ระบบฐานข้อมูล จำนวน 1 เครื่อง

1.5.2 ซอฟต์แวร์ (Software)

- Virtual Studio Code
- Golang
- Google Cloud Platform

1.6 ขั้นตอนการดำเนินงาน

ตารางที่ 1.1 แผนการดำเนินงาน

ID	หัวข้อ	2563							2564		
		Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar
1	ค้นหาหัวข้อโครงงาน	←									
2	ศึกษาหัวข้อโครงงาน		↔								
3	วิเคราะห์ระบบ			↔							
4	ออกแบบระบบ				↔						
5	พัฒนาระบบ					↔	↔	→			
6	ทดสอบระบบ							↔	→		

7	แก้ไขและปรับปรุงระบบ							←	—	→	
8	เขียนเอกสารโครงงาน					←	—	—	—	—	→

บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

2.1 ทฤษฎีเกี่ยวกับภาษาคอมพิวเตอร์

2.1.1 Machine Language

Machine Language คือภาษาเครื่อง เป็นรูปแบบพื้นฐานของบิตที่เครื่องคอมพิวเตอร์ได้รับการออกแบบให้รู้จักว่า เป็นคำสั่งและข้อมูลหรือคำสั่งที่ใช้ในเครื่อง มีความหมายเหมือน Computer Code ภาษาเครื่องเป็นภาษาระดับต่ำที่สุด เพราะใช้เลขฐานสองแทนข้อมูล และคำสั่งต่าง ๆ ทั้งหมด จะเป็นภาษาที่ขึ้นอยู่กับชนิดของเครื่องคอมพิวเตอร์ หรือหน่วยประมวลผลที่ใช้ นั่นคือแต่ละเครื่องก็จะมีรูปแบบของคำสั่งเฉพาะของตนเอง การเขียนโปรแกรมในสมัยก่อนนั้นมีความยุ่งยากเป็นอย่างมาก นักคำนวณและนักเขียนโปรแกรมจะต้องรู้จักวิธีการรวมตัวเลขเพื่อไปใช้แทนคำสั่งต่าง ๆ ดังนั้นนักคอมพิวเตอร์จึงได้พัฒนาภาษาโปรแกรมขึ้นมาเรียกว่า ภาษาแอสเซมบลี (Assembly Language) เพื่อให้สามารถเขียนโปรแกรมได้อย่างง่ายดายขึ้น

ภาษาเครื่อง เป็นภาษาโปรแกรมคอมพิวเตอร์ระดับต่ำที่สุด จะเขียนคำสั่งและแทนข้อมูลต่าง ๆ ด้วยเลขฐานสอง (Binary Code) ทั้งหมด จึงทำให้คอมพิวเตอร์สามารถเข้าได้ทันทีว่าเราได้สั่งอะไรไปโดยไม่ต้องใช้โปรแกรมแปลภาษาแต่อย่างใด ซึ่งเลขฐานสองเทียบได้กับลักษณะของสัญญาณทางไฟฟ้าเข้ากับหลักการทำงานของเครื่องซึ่งเครื่องสามารถเข้าใจ และพร้อมที่จะทำงานตามคำสั่งได้ทันที ซึ่งเป็นการเขียนคำสั่งด้วยเลข 0 หรือ 1 ดังตัวอย่างคำสั่งภาษาเครื่อง ดังนี้

ตารางที่ 2.1 ตัวอย่างคำสั่งภาษาเครื่อง

ภาษาเครื่อง Machine Language	ความหมาย
0010 0000	โหลดข้อมูลจากหน่วยความจำ
0100 0000	ดำเนินการบวกข้อมูล
00110000	เก็บข้อมูลลงในหน่วยความจำ

รหัสโครงสร้างของแต่ละคำสั่งของภาษาเครื่องจะประกอบด้วยส่วนสำคัญ 2 ส่วน คือ

1.) รหัสบอกประเภทของคำสั่ง (Operation Code หรือ Op - Code) เป็นส่วนที่บอกคำสั่งให้เครื่องทำการประมวลผล เช่นให้ทำการบวก ลบ คูณ หาร หรือเปรียบเทียบ

2.) รหัสบอกตำแหน่งข้อมูล (Operand) เป็นส่วนที่บอกว่าข้อมูลที่จะนำมาประมวลผลนั้นเก็บอยู่

ในตำแหน่ง (Address) ใดของหน่วยความจำ

ข้อดีของภาษาเครื่อง คือสามารถเขียนโปรแกรมควบคุมการทำงานคอมพิวเตอร์ได้โดยตรง และสั่งงานให้คอมพิวเตอร์ทำงานได้อย่างรวดเร็ว

2.1.2 Compiler

Compiler เปรียบเสมือนโปรแกรมแปลภาษา เป็นโปรแกรมคอมพิวเตอร์ที่ทำหน้าที่แปลงชุดคำสั่งภาษาคอมพิวเตอร์หนึ่งไปเป็นชุดคำสั่งที่มีความหมายเดียวกันในภาษาคอมพิวเตอร์อื่น หรือมันคือโปรแกรมที่เอาไว้แปลภาษาระดับสูงให้กลายเป็นภาษาเครื่องที่คอมพิวเตอร์เข้าใจได้

เดิมการทำงานต่าง ๆ ของคอมพิวเตอร์จะทำงานได้ตามสิ่งที่มนุษย์ทำการป้อนคำสั่งเข้าไป ซึ่งคอมพิวเตอร์จะรู้จักเพียงแค่เลขฐานสอง หรือเลขบิต 0 และ 1 เท่านั้น ซึ่งชุดคำสั่งนี้ว่า Machine Code หรือ Machine Language คือภาษาเครื่อง ดังนั้นคำสั่งที่คอมพิวเตอร์อ่านรู้เรื่อง เช่น 00011101011 เป็นต้น ซึ่งมนุษย์จะอ่านไม่รู้เรื่องแน่นอน ทำให้คนสมัยก่อนคิดวิธีการสั่งคำสั่งคอมพิวเตอร์โดยที่ไม่พิมพ์ 0 และ 1 โดยใช้รหัสซึ่งเป็นตัวอักษามาแทนที่หรือเรียกว่านีโมนิค (Mnemonic) ตัวอย่างเช่น

ต้องการเพิ่มค่าให้ตัวแปร X อีก 5 หน่วย จากที่ต้องสั่งโดย

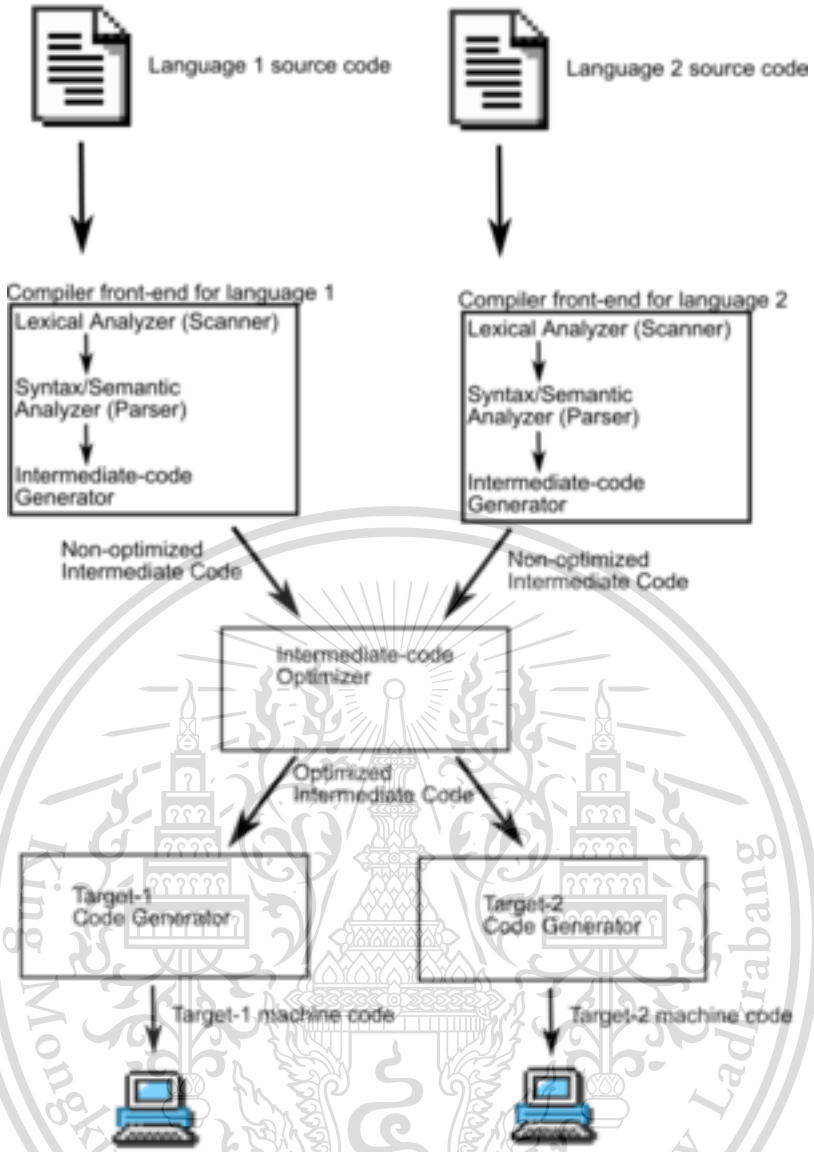
```
00011101011110101000001011111101000000110111101101011111111110011010
```

ก็จะเปลี่ยนเป็น

```
LOAD Register, X
ADD 5
STORE X, Register
```

ซึ่งการที่เครื่องคอมพิวเตอร์จะเข้าใจคำสั่งนีโมนิคนั้นก็จะต้องมีโปรแกรมมาแปลงคำสั่งให้กลายเป็นภาษาเครื่อง (Machine Language) อยู่ดี ดังนั้นจึงเป็นที่มาของคอมไพเลอร์ Compiler

ขั้นตอนการทำงานของคอมไพเลอร์ คือต้องทำการตัดคำ หรือ Tokenizer ตัวอย่างเช่น ประโยคว่า “ฉันออกเดินทาง” ประโยคนี้ประกอบไปด้วย 3 คำหลัก ๆ คือ “ฉัน” “ออก” “เดินทาง” ถ้าหากไม่ตัดคำก็จะไม่มีทางแปลออก ซึ่งการตัดคำนั้นทำโดยใช้ Syntax ซึ่งการเขียนโปรแกรมจะมีสิ่งที่เรียกว่า Syntax หรือไวยากรณ์ของโปรแกรม



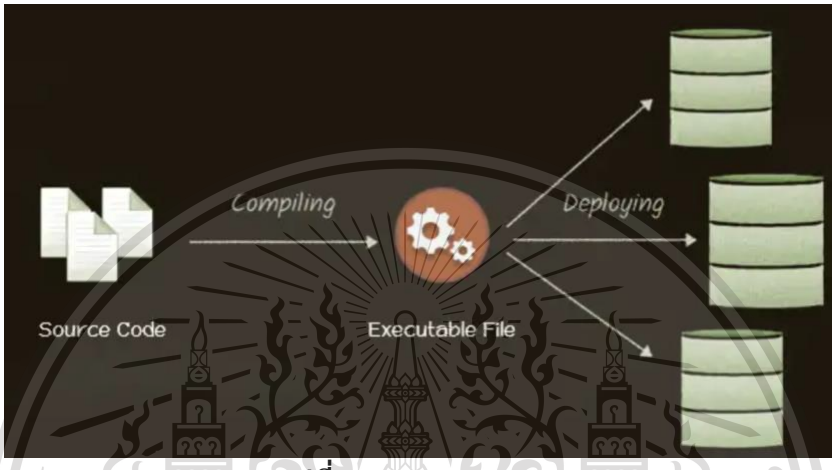
รูปที่ 1 ขั้นตอนการทำงานของตัวโปรแกรม

(**ที่มา:** https://images.app.goo.gl/41ooM8kTxtjNH6jv8)

2.1.3 Go

Go Language หรือ Golang เป็นภาษาโปรแกรมที่สร้างขึ้นโดย Rob pike, Ken Thompson และ Robert Griesemer จากบริษัท Google เกิดขึ้นมาตั้งแต่ปี 2006 โดยจุดประสงค์ที่ Google สร้างภาษานี้มาเพื่อแก้ไขปัญหาคอมไพล์ (Compiled) ช้าและเรื่องการรองรับมัลติคอร์โปรเซสเซอร์ (multi-core processor) ซึ่งเป็นการผสมผสานจุดเด่นของภาษา C++ และภาษา Python เข้าด้วยกันทำให้ภาษาโกสามารถคอมไพล์โปรแกรมที่มีความซับซ้อนได้ภายในเวลาอันรวดเร็ว และใช้ประสิทธิภาพของมัลติคอร์โปรเซสเซอร์ในการทำงานแบบขนานกันไปได้ อีกทั้งยังมีระบบจัดการหน่วยความจำที่แก้ปัญหา Garbage Collection

Golang เป็น static language คือการใช้ตัวแปรจะต้องมีการประกาศชนิดของข้อมูลและตัวแปรภาษาจะสามารถตรวจสอบชนิดข้อมูลได้ว่าเรากำหนดถูกต้องหรือไม่ตั้งแต่ช่วงของการคอมไพล์ อีกทั้งยังเป็นภาษาประเภท compiled language นั่นคือเราสามารถสั่ง compiler ให้อ่านซอร์สโค้ด (Source Code) แล้วสร้างผลลัพธ์ในรูปของไบนารี (Executable File) ที่สั่งให้ทำงานบนแพลตฟอร์ม (Platform) นั้น ๆ ได้เลยโดยไม่ทำงานอยู่บน VM หรือต้องติดตั้งรันไทม์ (Runtime) ทำให้ลดการใช้งานทรัพยากรคอมพิวเตอร์



รูปที่ 2 Executable File

(ที่มา: https://www.babelcoder.com/blog/articles/intro-to-golang)

2.1.4 JavaScript ES6

ES ย่อมาจากคำว่า ECMAScript คือมาตรฐานของภาษา JavaScript ที่กำหนดโดยองค์กร European Association for Standardizing Information and Communication Systems หรือ Ecma International เพื่อให้ภาษาสคริปต์ (Script) นั้นมีความเป็นสแตนดาส (Standard) และพัฒนาเรื่อยมาจนถึงปัจจุบัน

JavaScript เป็นภาษาประเภท Interpreted Language ซึ่งคือภาษาที่คอมพิวเตอร์จะประมวลผลและทำตามคำสั่งทีละบรรทัด ซึ่งข้อดีของภาษาประเภทนี้คือรายงานผลลัพธ์ได้ทันทีหากมีข้อผิดพลาดก็สามารถแก้ไขได้เลยทันที

JavaScript เป็นภาษาสคริปต์เชิงวัตถุที่เรียกกันว่า “สคริปต์” และเป็นภาษาโปรแกรมที่ใช้พัฒนาได้ทั้งฝั่งของ Front-end และ Back-end ซึ่งในฝั่งของ Front-end หรือทำงานในฝั่งของ Client นั้นสามารถใช้ร่วมกับภาษา HTML ในการสร้างและพัฒนาเว็บไซต์ (Web site) เพื่อให้เว็บไซต์มีการเคลื่อนไหวและสามารถตอบสนองผู้ใช้งานได้มากขึ้น และในฝั่งของ Back-end หรือทำงานในฝั่งของเซิร์ฟเวอร์ ซึ่งในโครงงานนี้จะใช้ Node.js เป็นรันไทม์

JavaScript รองรับการทำงานแบบ Asynchronous คือการทำตามคำสั่งของบรรทัดถัดไปทันทีโดยไม่ต้องหยุดรอจนกว่าคำสั่งจะเสร็จสมบูรณ์ ตัวอย่างคำสั่งแบบ Asynchronous ใน JavaScript ได้แก่ คำสั่ง setTimeout

2.1.5 Typescript

TypeScript เป็นภาษาโปรแกรมที่รวมความสามารถที่ ES2015 เองมีอยู่สิ่งที่เพิ่มขึ้นมาคือสนับสนุน Type System รวมถึงคุณสมบัติอื่น ๆ ที่เพิ่มมากขึ้น เช่น Enum และความสามารถที่เพิ่มขึ้นของการโปรแกรมเชิงวัตถุ TypeScript นั้นเป็น transpiler เหมือน Babel นั่นหมายความว่าตัวแปลภาษาของ TypeScript จะแปลโค้ดที่เราเขียนให้เป็น JavaScript อีกทีนึง จึงมั่นใจได้ว่าผลลัพธ์สุดท้ายจะสามารถใช้งานได้บนเว็บบราวเซอร์ (Browser) ทั่วไป Typescript สามารถระบุหรือไม่ระบุชนิดข้อมูลใน TypeScript ก็ได้ แต่เมื่อประกาศตัวแปรแล้วเราจะเปลี่ยนชนิดข้อมูลไม่ได้

2.1.5.1 ข้อดีของ Typescript

- Typescript สามารถทำให้ใช้ Javascript สมัยใหม่ได้ในปัจจุบัน โดย Typescript มีความสามารถของ ES2015 และอื่น ๆ รวมไว้อยู่แล้ว เช่น Typescript จะถูกแปลงเป็น Javascript ก็ต่อเมื่อถูกนำมาใช้งานจริง ๆ เพราะ Typescript นั้นมีคุณสมบัติของ Transpiler อยู่ ซึ่ง Transpiler ในที่นี้เสมือนตัวแปลง Javascript เวอร์ชัน (Version) ใหม่ให้กลายเป็นเวอร์ชันที่ NodeJS อ่านเข้าใจ

- Typescript จะเกิดข้อผิดพลาดน้องลงเนื่องจากตัวแปรที่ประกาศไว้แล้วใน Typescript จะเปลี่ยนชนิดข้อมูลไม่ได้

- Typescript จะมีช่วง compile time สำหรับการตรวจสอบโค้ดทำให้สามารถดักจับข้อผิดพลาดในโปรแกรมได้ตั้งแต่ต้นไม่ปล่อยให้ข้อผิดพลาดไปแสดงในตอนทำงานจริง

2.1.5.2 ชนิดข้อมูลพื้นฐานของ Typescript

Javascript มีชนิดข้อมูลอะไรใน Typescript ก็มีเช่นกัน ได้แก่ Boolean, Number และ String โดยเราสามารถกำกับชนิดข้อมูลให้ตัวแปรได้ด้วยการระบุชนิดข้อมูลที่ต้องการ ทั้งนี้ถ้าเรากำหนดชนิดข้อมูลให้กับตัวแปรผิดหรือไม่ตรง Typescript สามารถละชนิดข้อมูลที่ประกาศไปได้ Typescript จะอนุมานหรือคาดคะเนชนิดข้อมูลจากค่าของข้อมูลที่ระบุไว้

2.1.6 Node.js

Node.js คือ Cross Platform Runtime Environment สำหรับฝั่งเซิร์ฟเวอร์ที่เขียนด้วยภาษา JavaScript หรือก็คือ JavaScript Runtime โดยมีการรันด้วย Chrome's V8 JavaScript engine ในตอนเริ่มต้น V8 ถูกพัฒนาโดย Google เป็น Open source สามารถรัน JavaScript ได้อย่างรวดเร็ว ต่อมาจึงได้ทำเป็น Server Interpreter

Node.js มีส่วนเสริม (module/library/plugin/package) มากมายในคำสั่ง npm (Node Package Manager) ประโยชน์ก็คือสามารถติดตั้งและเป็นตัวจัดการ package เสริมต่าง ๆ ได้ในคำสั่งเดียว

2.2 ทฤษฎีเกี่ยวกับระบบฐานข้อมูล

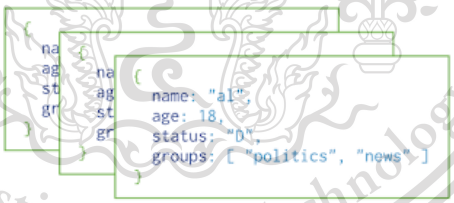
2.2.1 MongoDB

MongoDB เป็น Open Source Document Database โดยเป็นฐานข้อมูลแบบ NoSQL คือไม่มี Relation หรือความสัมพันธ์ของตารางแบบ SQL ทั่ว ๆ ไปแต่จะเก็บข้อมูลแบบ JSON (JavaScript Object Notation) แทนการบันทึกข้อมูลแบบทุก ๆ record ใน MongoDB จะเรียกว่า Document ซึ่งจะเก็บค่าเป็น Field หรือ Key และ Value หรือก็คือแบบ JSON

รูปที่ 3 รูปแบบการเก็บข้อมูลของ MongoDB

(**ที่มา:** https://docs.mongodb.com/manual/core/document/)

การเก็บข้อมูล Document ใน MongoDB จะถูกเก็บไว้ใน Collections เปรียบเทียบได้กับ Table ใน Relational Database ทั่ว ๆ ไปแต่แตกต่างกันที่ Collection ไม่จำเป็นต้องมีสกีมา (Schema) เหมือนกันก็สามารถบันทึกข้อมูลได้



