# Final Project Group3

**A) สมาชิก**

1. นายธำรง แซ่เฉิน 6509611775
2. นายธารา ศรีธราดล 6509681141
3. นางสาวนันท์นภัส งามดุษฎีพร 6509681166

**B) ชื่อระบบโซลูชันของกลุ่ม คำอธิบายที่มาและความสำคัญ ตอบโจทย์ SDG Goals ข้อใด**

ปัจจุบันการศึกษาเป็นปัจจัยสำคัญที่ช่วยยกระดับคุณภาพชีวิตของบุคคลและความก้าวหน้าของสังคม แต่ยังมีความท้าทายด้านการเข้าถึงทรัพยากรการเรียนรู้ที่มีคุณภาพ โดยเฉพาะในกลุ่มผู้เรียนที่มีข้อจำกัดทางเศรษฐกิจ ด้วยเหตุนี้ ทีมพัฒนาจึงได้สร้าง เว็บไซต์ BrainyBite เพื่อเป็นแหล่งเรียนรู้ที่เข้าถึงได้ง่าย โดยมุ่งเน้นการให้เนื้อหาการเรียนรู้ในรูปแบบที่ กระชับ เข้าใจง่าย ไม่คิดค่าใช้จ่ายและทันสมัย

BrainyBite ออกแบบมาเพื่อส่งเสริมการเรียนรู้ในทุกระดับ ให้ผู้เรียนสามารถเข้าถึงเนื้อหาบทความผ่านระบบที่ใช้งานง่ายและเสถียร โครงการนี้ยังคำนึงถึงความสำคัญของการศึกษาในฐานะเครื่องมือที่ช่วยลดความเหลื่อมล้ำและสนับสนุนการพัฒนาคุณภาพชีวิตในระยะยาว

โครงงานนี้มีความเกี่ยวข้องกับ เป้าหมายการพัฒนาที่ยั่งยืน SDG 4: Quality Education ซึ่งมุ่งเน้นการให้โอกาสทางการศึกษาที่เท่าเทียม โดยเฉพาะการสร้างความเท่าเทียมในการเข้าถึงแหล่งเรียนรู้และการพัฒนาทักษะที่จำเป็นสำหรับอนาคต

เลือกใช้บริการ AWS ได้แก่

1. Amazon S3: ใช้ในการจัดเก็บสื่อการเรียนรู้ต่าง ๆ เช่น รูปภาพ เอกสาร PDF
2. AWS Lambda: ช่วยในการประมวลผลข้อมูลแบบอัตโนมัติ ใช้เพื่อทริกเกอร์ Amazon SNS สำหรับส่งการแจ้งเตือนผ่านอีเมล
3. Amazon EC2: เป็นเซิร์ฟเวอร์สำหรับโฮสต์เว็บไซต์หลัก
4. Amazon RDS: ฐานข้อมูลสำหรับเก็บข้อมูลผู้ใช้ เนื้อหาการเรียน
5. Amazon SNS: ใช้ในการแจ้งเตือนผ่านอีเมล
6. Amazon API Gateway: เพื่อเรียกใช้ AWS Lambda ในการสั่งให้ SNS ส่งแจ้งเตือน

**C) กรณีการใช้งาน (Use cases) และประโยชน์สำหรับผู้ใช้**

**กรณีการใช้งาน**

ผู้ใช้

1. ลงทะเบียนเพื่อเข้าใช้งาน
2. เข้าสู่ระบบเพื่อเข้าถึงเนื้อหา
3. ดูบทความและเนื้อหาบนเว็บไซต์
4. ค้นหาบทความด้วยคำสำคัญหรือหมวดหมู่
5. เลือกดูบทความตามประเภทที่สนใจ
6. บันทึกบทความไว้ดูภายหลัง (Bookmark)
7. ลบ Bookmark ที่ไม่ต้องการ
8. ดูและแก้ไขข้อมูลโปรไฟล์ของตนเอง
9. ส่งบทความที่ต้องการเผยแพร่ผ่านระบบ

