



Missing Linking People

a) สมาชิกกลุ่ม

กิตติธัช เด่นสกุลประเสริฐ	6609650079
พิรญาณ์ เอนอ่อน	6609650087
จตุณัฐ รัตนมงคลกุล	6609650228
ณัฐ ศีรสุวรรณกุล	6609650327
ธนกฤต พิบูลย์สวัสดิ์	6609650400
พชร พรพงศ์	6609650509
พิมพ์พิชชา เกตุศรีระ	6609650525
รพินทร์ นะราช	6609650624
สุทธิพจน์ ประทุมทอง	6609650699
สุรบดี ผาสุข	6609650707

b) ชื่อระบบโซลูชันของกลุ่ม

Missing Linking People

c) ที่มาและความสำคัญ

ในปัจจุบัน การจัดเก็บข้อมูลหรือเบาะแสที่เกี่ยวข้องกับเหตุการณ์หรือเคสต่าง ๆ มักประสบปัญหาในหลายด้าน ไม่ว่าจะเป็น ข้อมูลซ้ำซ้อน ไม่เป็นระบบ ทำให้เกิดความสับสนและใช้เวลาในการจัดการข้อมูลเพิ่มขึ้น อีกทั้งยัง ยากต่อการค้นหาเคสเก่า ๆ ส่งผลให้การติดตามความคืบหน้าหรือการตรวจสอบย้อนหลังเป็นไปอย่างล่าช้าและไม่มีประสิทธิภาพ

นอกจากนี้ ยังไม่มี ช่องทางที่ชัดเจนและเป็นระบบสำหรับให้ประชาชนทั่วไปสามารถเข้ามามีส่วนร่วมในการแจ้งเบาะแสหรือข้อมูลที่เป็นประโยชน์ ซึ่งอาจทำให้ข้อมูลที่สำคัญตกหล่นหรือสูญหายไปโดยไม่ทันได้ดำเนินการใด ๆ

ด้วยเหตุนี้ จึงมีความจำเป็นต้องพัฒนา ระบบหรือเครื่องมือที่สามารถจัดการข้อมูลให้เป็นระบบมากขึ้น ค้นหาได้ง่าย มีช่องทางเปิดรับข้อมูลจากสาธารณชนอย่างมีประสิทธิภาพ และสามารถลดการตกหล่นของข้อมูลได้อย่างมีประสิทธิภาพ เพื่อสนับสนุนการทำงานให้รวดเร็วและแม่นยำยิ่งขึ้น อันจะนำไปสู่การแก้ไขปัญหาต่าง ๆ ได้อย่างตรงจุดและยั่งยืน

d) กรณีการใช้งาน (Use cases) และประโยชน์สำหรับผู้ใช้

1.กรอกตามปกติ

ผู้ใช้งานเข้ามาแจ้งข้อมูลคนหายหรือเบาะแสตามขั้นตอนที่กำหนดไว้ โดยกรอกข้อมูลครบถ้วน และเริ่มต้นด้วยคำว่า “แจ้งคนหาย” หรือ “แจ้งเบาะแส”

ประโยชน์สำหรับผู้ใช้

1. ระบบสามารถจัดหมวดหมู่ข้อมูลได้อย่างถูกต้องและแม่นยำ
2. ลดความผิดพลาดในการสื่อสารหรือการประมวลผลข้อมูล

2.ก่อนเริ่มไม่ได้พิมพ์”แจ้งคนหาย” หรือ “แจ้งเบาะแส”

หากผู้ใช้งานไม่ได้พิมพ์เริ่มต้นด้วยคำว่า “แจ้งคนหาย” หรือ “แจ้งเบาะแส” ระบบจะไม่สามารถดำเนินการต่อได้และ จะวนกลับมาถามซ้ำจนกว่าผู้ใช้งานจะพิมพ์คำที่กำหนดเพื่อเริ่มใช้งาน