รูปที่ 4 Collection

(**ที่มา:** https://images.app.goo.gl/WYNuZ3q11HJbJVEA9)

2.2.2 Kafka

Kafka คือ distributed message queue โดยเริ่มแรก Kafka ถูกสร้างขึ้นโดย LinkedIn เป็น open sourced ในช่วงต้นปี 2011 และถูกเผยแพร่ต่ออย่างช้าๆ ผ่านทาง Apache Incubator ตั้งแต่ปี 2012 จากนั้นจึงได้แยกบริษัทออกมาจาก LinkedIn ก่อตั้งเป็น บริษัท Confluent เพื่อพัฒนา Kafka โดยเฉพาะ

Kafka เป็นแพลตฟอร์มที่ถูกใช้เป็นตัวกลางในการเชื่อมต่อ เพื่อช่วยแยกและกระจายข้อความ (message) การสื่อสารระหว่างระบบแต่ละตัว ไม่ว่าระบบอะไรต้องการข้อมูลจากไหน ก็มาเรียกใช้ใน Kafka ตัวนี้เพียงตัวเดียว โดยการกระจายข้อความมีหลักการหลัก ๆ 3 ข้อ

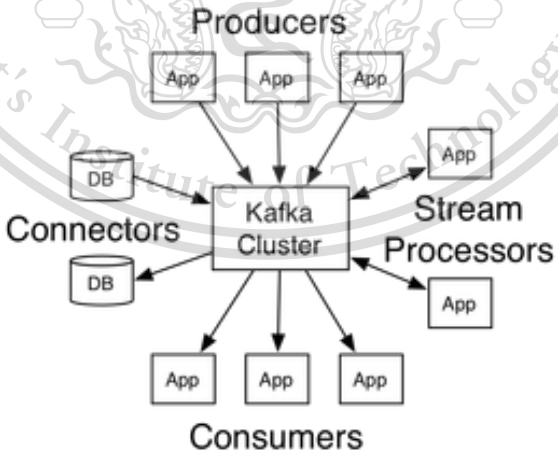
1. ทำการเผยแพร่ข้อมูลอย่างรวดเร็วทีละ records ในรูปแบบ message queue หรือระบบ messaging
2. ทนทานต่อความผิดปกติของสิ่งแวดล้อมรอบข้างด้วยระบบ cluster ทำให้มีโอกาสในการล่มน้อย
3. ทำงานตามการร้องขอของฝั่ง Client

2.2.2.1 จุดเด่นของ Kafka

จุดเด่นของ Kafka คือทำข้อมูลแบบ real-time pipline ซึ่งมีความเสถียรระหว่างตัวระบบและแอปพลิเคชันสามารถทำการแปลงข้อมูลแบบ real-time streaming application ไปเป็นแบบ streams data โดย Kafka ทำงานเป็น Cluster ซึ่ง Kafka cluster เก็บชุดข้อมูลลงในหมวดหมู่ที่เรียกว่า Topics ข้อมูลแต่ละ record จะประกอบไปด้วย key, value และ timestamp

2.2.2.2 Kafka Core API

Kafka Core API จะมีคือ Producer API ใช้สำหรับส่งข้อมูลเข้าถังของ kafka แต่ละ topics, Consumer API ใช้สำหรับให้ user เข้ามาดึงข้อมูลแต่ละ topics โดยจะส่งข้อมูลกลับไปทีละ records, Streams API สำหรับให้แอปพลิเคชันทำการดึงข้อมูล (consuming) จาก topics และส่งข้อมูลกลับมา (producing) และConnector API ใช้จัดการเชื่อต่อของ producers และ consumers ที่ต่อระหว่าง Kafka topics และแอปพลิเคชันภายนอก



รูปที่ 5 Kafka Core API

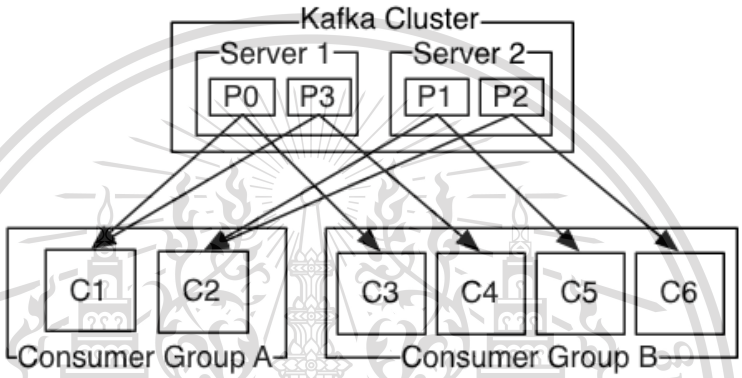
(**ที่มา:** https://saixiii.com/apache-kafka/)

2.2.2.3 Producers

Producers ทำหน้าที่ส่งข้อมูลให้กับ topics ที่กำหนดไว้ และต้องคอยแจกเลือก partition ภายใน topic เองด้วย ซึ่งจะปกติทำงานเป็นแบบ round-robin เพื่อความสมดุล หรือ อาจจะใช้ function อื่นในการกระจายก็ได้

2.2.2.4 Consumers

Consumers จะแทนตัวเองด้วย consumer group name โดยแต่ละ record ที่อยู่ใน topic จะถูกส่งให้เพียง 1 consumer ภายใต้ consumer group เท่านั้น เพราะฉะนั้นเราสามารถแบ่ง process ของ consumer ออกเป็นหลายเครื่องเพื่อช่วยในการทำงานได้ แต่ถ้า consumer groups ต่างกัน ทั้งหมด record ทั้งหมดจะถูก broadcast ให้กับ consumer ทั้งหมดเช่นกัน

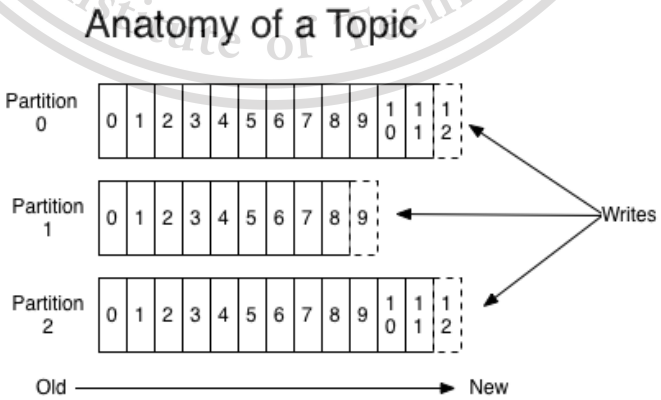


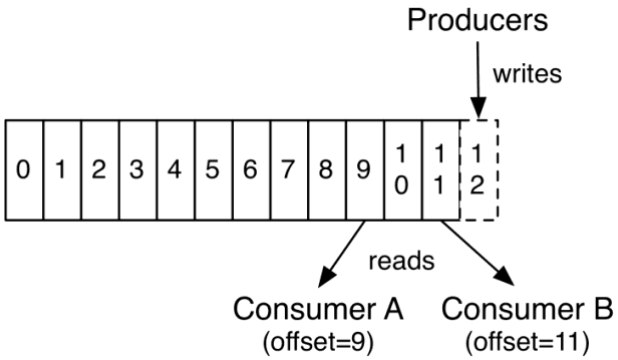
รูปที่ 6 Consumer Group

(**ที่มา:** https://saixiii.com/apache-kafka/)

2.2.2.5 Topics

Topics คือ หมวดหมู่หรือชื่อ group ของฐานข้อมูลที่จะเผยแพร่ โดยสามารถมีปลายทางได้หลายคน นั้นหมายถึงอาจจะมี cosumer ตั้งแต่ 0 หรือ หลายคนก็ได้





รูปที่ 7 อธิบายการเพิ่ม record

(**ที่มา:** https://saixiii.com/apache-kafka/)

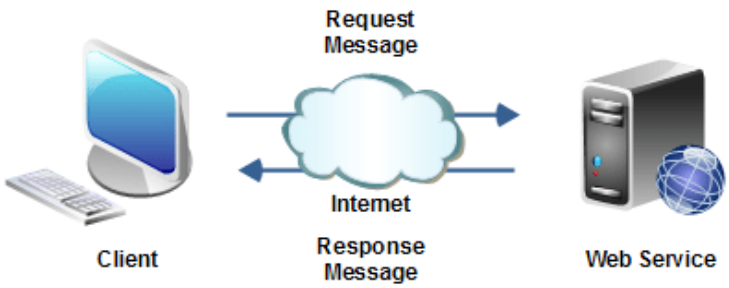
จะเห็นได้ว่าการทำงานของแต่ละ topics ฝั่ง producer ก็จะทยอยเพิ่ม record เข้าไปตาม sequence ส่วน Consumer แต่ละอันก็จะมี offset ของตัวเองที่จะค่อยเข้ามาดึงตาม sequence ของตนเอง

2.3 ทฤษฎีเกี่ยวกับ Web Service

Web service คือ บริการแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์ใน Internet ซึ่งในตอนแรกถูกออกแบบมาเพื่อใช้ระหว่าง human-to-machine ผ่าน Protocol HTTP (Hypertext Transfer Protocol) ต่อมาถูกพัฒนามาใช้กับ machine-to-machine

Web service เป็นมาตรฐานในการเชื่อมต่อ web-base-application ที่ทำงานอยู่บน Internet protocol โดยอาศัย REST, XML, SOAP, WSDL และ UDDI

การทำงานของ Web service ฝั่ง Client จะทำการโหลดข้อมูลต่าง ๆ ที่ใช้ในการติดต่อไปยังเซิร์ฟเวอร์เข้าสู่ SOAP message ส่ง SOAP message ไปยัง Web service ด้วยการทำ HTTP POST ฝั่ง Web service ก็จะทำการถอดข้อมูลออกจาก SOAP และเปลี่ยนเป็นคำสั่งที่แอปพลิเคชันเข้าใจ ในส่วน แอปพลิเคชันก็จะนำคำสั่งหรือข้อมูลที่ได้ไปทำงานจนได้ผลลัพธ์ ที่ต้องส่งกลับไปหาฝั่ง Client จัดใส่ใน HTTP format ฝั่ง Client ก็จะแกะข้อมูลออกจาก SOAP message



รูปที่ 8 รูปแบบการแลกเปลี่ยนข้อมูลระหว่าง Client และ Web service

(**ที่มา:** https://saixiii.com/what-is-webservice/)

จุดเด่นของ Web service คือมีมาตรฐานในการใช้งาน มีความยืดหยุ่นสามารถปรับเปลี่ยนตามความต้องการและใช้งานได้ ไม่ยึดติดกับ Operating System และภาษาโปรแกรมมิ่ง (Programming) ชนิดใด

2.3.1 RESTful

REST หรือ Representational State Transfer เกิดขึ้นมาปี 2000 เป็น “รูปแบบสถาปัตยกรรม” ที่ใช้ประโยชน์จากเทคโนโลยี Web Protocol เป็นมาตราฐานในการสร้าง Web Service รูปแบบหนึ่ง โดย REST ต้องทำให้ข้อมูลอยู่ในรูปแบบของ resource ส่วนการกระทำต่าง ๆ เป็นไปตาม HTTP Verb หรือ HTTP Method (GET, POST, PUT, DELETE) มีหน้าที่ดังนี้

- GET: ใช้ในเรียกข้อมูลเหมือน select ข้อมูลใน SQL
- POST: ใช้ในการส่งข้อมูลเพื่อทำการเพิ่มข้อมูล
- PUT: ใช้ในการส่งข้อมูลเพื่อทำการแก้ไขข้อมูลหรืออัปเดตข้อมูล
- DELETE: ใช้ในการลบข้อมูล

2.3.1.1 Architectural Constraints of RESTful API 6 ข้อกำหนดของ RESTful API ซึ่งถือเป็นสิ่งที่สำคัญในการสร้าง RESTful API ตามมาตรฐานซึ่งทำให้ง่ายต่อการพัฒนา

1. Client-server Architecture: Client ไม่จำเป็นต้องรู้อะไรเกี่ยวกับ Business logic ภายในไม่มีหน้าที่เกี่ยวกับการจัดเก็บข้อมูล ส่วนเซิร์ฟเวอร์มีหน้าที่เก็บทรัพยากร (Resource) และไม่จำเป็นต้องรู้อะไรเกี่ยวกับ UI Frontend หรือสถานะของผู้เรียก

2. Statelessness: ส่ง Request รับ Response จากเซิร์ฟเวอร์

3. Cacheability: สามารถ cache response ได้ การ Response จะต้องสามารถกำหนดได้ว่าจะ Cache หรือไม่ เพื่อป้องกันไม่ให้ User หรือ Client ได้รับข้อมูลเก่า

4. Layered System: ปกติ Client ไม่รู้ว่าที่ทำการเชื่อมต่อนั้น ได้เชื่อมต่อโดยตรงกับเซิร์ฟเวอร์ปลายทาง หรือไปยังตัวกลางอื่น ๆ ระหว่างทาง, เซิร์ฟเวอร์ตัวกลางควรสามารถปรับปรุงความสามารถในการขยายระบบได้ โดยการใช้งานการทำ Load balance

5. Code on Demand (optional): เซิร์ฟเวอร์สามารถขยายได้ชั่วคราว หรือปรับแต่งการทำงานของ Client ได้ ตัวอย่างเช่น ทำ client-side scripts ใน JavaScript

6. Uniform Interface: ถือเป็นข้อสำคัญจะที่แยกระหว่าง REST API และ Non-REST API มันแสดงให้เห็นถึงวิธีการที่จะคุยกับ เซิร์ฟเวอร์โดยไม่คำนึงถึงประเภทของอุปกรณ์ หรือประเภทของ Application Uniform Interface ได้แยกออกไปอีก 4 อย่าง

1.) Resource-Based เช่น API/Users

2.) Manipulation of Resources Through Representations: เช่น User get user_id หรือ Request list of users แล้วทำการ Delete หรือ Modify User

3.) Self-descriptive Messages: แต่ละ Message มีข้อมูลเพียงพอที่จะนำมาอธิบายวิธีการ Process message เพื่อให้เซิร์ฟเวอร์ทำการวิเคราะห์ได้ง่าย

4.) Hypermedia as the Engine of Application State (HATEOAS): จำเป็นต้องมี Links สำหรับทุก ๆ Response เพื่อให้ Client สามารถค้นหาได้ง่าย

2.3.2 MQTT API

MQTT ย่อมาจาก Message Queuing Telemetry Transport เป็นโพรโตคอล (Protocol) สำหรับใช้ในสื่อสารข้อมูลระหว่าง Machine to Machine (M2M) ถูกคิดค้นขึ้นในปี ค.ศ. 1999 โดย Dr Andy Stanford-Clark จาก IBM และ Arlen Nipper จาก Arcom (Now Eurotech) ออกแบบมาเพื่อใช้สื่อสารในระบบเครือข่ายที่มีทรัพยากรค่อนข้างจำกัด ใช้งานแบนด์วิดธ์ (Bandwidth) ต่ำ สามารถส่งข้อมูล (Publish) และรับข้อมูล (Subscribe) ระหว่างอุปกรณ์ เพื่อสื่อสารกันระหว่างอุปกรณ์

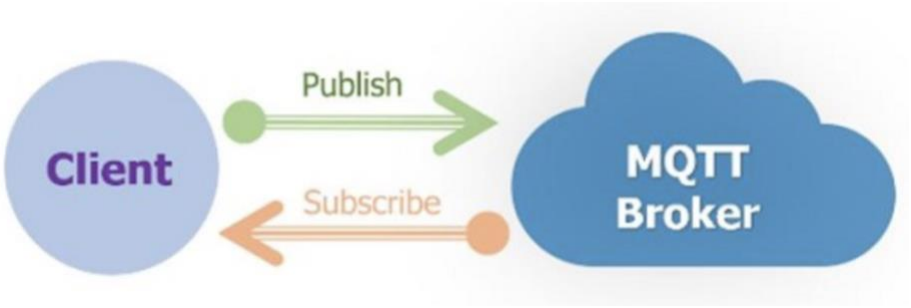
2.3.2.1 องค์ประกอบของ MQTT API

1. MQTT Client

เป็นส่วนส่งข้อมูล (Publish) ต่าง ๆ ขึ้นไปยัง MQTT Broker และสามารถรับข้อมูล (Subscribe) ต่าง ๆ จาก MQTT Broker ผ่านทาง TCP/IP Protocol

2. MQTT Broker หรือ MQTT Server

เป็นซอฟต์แวร์สำหรับรับข้อมูลจาก MQTT Client ที่ได้ส่งข้อมูล (Publish) เข้ามาและสามารถส่งข้อมูล (Publish) จาก MQTT Broker ไปยัง MQTT Client ที่ได้ รับข้อมูล (Subscribe) ไว้ได้



รูปที่ 9 องค์ประกอบของ MQTT API

(**ที่มา:** https://blog.thaieasyelec.com/introduction-to-mqtt/)

2.3.2.2 หลักการทำงานของ MQTT API

หลักการทำงานของ MQTT คือการการรับส่งข้อมูลระหว่าง Server (Broker) และ Client (Publisher/Subscriber) โดยการประกาศหัวข้อการรับส่งข้อมูลเรียกว่า Topic ไว้ใน Server (Broker) จากนั้น Publisher จะส่งข้อมูลไปยัง Topic นั้น ๆ และ Subscriber ก็จะได้รับข้อมูลทั้งหมดใน Topic นั้น ๆ เช่นกัน

2.3.3 HTTP และ HTTPS

HTTP (Hypertext Transport Protocol) คือโพรโตคอลสำหรับสื่อสารใช้เรียกโปรแกรมบนบราวเซอร์เพื่อเรียกดูข้อมูลต่าง ๆ ของเว็บนั้น ๆ หรือเรียกได้ว่าเป็นการถ่ายโอนข้อมูลผ่านบราวเซอร์ และจะใช้ HTTP เป็นตัวเรียกเซิร์ฟเวอร์ให้ส่งข้อมูลมาแสดงที่หน้าจอ โดยการส่งข้อมูลวิธีนี้จะไม่มีการเข้ารหัสข้อมูล ซึ่งอาจจะทำให้มีโอกาสเกิดการดักจับข้อมูลได้ง่าย

HTTPS (Hypertext Transfer Protocol over Secure Socket Layer หรือ Http over SSL) เป็นการถ่ายโอนหรือส่งข้อมูลระหว่างบราวเซอร์กับเซิร์ฟเวอร์ เช่นเดียวกับ HTTP แต่สำหรับ HTTPS นั้นจะมีการนำ SSL (Secure Socket Layer) และ TLS (Transport Layer Security) หรือการเข้ารหัสข้อมูล เข้ามาใช้ในการส่งข้อมูลด้วย ดังนั้นเมื่อมีการเข้ารหัสข้อมูลระหว่างการส่งข้อมูลนั้นก็ทำให้ไม่สามารถถูกดักจับข้อมูลระหว่างการส่งข้อมูลได้ หรือถูกดักจับไปก็ไม่สามารถเข้าดูข้อมูลได้ มันจะแสดงเป็นเพียงอักษร ตัวเลข ที่ไม่สามารถอ่านได้จะต้องทำการถอดรหัสก่อนจึงจะสามารถอ่านข้อความต่าง ๆ ได้ ซึ่งก็มีโอกาสเกิดขึ้นได้ยาก เพราะการถอดรหัสจะต้องใช้รหัสที่เหมาะสมและตรงกันเท่านั้น

2.4 ทฤษฎีเกี่ยวกับรูปแบบของข้อมูล

2.4.1 JSON

JSON ย่อมาจาก JavaScript Object Notation เป็นข้อมูลรูปแบบ text ที่มีรูปแบบที่จะเก็บข้อมูลแบบ Key และ Value ใช้ในการสร้างออบเจ็กต์ (Object) ขึ้นมาเพื่อส่งข้อมูลระหว่างแอปพลิเคชันหรือ Applications Program Interface (API)

```
'{"key": "value"}'
```

รูปที่ 10 รูปแบบการเก็บข้อมูลของ JSON

(**ที่มา:** https://www.borntodev.com/2020/02/28/what-is-json/)

2.4.1.1 ประเภทของข้อมูลที่ JSON สามารถเก็บได้

1. Number: ตัวเลข

2. String: Unicode ใช้เครื่องหมาย Double-Quote (“) เป็นตัวบ่งบอก และสามารถใช้ Backslash Syntax ได้

3. Boolean: True or False

4. Array: ชุดข้อมูล ซึ่งจะเป็นชนิดใดก็ได้ ใช้สัญลักษณ์ Square Bracket [var1,var2] เป็นตัวแสดง และคั่นด้วย Comma แต่ละค่าใน Array

5. Object: ชุดข้อมูลที่เป็นคู่ Key-Value แบบ Strings ใช้สัญลักษณ์ปีกกา {key1: value1, key2: value2} ใช้ Comma เป็นตัวแบ่งแต่ละคู่ และใช้ Colon เป็นตัวแบ่งระหว่าง Key และ Value