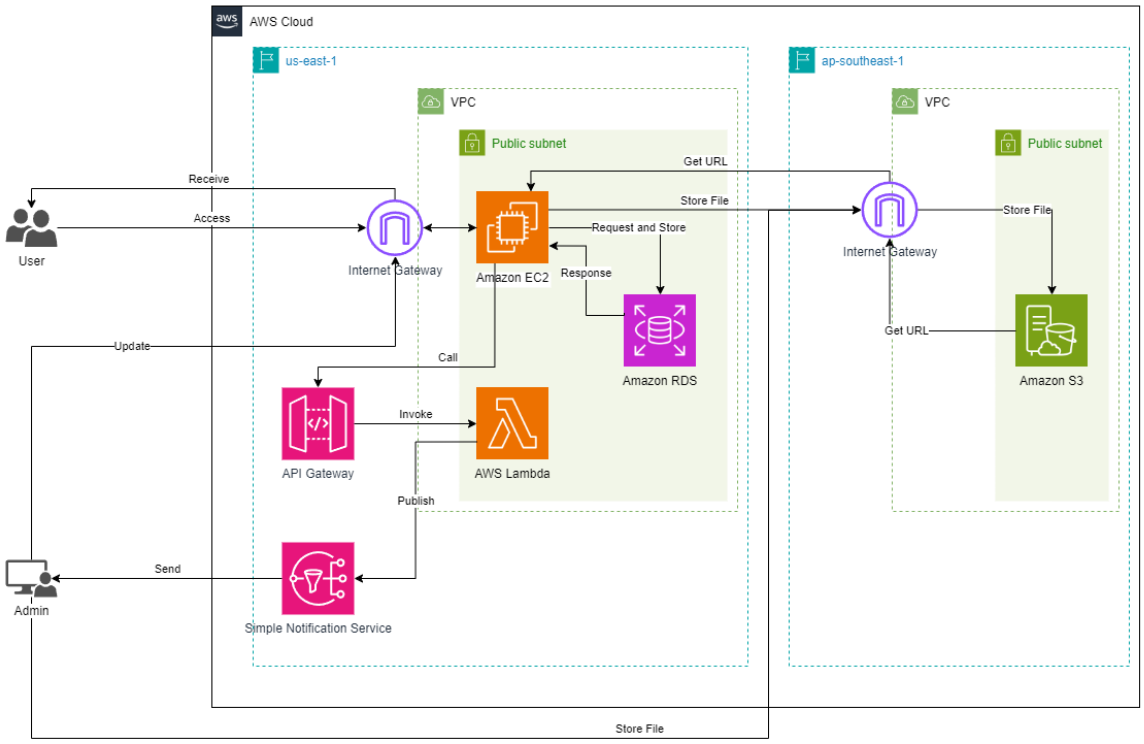
ผู้ดูแล

1. โหลดเนื้อหาใหม่หรือจัดการบทความที่ส่งเข้ามา
2. ตรวจสอบและแก้ไขข้อมูลระบบ

**ประโยชน์สำหรับผู้ใช้**

1. เพิ่มการเข้าถึงการเรียนรู้ให้แก่ผู้เรียนทุกพื้นที่
2. ช่วยลดเวลาและค่าใช้จ่ายในการเข้าถึงเนื้อหาการเรียน
3. เพิ่มความสะดวกและความรวดเร็วในการจัดการข้อมูล

**D) สถาปัตยกรรมระบบที่แสดงให้เห็นถึงบริการ AWS ที่ใช้พร้อมคำอธิบายการทำงานร่วมกันขององค์ประกอบต่าง ๆ ในสถาปัตยกรรม**




**การทำงานร่วมกันขององค์ประกอบต่าง ๆ**

1. User หรือ ผู้ใช้งาน
    - ผู้ใช้งานเข้าถึง Web Application ซึ่งถูก Host บน Amazon EC2
    - Amazon EC2 ตอบรับคำขอและแสดงผลหน้าเว็บไซต์ให้แก่ผู้ใช้
2. Admin หรือผู้ดูแลระบบ
    - รับการแจ้งเตือนจาก SNS เมื่อมีผู้ใช้งานต้องการเพิ่มบทความ
    - ผู้ดูแลระบบตรวจสอบบทความ ถ้าอนุมัติ จะทำการเพิ่มไฟล์ไปที่ Amazon S3 และอัปเดตฐานข้อมูล Amazon RDS MySQL
3. Internet Gateway
    - คอยรับข้อมูลเข้าและส่งข้อมูลออกสู่อินเตอร์เน็ต
4. Amazon EC2 หรือ Web Server