ประโยชน์สำหรับผู้ใช้

1. ช่วยให้ผู้ใช้ไม่กรอกข้อมูลผิดขั้นตอน
2. ป้องกันความสับสนในการใช้งานระบบ

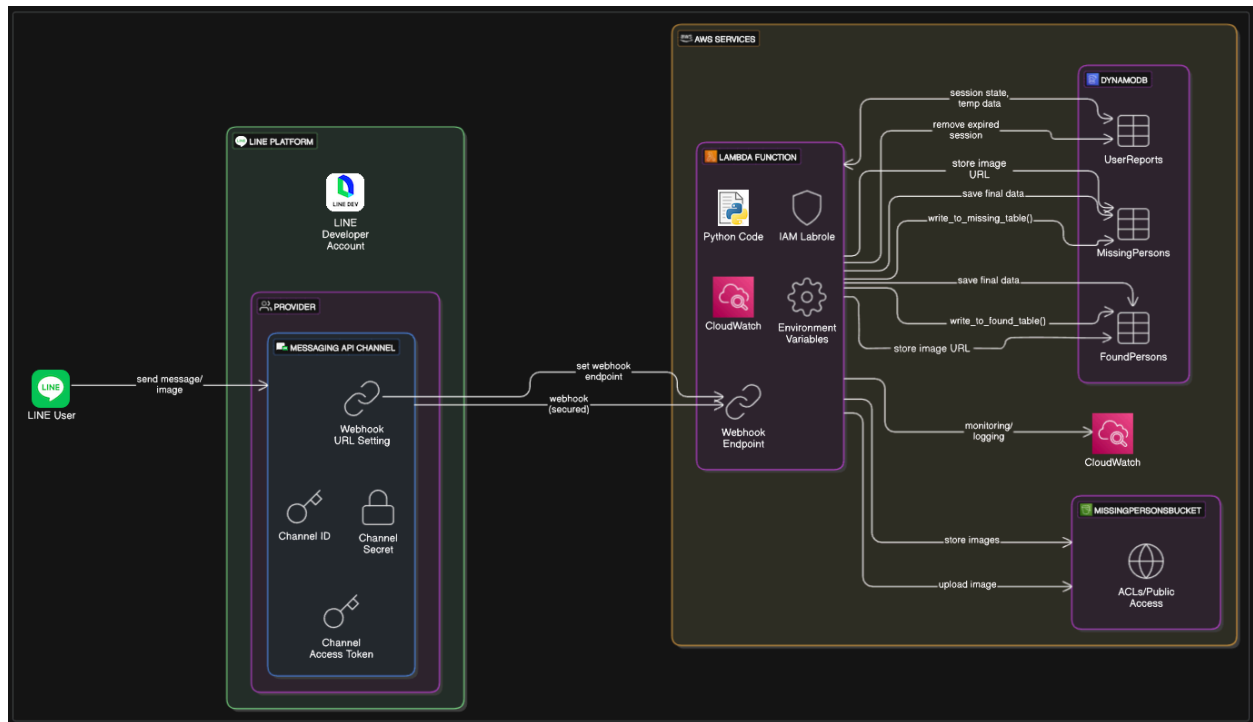
3.กรณีผู้ใช้ไม่อัปเดตจากข้อความล่าสุดภายใน 5 นาที

ผู้ใช้งานหยุดกรอกข้อมูลกลางคันหรือลืมส่งข้อมูลต่อ หลังพิมพ์ข้อความล่าสุดแล้ว ไม่มีการตอบสนองภายใน 5 นาที

ประโยชน์สำหรับผู้ใช้

1. เพิ่มความปลอดภัยของข้อมูลโดยลดความเสี่ยงจากการกรอกค้าง
2. ช่วยให้ข้อมูลที่เข้าสู่ระบบมีความครบถ้วนมากขึ้น

e) สถาปัตยกรรมระบบที่แสดงให้เห็นถึงบริการ AWS ที่ใช้ พร้อมคำอธิบายการทำงานร่วมกันขององค์ประกอบต่าง ๆ ในสถาปัตยกรรม



AWS Lambda : ใช้สำหรับการสร้างฟังก์ชัน และ Create webhook URLสำหรับการทำงานร่วมกับ LINE

Amazon CloudWatch : ใช้ตรวจสอบ Debug และเช็คข้อมูลที่จะใส่ใน Database เพื่อในกรณีที่เกิดปัญหาต่าง ๆ ขึ้นมา