6. Null: ค่าว่าง

2.4.1.2 JSON Schema

JSON Schema ใช้สำหรับแสดง Format โครงสร้างของ JSON เพื่อทำ Validation, Documentation และ Interaction Control หรือคือการติดต่อไปยังแอปพลิเคชันที่เราจำเป็นต้องส่ง Request ที่ทางแอปพลิเคชันต้องการ ไปให้ครบถ้วน ซึ่ง Schema จะเป็นตัวบอกว่าข้อมูลต้องมีอะไรบ้าง ซึ่งใช้หลักการเดียวกับ XML Schema (XSD)

2.4.2 Protocol Buffer

Protocol Buffer หรือ Protobuf เป็นเครื่องมือกำหนดโครงของ Serialized Data จาก Google ที่สามารถสร้างโค้ดตัวอ่านข้อมูลส่วนตัวของเราได้ในภาษาเป้าหมาย เป็นวิธีการจัดลำดับข้อมูลที่สามารถส่งผ่านสายไฟหรือเก็บไว้ในไฟล์ รูปแบบอื่น ๆ เช่น JSON และ XML ยังใช้สำหรับการจัดลำดับข้อมูล แม้ว่าแพลตฟอร์มเหล่านี้จะมีความยืดหยุ่นและมีประสิทธิภาพอย่างมาก แต่สถานที่หนึ่งที่ไม่ได้รับการปรับให้เหมาะสมอย่างเต็มที่คือสถานการณ์ที่ข้อมูลจะถูกส่งระหว่างไมโครเซอร์วิส

(Microservice) หลาย ๆ ตัวด้วยวิธีที่เป็นกลางของแพลตฟอร์ม นี่คือสาเหตุที่ทำให้ Google สร้างรูปแบบ Protobuf ในปี 2008 ตั้งแต่นั้นมา Google ก็มีการใช้งานภายในกันอย่างแพร่หลายและเป็นรูปแบบข้อมูลเริ่มต้นสำหรับเฟรมเวิร์ค (Framework) gRPC ในขั้นต้น Protobuf ถูกสร้างขึ้นสำหรับภาษาหลักสามภาษา ได้แก่ C++, Java และ Python ในช่วงหลายปีที่ผ่านมาหลายภาษาเช่น Go, Ruby, JS, PHP, C# และ Objective-C ก็เริ่มรองรับ Protobufs เช่นกัน

2.4.2.1 คุณสมบัติของ Protobuf

Protobuf เป็นรูปแบบการถ่ายโอนไบนารี (Binary) ซึ่งหมายความว่าข้อมูลจะถูกส่งเป็นไบนารี ซึ่งจะช่วยเพิ่มความเร็วในการส่งข้อมูลมากกว่าสตริง (String) ดิบเนื่องจากใช้พื้นที่และแบนด์วิดธ์น้อยกว่า เนื่องจากข้อมูลถูกบีบอัดการใช้งาน CPU ก็จะน้อยลง ดังนั้น Protobuf มักจะนิยมใช้ในการส่งข้อมูลแบบ Server- Server (Private API) เพื่อความรวดเร็วในการส่งข้อมูล

2.4.3 YMAL

YAML ย่อมาจาก YAML Ain't Markup Language เป็นรูปแบบข้อมูลแบบอนุกรม (Data Serialization Format) YMAL นิยมใช้กับข้อมูลที่เป็นพวก Configuration ชนิดของ Value ใน YAML ได้แก่ String, Number, List (Array), Dictionary, Boolean, Null ลักษณะข้อมูลเป็นแบบ Key: Value โดย Key จะใช้บอกหรืออธิบายว่า Value นั้นคืออะไร ซึ่งจะมีโครงสร้างเหมือน JSON แต่จะมี Syntax ที่แตกต่างกัน

2.4.3.1 สิ่งที่แตกต่างกันระหว่าง JSON และ YAML

- JSON จะต้องครอบด้วยปีกกา {}
- JSON Key และ Value ที่เป็น String จะต้องอยู่ใน Double Quotes (“”)
- JSON แยก Item โดยใช้ Comma (,)
- JSON ข้อมูลที่เป็น List ต้องครอบด้วย Bracket ([]) และแยก Element ด้วย Comma (,)
- YAML ไม่ต้องมี Double Quotes (“”), ปีกกา ({})
- YAML แยก Item โดยขึ้นบรรทัดใหม่
- YAML แยก Key กับ Value โดยใช้ Colon (:) และต้องมีอีก 1 Space
- YAML ข้อมูลที่เป็น List ไม่ต้องครอบ Bracket แต่จะแยก element โดยการขึ้นบรรทัดใหม่และขึ้นต้นด้วย dash (-)

2.5 ทฤษฎีเกี่ยวกับการจัดการ Service

2.5.1 Kubernetes

Kubernetes หรือ k8s เป็นแพลตฟอร์มแบบ Open-Source สำหรับช่วยให้การปฏิบัติงานต่าง ๆ ที่เกี่ยวข้องกับ Linux Container สามารถทำได้โดยอัตโนมัติ ลดกระบวนการติดตั้งหรือขยายแอปพลิเคชันที่รันบน Container ที่นักพัฒนาต้องลงมือทำด้วยตนเองให้เหลือน้อยที่สุด

หรือกล่าวได้ว่า ช่วยให้นักพัฒนาสามารถทำคลัสเตอร์ (Cluster) กลุ่มของโฮสต์ที่รัน Linux Container ได้ และ Kubernetes ก็เข้ามาช่วยบริหารจัดการคลัสเตอร์เหล่านั้นสำหรับใช้งานบน Public, Private และ Hybrid Cloud ได้อย่างสะดวกรวดเร็วและมีประสิทธิภาพมากขึ้น

Kubernetes ถูกพัฒนาและออกแบบโดยวิศวกรจาก Google ซึ่งเป็นนักพัฒนาทีมแรก ๆ ที่เริ่มนำเทคโนโลยี Linux Container มาใช้ ก่อนหน้านี้ Google ใช้แพลตฟอร์ม Borg ในการบริหารจัดการกว่า 2,000 ล้าน Container ที่ถูกสร้างขึ้นในแต่ละสัปดาห์ ก่อนที่จะนำประสบการณ์ที่ได้มาพัฒนาต่อยอดเป็น Kubernetes เพื่อบริหารจัดการระบบ Cloud ทั้งหมดในปัจจุบัน

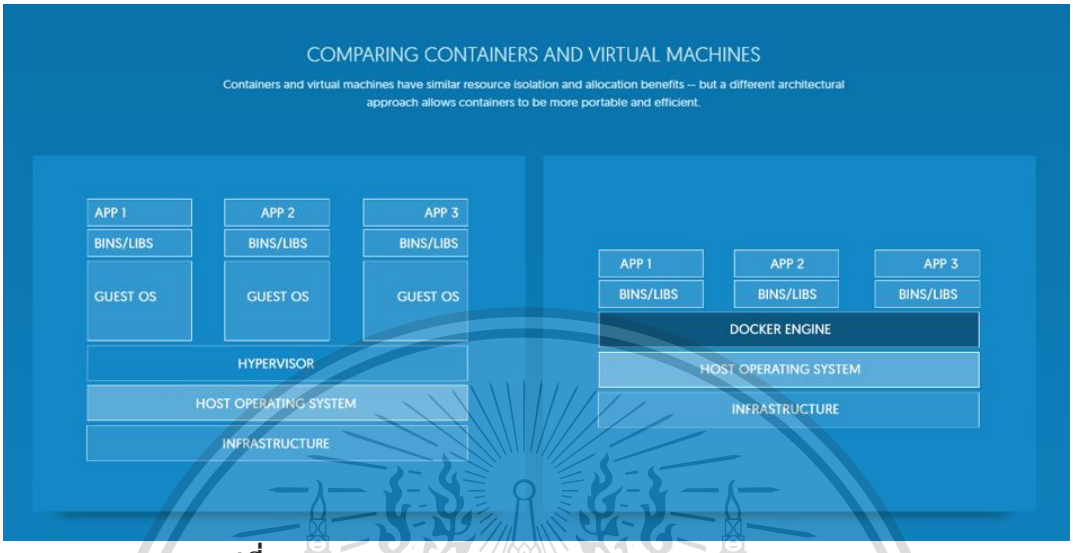
สำหรับบริษัทที่นำเทคโนโลยี Container มาใช้งานแล้ว จะพบว่าแอปพลิเคชันบน Production มักมีการเชื่อมต่อหลาย ๆ Containers เข้าด้วยกัน ซึ่ง Containers เหล่านั้นมักถูกติดตั้งบนหลาย ๆ โฮสต์ (Host) Kubernetes จะเข้ามาช่วยประสานการทำงาน (Orchestration) และบริหารจัดการ Containers ทั้งการติดตั้งและการขยายระบบในอนาคต ส่งผลให้นักพัฒนาสามารถสร้างแอปพลิเคชันที่เชื่อมต่อหลาย ๆ Containers เข้าด้วยกัน กำหนดการทำงานต่าง ๆ ขยายระบบ และตรวจสอบการทำงานทั้งหมดได้อย่างง่ายดาย ที่สำคัญคือสามารถดำเนินคำสั่งต่าง ๆ ได้โดยอัตโนมัติอีกด้วย

คุณสมบัติเด่นของ Kubernetes มีดังนี้ ประสานการทำงานของ Containers ระหว่างแต่ละโฮสต์เข้าด้วยกัน ช่วยให้ใช้ทรัพยากรของฮาร์ดแวร์ในการรันแอปพลิเคชันได้อย่างมีประสิทธิภาพสูงสุดเพิ่มสตอเรจ (Storage) สำหรับรัน Stateful Apps ได้อย่างง่ายดายขยายแอปพลิเคชันที่รันบน Container และทรัพยากรต่าง ๆ ที่จำเป็นต้องใช้งานได้ตามต้องการช่วยรับรองว่าแอปพลิเคชันสามารถทำงานได้ตรงตามที่กำหนดไว้ มีระบบ Health-Check และ Self-Heal รวมไปถึง Autoreplacement, Autorestart, Autoreplication และ Autoscaling

2.5.2 Docker

Docker คือ Engine ตัวหนึ่งที่การทำงานจะมีลักษณะจำลองสภาพแวดล้อมขึ้นมาบนเครื่องเซิร์ฟเวอร์เพื่อใช้ในการ Run Service ที่ต้องการ จะทำงานคล้ายกับ Virtual Machine เช่น VMWare, VirtualBox, XEN, KVM เป็นต้น ซึ่งจะมีข้อแตกต่างที่ชัดเจนคือ Virtual Machine เป็นการจำลอง OS เพื่อใช้งาน และเมื่อต้องการใช้งาน Container ในการจำลองสภาพแวดล้อม

ขึ้นมา เพื่อใช้งานสำหรับ 1 Service ที่ต้องการใช้งานเท่านั้น โดยไม่ต้องมีส่วนของ OS เข้าไปเกี่ยวข้องเหมือน Virtual Machines อื่น ๆ ดังรูป



รูปที่ 11 Comparing Containers and Virtual Machines

(ที่มา: https://miro.medium.com/max/1400/0*yVeFXrQ098IERiLG.png)

Docker สามารถใช้งานได้สะดวก ตอบสนองความต้องการของนักพัฒนาโปรแกรม (Developer) หรือผู้ดูแลระบบ (System Admin)

2.5.2.1 สิ่งที่ Docker น่าสนใจ

- Docker Engine สามารถใช้งานได้บนหลายแพลตฟอร์มทั้งบน Linux, Mac และ Windows
- Docker มีขนาดเล็ก สามารถใช้งาน และติดตั้งได้อย่างรวดเร็ว และสะดวกในการ Start / Stop หรือแม้แต่การย้ายไปใช้งานสำหรับเครื่องเซิร์ฟเวอร์อื่นที่มีการ Run Docker Engine ก็สามารถทำได้โดยไม่ซับซ้อน
- ผู้ใช้งาน Docker ไม่จำเป็นต้องติดตั้ง OS อีกครั้งเพื่อติดตั้ง Container รวมทั้งไม่จำเป็นต้องคอนฟิก (Config) เพิ่มเติมในส่วนที่ไม่จำเป็นอีกด้วย
- Docker มีความต้องการในการใช้ CPU, RAM และพื้นที่น้อยกว่า Virtual Machine ทั้งนี้ในทรัพยากรที่มีเท่ากัน Docker สามารถใช้งาน Container ได้มากกว่า Virtual Machine
- เนื่องจากผู้ใช้งาน สามารถสร้าง Docker Image ได้เอง จาก Dockerfile ดังนั้นการใช้งาน Docker ยังช่วยลดปัญหาสภาพแวดล้อมที่ต่างกัน ที่มักพบเมื่อบางแอปพลิเคชันสามารถทำงานได้บน Development Server แต่ไม่สามารถใช้งานบน Production Server ได้

- Docker ยังมี Docker Registry ซึ่งผู้ใช้งานสามารถเลือก Pull Image ต่าง ๆ ที่มีการสร้างไว้ให้แล้วมาใช้งาน โดยมี Docker Hub เป็น Registry หลักในการเรียกใช้ Image

2.5.2.2 ข้อดีของ Docker เมื่อเทียบกับ VMs

- Docker ไม่ต้องเสียเวลาในสร้าง OS ใหม่ และการคอนฟิกแต่ละ OS เลย
- Docker เบาและเร็วกว่ามาก ไม่ว่าจะเป็น start stop และ restart เพราะมันใช้ OS, CPU และ RAM ร่วมกันกับโฮสต์ OS
- Docker สามารถรัน container ได้มากกว่า VMs ในเครื่องที่มีทรัพยากรที่เท่ากัน
- Docker มีระบบ Registry ทำให้สามารถเคลื่อนย้าย หรือติดตั้ง Container ได้สะดวก และรวดเร็วกว่ามาก
- Containers จะรันอยู่บน Docker Engine ทำให้ไม่ต้องสนใจว่าโครงสร้าง (Infrastructure) หรือโฮสต์ OS ว่าจะเป็นอะไรยังไง ทำให้แก้ไขปัญหาว่าเครื่องนักพัฒนารันได้ แต่เครื่อง Production มันรันไม่ได้บ้าง หรือเครื่องนักพัฒนาแต่ละคนติดตั้งเครื่องมือคนละเวอร์ชันกัน เราก็ Build Container เป็น Image แล้วส่งให้คนในทีมใช้งาน

2.6 ทฤษฎีเกี่ยวกับการทำ Version Control

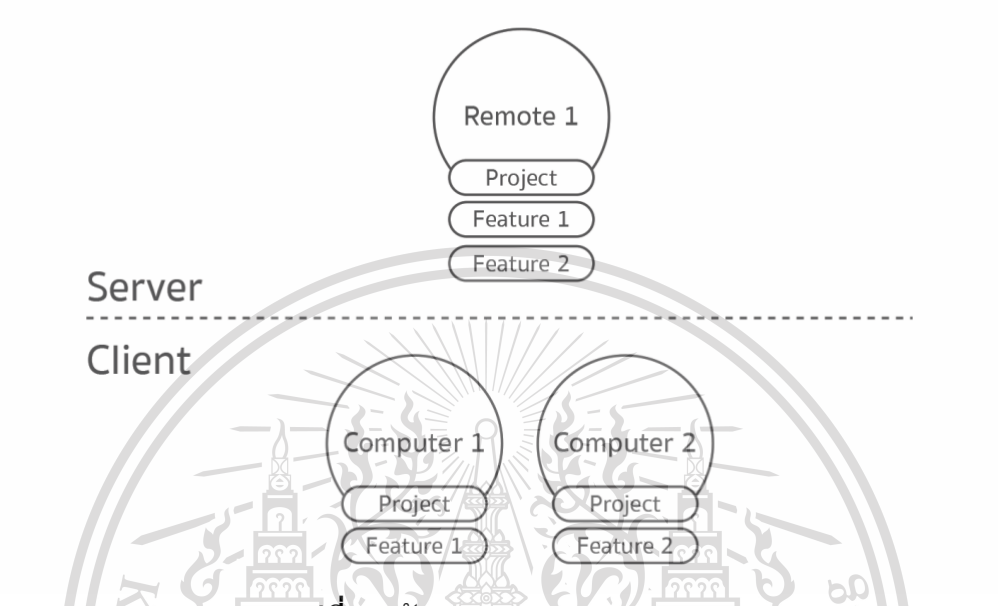
Version Control เกิดมาจากการที่ต้องการพัฒนาโปรเจคที่มีขนาดใหญ่ ใช้ผู้พัฒนาหลายคน แล้วต้องการรวบรวมโค้ดของแต่ละคนที่เขียนเข้าด้วยกัน ซึ่งแต่ก่อนนั้นก็จะใช้วิธีเดิม ๆ ก็คือการคัดลอกของแต่ละคนมารวมไว้ที่เครื่องเดียว ซึ่งการทำแบบนี้ก็จะก่อให้เกิดปัญหาตามมา คือใครเขียนโค้ดส่วนไหน โค้ดของคนแรกอาจจะไปกระทบกับโค้ดของคนอื่น ๆ พอเกิดเหตุการณ์แบบนี้ขึ้นก็ทำให้ต้องมานั่งแก้ไข หรืออาจจะเกิดปัญหาลืมว่าตัวเองนั้นได้แก้โค้ดส่วนไหนไปบ้าง ทำให้เกิดความวุ่นวาย ปัญหาเหล่านี้เป็นที่มาของ Version Control ขึ้นมาเพื่อควบคุมการเปลี่ยนแปลงต่าง ๆ ของโค้ดในโปรเจค

ประโยชน์ของ Version Control ได้แก่ สามารถเก็บประวัติการแก้ไขโค้ดได้และทำให้เห็นอีกว่าแก้ไขส่วนไหนไปบ้างทุกครั้ง สามารถรวมโค้ดจากหลาย ๆ คนเข้าด้วยกันได้ง่ายขึ้น สามารถดูได้ว่าโค้ดเดิมคืออะไรและแก้ไขเป็นอะไร เมื่อเกิดปัญหาสามารถดูประวัติการแก้ไขโค้ดในแต่ละบรรทัดได้ง่าย สามารถช่วยจัดการโปรเจคให้เป็นระบบระเบียบได้ ไม่สะเปะสะปะ ไม่เขียนโค้ดข้ามฟีเจอร์ในโค้ดชุดเดียว Version Control ถือว่าเป็นแบ็กอัพอีกด้วย หากโค้ดเกิดปัญหาต้อง Rollback กลับไปใช้โค้ดเดิม อีกทั้งใช้พื้นที่ในการเก็บข้อมูลน้อยถ้าเทียบกับการแบ็กอัพเก็บข้อมูลทั้งโปรเจคโดยช่องทางอื่น ๆ และสามารถ Track การทำงานของทุกคนภายในทีมได้จาก History

2.6.1Git

Git เป็น Version Control แบบ Distributed Version Control Systems ที่ใช้ในการพัฒนา

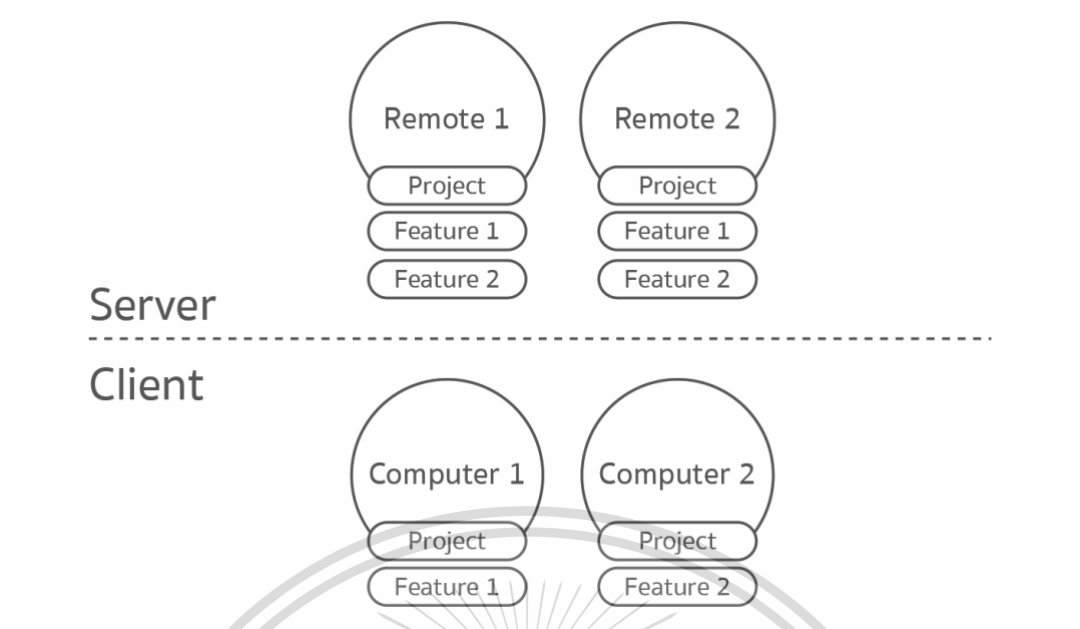
ซอฟต์แวร์สามารถช่วยเก็บไฟล์ ติดตามการเปลี่ยนแปลงต่าง ๆ และกลับไปดูรายละเอียด Version ต่าง ๆ ได้ เปรียบเสมือนเซิร์ฟเวอร์กลางหรือรีโมต (Remote) ที่คอยเก็บข้อมูลจากผู้ใช้แต่ละเครื่องที่ใช้งานแตกต่างกันไป เช่น มี Remote ที่คอยเก็บข้อมูลจาก Developer 2 คนที่ใช้ Computer 2 เครื่อง ทำงานฟีเจอร์ที่แตกต่างกันไป ดังรูป