- รับคำร้องและแสดงผลหน้าเว็บไซต์ให้แก่ผู้ใช้งาน
- ติดต่อกับฐานข้อมูลเมื่อมีต้องการเรียกใช้หรือเปลี่ยนแปลงข้อมูล
- ติดต่อกับ Amazon S3 ผ่านทาง Internet Gateway เพื่ออัปโหลดไฟล์ต่าง ๆ

5. Amazon RDS
    - รอรับคำร้องต่าง ๆ และตอบรับ
    - คอยรับข้อมูลที่ส่งเข้ามาเพื่อบันทึกลงฐานข้อมูล
6. Amazon API Gateway
    - เมื่อมีผู้ใช้เพิ่มบทความ Amazon EC2 จะทำการเรียก API Gateway เพื่อเรียกการทำงานของ AWS Lambda
7. AWS Lambda
    - ส่งข้อมูลต่าง ๆ ไปให้ Simple Notification Service และสั่งให้ทำงาน
8. Simple Notification Service
    - แจ้งเตือนไปที่ผู้ดูแลระบบตามอีเมลที่ subscription เอาไว้
9. Amazon S3
    - เก็บไฟล์ต่าง ๆ ที่ Amazon EC2 ส่งเข้ามา
    - ส่ง URL ไปให้ Amazon EC2 เพื่อนำไปเก็บใน Amazon RDS

**E) อธิบายขั้นตอนหลักที่สำคัญในการสร้างและตั้งค่าแต่ละองค์ประกอบ**

**Web Application**

1. สร้างโปรเจกต์ด้วย Spring Initializer
    - เข้าไปที่ https://start.spring.io/
    - ตั้งค่าพื้นฐาน:

        Project: Maven

        Language: Java 17

Spring Boot Version: 3.3.5

2. ดาวน์โหลดและนำไฟล์โปรเจกต์เข้า IDE
    - ดาวน์โหลดไฟล์ .zip จาก Spring Initializer
    - แตกไฟล์และนำเข้าใน IDE IntelliJ IDEA
3. ตั้งค่าไฟล์ application.properties
    - ตั้งค่าการเชื่อมต่อกับฐานข้อมูล เช่น Amazon RDS
    - กำหนดค่าการใช้งาน AWS เช่น Amazon S3
4. พัฒนาโค้ดส่วน Backend
    - Controller: รับคำขอ (HTTP Requests) จากผู้ใช้
    - Model: สร้างคลาสที่ใช้เป็นตัวแทนข้อมูล
    - Service: จัดการตรรกะของแอปพลิเคชัน เช่น ประมวลผลข้อมูลหรือเรียกใช้งาน AWS S3
    - Repository: เชื่อมต่อกับฐานข้อมูลเพื่อจัดการข้อมูล
5. พัฒนาโค้ดส่วน Frontend
    - พัฒนาโค้ด html css แล้วเชื่อมต่อผ่าน API ที่พัฒนาใน Backend
6. ทดสอบและปรับปรุงระบบ
    - ใช้ Postman สำหรับทดสอบ API
    - Debug และตรวจสอบการทำงานของ Web Application
7. Deploy Web Application

**AWS Service**

1. สร้าง Amazon EC2 Instance พร้อมกำหนดค่าต่าง ๆ
    - กำหนด Security Group เพื่อกำหนด Inbound และ outbound ให้รองรับ traffic จาก SSH, Spring Boot, และ MySQL
    - ติดตั้ง Java 17 และ Maven

2. สร้าง Amazon S3 Bucket
    - เปิด Public Access
    - กำหนด Policy ให้สามารถเรียกใช้ Object ได้
    - กำหนด Cross-Origin Resource Sharing ให้เซิร์ฟเวอร์สามารถดึงข้อมูลได้
3. สร้าง Access Key จาก IAM
    - สร้าง Access Key ที่ให้สิทธิ์ Amazon S3 Full Access
    - นำ Key ที่ได้ไปกำหนดค่า application.properties
4. สร้าง Topic ใน SNS
    - สร้าง Topic แบบ Standard เพื่อให้รองรับการส่งอีเมล
    - สร้าง Subscriptions ด้วยอีเมลที่ต้องการรับ SNS
5. สร้าง AWS lambda เพื่อกำหนดข้อมูลที่ต้องการส่งให้กับ SNS
6. สร้าง API จาก Amazon API Gateway
    - เลือกฟังก์ชันจาก AWS lambda
    - เลือก method ที่ต้องการใช้
7. สร้าง MySQL Instance จาก Amazon RDS
    - กำหนด User และ Password
    - เปิด Public Access
    - นำมูลที่ได้ไปกำหนดค่าใน application.properties