Amazon S3 : ใช้ในการเก็บรูปภาพที่ผู้ใช้ส่งมาใน Line

Amazon DynamoDB Database : ใช้เก็บข้อมูลทั้งหมดที่ User ตอบลงใน Database กลาง

f) ขั้นตอนหลักที่สำคัญในการสร้างและตั้งค่าแต่ละองค์ประกอบ (เน้นให้ผู้อ่านสามารถดำเนินการตามได้ แม้จะไม่ได้ลงรายละเอียดทั้งหมด)

1. การสร้าง LINE Messaging API

- เข้าเว็บไซต์ <https://developers.line.biz/en/>
- สร้าง Provider ใหม่ จากนั้นสร้าง Channel ประเภท **Messaging API**
 - ชื่อ Channel: MissingLinking
 - คำอธิบาย: MissingLinking
 - ประเทศ: Thailand
- คัดลอกค่าต่อไปนี้มาใช้ภายหลัง:
 - Channel ID
 - Channel Secret (จากแท็บ Basic settings)
 - Bot Basic ID และ Channel Access Token (long-lived) (จากแท็บ Messaging API)

2. การสร้าง Lambda Function

- เข้าหน้า AWS Lambda และกด Create function
 - Function name: MissingPerson
 - Runtime: Python 3.13
 - Execution role: เลือก Use an existing role → LabRole
 - เปิดใช้งาน Function URL → Auth type: NONE
- บันทึกค่า Function URL ไว้ใช้เป็น Webhook URL
 - อัปโหลดโค้ด Lambda เป็น .zip file
 - ตั้งค่า Environment Variables:
 - YOUR_CHANNEL_ACCESS_TOKEN → ใส่ token จาก LINE
 - YOUR_CHANNEL_SECRET → ใส่ secret จาก LINE

3. การเชื่อมต่อ Lambda กับ LINE Webhook

- กลับไปที่ LINE Developers → Tab Messaging API
- เปิด Use webhook เป็น True
- วาง Lambda Function URL ลงใน Webhook URL

4. การสร้างฐานข้อมูล DynamoDB

- สร้างทั้งหมด 3 ตารางใน AWS DynamoDB:
 1. MissingPersons → Partition key: person_id (String)
 2. FoundPersons → Partition key: person_id (String)
 3. UserReports → Partition key: user_id (String)
- ใช้ Default settings ทั้งหมด
- หากเปลี่ยนชื่อตาราง ต้องแก้ใน Lambda โค้ดบรรทัดที่ 13–15 แล้วกด Deploy

5. การสร้าง S3 Bucket

- เข้าหน้า Amazon S3 → Create bucket
 - Bucket name: missingpersonsbucket
 - เปิด Object Ownership แบบ ACLs enabled
 - ปิดการตั้งค่า Block all public access และยอมรับคำเตือน
 - หากเปลี่ยนชื่อ bucket ต้องแก้ในโค้ด Lambda บรรทัดที่ 10 แล้ว Deploy ใหม่
- ใช้สำหรับเก็บรูปภาพที่อัปโหลดจาก LINE โดยจะมีการเชื่อมโยงกับข้อมูลใน DynamoDB ผ่าน URL ของรูป

6. การตรวจสอบการทำงานผ่าน CloudWatch

- กลับไปหน้า Lambda → Tab Monitor → เลือก View CloudWatch Logs
 - ตรวจสอบ Log Streams ล่าสุด เพื่อดูว่า Lambda รับข้อความ/รูปภาพจาก LINE แล้วทำงานถูกต้องหรือไม่

g) ผลการทดสอบการทำงานของระบบโซลูชันตามมุมมองของผู้ใช้ (end-to-end) ทั้งหมด 3 กรณี ควรมีอย่างน้อย 1 กรณีในนี้ที่แสดงถึงกรณีที่สามารถเกิดข้อผิดพลาด และระบบรับมือข้อผิดพลาดอย่างไร

1. ระบบมีการจำกัดเวลาทำให้มีปัญหาสำหรับผู้ที่พิมพ์ช้าหรือมีรายละเอียดของข้อมูล ทำให้การทำผ่านฟอร์มในบางสถานการณ์ตอบโทย่มากกว่าเนื่องจากถ้าไม่มีการจำกัดเวลาจะทำให้ข้อมูลใน lamda จะเก็บตัวแปรเข้า table ต่อเนื่องไปเรื่อยๆทำให้ข้อมูลที่ยังกรอกไม่เสร็จยังคงอยู่