รูปที่ 12 ตัวอย่างการทำงานของ Git

(**ที่มา:** https://miro.medium.com/max/1400/1*ipYDCTSrL8Vyca109yaTEQ.png)

โดย Remote จะคอยแบ็กอัพ (Backup) ข้อมูลหรือเก็บข้อมูลของคอมพิวเตอร์ทั้งสองเครื่องอยู่ตลอดเวลาแต่ในความจริงแล้ว Git ก็มีการทำงานที่มากกว่านั้น Git ออกแบบมาให้ทำงานกระจายแบบไม่มีศูนย์กลาง ทุกเครื่องจะทำงานเป็น VCS (Version Control System) ด้วยตัวเองได้ ซึ่งหมายความว่ามันไม่จำเป็นต้องมีเซิร์ฟเวอร์กลางใด ๆ ก็ได้เพราะสามารถทำงานได้ด้วยตัวเอง สามารถใช้เครื่องส่วนตัวทำเป็น VCS ได้เลย แต่ถ้าต้องทำงานร่วมกันหลาย ๆ เครื่อง ก็ต้องใช้เซิร์ฟเวอร์เป็นตัวกลางในการรวมข้อมูลซึ่งจะทำให้ข้อมูลของเราไม่ผูกขาดกับเซิร์ฟเวอร์จนเกินไป ในเวลาที่เซิร์ฟเวอร์กลางมีปัญหาหรือว่าทำงานแบบออฟไลน์ (Offline) เราก็ยังคงทำงานได้อยู่ โดยใช้ข้อมูลจาก VCS ภายในเครื่องตัวเอง พอเชื่อมต่อกับเซิร์ฟเวอร์กลางก็ค่อยซิงก์ (Sync) ข้อมูลทีหลังได้ และเมื่อทุก ๆ เครื่องทำงานเป็น VCS อยู่แล้ว จึงทำให้เราสามารถมีเซิร์ฟเวอร์กลางมากกว่า 1 ตัวได้เช่นกัน ดังรูป

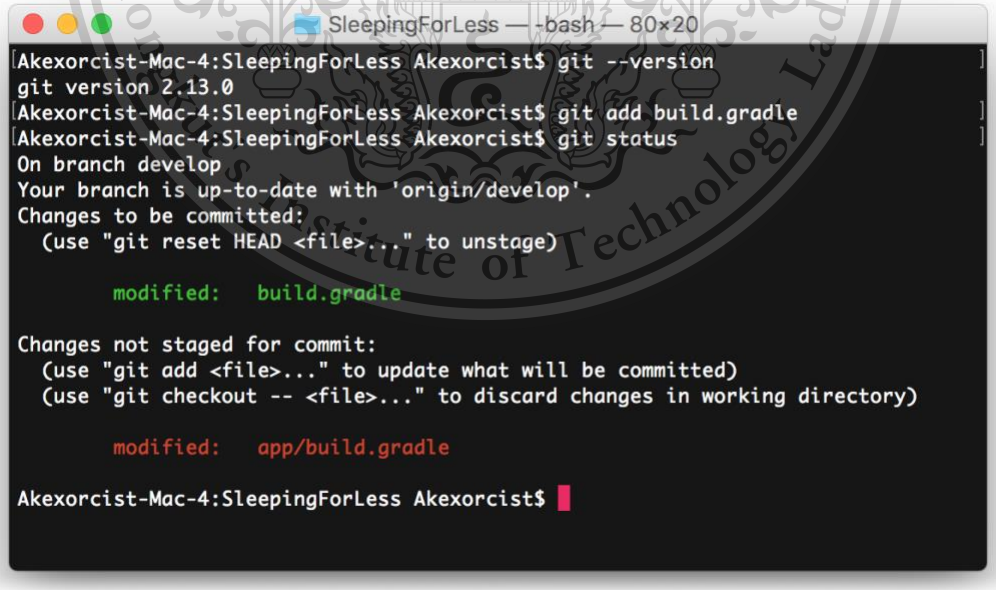


รูปที่ 13 ตัวอย่างการทำงานของ Git ที่มี Remote หลายตัว

(**ที่มา:** https://miro.medium.com/max/1400/1*MT4lWT4oycU5FY3GiQGQeA.png)

การเรียกใช้งาน Git มี 2 แบบคือ Command Line กับ GUI

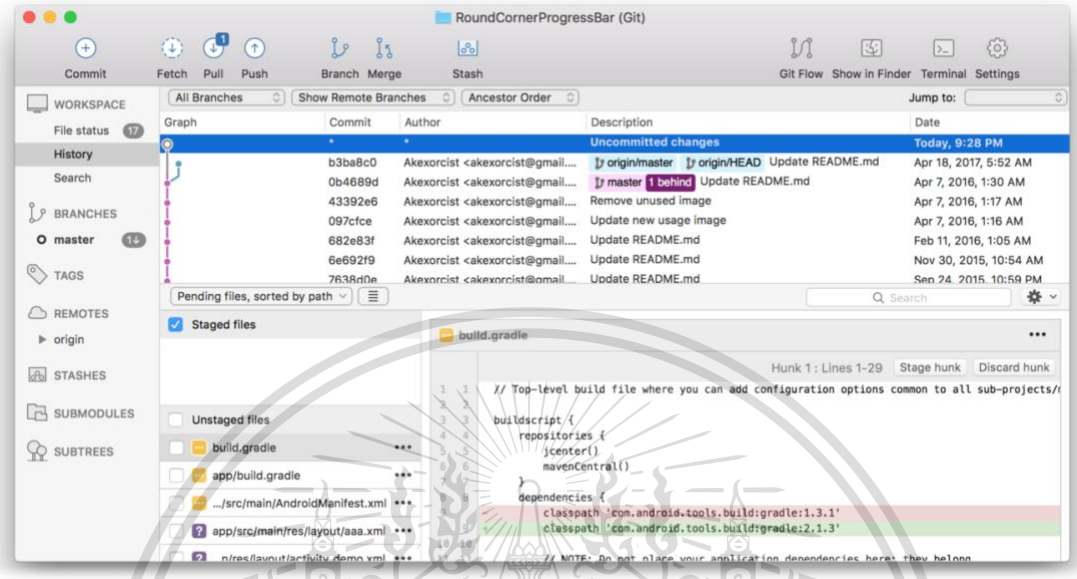
- Command Line พิมพ์คำสั่งของ Git จากเทอร์มินอล (Terminal) หรือ Command Prompt โดยตรง



รูปที่ 14 การใช้งานแบบ Command Line

(**ที่มา:** https://miro.medium.com/max/1400/1*aZKqF6i4wGmvLodBarQa-A.png)

- GUI จะใช้โปรแกรมพวก Git GUI อย่าง Source Tree, TortoiseGit หรือ GitHub Desktop เป็นต้น



รูปที่ 15 การใช้งานแบบ GUI

(**ที่มา:** https://miro.medium.com/max/1400/1*CMSWJ1S2YQZykGTdXtHW_w.png)

เซิร์ฟเวอร์สำหรับให้บริการ Git (Version Control Repository Hosting Service) ซึ่งการใช้ Git ควรต้องมีเซิร์ฟเวอร์ที่ทำหน้าที่เป็น VCS ด้วย ซึ่งปัจจุบันจะมีเซิร์ฟเวอร์ที่ให้บริการ Git ด้วย ยกตัวอย่าง เช่น GitHub, Bitbucket, Gitlab เป็นต้น เว็บพวกนี้จะให้บริการทั้งแบบ Public คือทุกคนสามารถเข้ามาดูโค้ดเราได้ และ Private คือจะสามารถดูโค้ดได้เฉพาะคนที่อนุญาตเท่านั้น

คำสั่งของ Git จะมีคำสั่ง Repository, Clone, Commit, Unstaged, Staged, Push, Pull, Fetch, Merge Commit ซึ่งแต่ละคำสั่งจะทำงานดังนี้

Repository: เมื่อต้องการพัฒนาโปรแกรม จะต้องสร้างสิ่งที่เรียกว่า Project ซึ่งการสร้าง Project สำหรับใช้งานใน Git เรียกว่า Repository หรือที่มักเรียกกันว่า “สร้าง Repo” ซึ่งก็หมายความว่าสร้างโฟลเดอร์ (Folder) นั้นเองเอาไว้ใช้เก็บข้อมูล และใน 1 Repository ก็สามารถเก็บ Project เท่าไหร่ก็ได้แต่ส่วนใหญ่มักจะเก็บ 1 Project ต่อ 1 Repository

Clone: เมื่อมี Repository อยู่บน Remote สักที่อยู่แล้ว และต้องการซิงก์ข้อมูลหรือโค้ดลงมาที่เครื่อง จะต้องทำสิ่งที่เรียกว่า Clone Repository ซึ่งมันก็คือการ Copy Repository จาก Remote ลงมาที่เครื่อง โดยถ้าใช้ในรูปแบบ Command Line ก็จะให้คำสั่งว่า git clone (url)

Commit: เวลาที่นักพัฒนาแก้ไขโค้ดหรือทำงานส่วนไหนเสร็จแล้ว และต้องการแบ็กอัพเก็บข้อมูลไว้ใน VCS จะเรียกสิ่งนี้ว่า Commit โดยถ้าใช้ในรูปแบบ Command Line ก็จะให้คำสั่งว่า git commit -m “message” คือพิมพ์คำสั่ง git commit -m และตามด้วยข้อความเพื่ออธิบายรายละเอียด

ของข้อมูลที่ Commit ไปว่าได้ทำอะไรไปบ้างเวลาที่ต้องการกลับมาดูจะได้รู้ได้ทันทีว่าแก้ไขอะไรส่วนไหนไปบ้างไม่ต้องกลับไปเช็กอย่างโค้ดเพียงอย่างเดียว ซึ่งการ Commit สามารถเลือกได้ว่าต้องการเก็บไฟล์ไหนบ้างโดยไม่จำเป็นต้องเลือกทุกไฟล์ และการ Commit นั้นมันจะเก็บว่ามีข้อมูลส่วนไหนถูกแก้ไขเปลี่ยนแปลงไปบ้าง รวมถึงเปลี่ยนแปลงไปอย่างไร สามารถดูโค้ดในg Version ก่อนหน้าและปัจจุบันได้ หรือสามารถย้อนดูประวัติ (History) ได้

Unstaged และ Staged: สองอย่างนี้มันคือ สถานะของไฟล์งาน ซึ่งเวลาที่ต้องการแก้ไขโค้ดหรือแก้ไขข้อมูล ไฟล์ที่ทำการแก้ไขจะอยู่สถานะ Unstaged อัตโนมัติ และเมื่อทำการแก้ไขเสร็จเรียบร้อย และต้องการ Commit โค้ดไปเก็บไว้ จะต้องเลือกไฟล์ที่ต้องการเพื่อย้ายไปสู่สถานะ Staged ก่อนถึงจะ Commit ได้ ซึ่งสถานะ Unstaged และ Staged ทำให้เราสามารถเลือกเฉพาะบางไฟล์สำหรับ Commit ได้นั่นเอง จะได้ Commit เฉพาะไฟล์ที่เราเขียนเสร็จแล้ว ซึ่งเราสามารถเช็กสถานะของไฟล์ได้โดยการพิมพ์คำสั่งใน Command Line คือ git status ได้ git status จะแสดงไฟล์ที่มีการเปลี่ยนแปลงอยู่ หรือยังต้องการ add หรือ commit อยู่นั้นเอง

Push: หากเวลาที่มี Commit อยู่ในเครื่องและนักพัฒนาต้องการที่จะซิงค์ข้อมูลไปเก็บไว้ใน Remote จะเรียกขั้นตอนนี้ว่า Push หรือการส่งการเปลี่ยนแปลงของไฟล์ที่ยังอยู่ในเครื่องไปไว้บน Remote Repository หรืออธิบายได้อีกว่าก่อนจะ Push ได้ต้องทำการ Commit มาก่อนแล้ว โดยการพิมพ์คำสั่งใน Command Line คือ git push

Pull: เวลาต้องการซิงค์ข้อมูลจาก Remote เพื่อดึงข้อมูล Commit ใหม่ ๆ ลงมาเก็บไว้ในเครื่อง จะเรียกขั้นตอนนี้ว่า Pull โดยคำสั่งใน Command Line คือ git pull นั้นจะทำการ git fetch และ git merge ไปด้วย โดยเราจะมักเห็นใช้ git pull –rebase เพื่อทำการเปลี่ยนฐานแทนการ merge

Fetch: มันจะมีกรณีที่เราไม่ได้ต้องการ Pull ข้อมูลมาไว้ที่เครื่องทันทีเลย แต่เพียงแค่ต้องการจะเช็กสถานะของ Remote เท่านั้นว่ามีใครได้ Push ข้อมูลใหม่ขึ้นไปที่ Remote หรือไม่ เราจะเรียกวิธีนี้ว่า Fetch ซึ่งการ Fetch จะสามารถดูและอัปเดตประวัติ (History) ทั้งหมดที่อยู่บน Remote ได้โดยที่ไม่ต้องดึงข้อมูลลงมาที่เครื่อง หรืออธิบายได้ว่ามันคือการตรวจสอบไฟล์ภายในเครื่องและ Remote ว่าตรงกันหรือไม่ ซึ่งขั้นตอนการ Fetch จะถูกเรียกทุกครั้งที่ทำการ Pull เราจะสามารถเลือกได้ว่าจะ Pull ข้อมูลเลยหรือเพียงแค่จะ Fetch ดูก่อนว่ามี Commit อะไรเพิ่มเข้ามาหรือไม่ แล้วค่อย Pull ข้อมูลลงมาที่หลังก็ได้ โดยการพิมพ์คำสั่งใน Command Line คือ git fetch

Merge Commit: อธิบายได้ตามตัวอย่างนี้ มีนักพัฒนาสองคนเขียนโค้ดด้วยกัน และทั้งเขียนในไฟล์ เดียวกัน นาย B ทำงานในส่วนที่ 2 เสร็จก็ทำการ Commit ไฟล์แล้ว Push ขึ้นไปยัง Remote นาย A ทำงานส่วนที่ 1 เสร็จทีหลังจะ Push ขึ้น Remote แต่ก็ไม่สามารถทำได้ เพราะนาย B ได้ Push ขึ้นไปแล้วก่อนหน้านี้ ดังนั้นสิ่งที่นาย A ต้องทำก่อนที่จะ Push ของตัวเองขึ้นไปได้ ก็คือจะต้อง Pull จาก Remote ลงมาใหม่ก่อนเพื่ออัปเดต Commit ที่นาย B ได้ Push ขึ้นไป ซึ่งเราเรียกขั้นตอนนี้ว่า Merge Commit นั่นเอง ซึ่งนี่คือการ Merge รวมกันได้โดยไม่มีปัญหาอะไร เมื่อ Merge เสร็จแล้วก็จะกลายเป็น Commit ตัวหนึ่งที่ให้เราเก็บและ Push ขึ้น Remote เท่านั้น แต่ในกรณีที่มีปัญหาเช่นโค้ดของนาย A และ B เกิดชนกันไปแก้โค้ดที่เดียวกัน ดังนั้น Git จะแจ้งว่าเกิด Conflict หรือโค้ดที่ทับ

ซ้อนกัน ซึ่งนาย A ก็ต้องแก้ Conflict นี้ให้เรียบร้อยถึงจะ Merge Commit แล้ว Push ขึ้น Remote ได้ ขั้นตอนของการแก้ Conflict จะมีดังนี้ Pull จาก Remote ลงเครื่อง > เกิด Conflict > แก้ไขโค้ดให้เหมาะสมก่อน > ทำการ Merge Commit ใหม่ > Push ขึ้น Remote

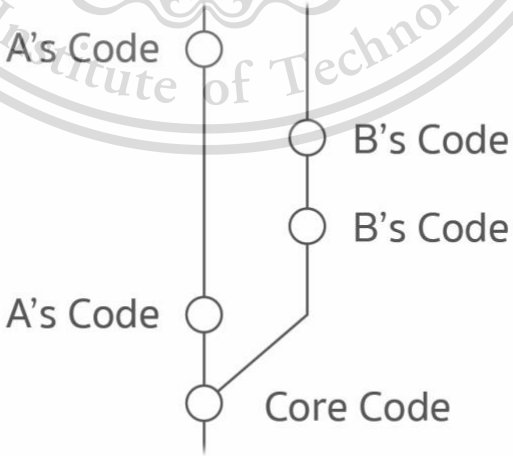
คำที่ควรรู้จักในการใช้งาน Git มีดังนี้ BranchPull Request, Merge Branch, Stash, Unstash, Git Flow, Tag, Fork, Pull Request

Branch เมื่อทำการ Commit ไปเรื่อย ๆ จะกลายเป็น History ขึ้นมา Commit จะเรียงกันเป็นเส้นยาว ๆ ที่เรียกว่า Branch ซึ่งเป็นดังรูป

รูปที่ 16 ตัวอย่าง Branch

(**ที่มา:** https://miro.medium.com/max/4800/1*2-9DuAKBYC4AaYdW_cfFOA.png)

แต่ในความเป็นจริง Branch ไม่ได้มีเพียง Branch เดียวจะสามารถมีได้หลาย Branch หลายเส้น ซึ่งอาจจะเกิดขึ้นในกรณีที่ทำงานร่วมกันหลายคนเขียนโค้ดกันคนละส่วนคนละฟีเจอร์ ถ้าหากให้เพียง Branch เดียว คงจะต้องเสียเวลา Pull ข้อมูลของกันและกัน และเส้น Branch ก็จะสลับกันมั่วและเกิดความไม่เรียบร้อย ดังนั้น Git จึงออกแบบมาให้สร้าง Branch แยกออกมาได้หลายเส้น ดังตัวอย่าง

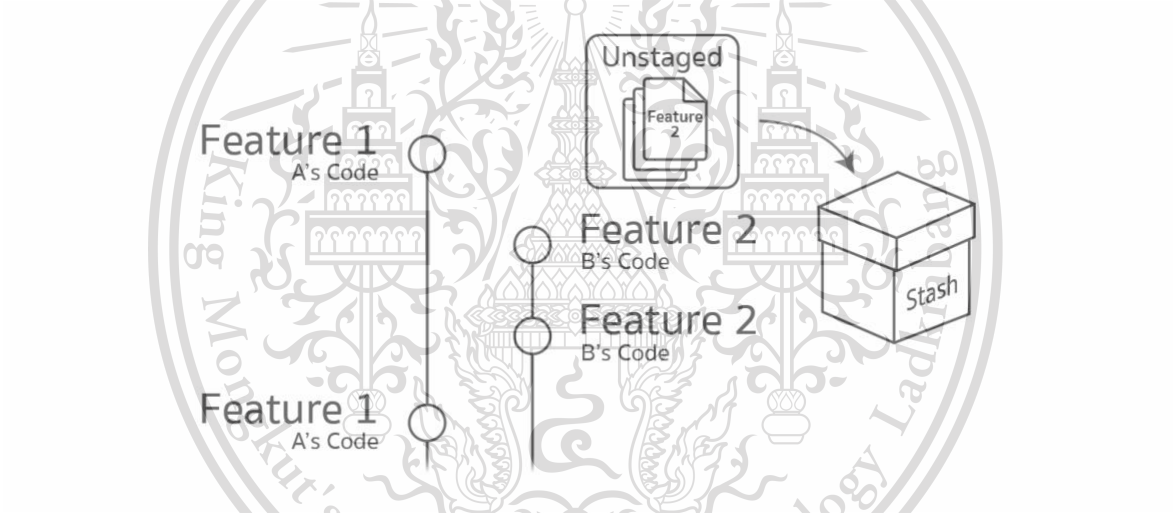


รูปที่ 17 Branch หลายเส้น

(**ที่มา:** https://miro.medium.com/max/4800/1*uqJz73zez9hepV4-8fvIPw.png)

Merge Branch คือการที่ต้องการจับเอา Branch มารวมกัน จะเรียกวิธีนี้ว่า Merge Branch แต่ในระหว่างการ Merge Branch อาจจะเกิด Conflict ขึ้นได้ แต่เนื่องจากการแยก Branch มักจะใช้สำหรับกรณีที่เขียนโค้ดแยกส่วนหรือแยกฟีเจอร์กันอยู่แล้ว ดังนั้นตำแหน่งของโค้ดที่เกิด Conflict ก็จะเป็นเพียงบางจุดเท่านั้น และเมื่อแก้ไขโค้ดแล้วก็ Merge Branch ทั้งสองเข้าด้วยกันได้ ก็จะเกิดเป็น Merge Commit