**F) ผลการทดสอบการทำงานของระบบโซลูชันแบบ end-to-end ทั้งหมด 3 สถานการณ์**

**สถานการณ์ทดสอบที่ 1 :** การแสดงบทความพร้อมการค้นหาหรือกรองหมวดหมู่

**เป้าหมายการทดสอบ :** ทดสอบกระบวนการตั้งแต่ผู้ใช้เข้าใช้งานเว็บไซต์ เลือกหมวดหมู่หรือค้นหาบทความ จนไปถึงการแสดงบทความสำเร็จ

**ขั้นตอนการทดสอบ :**

1. ผู้ใช้เปิดเว็บไซต์และเข้าสู่หน้าหลักของเว็บไซต์
2. เลือกหมวดหมู่หรือค้นหาจากช่องค้นหา เช่น “Cloud Computing” หรือ “AI”
3. ระบบส่งคำร้อง (API Request) ไปยังเซิร์ฟเวอร์เพื่อดึงข้อมูลจาก RDS MySQL
4. แสดงรายการบทความที่ตรงกับหมวดหมู่หรือคำค้นหา
5. ผู้ใช้เลือกบทความเพื่อเปิดอ่าน
6. ระบบส่งคำขอ (API Request) ไปยัง RDS MySQL เพื่อดึงข้อมูลบทความ พร้อม URL Markdown และโหลดเนื้อหาบทความ
7. แสดงหน้าหาในรูปแบบ Markdown บนหน้าเว็บ

**ผลลัพธ์ที่คาดหวัง :**

1. ระบบหมวดหมู่ทำงานได้ถูกต้อง
2. ระบบค้นหาบทความได้ตรงคำค้นหา
3. บทความแสดงผลได้อย่างถูกต้องและสมบูรณ์

**ผลการทดสอบ :** ผ่าน

**รายละเอียด :**

- การค้นหาบทความหมวดหมู่ทำงานได้ถูกต้อง เช่น เมื่อเลือกหมวดหมู่ “Cloud Computing” ระบบจะแสดงเฉพาะบทความที่เกี่ยวข้องกับ Cloud Computing
- การค้นหาบทความด้วยคำค้นหา เช่น “AI” แสดงผลลัพธ์ที่ประกอบด้วยคำค้นหาถูกต้อง
- การแสดงบทความในหน้าบทความถูกต้อง โดยเนื้อหาโหลดจาก S3 และแสดงในรูปแบบ Markdown ได้อย่างสมบูรณ์
- ไม่มีข้อผิดพลาดหรือการโหลดข้อมูลล่าช้า

**สถานการณ์ทดสอบที่ 2 :** การเพิ่มบทความโดยผู้ใช้ที่เข้าสู่ระบบ

**เป้าหมายการทดสอบ :** ตรวจสอบกระบวนการเพิ่มบทความและการแจ้งเตือนไปยังผู้ดูแลระบบ

**ขั้นตอนการทดสอบ :**

1. ผู้ใช้เข้าสู่ระบบด้วยบัญชีที่มีอยู่หรือลงทะเบียนใหม่
2. เข้าหน้าเพิ่มบทความและกรอกข้อมูล เช่น
    - รูปภาพ : อัปโหลดภาพประกอบ
    - ชื่อบทความ : การใช้งาน AWS Lambda
    - รายละเอียด : AWS Lambda คืออะไร และใช้งานอย่างไร
    - หมวดหมู่ : Cloud Computing
    - เนื้อหา : PDF, Markdown, DOCS, หรือไฟล์ที่เกี่ยวข้องกับข้อมูลเนื้อหา
3. กดส่งบทความ
4. ระบบส่งคำร้อง (API Request ) ไปที่เซิร์ฟเวอร์เพื่ออัปโหลดไฟล์ข้อมูลเนื้อหาและรูปภาพไปที่ S3 และบันทึก Metadata ของบทความลงใน RDS MySQL
5. ระบบเรียก AWS Lambda ผ่าน API Gateway เพื่อส่งข้อความแจ้งเตือนไปยังอีเมลของผู้ดูแลระบบ
6. ผู้ดูแลระบบได้รับการแจ้งเตือนและตรวจสอบบทความ
7. ผู้ดูแลระบบแปลงบทความเป็น Markdown และอัปบทความไปที่ RDS MySQL