2. ผู้ใช้เริ่มต้นใช้งานโดยไม่ได้พิมพ์คำว่า “แจ้งคนหาย” หรือ “แจ้งเบาะแส” ตามที่ระบบกำหนด ส่งผลให้ระบบไม่สามารถเข้าสู่ขั้นตอนการแจ้งข้อมูลได้ และจะวนเข้าคำถามเดิม แสดงให้เห็นถึงกรณีที่ผู้ใช้ไม่ทำตามรูปแบบที่กำหนด ซึ่งอาจก่อให้เกิดความสับสน

3. ผู้ใช้กรอกข้อมูลครบถ้วนตามที่ระบบถามผ่าน LINE Bot จนสิ้นสุดขั้นตอน ระบบสามารถจัดเก็บข้อมูลใน DynamoDB และ ไฟล์รูปภาพใน S3 ได้อย่างถูกต้อง

h) บทวิเคราะห์ข้อดีและข้อเสียของระบบโดยอิงตาม AWS Well-Architected Framework ทั้งในด้านการปฏิบัติงาน ความปลอดภัย ความน่าเชื่อถือ ประสิทธิภาพการทำงาน ค่าใช้จ่าย และความยั่งยืน

1. การปฏิบัติงาน (Operational Excellence)

ข้อดี:

- มีการใช้ CloudWatch ซึ่งรองรับการตรวจสอบการทำงานได้ง่ายในกรณีที่เกิดปัญหา
- การแยกฟังก์ชันออกเป็น Lambda และการใช้ DynamoDB ตามชนิดของข้อมูล ช่วยให้ระบบดูแลง่ายและแยกส่วนการ debug ได้สะดวก

ข้อเสีย:

- ระบบไม่มีการทวนข้อมูลที่ผู้ใช้ส่งมา ทำให้ยากต่อการตรวจสอบย้อนกลับ
- ระบบไม่สามารถย้อนกลับไปแก้ข้อความใน State ก่อนหน้าได้

2. ความปลอดภัย (Security)

ข้อดี:

- ใช้ IAM ในการควบคุมสิทธิ์การเข้าถึงแต่ละ **Role** อย่างเหมาะสม
- รองรับการเข้ารหัสข้อมูลทั้งระหว่างส่งและจัดเก็บ (Encryption in Transit and at Rest)
- สามารถใช้บริการ AWS เช่น **IAM** เพื่อจัดการ Authentication

ข้อเสีย:

- หากไม่มีการจัดการ Secrets อย่างเหมาะสม อาจเกิดช่องโหว่ด้านความปลอดภัย
- ระบบอาจเสี่ยงต่อการโจมตีหากไม่มีการกำหนด Security Group และ WAF อย่างรัดกุม

3. ความน่าเชื่อถือ (Reliability)

ข้อดี:

- ใช้บริการที่รองรับ High Availability เช่น **DynamoDB** , **S3** และ **Lambda** (รองรับ Auto Retry)
- สามารถสำรองข้อมูลผ่าน S3 versioning และ **DynamoDB** backup

ข้อเสีย:

- หากไม่ได้ออกแบบ Error Handling และการ retry อย่างเหมาะสม อาจทำให้ระบบล่มเหลวเมื่อเกิดปัญหาเล็กน้อย
- ขาดการออกแบบ Disaster Recovery หรือการซ่อมแผน DR อย่างจริงจัง

4. ประสิทธิภาพการทำงาน (Performance Efficiency)

ข้อดี:

- ใช้บริการแบบ Serverless เช่น **Lambda** ที่สามารถปรับสเกลตามจำนวนผู้ใช้งานโดยอัตโนมัติ
- การใช้ **DynamoDB** ช่วยให้ Query ทำงานได้รวดเร็วแม้ในปริมาณข้อมูลมาก

ข้อเสีย:

- หากการออกแบบ Schema หรือ Query ของ **DynamoDB** ไม่ดี อาจทำให้ประสิทธิภาพลดลง

- Lambda มีข้อจำกัดด้านเวลาทำงาน (timeout) และขนาด payload

5. ค่าใช้จ่าย (Cost Optimization)

ข้อดี:

- ใช้บริการแบบ Pay-as-you-go เช่น **Lambda** และ **S3** ที่ช่วยลดต้นทุนเมื่อไม่มีการใช้งาน
- สามารถกำหนด Budget และใช้ Cost Explorer เพื่อควบคุมค่าใช้จ่ายได้