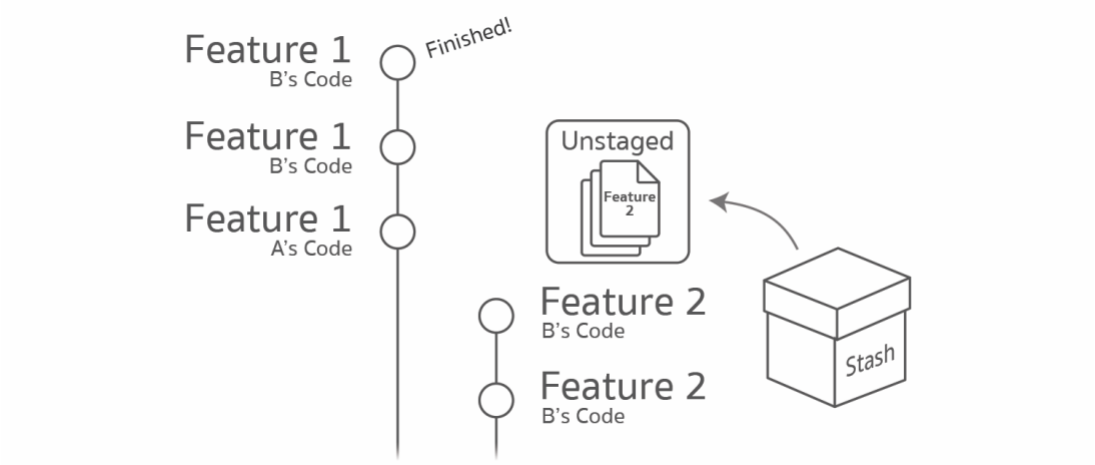
Stash และ Unstash เปรียบเสมือนที่พักข้อมูลหรือที่เก็บข้อมูล และ Stash ยังสามารถใช้ย้ายไฟล์ข้าม Branch ได้อีกด้วย ยกตัวอย่างเช่น นักพัฒนาทำงานร่วมกัน 2 คน และทำงานคนละ Branch แต่เกิดกรณีที่นาย A ต้องไปช่วยนาย B ทำงานในขณะที่งานของนาย A ก็ยังไม่เสร็จเรียบร้อย แต่ต้องพักเพื่อสลับ Branch ไปช่วยนาย B ทำงานก่อน ซึ่งนาย A จะไม่สามารถสลับ Branch ได้ทันทีเลย สิ่งที่ต้องทำคือ ต้องเอาไฟล์ที่ยังทำไม่เสร็จซึ่งเดิมอยู่ที่ Unstaged ไปเก็บไว้ใน Stash ชั่วคราวก่อน เพื่อจะได้สลับ Branch ไปทำงานอีกส่วนได้ ดังรูป



รูปที่ 18 ย้ายไฟล์จาก Unstaged ไปที่ Stash ชั่วคราว

(**ที่มา:** https://miro.medium.com/max/1400/1*VnAJvnOSX5ZzsR2_5RwJ0A.png)

เมื่อทำงานส่วนของนาย B เรียบร้อยแล้ว ก็กลับมาสลับ Branch ทำงานในส่วนของตัวเองต่อโดยกลับมาที่ Branch ตัวเองแล้ว Unstash เพื่อเอาไฟล์ที่เคยทำค้างไว้มาทำต่อให้เสร็จ ดังรูป

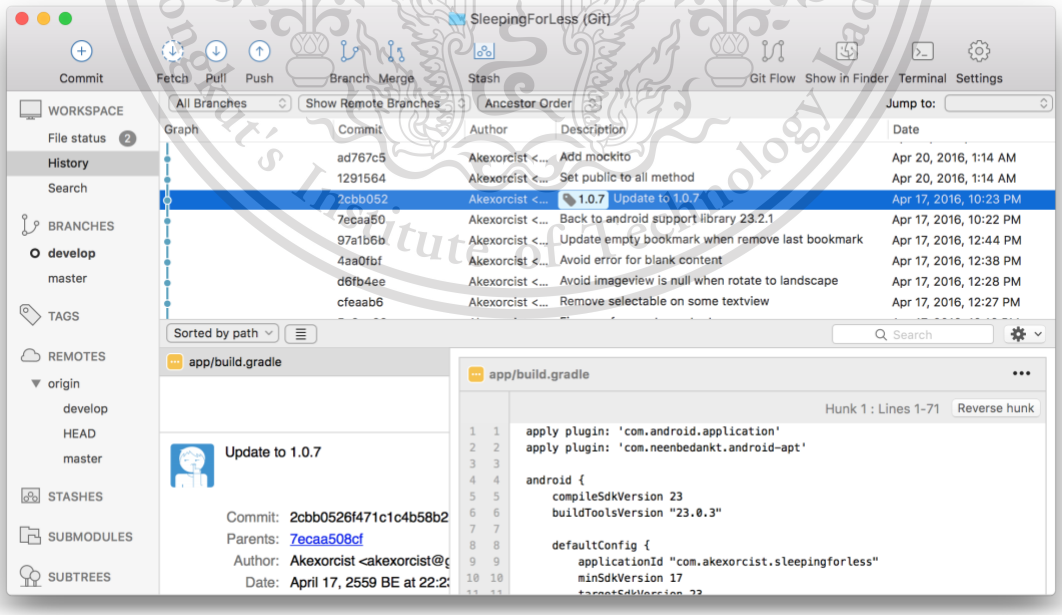


รูปที่ 19 ทำการ Unstash ไฟล์เดิมมาทำงานต่อ

(ที่มา: https://miro.medium.com/max/1400/1*1mryyy3rk3gv-1ZZEDHLRg.png)

Git Flow คือมาตรฐานที่กำหนดรูปแบบของการแตก Branch ให้เป็นระเบียบเพื่อให้ตอบโจทย์ใน Development Process โดยจะแบ่ง Branch ออกเป็น 5 กลุ่ม คือ Master, develop, feature, release, hotfix

Tag จะอยู่ใน Master ซึ่ง Tag จะเป็นตัวช่วยบอกว่าอันไหนอยู่ที่ Commit ไหน เพื่อที่จะได้ดูรายละเอียดต่าง ๆ ได้ง่าย ในกรณีที่ใช้งาน Git Flow จะมีการใส่ Tag ให้อัตโนมัติตามชื่อ Branch ที่กำหนดไว้ตอน Release และ Hotfix



รูปที่ 20 รูปแบบของ Tag

(ที่มา: https://miro.medium.com/max/4800/1*l5EHuiWChGwotsN15bE4Xg.png)

Fork คือการ Copy Repository ของคนอื่นที่เขาอนุญาตให้เราเข้าไปดูได้มาเก็บไว้เป็น Repository ของตัวเอง ซึ่งจะมีทุกอย่างเหมือนกันทั้งหมด แต่จะต่างกันตรงที่เราจะเป็นเจ้าของเพียงแค่ Repository ที่เราคัดลอกมาเท่านั้น ดังนั้นก็จะสามารถแก้ไขโค้ดต่าง ๆ ได้เพียงแค่ในของตัวเองเช่นกัน

Pull Request เปรียบเสมือนการช่วยคนอื่นแก้ Bug โดยการส่ง Pull Request ไปให้ต้นทางเพื่อให้เจ้าของ Library นั้น ๆ เห็น Commit ของเราที่แก้ Bug ส่วนนั้นไป โดยที่เจ้าของ Library ก็สามารถกดรับ Pull Request ของเราที่ส่งไปให้เพื่อรวม Commit ที่แก้ Bug แล้วเข้าไปใน Repository ของเขาหรือจะไม่กดรับ Pull Request ก็ได้

2.7 ทฤษฎีเกี่ยวกับความปลอดภัย

2.7.1 SSL (Secure Socket Layer)

Secure Socket Layer คือเทคโนโลยีการเข้ารหัสข้อมูล เพื่อเพิ่มความปลอดภัยในการสื่อสารหรือส่งข้อมูลบนเครือข่ายอินเทอร์เน็ต ระหว่างเครื่องเซิร์ฟเวอร์กับเว็บบราวเซอร์หรือแอปพลิเคชันที่ใช้งาน เพื่อให้ข้อมูลปลอดภัยจากการเข้าถึง การเรียกใช้งานจะเรียกผ่านโพรโตคอล HTTPS

SSL certificate คือใบรับรองอิเล็กทรอนิกส์ เป็นไฟล์ขนาดเล็กที่มีการผูกไว้กับ Private Key ของเครื่องเซิร์ฟเวอร์เพื่อยืนยันตัวตนและความถูกต้องในการส่งข้อมูลในการส่งข้อมูลระหว่างเครื่องเซิร์ฟเวอร์กับเว็บบราวเซอร์ หรือแอปพลิเคชันที่ใช้งาน มีการเข้ารหัสและถอดรหัสผ่านเทคโนโลยี SSL/TLS หากมีการดักจับข้อมูลไปได้แต่ข้อมูลก็จะปลอดภัย เนื่องจากมีการเข้ารหัสไว้และต้องมีคีย์ถอดรหัสที่เหมาะสมและตรงกันเท่านั้นถึงจะสามารถถอดรหัสได้ สิ่งที่แฮกเกอร์เห็นจะเป็นรูปแบบที่อ่านไม่ออก

ประเภทของ SSL certificate มีดังนี้ Self – sign SSL certificate, Shared SSL certificate, Dedicated SSL certificate

Self – sign SSL certificate คือ SSL certificate ที่ถูกสร้างโดยคอมพิวเตอร์เครื่องใดก็ได้ ซึ่งก็หมายถึงว่าใครก็สามารถสร้าง SSL certificate นี้ได้เช่นกัน แต่ SSL certificate ที่ได้นับมาจะไม่ผ่านการรับรองที่เป็นมาตรฐานจาก CA ดังนั้นเมื่อนำไปใช้งานจริง Browser ที่เข้าใช้งานเว็บไซต์ดังกล่าว จะขึ้นเตือนว่า SSL certificate ไม่ปลอดภัย และต้องกด Continue เพื่อยอมรับความเสี่ยงถึงจะเข้าใช้งานได้

Shared SSL certificate คือ SSL certificate ที่ใช้งานภายใต้ชื่อของผู้ที่ให้บริการ มักนิยมจะใช้กับ Shared Host ทั่วไป หลักการทำงานคือ ไม่สามารถเรียกใช้งาน Domain ของตัวเองได้

Dedicated SSL certificate คือ SSL certificate ที่มีความน่าเชื่อถือ และเป็นที่นิยมมากที่สุด จะเป็นการระบุเจาะจงเฉพาะ Domain ที่ต้องการสั่งซื้อ โดยจะออกให้โดย CA ที่มีความน่าเชื่อถือเท่านั้น ใบรับรองดังกล่าวจะมีการตรวจสอบความเป็นเจ้าของ Domain หรือองค์กรก่อนที่จะสามารถ

ออก SSL certificate ให้ได้ และการใช้งานผ่าน Browser ต่าง ๆ จะขึ้นรูปกุญแจสีเขียว ซึ่งหมายความว่าความปลอดภัยในการเข้าใช้งานเว็บไซต์

รูปที่ 21 วิธีการส่งข้อมูลรูปแบบปกติ

(**ที่มา:** https://ssl.in.th/images/tools/Transfer-01.png)

รูปที่ 22 วิธีการส่งข้อมูลรูปแบบ SSL

(**ที่มา:** https://ssl.in.th/images/tools/Transfer-02.png)

2.7.2 TLS (Transport Layer Security)

Transport Layer Security คือเทคโนโลยีการเข้ารหัสข้อมูล TLS จะคล้ายกับ SSL แต่ TLS นั้นจะเป็นโพรโตคอลที่ใหม่กว่า และเป็นมาตรฐานเปิดที่พัฒนามาจาก SSL นั้นเอง ซึ่ง TLS จะสามารถลดระดับของตัวเองลงไปเป็น SSL ได้ต่อเมื่ออุปกรณ์ที่เชื่อมต่ออยู่นั้นไม่ได้รองรับ TLS และขั้นตอนการเข้ารหัส และเรียกใช้งานของ TLS จะแยก Port หรือไม่แยก Port ของ Data ที่เข้ารหัสกับไม่เข้ารหัสออกจากกันก็ได้

2.7.3 XSS (Cross site Scripting)

Cross site Scripting คือการฝังโค้ดเข้าไปกับหน้าเว็บเพจที่มีช่องโหว่ เมื่อผู้ใช้งานโหลดหน้าเว็บเพจไป สิ่งสำคัญบางอย่างของผู้ใช้งาน เช่น ค่า Cookie, Username, Password ก็มีโอกาสที่จะถูกขโมยไปได้ โดยช่องโหว่ที่สามารถทำให้เกิด XSS ได้นั้นเนื่องมาจาก Web Application รับข้อมูลเข้ามาจากผู้ใช้ ซึ่งโปรแกรมเมอร์ที่พัฒนา Web Application ดังกล่าวไม่ได้ทำการตรวจสอบความถูกต้องของข้อมูลนั้น แล้วนำข้อมูลดังกล่าวไปแสดงบนเว็บเพจ XSS มี 2 รูปแบบดังนี้ Non-persistent XSS กับ Persistent XSS

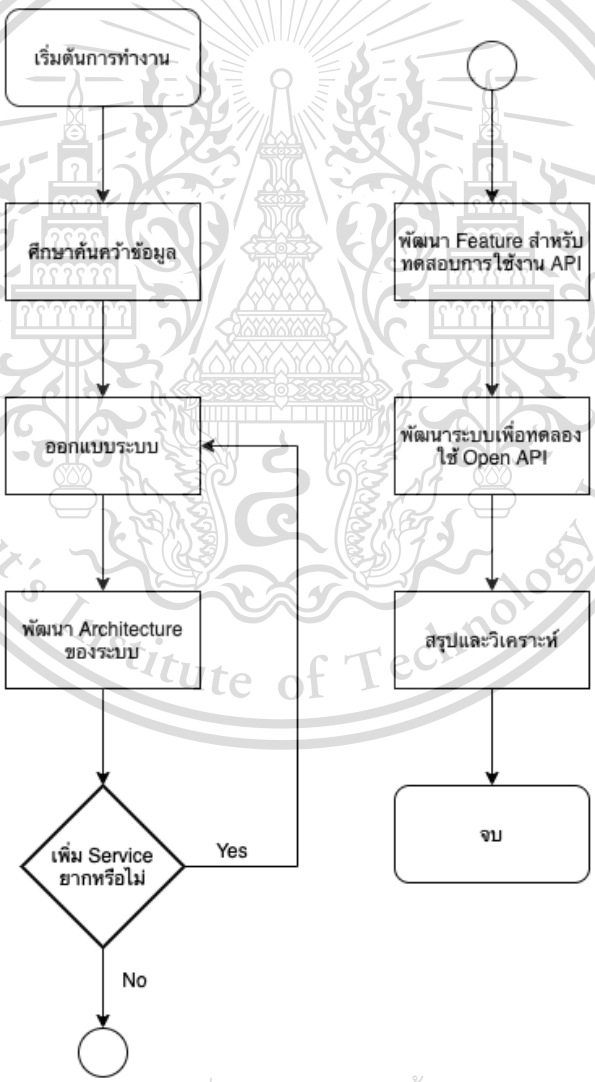
Non-persistent XSS จะขึ้นกับค่าที่เรา Inject ลงไปใน Get หรือ Post Request ด้วย โดยเมื่อ script ที่เราใส่ลงไปใน get หรือ Post Request ไปโผล่บนหน้า html มันก็จะเกิดการรัน script ขึ้น แต่การใช้เทคนิคนี้บางครั้งเราต้องใส่ script ของเราลงไปใน url ซึ่งอาจจะทำให้สังเกตได้ง่าย

Persistent XSS คือ XXS คือเหมือนการที่ใส่ Script เข้าไปในหน้าเว็บเลย หรือใส่เข้าไปในฐานข้อมูลของเว็บเลย

บทที่ 3
การออกแบบและพัฒนา

3.1 บทนำ

ปริญญานิพนธ์นี้ได้ออกแบบและพัฒนาการทำงานเพื่อให้บรรลุตามวัตถุประสงค์และขอบเขตของปริญญานิพนธ์ข้างต้น โดยเลือกให้หลักการต่าง ๆ และออกแบบโครงสร้างเพื่อให้เป็นประโยชน์และใช้งานได้จริงแก่นักพัฒนา ในการทำระบบประมวลผลข้อความผ่านช่องทางเอพีไอ จะมีการวางแผนโครงงานเกี่ยวกับแนวทางการดำเนินงานให้ทันตามระยะเวลาที่กำหนดดังนี้



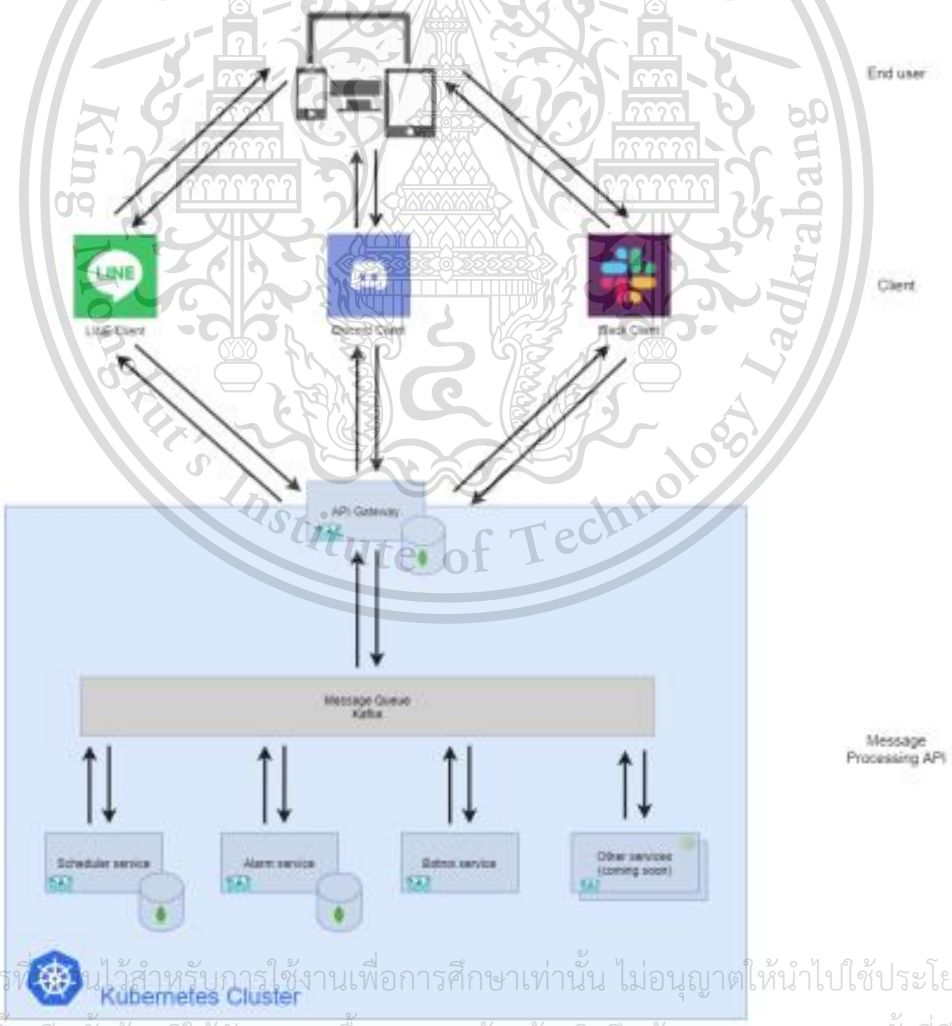
รูปที่ 23 แนวทางการดำเนินงาน

3.2 ภาพรวมของระบบ

ปริญญานิพนธ์นี้เป็นการออกแบบและสร้าง Open API ที่มีชื่อว่า ระบบประมวลผลข้อความผ่านช่องทางเอพีไอ Open API คือ อินเตอร์เฟสที่เผยแพร่ต่อสาธารณะซึ่งมีฟีเจอร์ที่พร้อมใช้งาน ช่วยให้นักพัฒนาซอฟต์แวร์สามารถเข้าถึงและนำไปใช้งานได้ ซึ่งฟีเจอร์ต่าง ๆ ที่มีในระบบของประมวลผลข้อความผ่านช่องทางเอพีไอเช่น ฟีเจอร์ตรวจสอบผลสลากกินแบ่งรัฐบาล, แชตบอตสำหรับคุยเล่น โดยจะใช้งานผ่าน RESTful API