**ผลลัพธ์ที่คาดหวัง :**

1. ข้อมูลบทความที่ผู้ใช้เพิ่มถูกบันทึกลงฐานข้อมูล
2. ไฟล์ข้อมูลเนื้อหา และรูปภาพถูกอัปโหลดไปยัง S3
3. ผู้ดูแลระบบได้รับข้อความแจ้งเตือนผ่านอีเมล

**ผลการทดสอบ :** ผ่าน

**รายละเอียด :**

- ผู้ใช้สามารถเข้าสู่ระบบและลงทะเบียนสำเร็จ และเข้าถึงฟีเจอร์เพิ่มบทความได้
- ข้อมูลบทความถูกบันทึกลงในฐานข้อมูล RDS MySQL และไฟล์บทความรวมถึงรูปภาพถูกอัปโหลดไปยัง S3 อย่างสมบูรณ์
- Lambda เรียกใช้ AWS SNS ได้สำเร็จ และผู้ดูแลระบบได้รับการแจ้งเตือนผ่านอีเมลทันที

- หลังผู้ดูแลตรวจสอบและอนุมัติ ผู้ดูและจะอัปบทความไปที่ RDS MySQL และแสดงบนหน้าเว็บตามที่คาดหวัง

**สถานการณ์ทดสอบที่ 3 :** การแสดงเนื้อหาบทความ Markdown จาก S3 เกิดข้อผิดพลาด

**เป้าหมายการทดสอบ :** ผู้ใช้เปิดบทความที่มีเนื้อเก็บใน S3 แต่ไฟล์ Markdown ที่เกี่ยวข้องไม่สามารถเข้าถึงได้ เช่น ไฟล์ถูกลบหรือ URL หมดอายุ

**ขั้นตอนการทดสอบ :**

1. ลบไฟล์ Markdown ที่เกี่ยวข้องกับบทความออกจาก S3
2. ผู้ใช้พยายามเข้าถึงบทความผ่านหน้าเว็บ

**ผลลัพธ์ที่คาดหวัง :**

1. ระบบแสดงข้อความเตือนว่า “Failed to load content.”
2. หน้าเว็บยังแสดงข้อมูลพื้นฐานของบทความ เช่น ชื่อ, รายละเอียดและภาพประกอบ โดยไม่แสดงหน้าว่างหรือ Error Code จากเซิร์ฟเวอร์
3. แสดงข้อผิดพลาดลงใน Log ของระบบ เพื่อให้ทีมพัฒนาตรวจสอบปัญหา

**ผลการทดสอบ :** ไม่ผ่าน

**รายละเอียด :**

- ผู้ใช้พยายามเข้าถึงบทความที่เชื่อมโยงกับไฟล์ Markdown ที่ถูกลบ
- ระบบแสดงข้อความเตือนว่า “Failed to load content.”
- หน้าเว็บยังคงแสดงข้อมูลพื้นฐานของบทความ เช่น ชื่อบทความ, ผู้เขียน, วันที่เผยแพร่ โดยไม่มีการแสดงหน้าว่างหรือข้อความ Error Code ที่ไม่เหมาะสม
- แสดงข้อผิดพลาดลงใน Log ของระบบ เพื่อให้ทีมพัฒนาตรวจสอบปัญหา

**สรุปผลการทดสอบ**

การทดสอบทั้งหมด 3 มีสถานการณ์ที่ผ่าน 2 สถานการณ์ และไม่ผ่าน 1 สถาการณ์ ระบบสามารถตอบสนองตามความคาดหวังส่วนใหญ่ได้ รวมถึงส่งผลต่อประสบการณ์ของผู้ใช้ไม่มากนัก