ข้อเสีย:

- หากไม่มีการ Monitor และ Optimize การใช้งาน เช่น **Lambda** ถูกเรียกซ้ำซ้อนหรือใช้ storage มากเกินไปจะทำให้ค่าใช้จ่ายพุ่งสูง
- การใช้บริการบางรายการอาจมีค่าใช้จ่ายแฝงสูงหากใช้บ่อย

6. ความยั่งยืน (Sustainability)

ข้อดี:


- ระบบ Serverless ลดการใช้พลังงานจากการตั้งเซิร์ฟเวอร์ตลอดเวลา
- ใช้ทรัพยากรตามความจำเป็น ลดการสูญเสียของพลังงานและทรัพยากร

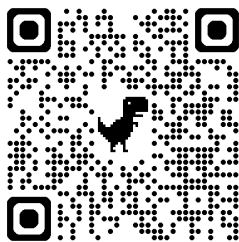
ข้อเสีย:

- ยังไม่ได้มีการประเมินหรือปรับแต่งทรัพยากรให้เหมาะสมกับการใช้งานจริง เช่น memory size ของ Lambda
- การเก็บข้อมูลที่ไม่มีประโยชน์ไว้ใน **S3** หรือ **DynamoDB** อาจสิ้นเปลืองทรัพยากรโดยไม่จำเป็น

i) URL ไปยัง Drive หรือ Git Repository ที่ต้องใช้ของระบบ (หากมี) รวมทั้งคลิปวิดีโอเื่อการทำงานของระบบ ความยาวไม่เกิน 5 นาที ซึ่งแสดงให้เห็นทรัพยากรหรือบริการที่เกี่ยวข้องใน AWS ที่ใช้ในระบบในขณะที่ระบบทำงานด้วย

GitHub Repository : <https://github.com/Jutinut/CS232>

คลิปวิดีโอเื่อการทำงานของระบบ :  cs232_Final Project.mov



QR code วิดีโอเื่อ

i) สรุปผลการดำเนินโครงการ และแนวทางพัฒนาต่อในอนาคต

โครงการนี้ได้พัฒนาระบบแจ้งคนหายและเบาะแสผ่าน LINE Chatbot โดยใช้เทคโนโลยี AWS เช่น Lambda, DynamoDB และ S3 เพื่อจัดเก็บข้อมูลและประมวลผลแบบไร้เซิร์ฟเวอร์ (Serverless) ซึ่งทำให้ระบบสามารถทำงานได้อย่างมีประสิทธิภาพ จากการทดสอบระบบพบว่าสามารถตอบสนองต่อคำสั่งของผู้ใช้ได้อย่างถูกต้องและรวดเร็ว ระบบสามารถจัดเก็บข้อมูลผู้แจ้งและรูปภาพที่เกี่ยวข้องได้อย่างครบถ้วน

แนวทางพัฒนาต่อ

ระบบสามารถพัฒนาเพิ่มเติมโดยการเชื่อมต่อ AWS Lambda เข้ากับ EC2 ที่ใช้สำหรับโฮสต์เว็บไซต์เพื่อให้ผู้ใช้งานสามารถแจ้งข้อมูลผ่านแบบฟอร์มบนหน้าเว็บไซต์ได้อีกหนึ่งช่องทาง ซึ่งจะเป็นการขยายจากเดิมที่รองรับเฉพาะช่องทาง LINE Official Account โดยเมื่อผู้ใช้งานกรอกข้อมูลผ่านแบบฟอร์ม ข้อมูลจะถูกส่งเป็น payload ไปยัง Lambda ซึ่งจะประมวลผลและบันทึกลงในฐานข้อมูล DynamoDB แบบไร้สถานะ (stateless) การออกแบบลักษณะนี้จะช่วยเพิ่มความยืดหยุ่นของระบบรองรับผู้ใช้งานในหลากหลายช่องทางและสามารถขยายฟังก์ชันเพิ่มเติมในอนาคต

ได้ง่ายยิ่งขึ้น เช่น การแสดงผลรายงานคนหายบนเว็บไซต์หรือการแจ้งเตือนผ่านหลายแพลตฟอร์ม