ระบบนี้ถูกออกแบบมาเป็น Micro Service โดยอ้างอิงแนวคิด Event Driven Architecture โดยโครงสร้างของระบบประมวลผลข้อความผ่านช่องทางเอพีไอจะแบ่งออกเป็น

1. API Gateway Service สำหรับรับคำสั่งจากภายนอก
2. Message Queue โดย Message Queue จะทำหน้าที่เป็น Event bus คอยส่ง-รับ Event จาก API Gateway ส่งให้ Service ปลายทาง
3. Service ปลายทาง/อื่น ๆ ในระบบ คือ Service ที่พัฒนา Feature ต่าง ๆ ให้กับ Open API



รูปที่ 24 Diagram แสดงโครงสร้างของระบบ

3.3 ขั้นตอนการดำเนินงานโดยละเอียด

3.3.1 ศึกษาปัญหาของการออกแบบ Open API ของรูปแบบปกติ

การออกแบบ Micro Service ของ Open API ในรูปแบบปกตินั้นส่วนใหญ่จะทำงานในรูปแบบ Synchronous โดยจะต้องมีการรอการตอบสนองกันถึงจะประมวลขั้นตอนถัดไปได้

รูปที่ 25 รูปแบบการทำงานแบบ Synchronous

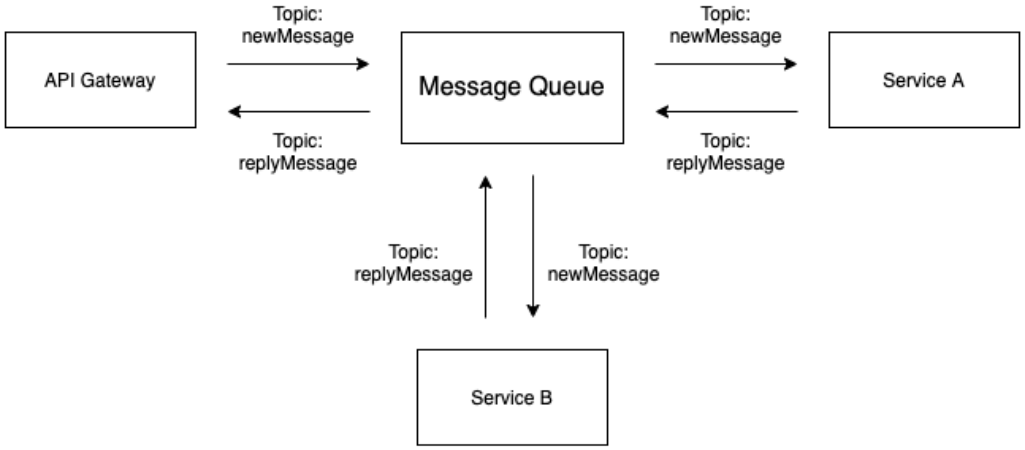
ในปกติการทำงานในรูปแบบ Synchronous API Gateway จำเป็นต้องรู้จักกับ Service ปลายทางทำให้ไม่มีความยืดหยุ่นและเมื่อต้องการเพิ่ม Service ใหม่เข้าไปอาจจะต้องกลับไปแก้ไขหรืออัปเดต Open API ด้วย

รูปที่ 26 แสดงให้เห็นว่า API Gateway ต้องรู้จักกับ Service อื่น ๆ

3.3.2 ออกแบบโครงสร้างและอินเตอร์เฟสการเชื่อมต่อใหม่

มีการเปลี่ยนแปลง Protocol การสื่อสารที่ใช้งานจากเดิมทั่ว ๆ ไปใช้ HTTP (Hypertext Transfer Protocol) มาเป็น MQTT (Message Queuing Telemetry Transport) โดยกำหนด Topic ของ Message Queue ไว้ 2 Topic คือ newMessage และ replyMessage

Topic newMessage จะถูกใช้งานจาก API Gateway เมื่อมีการเรียกใช้งาน Open API และปลายทางของ Topic นี้คือ Service ภายใน ส่วน Topic replyMessage จะเป็น Topic ที่ถูกใช้งานจาก Service ภายในเมื่อมีการประมวลผลการทำงานเสร็จแล้ว และปลายทางของ Topic นี้คือ API Gateway โดย API Gateway จะนำ Topic นี้ไปส่งต่อ/ต่อกลับผู้เรียกใช้งาน มีการกำหนด Schema ของ Message โดยประกอบด้วย Refferrent ID สำหรับการอ้างอิงเจ้าของ Event หรือหมายเลข Event, Feature ที่ต้องการใช้งาน, ข้อมูลที่ต้องการส่งให้ Feature, รูปแบบของการทำงาน Asynchronous, Synchronous โดยจะใช้ Protobuf เป็นรูปแบบข้อมูลของ Message ที่ใช้งาน



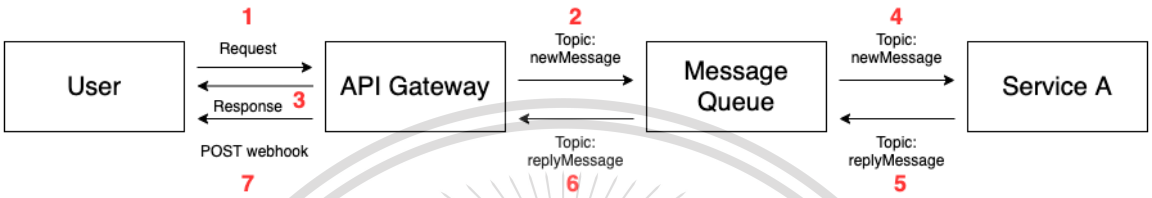
รูปที่ 27 รูปการทำงานของ Topic newMessage และ replyMessage

message-schema.proto message-schema.proto

```
syntax = "proto3";

package messagechema;

import "google/protobuf/timestamp.proto";

option go_package = ".;messageschema";


message DefaultMessage {
    string message = 1; //message
    string ref1 = 2; //client reference
    string ref2 = 3; //message reference
    string ref3 = 4; //end-user reference
    string owner = 5; //service reference
    string publishedBy = 6;
    google.protobuf.Timestamp publishedAt = 7;
    string feature = 8;
    bytes data = 9; //attachment
    string type = 10; //message type eg. reply message, notification
    ExecuteMode excuteMode = 11;
    string callbackTopic = 12; //topic for callback
    string errorInternal = 13;
    string error = 14;
}

message HealthCheckMessage {
    string feature = 1;
    string description = 2;
    repeated ExecuteMode executeMode = 3;
    string serviceName = 4;
}


enum ExecuteMode {
    Asynchronous = 0;
    Synchronous = 1;
}
```

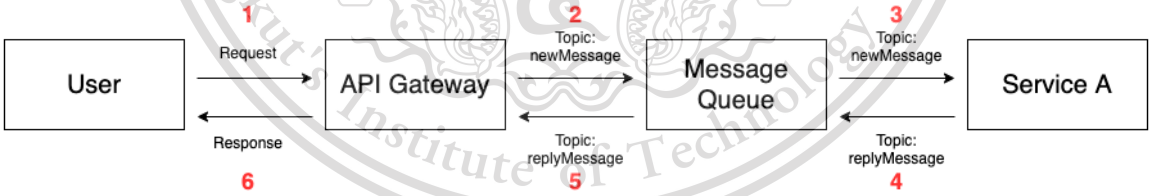
รูปที่ 28 Schema ของ Message ในรูปแบบ Protobuf

สำหรับการทำงานแบบ Asynchronous API Gateway จะสร้าง Message จาก Schema ที่กำหนดโดยระบุรูปแบบการทำงานเป็น Asynchronous และส่ง Message นี้เข้าไปยัง Message Queue แล้วจะตอบกลับไปยังผู้ที่เรียกใช้งาน Open API ทันที และหลังจากที่เซอร์วิสภายในประมวลผลข้อความเสร็จแล้วจะส่ง message ผ่าน Message Queue กลับมาที่ API Gateway และ API Gateway จะดูข้อมูลของเจ้าของ Message นี้ใน Database และนำข้อมูลใน Message ส่งกลับไปยังเจ้าของ Message ผ่าน RESTful API



รูปที่ 29 รูปแบบการทำงานของ Asynchronous

การทำงานของ Synchronous เมื่อมีการเรียกใช้งาน Open API ตัว API Gateway จะสร้าง Message จาก Schema ที่กำหนดโดยระบุการทำงานเป็น Synchronous และระบุ Topic ที่ใช้สำหรับส่ง Message กลับมาโดยเฉพาะและส่ง Message นี้เข้าไปยัง Message Queue แล้วทำการติดตาม Topic ที่ถูกสร้างขึ้นมาใหม่โดยเฉพาะ หลังจากเซอร์วิสปลายทางประมวลผลเสร็จแล้วจะส่งกลับมาที่ Message ใหม่ที่ระบุไปเมื่อ API Gateway ได้รับ Message ผ่านจาก Topic นั้นแล้วจะทำการตอบกลับไปยังผู้ที่เรียกใช้งาน Open API ทันที



รูปที่ 30 รูปแบบการทำงานของ Synchronous

3.3.3 พัฒนาระบบตามที่ได้ออกแบบไว้

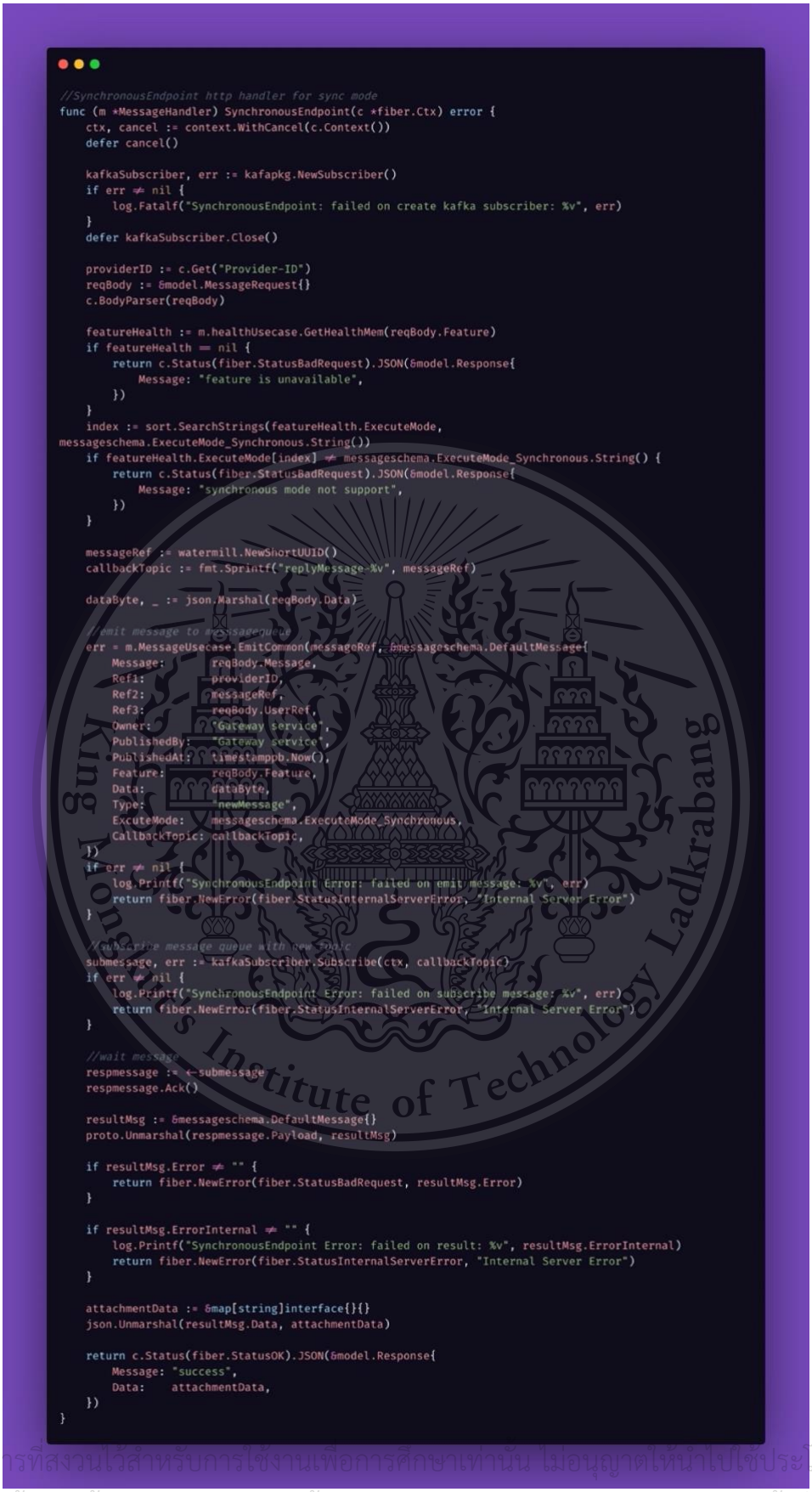
เมื่อออกแบบเสร็จแล้วจะทำการพัฒนา API Gateway ด้วยภาษา Golang โดยพัฒนาเพื่อให้สามารถทำงานตามที่ต้องการ โดยมีการใช้ Library ดังนี้

1. Fiber เป็น Web Framework สำหรับพัฒนา RESTful API
2. Watermill สำหรับติดต่อสื่อสารกับ Message Queue
3. QMGO สำหรับการติดต่อสื่อสารกับ Database MongoDB
4. Protobuf สำหรับการสร้างข้อมูลในรูปแบบ Protobuf

โดย API Gateway จะทำหน้าที่สำหรับตรวจสอบผู้ใช้งานและจัดเก็บข้อมูลผู้ใช้งานลง Database ด้วย

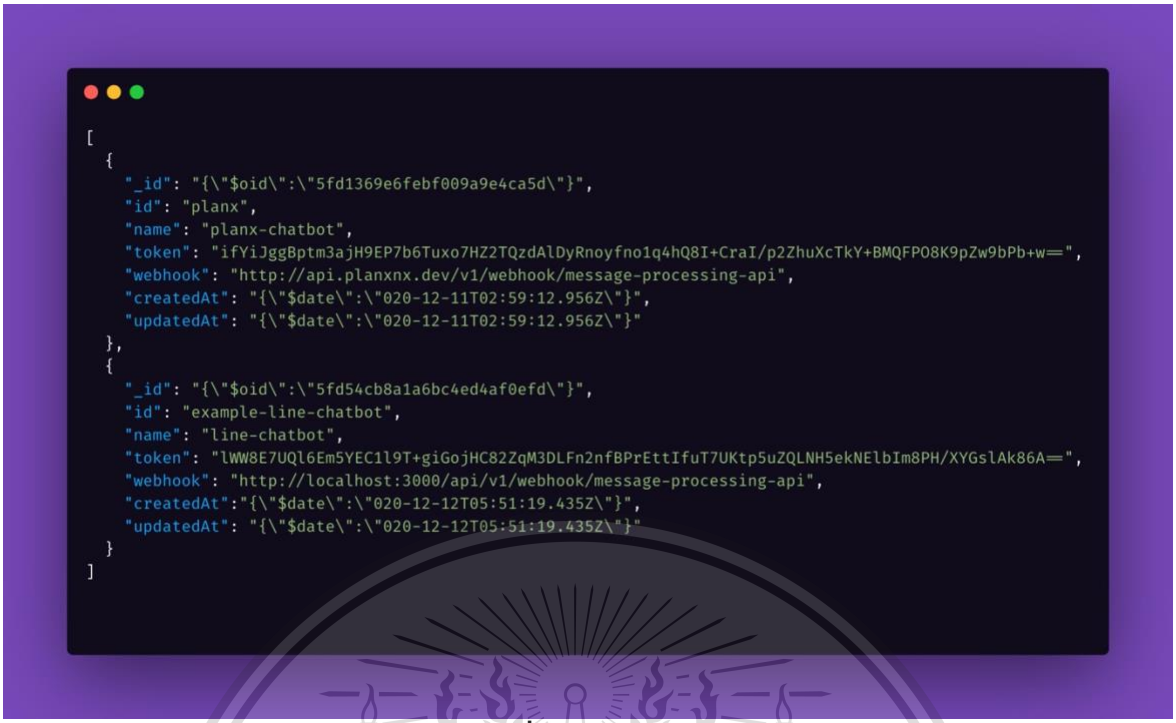
```go
//AsynchronousHandler http handler for async mode
func (m *MessageHandler) AsynchronousHandler(c *fiber.Ctx) error {
    providerID := c.Get("Provider-ID")
    reqBody := &model.MessageRequest{}
    c.BodyParser(reqBody)

    featureHealth := m.healthUsercase.GetHealthMem(reqBody.Feature)
    if featureHealth == nil {
        return c.Status(fiber.StatusBadRequest).JSON(&model.Response{
            Message: "feature is unavailable",
        })
    }
    index := sort.SearchStrings(featureHealth.ExecuteMode,
messageschema.ExecuteMode_Asynchronous.String())
    if featureHealth.ExecuteMode[index] != messageschema.ExecuteMode_Asynchronous.String() {
        return c.Status(fiber.StatusBadRequest).JSON(&model.Response{
            Message: "asynchronous mode not support",
        })
    }

    messageRef := watermill.NewUUID()

    dataByte, err := json.Marshal(reqBody.Data)
    if err != nil {
        log.Printf("AsynchronousHandler Error: failed on marshal req data: %v", err)
        return fiber.NewError(fiber.StatusInternalServerError, "Internal Server Error or request.data is
invalid format")
    }

    //emit message to messagequeue
    err = m.MessageUsecase.EmitCommon(messageRef, &messageschema.DefaultMessage{
        Message:     reqBody.Message,
        Ref1:        providerID,
        Ref2:        messageRef,
        Ref3:        reqBody.UserRef,
        Owner:       "Gateway service",
        PublishedBy: "Gateway service",
        PublishedAt: timestamppb.Now(),
        Feature:     reqBody.Feature,
        Data:        dataByte,
        Type:        "newMessage",
    })
    if err != nil {
        log.Printf("AsynchronousHandler Error: failed on emit message: %v", err)
        return fiber.NewError(fiber.StatusInternalServerError, "Internal Server Error")
    }
    return c.Status(fiber.StatusOK).JSON(&model.Response{
        Message: "Success",
        Data: model.MessageResponseData{
            MessageRef: messageRef,
        },
    })
}
```

รูปที่ 31 Code API Gateway ในส่วนของ Http Handler สำหรับการทำงานแบบ Asynchronous



```go
//SynchronousEndpoint http handler for sync mode
func (m *MessageHandler) SynchronousEndpoint(c *fiber.Ctx) error {
    ctx, cancel := context.WithCancel(c.Context())
    defer cancel()

    kafkaSubscriber, err := kafapkg.NewSubscriber()
    if err != nil {
        log.Fatalf("SynchronousEndpoint: failed on create kafka subscriber: %v", err)
    }
    defer kafkaSubscriber.Close()

    providerID := c.Get("Provider-ID")
    reqBody := &model.MessageRequest{}
    c.BodyParser(reqBody)

    featureHealth := m.healthUsecase.GetHealthMem(reqBody.Feature)
    if featureHealth == nil {
        return c.Status(fiber.StatusBadRequest).JSON(&model.Response{
            Message: "feature is unavailable",
        })
    }
    index := sort.SearchStrings(featureHealth.ExecuteMode,
messageschema.ExecuteMode_Synchronous.String())
    if featureHealth.ExecuteMode[index] != messageschema.ExecuteMode_Synchronous.String() {
        return c.Status(fiber.StatusBadRequest).JSON(&model.Response{
            Message: "synchronous mode not support",
        })
    }

    messageRef := watermill.NewShortUUID()
    callbackTopic := fmt.Sprintf("replyMessage-%v", messageRef)

    dataByte, _ := json.Marshal(reqBody.Data)

    //emit message to messagequeue
    err = m.MessageUsecase.EmitCommon(messageRef, &messageschema.DefaultMessage{
        Message:      reqBody.Message,
        Ref1:         providerID,
        Ref2:         messageRef,
        Ref3:         reqBody.UserRef,
        Owner:        "Gateway service",
        PublishedBy:  "Gateway service",
        PublishedAt:  timestamppb.Now(),
        Feature:      reqBody.Feature,
        Data:         dataByte,
        Type:         "newMessage",
        ExcuteMode:   messageschema.ExecuteMode_Synchronous,
        CallbackTopic: callbackTopic,
    })
    if err != nil {
        log.Printf("SynchronousEndpoint Error: failed on emit message: %v", err)
        return fiber.NewError(fiber.StatusInternalServerError, "Internal Server Error")
    }

    //subscribe message queue with new topic
    submessage, err := kafkaSubscriber.Subscribe(ctx, callbackTopic)
    if err != nil {
        log.Printf("SynchronousEndpoint Error: failed on subscribe message: %v", err)
        return fiber.NewError(fiber.StatusInternalServerError, "Internal Server Error")
    }

    //wait message
    respmessage := <-submessage
    respmessage.Ack()

    resultMsg := &messageschema.DefaultMessage{}
    proto.Unmarshal(respmessage.Payload, resultMsg)

    if resultMsg.Error != "" {
        return fiber.NewError(fiber.StatusBadRequest, resultMsg.Error)
    }

    if resultMsg.ErrorInternal != "" {
        log.Printf("SynchronousEndpoint Error: failed on result: %v", resultMsg.ErrorInternal)
        return fiber.NewError(fiber.StatusInternalServerError, "Internal Server Error")
    }

    attachmentData := &map[string]interface{}{}
    json.Unmarshal(resultMsg.Data, attachmentData)

    return c.Status(fiber.StatusOK).JSON(&model.Response{
        Message: "success",
        Data:    attachmentData,
    })
}
```