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

**Operational excellence**

- การใช้บริการจัดการของ AWS เช่น Amazon S3 และ Amazon RDS ช่วยลดความซับซ้อนในการจัดการโครงสร้างพื้นฐาน ทำให้ทีมสามารถมุ่งเน้นที่การพัฒนาระบบได้มากขึ้น
- มีเครื่องมือสำหรับการตรวจสอบและวิเคราะห์ (Monitoring) เช่น Amazon CloudWatch เพื่อการติดตามสถานะระบบ

**Security**

- มีการป้องกันการเข้าถึงที่ไม่ได้รับอนุญาตโดย AWS เป็นคนดูแล hardware ของระบบ
- ตั้งค่า AWS resource เพื่อจำกัดสิทธิ์ในการเข้าถึง

**Reliability**

- หาก AWS มีปัญหาสามารถนำ spring boot ไปรันที่อื่นได้
- ข้อเสีย ไม่ได้สำรองข้อมูลของ RDS และ S3

**Performance efficiency**

- การใช้บริการของ AWS (เช่น EC2, Lambda) ทำให้เราไม่จำเป็นต้องเครื่อง Server ทำให้การประหยัด cost ในการดูแล
- การใช้บริการของ AWS ทำให้เราสามารถปรับเปลี่ยนการใช้ resource เมื่อ demand มีการเปลี่ยนแปลง

**Cost optimization**

- ใช้ Lambda ซึ่งคิดค่าใช้จ่ายตามการใช้งานจริง ลดค่าใช้จ่ายเมื่อมีการใช้งานต่ำ
- การใช้ S3 สำหรับจัดเก็บข้อมูลช่วยลดต้นทุนเมื่อเทียบกับการใช้เซิร์ฟเวอร์แบบดั้งเดิม

**Sustainability**

- การใช้ Lambda ซึ่งรองรับ Serverless Architecture ช่วยลดการใช้ทรัพยากรฮาร์ดแวร์ ทำให้ลดผลกระทบต่อสิ่งแวดล้อม

**H) URL ไปยัง Drive หรือ Git Repository ที่ต้องใช้ของระบบ (หากมี) รวมทั้งคลิปวิดีโอเดโมการทำงานของระบบความยาวไม่เกิน 3 นาทีซึ่งแสดงให้เห็นทรัพยากรหรือบริการที่เกี่ยวข้องใน AWS ที่ใช้ในระบบในขณะที่ระบบทำงาน**

Link Video:
https://drive.google.com/file/d/1E3B3YRvNkNqZ43OWBNB8CZwPwN0kYmc5/view?usp=sharing

**I) สรุปผลการดำเนินโครงงาน และแนวทางพัฒนาต่อในอนาคต**

โครงงานที่พัฒนาขึ้นมีการนำเทคโนโลยี Cloud Computing สามารถพัฒนาเว็บไซต์ BrainyBite ซึ่งเป็นแหล่งรวบรวมเนื้อหาสาระสั้น ๆ และให้ความรู้ที่เข้าใจง่าย โดยเว็บไซต์ BrainyBite สามารถบรรลุเป้าหมายได้ดังนี้

1. พัฒนาเว็บไซต์ BrainyBite ซึ่งเป็นแหล่งรวบรวมเนื้อหาสาระสั้น ๆ และให้ความรู้ที่เข้าใจง่าย
2. รองรับการอัปโหลดและจัดการเนื้อหา ได้อย่างมีประสิทธิภาพผ่าน AWS
3. ได้ใช้เทคโนโลยี Cloud เช่น AWS (S3, Lambda, SNS, EC2, RDS)

การนำเทคโนโลยี Cloud Computing มาใช้งานนี้ไม่เพียงช่วยลดภาระด้านโครงสร้างพื้นฐาน แต่ยังช่วยให้การพัฒนาระบบเป็นไปอย่างรวดเร็วและยั่งยืน ตอบโจทย์การพัฒนาตามเป้าหมาย SDG 4 (Quality Education)

**แนวทางพัฒนาต่อในอนาคต**

1. รองรับหลายภาษาเพื่อขยายกลุ่มเป้าหมายไปยังผู้ใช้งานต่างประเทศ
2. เพิ่มฟีเจอร์สถานะเมื่อผู้ใช้งานเพิ่มบทความ
3. เพิ่มช่องทางการติดต่อกับผู้ดูแล
4. พัฒนาแอปพลิเคชันบนมือถือและแท็บเล็ต
5. เพิ่มระบบวิเคราะห์ข้อมูลผู้ใช้งาน