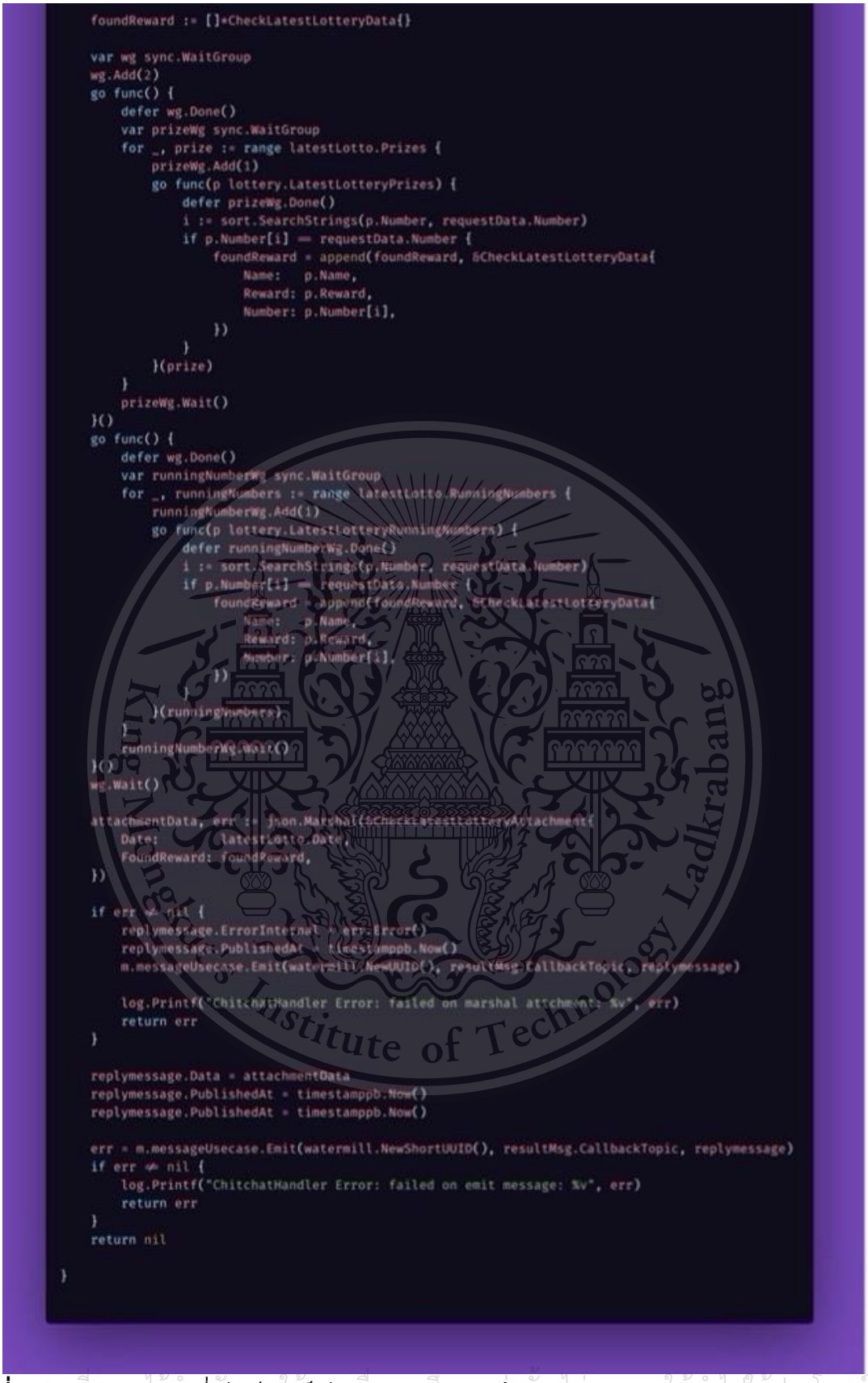
รูปที่ 32 Code API Gateway ในส่วนของ Http Handler สำหรับการทำงานแบบ Synchronous



รูปที่ 33 Database

พัฒนาเซอร์วิสเพื่อฟีเจอร์ตรวจสอบผลสลากกินแบ่งรัฐบาลโดยฟีเจอร์นี้จะรองรับการทำงานแบบ Synchronous เซอร์วิสนี้จะทำการติดตาม Topic newMessage จาก Message Queue เมื่อได้รับ Message ใหม่จะทำการตรวจสอบว่าฟีเจอร์ของ Message นี้ตรงกับฟีเจอร์ของเซอร์วิสหรือไม่ หากไม่ตรงก็จะข้ามไป แต่ถ้าหากตรงก็ทำการตรวจสอบรูปแบบการทำงานที่แนบมาด้วยว่าถูกต้องหรือไม่ หากไม่ถูกต้องจะตอบกลับไปพร้อมกับแนบข้อความความผิดพลาดกลับไป ถ้าตรวจสอบแล้วทั้งหมด ถัดมาจะตรวจสอบข้อมูลที่แนบมาด้วยว่ามี Field ข้อมูลตรงตามต้องการหรือไม่สำหรับเซอร์วิสนี้ Field ที่ต้องการคือ Number หากไม่มีก็จะแจ้งข้อผิดพลาดกลับไป หากมีเซอร์วิสก็จะทำการดึงผลรางวัลของงวดล่าสุดและนำเลขที่ส่งมานำมาค้นหาว่าตรงกับผลรางวัลไหนหรือไม่ เมื่อหาเสร็จแล้วจะส่งผลลัพธ์การค้นหากลับไปยัง Message Queue โดยใช้ Topic ที่กำหนดมา

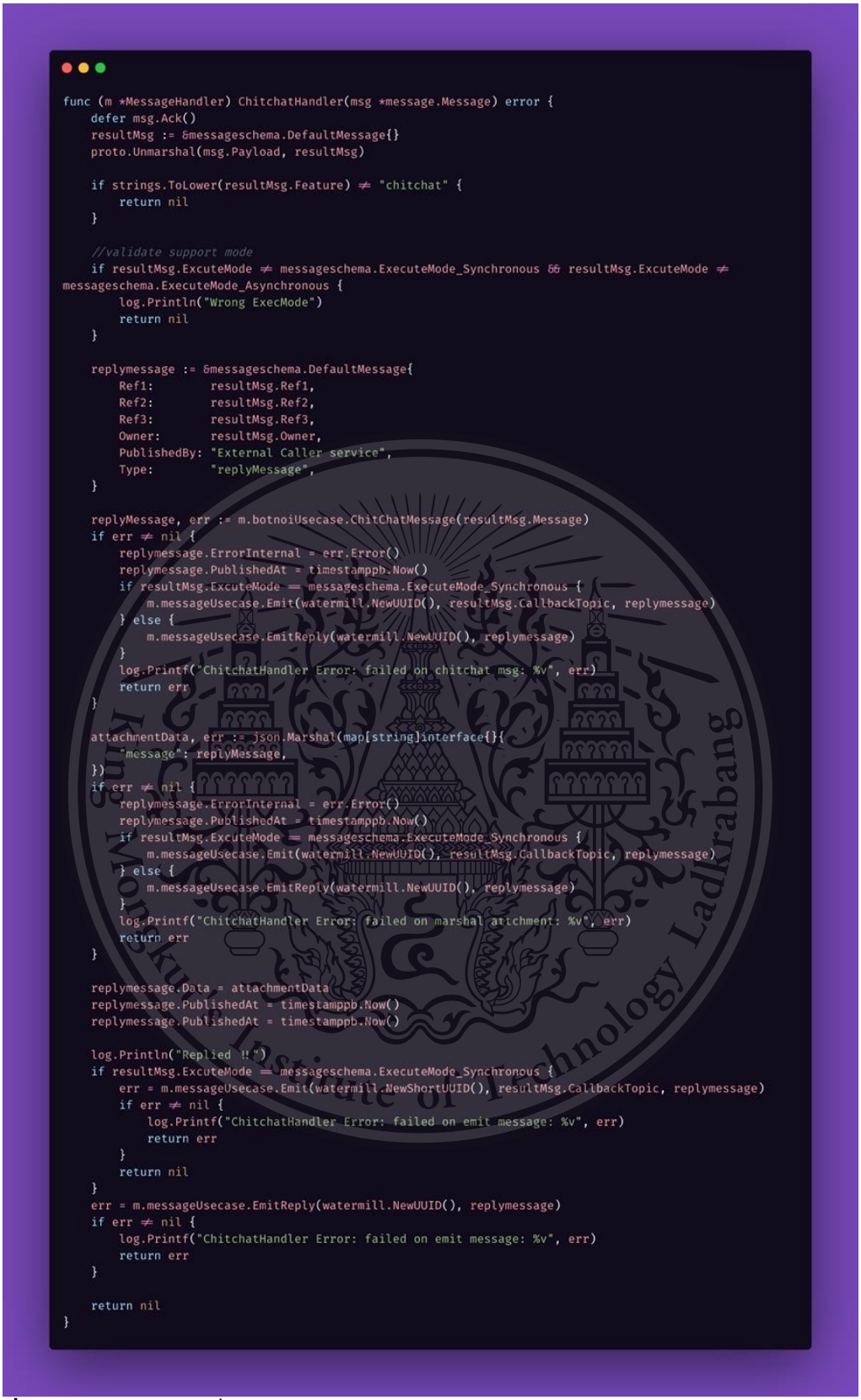
```go
type CheckLatestLotteryRequestData struct {
    Number string `json:"number,omitempty"`
}

type CheckLatestLotteryAttachment struct {
    Date        string                    `json:"date,omitempty"`
    FoundReward []*CheckLatestLotteryData `json:"foundReward,omitempty"`
}

type CheckLatestLotteryData struct {
    Name   string `json:"name,omitempty"`
    Reward string `json:"reward,omitempty"`
    Number string `json:"number,omitempty"`
}

func (m *MessageHandler) CheckLatestLotteryHandler(msg *message.Message) error {
    defer msg.Ack()
    resultMsg := &messageschema.DefaultMessage{}
    proto.Unmarshal(msg.Payload, resultMsg)

    if strings.ToLower(resultMsg.Feature) != featureName {
        return nil
    }

    replymessage := &messageschema.DefaultMessage{
        Ref1:        resultMsg.Ref1,
        Ref2:        resultMsg.Ref2,
        Ref3:        resultMsg.Ref3,
        Owner:       resultMsg.Owner,
        PublishedBy: fmt.Sprintf("External Caller service: %v", featureName),
        Type:        "ReplyMessage",
    }

    //validate supported mode
    if resultMsg.ExcuteMode != messageschema.ExecuteMode_Synchronous {
        log.Println("Wrong ExecMode")
        replymessage.Error = "Wrong exec mode"
        replymessage.PublishedAt = timestamppb.Now()
        m.messageUsecase.Emit(watermill.NewUUID(), resultMsg.CallbackTopic, replymessage)
        return nil
    }

    requestData := &CheckLatestLotteryRequestData{}
    err := json.Unmarshal(resultMsg.Data, requestData)
    if err != nil {
        replymessage.ErrorInternal = err.Error()
        replymessage.PublishedAt = timestamppb.Now()
        m.messageUsecase.Emit(watermill.NewUUID(), resultMsg.CallbackTopic, replymessage)
        log.Printf("CheckLatestLotteryHandler Error: failed on get latest lotto: %v", err)
        return err
    }

    if requestData.Number == "" {
        replymessage.Error = "required lottery number"
        replymessage.PublishedAt = timestamppb.Now()
        m.messageUsecase.Emit(watermill.NewUUID(), resultMsg.CallbackTopic, replymessage)
        return nil
    }

    latestLotto, err := m.lotteryUsecase.GetLatestLottery()
    if err != nil {
        replymessage.ErrorInternal = err.Error()
        replymessage.PublishedAt = timestamppb.Now()
        m.messageUsecase.Emit(watermill.NewUUID(), resultMsg.CallbackTopic, replymessage)
        log.Printf("CheckLatestLotteryHandler Error: failed on get latest lotto: %v", err)
        return err
    }
```



```
foundReward := []*CheckLatestLotteryData{}

var wg sync.WaitGroup
wg.Add(2)
go func() {
    defer wg.Done()
    var prizeWg sync.WaitGroup
    for _, prize := range latestLotto.Prizes {
        prizeWg.Add(1)
        go func(p lottery.LatestLotteryPrizes) {
            defer prizeWg.Done()
            i := sort.SearchStrings(p.Number, requestData.Number)
            if p.Number[i] == requestData.Number {
                foundReward = append(foundReward, &CheckLatestLotteryData{
                    Name:   p.Name,
                    Reward: p.Reward,
                    Number: p.Number[i],
                })
            }
        }(prize)
    }
    prizeWg.Wait()
}()
go func() {
    defer wg.Done()
    var runningNumberWg sync.WaitGroup
    for _, runningNumbers := range latestLotto.RunningNumbers {
        runningNumberWg.Add(1)
        go func(p lottery.LatestLotteryRunningNumbers) {
            defer runningNumberWg.Done()
            i := sort.SearchStrings(p.Number, requestData.Number)
            if p.Number[i] == requestData.Number {
                foundReward = append(foundReward, &CheckLatestLotteryData{
                    Name:   p.Name,
                    Reward: p.Reward,
                    Number: p.Number[i],
                })
            }
        }(runningNumbers)
    }
    runningNumberWg.Wait()
}()
wg.Wait()

attachmentData, err := json.Marshal(&CheckLatestLotteryAttachment{
    Date:        latestLotto.Date,
    FoundReward: foundReward,
})

if err != nil {
    replymessage.ErrorInternal = err.Error()
    replymessage.PublishedAt = timestamppb.Now()
    m.messageUsecase.Emit(watermill.NewUUID(), resultMsg.CallbackTopic, replymessage)

    log.Printf("ChitchatHandler Error: failed on marshal attchment: %v", err)
    return err
}

replymessage.Data = attachmentData
replymessage.PublishedAt = timestamppb.Now()
replymessage.PublishedAt = timestamppb.Now()

err = m.messageUsecase.Emit(watermill.NewShortUUID(), resultMsg.CallbackTopic, replymessage)
if err != nil {
    log.Printf("ChitchatHandler Error: failed on emit message: %v", err)
    return err
}
return nil

}
```

รูปที่ 34 Code Service ที่เป็นฟีเจอร์เช็กผลสลากกินแบ่งรัฐบาลส่วนของ Message Handler สำหรับการทำงานแบบ Synchronous

พัฒนาเซอร์วิสเพื่อฟีเจอร์แชตบอตสำหรับคุยเล่น ฟีเจอร์นี้จะรองรับการทำงานทั้งแบบ Synchronous และ Asynchronous เซอร์วิสนี้จะทำการติดตาม Topic newMessage จาก Message Queue เมื่อได้รับ Message ใหม่ จะทำการตรวจสอบว่าฟีเจอร์ของ Message นี้ตรงกับฟีเจอร์ของเซอร์วิสหรือไม่ หากไม่ตรงทำการข้ามไป เนื่องจากเซอร์วิสนี้รองรับการทำงานทั้ง Synchronous และ Asynchronous จึงไม่จำเป็นต้องตรวจสอบการทำงานหลังจากนั้น เซอร์วิสจะนำข้อความนี้แบบมาใน Message ส่งไปยังเซอร์วิสของ Botnoi เพื่อให้ประมวลผลข้อความและเมื่อได้ข้อความผลลัพธ์ของการประมวลผลหากการทำงานเป็นแบบ Synchronous ก็ทำการส่ง Message ตอบกลับไปยัง Topic ที่กำหนดแต่ถ้าเป็นแบบ Asynchronous ก็ทำการส่ง Message ตอบกลับไปยัง Topic replyMessage



```go
func (m *MessageHandler) ChitchatHandler(msg *message.Message) error {
    defer msg.Ack()
    resultMsg := &messageschema.DefaultMessage{}
    proto.Unmarshal(msg.Payload, resultMsg)

    if strings.ToLower(resultMsg.Feature) != "chitchat" {
        return nil
    }

    //validate support mode
    if resultMsg.ExcuteMode != messageschema.ExecuteMode_Synchronous && resultMsg.ExcuteMode !=
messageschema.ExecuteMode_Asynchronous {
        log.Println("Wrong ExecMode")
        return nil
    }

    replymessage := &messageschema.DefaultMessage{
        Ref1:        resultMsg.Ref1,
        Ref2:        resultMsg.Ref2,
        Ref3:        resultMsg.Ref3,
        Owner:       resultMsg.Owner,
        PublishedBy: "External Caller service",
        Type:        "replyMessage",
    }

    replyMessage, err := m.botnoiUsecase.ChitChatMessage(resultMsg.Message)
    if err != nil {
        replymessage.ErrorInternal = err.Error()
        replymessage.PublishedAt = timestamppb.Now()
        if resultMsg.ExcuteMode == messageschema.ExecuteMode_Synchronous {
            m.messageUsecase.Emit(watermill.NewUUID(), resultMsg.CallbackTopic, replymessage)
        } else {
            m.messageUsecase.EmitReply(watermill.NewUUID(), replymessage)
        }
        log.Printf("ChitchatHandler Error: failed on chitchat msg: %v", err)
        return err
    }

    attachmentData, err := json.Marshal(map[string]interface{}{
        "message": replyMessage,
    })
    if err != nil {
        replymessage.ErrorInternal = err.Error()
        replymessage.PublishedAt = timestamppb.Now()
        if resultMsg.ExcuteMode == messageschema.ExecuteMode_Synchronous {
            m.messageUsecase.Emit(watermill.NewUUID(), resultMsg.CallbackTopic, replymessage)
        } else {
            m.messageUsecase.EmitReply(watermill.NewUUID(), replymessage)
        }
        log.Printf("ChitchatHandler Error: failed on marshal attchment: %v", err)
        return err
    }

    replymessage.Data = attachmentData
    replymessage.PublishedAt = timestamppb.Now()
    replymessage.PublishedAt = timestamppb.Now()

    log.Println("Replied !!")
    if resultMsg.ExcuteMode == messageschema.ExecuteMode_Synchronous {
        err = m.messageUsecase.Emit(watermill.NewShortUUID(), resultMsg.CallbackTopic, replymessage)
        if err != nil {
            log.Printf("ChitchatHandler Error: failed on emit message: %v", err)
            return err
        }
        return nil
    }
    err = m.messageUsecase.EmitReply(watermill.NewUUID(), replymessage)
    if err != nil {
        log.Printf("ChitchatHandler Error: failed on emit message: %v", err)
        return err
    }

    return nil
}
```

รูปที่ 35 Code Service ที่เป็นฟีเจอร์แชตบอตคุยเล่น ส่วนของ Message Handler สำหรับการทำงานที่รองรับทั้งแบบ Synchronous, Asynchronous

บทที่ 4

ผลการดำเนินการ

4.1 ภาพรวมระบบ

ระบบประมวลผลข้อความผ่านช่องทางเอพีไอนี้ช่วยให้นักพัฒนาซอฟต์แวร์เข้าถึงและนำไปใช้งานได้อย่างง่ายดายเพราะเป็นระบบที่มีฟีเจอร์พร้อมใช้งาน โดยระบบนี้ออกแบบให้เป็น Microservice โดยอ้างอิงแนวคิด Event Drive Architecture

Microservice หรือ Microservice Architecture คือสถาปัตยกรรมการออกแบบ Service หรือการออกแบบซอฟต์แวร์ โดยที่ชื่อมีคำว่า Micro นำหน้าอยู่ก็เพราะว่าเป็นการออกแบบที่ทำให้ Service มีขนาดเล็กเพื่อแก้ไขจุดด้อยของสถาปัตยกรรมการออกแบบอื่น ๆ ซึ่ง Microservice จะต้องมี 1 ดาต้าเบส (Database) มักจะไม่ใช้ดาต้าเบสร่วมกันแต่ในโครงงานชิ้นนี้เราได้ใช้ดาต้าเบสตัวเดียวเพราะโครงงานชิ้นนี้เป็นเพียงตัวทดลองและเนื่องจากมีรีซอร์ส (Resource) และค่าใช้จ่ายไม่เพียงพอต่อการใช้หลายดาต้าเบสจึงใช้เพียงแค่ดาต้าเบสเดียวเท่านั้น

จากการดำเนินการจะได้ระบบประมวลผลข้อความผ่านช่องทางเอพีไอที่มีฟีเจอร์ตรวจสอบผลสลากกินแบ่งรัฐบาล, แชตบอตสำหรับคุยเล่น ซึ่งจะใช้งานผ่าน RESTful API และระบบนี้จะแบ่งโครงสร้างออกเป็น 3 ส่วน ดังนี้

1. API Gateway Service สำหรับรับคำสั่งจากภายนอก
2. Message Queue โดย Message Queue จะทำหน้าที่เป็น Event bus คอยส่ง-รับ Event จาก API Gateway ส่งให้ Service ปลายทาง
3. Service ปลายทาง/อื่น ๆ ในระบบ คือ Service ที่พัฒนา Feature ต่าง ๆ ให้กับ Open API

4.2 ความสามารถของระบบในการยืนยันตัวตน

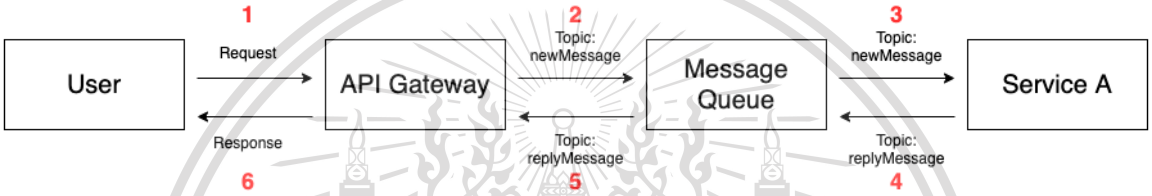
สามารถลงทะเบียนเพื่อเพิ่มผู้ใช้งานได้ การยืนยันตัวตนทำให้สามารถจำกัดผู้ใช้งานในระบบและระบุตัวตนได้ ทำให้เข้าถึงข้อมูลของผู้ใช้งานได้จากเจ้าของเพียงผู้เดียว สามารถป้องกันการเข้าถึงข้อมูลผู้ใช้งานจากผู้ไม่หวังดีได้ โดยการใช้งานบริการแต่ละครั้ง ผู้ใช้งานต้องแนบ token มาในคำขอการใช้งานเพื่อยืนยันตัวตนทุกครั้งด้วย

4.3 ความสามารถของระบบในการทำงานแบบ Synchronus และ Asynchronus

การทำงานแบบ Synchronous คือเมื่อมีการทำงานจะเริ่มทำงานนั้นจนกระทั่งงานนั่นเสร็จสิ้นถึงจะทำงานอื่นต่อได้เพราะมีการใช้สัญญาณนาฬิการ่วมกัน อีกทั้งงานที่เริ่มทำไปแล้วก็ไม่สามารถที่

จะสลับไปทำงานอื่นได้ หากมีการส่งค่าให้หน่วยประมวลผลอื่นต้องรอจนกว่าผลจะกลับมาแล้วจึงทำงานต่อ ไม่สามารถสลับงานได้จึงทำให้การทำงานเกิดความล่าช้าและในปกติการทำงานในรูปแบบ Synchronous API Gateway จำเป็นต้องรู้จักกับ Service ปลายทางทำให้ไม่มีความยืดหยุ่นและเมื่อต้องการเพิ่ม Service ใหม่เข้าไปอาจจะต้องกลับไปแก้ไขหรืออัปเดต Open API ด้วย

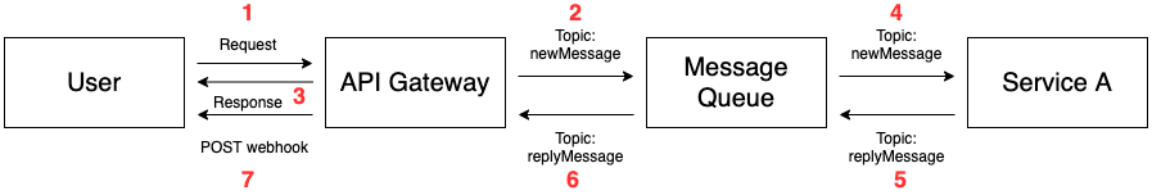
ซึ่งในโครงงานนี้การทำงานของ Synchronous เมื่อมีการเรียกใช้งาน Open API ตัว API Gateway จะสร้าง Message จาก Schema ที่กำหนดโดยระบุการทำงานเป็น Synchronous และระบุ Topic ที่ใช้สำหรับส่ง Message กลับมาโดยเฉพาะและส่ง Message นี้เข้าไปยัง Message Queue แล้วทำการติดตาม Topic ที่ถูกสร้างขึ้นมาใหม่โดยเฉพาะ หลังจากเซอร์วิสปลายทางประมวลผลเสร็จแล้วจะส่งกลับมาที่ Message ใหม่ที่ระบุไปเมื่อ API Gateway ได้รับ Message ผ่านจาก Topic นั้นแล้วจะทำการตอบกลับไปยังผู้ที่เรียกใช้งาน Open API ทันที



รูปที่ 36 รูปแบบการทำงานของ Synchronous

การทำงานแบบ Asynchronous คือการทำงานที่เราสามารถจะสลับไปทำงานอื่นได้ ในขณะที่รอให้งานหนึ่งเสร็จเพราะไม่จำเป็นต้องพึ่งสัญญาณนาฬิกาในการทำงาน เมื่องานนั้นประมวลผลเสร็จแล้วเราก็จะรับผลของงานนั้น แล้วทำต่อได้โดยปกติ งานที่เราทำค้างไว้จะถูกเก็บสถานะและกลับมาทำต่อภายหลังได้

ซึ่งในโครงงานนี้การทำงานแบบ Asynchronous API Gateway จะสร้าง Message จาก Schema ที่กำหนดโดยระบุรูปแบบการทำงานเป็น Asynchronous และส่ง Message นี้เข้าไปยัง Message Queue แล้วจะตอบกลับไปยังผู้ที่เรียกใช้งาน Open API ทันที และหลังจากที่เซอร์วิสภายในประมวลผลข้อความเสร็จแล้วจะส่ง message ผ่าน Message Queue กลับมาที่ API Gateway และ API Gateway จะดูข้อมูลของเจ้าของ Message นี้ใน Database และนำข้อมูลใน Message ส่งกลับไปยังเจ้าของ Message ผ่าน RESTful API



รูปที่ 37 รูปแบบการทำงานของ Asynchronous

4.4 ความสามารถของฟีเจอร์ตรวจสอบผลสลากกินแบ่งรัฐบาล

ความสามารถของฟีเจอร์ตรวจสอบผลสลากกินแบ่งรัฐบาลคือ สามารถตรวจเช็กเลขผลสลากกินแบ่งโดยกรอกเลขผลสลากกินแบ่งเข้าไปก็สามารถรู้ว่าถูกรางวัลหรือไม่ ซึ่งสามารถตรวจเช็กได้ทุก ๆ รางวัลแต่จะสามารถตรวจเช็กได้เพียงแค่งวดล่าสุดเท่านั้น โดยจุดประสงค์ของการทำฟีเจอร์ตรวจสอบผลสลากกินแบ่งรัฐบาลนี้ขึ้นมาเพื่อเป็นตัวอย่างการทำฟีเจอร์สำหรับ Open API เพียงเท่านั้น ทำให้เห็นว่าเวลานักพัฒนาจะพัฒนาฟีเจอร์ใหม่ ๆ ไม่จำเป็นต้องเข้าไปแก้ไขหรืออัปเดตเซอร์วิสเก่าที่เคยทำไว้ และเป็นการบอกว่า Architecture ของระบบนี้ให้มีความยืดหยุ่นและไม่ยึดติดอยู่กับสิ่งใดในการใช้งาน

ฟีเจอร์ตรวจสอบผลสลากกินแบ่งรัฐบาลคือฟีเจอร์นี้รองรับการทำงานแบบ Synchronous เซอร์วิสนี้จะทำการติดตาม Topic newMessage จาก Message Queue เมื่อได้รับ Message ใหม่จะทำการตรวจสอบว่าฟีเจอร์ของ Message นี้ตรงกับฟีเจอร์ของเซอร์วิสหรือไม่ หากไม่ตรงก็จะข้ามไป แต่ถ้าหากตรงก็ทำการตรวจสอบรูปแบบการทำงานที่แนบมาด้วยว่าถูกต้องหรือไม่ หากไม่ถูกต้องจะตอบกลับไปพร้อมกับแนบข้อความความผิดพลาดกลับไป ถ้าตรวจสอบแล้วทั้งหมด ถัดมาจะตรวจสอบข้อมูลที่แนบมาด้วยว่ามี Field ข้อมูลตรงตามต้องการหรือไม่สำหรับเซอร์วิสนี้ Field ที่ต้องการคือ Number หากไม่มีก็จะแจ้งข้อผิดพลาดกลับไป หากมีเซอร์วิสก็จะทำการดึงผลรางวัลของงวดล่าสุดและนำเลขที่ส่งมานำมาค้นหาว่าตรงกับผลรางวัลไหนหรือไม่ เมื่อหาเสร็จแล้วจะส่งผลลัพธ์การค้นหากลับไปยัง Message Queue โดยใช้ Topic ที่กำหนดมา

4.5 ความสามารถของฟีเจอร์แชตบอตคุยเล่น

ความสามารถของฟีเจอร์แชตบอตสำหรับคุยเล่นคือ ซอฟต์แวร์ที่พัฒนาขึ้นมาเพื่อช่วยตอบกลับการสนทนาผ่านข้อความแบบอัตโนมัติและรวดเร็ว ฟีเจอร์นี้สามารถพูดคุยเล่นได้ทั่วไปเมื่อใดก็ได้ที่เราต้องการ โดยจุดประสงค์ของการทดลองสร้าง LINE Chatbot เพื่อลองใช้ Message Processing API ที่สร้างขึ้นมาทำให้รู้สึกว่าการสร้าง LINE Chatbot นั้นง่าย สะดวก และรวดเร็วขึ้นเป็นอย่างมาก เพราะในส่วนของ Message Processing API มีการรองรับการใช้งานมากพออยู่แล้ว ทำให้ไม่ต้องเสียเวลาเขียน code หรือพัฒนาเพื่อสร้างฟีเจอร์ขึ้นมาเองใหม่ทั้งหมด นักพัฒนาสามารถนำ Open API นี้ไปใช้งานได้เลยหากต้องการ อีกทั้งทำให้เห็นว่า Architecture ของระบบนี้ให้มีความยืดหยุ่นและไม่ยึดติดอยู่กับสิ่งใดในการใช้งาน

ฟีเจอร์แชตบอตสำหรับคุยเล่น ฟีเจอร์นี้จะรองรับการทำงานทั้งแบบ Synchronous และ Asynchronous เซอร์วิสนี้จะทำการติดตาม Topic newMessage จาก Message Queue เมื่อได้รับ Message ใหม่ จะทำการตรวจสอบว่าฟีเจอร์ของ Message นี้ตรงกับฟีเจอร์ของเซอร์วิสหรือไม่ หาก

ไม่ตรงทำการข้ามไป เนื่องจากเซอร์วิสนี้รองรับการทำงานทั้ง Synchronous และ Asynchronous จึงไม่จำเป็นต้องตรวจสอบการทำงานหลังจากนั้น เซอร์วิสจะนำข้อความนี้แบบมาใน Message ส่งไปยังเซอร์วิสของ Botnoi เพื่อให้ประมวลผลข้อความและเมื่อได้ข้อความผลลัพธ์ของการประมวลผลหากการทำงานเป็นแบบ Synchronous ก็ทำการส่ง Message ตอบกลับไปยัง Topic ที่กำหนดแต่ถ้าเป็นแบบ Asynchronous ก็ทำการส่ง Message ตอบกลับไปยัง Topic replyMessage

รูปที่ 38 ตัวอย่างฟีเจอร์แชตบอตคุยเล่น

บทที่ 5

สรุปผลการดำเนินงานและข้อเสนอแนะ

ตลอดระยะเวลาการดำเนินงานวิเคราะห์รูปแบบการทำงานไปจนถึงการปรับปรุงกระบวนการต่าง ๆ จนกระทั่งสิ้นสุดปริญญานิพนธ์ สามารถสรุปผลการดำเนินงาน ปัญหาและอุปสรรคในการดำเนินงาน วิธีการแก้ปัญหา และข้อเสนอแนะ ในอนาคตได้ดังต่อไปนี้

5.1 สรุปผลการดำเนินงาน

จากการทดลองออกแบบโครงสร้างของ Open API ในรูปแบบ Microservice โดยใช้ Message Queue เป็นสื่อกลางสำหรับการเชื่อมต่อภายใน และมีการกำหนดรูปแบบของดาต้าโมเดลสำหรับสร้างข้อความที่ใช้ส่งผ่าน message queue ส่งผลทำให้การพัฒนา Open API ทำได้อย่างราบรื่นขึ้น, ใช้เวลาพัฒนาน้อยลง และรองรับการทำงานทั้งรูปแบบ Synchronous และ Asynchronous โดยเมื่อเราต้องการเพิ่มฟีเจอร์หรือเซอร์วิสใหม่ ๆ เข้าไปในระบบเราไม่จำต้องกลับไปแก้ไขโค้ดเดิมเลยแม้แต่น้อย และการกำหนดรูปแบบการทำงานก็สามารถทำได้อย่างง่ายดายและรองรับได้ทั้งสองโหมดการทำงานพร้อมกัน ไม่ต้องพัฒนาหลาย ๆ รอบแยกกัน

5.2 ปัญหาและอุปสรรคในการดำเนินงาน

5.2.1 Architecture ที่ใช้ในการออกแบบและรีซอร์ส (resource) ที่ใช้ เช่น Message Queue, Kubernetes เป็นต้น ต้องการความเชี่ยวชาญสูงศึกษายากต้องใช้ระยะเวลาในการศึกษาและประสบการณ์ในการตั้งค่าในขั้นตอนที่จะติดตั้งระบบจะใช้เวลานาน และใช้งานยาก

5.2.2 ทรัพยากรที่ใช้มีค่าใช้จ่ายสูงมีแหล่งเครื่องมือที่ไม่คิดค่าใช้จ่ายมีให้ใช้น้อยทำให้ขั้นตอนการทดสอบบนคลาวด์จริงมีค่าใช้จ่ายสูง

5.2.3 วิธีการแก้ปัญหา (solution)สำหรับแก้ปัญหามีหลากหลายรูปแบบ และมีข้อจำกัดแตกต่างกันไป ไม่มีวิธีการใดไม่มีปัญหาหรือไม่มีข้อจำกัด ทำให้ขั้นตอนการพัฒนาใช้เวลาทดสอบลองผิดลองถูกเยอะ ทำให้กว่าจะหาวิธีแก้ปัญหาที่เหมาะสมที่สุดใช้เวลานาน

5.3 ข้อเสนอแนะ

5.3.1 ทางคณะควรจะมีงบประมาณสำหรับสนับสนุนการทำโครงงานที่มากกว่านี้

5.3.2 ลดขอบเขต (Scope) ของโครงงานลงเพื่อทรัพยากรที่ใช้จะได้ราคาถูกลง หรือไม่จำเป็นต้องใช้ความแม่นยำสูง

เอกสารอ้างอิง

[1] Machine Language. [ออนไลน์]. เข้าถึงได้จาก: https://sites.google.com/site/kzsq27/1.(วันที่ค้นข้อมูล: 9 มกราคม 2564)

[2] Compiler. [ออนไลน์]. เข้าถึงได้จาก: https://www.tamemo.com/post/47/compiler-1-intro/.(วันที่ค้นข้อมูล: 9 มกราคม 2564)

[3] Go Language. [ออนไลน์]. เข้าถึงได้จาก: https://www.webmasty.com/.(วันที่ค้นข้อมูล: 9 มกราคม 2564)

[4] JavaScript ES6. [ออนไลน์]. เข้าถึงได้จาก: https://www.seibottech.co.th/news/javascript-%E0%B8%84%E0%B8%B7%E0%B8%AD%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3/.(วันที่ค้นข้อมูล: 9 มกราคม 2564)

[5] Typescript. [ออนไลน์]. เข้าถึงได้จาก: https://www.babelcoder.com/blog/articles/typescript-data-types. (วันที่ค้นข้อมูล: 9 มกราคม 2564)

[6] Node.js. [ออนไลน์]. เข้าถึงได้จาก: https://beyourcyber.com/2016/node-js-is-programming-language-by-javascript/. (วันที่ค้นข้อมูล: 10 มกราคม 2564)

[7] **MongoDB**. [ออนไลน์]. เข้าถึงได้จาก: https://devahoy.com/blog/2015/08/getting-started-with-mongodb/#step1 (วันที่ค้นข้อมูล: 10 มกราคม 2564)

[8] **Kafka**. [ออนไลน์]. เข้าถึงได้จาก: https://saixiii.com/apache-kafka/. (วันที่ค้นข้อมูล: 10 มกราคม 2564)

[9] Web service. [ออนไลน์]. เข้าถึงได้จาก: https://saixiii.com/what-is-webservice/. (วันที่ค้นข้อมูล: 10 มกราคม 2564)

[10] **RESTful API**. [ออนไลน์]. เข้าถึงได้จาก: https://iamgique.medium.com/restful-api-%E0%B8%81%E0%B8%B1%E0%B8%9A-rest-api-%E0%B8%95%E0%B9%88%E0%B8%B2%E0%B8%87%E0%B8%81%E0%B8%B1%E0%B8%99%E0%B8%99%E0%B8%B0%E0%B8%A3%E0%B8%B9%E0%B9%89%E0%B8%A2%E0%B8%B1%E0%B8%87-2c70c42990e3. (วันที่ค้นข้อมูล: 15 มกราคม 2564)

[11] MQTT API. [ออนไลน์]. เข้าถึงได้จาก: https://blog.thaieasyelec.com/introduction-to-mqtt/. (วันที่ค้นข้อมูล: 15 มกราคม 2564)

[12] HTTP และ HTTPS. [ออนไลน์]. เข้าถึงได้จาก: https://www.dharmniti.co.th/https/. (วันที่ค้นข้อมูล: 15 มกราคม 2564)

[13] Protocol Buffer. [ออนไลน์]. เข้าถึงได้จาก: https://ichi.pro/th/thakhwam-kheaci-kab-protocol-buffers-74516557270343 https://www.howtoautomate.in.th/protobuf-101/. (วันที่ค้นข้อมูล: 15 มกราคม 2564)

[14] Kubernetes. [ออนไลน์]. เข้าถึงได้จาก: https://blog.nextzy.me/kubernetes-in-60-seconds-36666e1e3ef8. (วันที่ค้นข้อมูล: 15 มกราคม 2564)

[15] Docker. [ออนไลน์]. เข้าถึงได้จาก: https://www.hostpacific.com/using-docker-on-centos7/. (วันที่ค้นข้อมูล: 15 มกราคม 2564)

[16] Git. [ออนไลน์]. เข้าถึงได้จาก: https://akexorcist.medium.com/%E0%B8%A1%E0%B8%B2%E0%B9%80%E0%B8%A3%E0%B8%B5%E0%B8%A2%E0%B8%99%E0%B8%A3%E0%B8%B9%E0%B9%89-git-%E0%B9%81%E0%B8%9A%E0%B8%9A%E0%B8%87%E0%B9%88%E0%B8%B2%E0%B8%A2%E0%B9%86%E0%B8%81%E0%B8%B1%E0%B8%99%E0%B9%80%E0%B8%96%E0%B8%AD%E0%B8%B0-427398e62f82. (วันที่ค้นข้อมูล: 21 กุมภาพันธ์ 2564)

[17] SSL. [ออนไลน์]. เข้าถึงได้จาก: https://ssl.in.th/tools/about-ssl/. (วันที่ค้นข้อมูล: 21 กุมภาพันธ์ 2564)

[18] TLS. [ออนไลน์]. เข้าถึงได้จาก: https://arnondora.in.th/what-is-https-ssl-tls/ (วันที่ค้นข้อมูล: 21 กุมภาพันธ์ 2564)

[19] XSS. [ออนไลน์]. เข้าถึงได้จาก: https://www.softmelt.com/article.php?id=66#:~:text=Cross%2Dsite%20Scripting%20(XSS)%20%E0%B9%80%E0%B8%9B%E0%B9%87%E0%B8%99%E0%B8%A7%E0%B8%B4%E0%B8%98%E0%B8%B5%E0%B8%81%E0%B8%B2%E0%B8%A3%E0%B8%A2%E0%B8%AD%E0%B8%94%E0%B8%99%E0%B8%B4%E0%B8%A2%E0%B8%A1,%E0%B8%88%E0%B8%B0%E0%B8%96%E0%B8%B9%E0%B8%81%E0%B8%82%E0%B9%82%E0%B8%A1%E0%B8%A2%E0%B9%84%E0%B8%9B%E0%B9%84%E0%B8%94%E0%B9%89. (วันที่ค้นข้อมูล: 21 กุมภาพันธ์ 2564)

[20] Synchronous และ Asynchronous. [ออนไลน์]. เข้าถึงได้จาก:

https://www.gotoknow.org/posts/561181. (วันที่ค้นข้อมูล: 21 มกราคม 2564)