สมาชิกกลุ่ม

1.นายธิปไตย ศิริรักษ์	6609650434
2.นางสาวธนัชพร วงค์ตลาด	6609650012
3.นายจันทร์พงศ์ วิทยอรุณธานี	6609650210
4.นางสาววรรณพร เทพชนะ	6609650632
5.นางสาวมนัสนันท์ อุดมรัตน์	6609650608
6.นายสรวิศ ตันศรีเจริญ	6609650673
7.นายภูมิภากร โกเมนไปรรินทร์	6609650582

“ RaPunSale : ปล่อยผมแล้วปล่อยยอดขายให้พุ่ง! ”

ที่มาและความสำคัญ

ในปัจจุบันองค์กรธุรกิจที่มีสินค้าหลากหลายประเภท เช่น เครื่องใช้ไฟฟ้า หรือวัสดุก่อสร้าง มักประสบปัญหาในการบริหารจัดการข้อมูลด้านการขายและการให้ข้อมูลลูกค้า เนื่องจากข้อมูลสินค้า สต๊อก คลังสินค้า และเอกสารสเปกผลิตภัณฑ์มักกระจัดกระจายอยู่ในหลายระบบ ส่งผลให้พนักงานขายต้องสลับใช้งานหลายระบบเพื่อตอบคำถามลูกค้าอย่างครบถ้วน ทั้งยังมีความเสี่ยงที่จะให้ข้อมูลผิดพลาด เช่น การเสนอขายสินค้าที่หมดสต๊อก หรือส่งเอกสารข้อมูลผิดรุ่นให้แก่ลูกค้า ซึ่งอาจส่งผลกระทบต่อความเชื่อมั่นของลูกค้าและภาพลักษณ์ขององค์กร

ยิ่งไปกว่านั้น การขาดระบบติดตามลูกค้าอย่างเป็นระบบ ทำให้ไม่สามารถตรวจสอบสถานะการขายของลูกค้าแต่ละรายได้อย่างมีประสิทธิภาพ พนักงานขายไม่สามารถทราบได้ว่าลูกค้ารายใดอยู่ในขั้นตอนการตัดสินใจ สั่งซื้อ หรือส่งมอบสินค้าแล้ว นอกจากนี้ ผู้บริหารยังไม่สามารถเข้าถึงข้อมูลภาพรวมของการขายในแต่ละช่วงเวลาเพื่อใช้ประกอบการวางแผนและตัดสินใจได้อย่างทันท่วงที ปัญหาเหล่านี้สะท้อนให้เห็นถึงความจำเป็นของระบบกลางในการจัดการข้อมูลการขายที่ครบวงจร

จากเหตุผลข้างต้น จึงมีแนวคิดในการพัฒนา “ระบบรวมศูนย์ข้อมูลการขาย” โดยใช้เทคโนโลยีคลาวด์ของ Amazon Web Services (AWS) เพื่อเป็นเครื่องมือในการจัดเก็บข้อมูลสินค้าแบบรวมศูนย์ แสดงผลข้อมูลแบบเรียลไทม์ สนับสนุนการค้นหาและส่งข้อมูลสเปกสินค้าแก่ลูกค้า พร้อมทั้งติดตามสถานะของลูกค้าได้อย่างเป็นระบบ นอกจากนี้ยังสามารถนำข้อมูลไปวิเคราะห์ผ่าน Dashboard เพื่อช่วยให้ผู้บริหารมีข้อมูลเชิงลึกประกอบการตัดสินใจ ทั้งนี้ ยังสามารถขยายความสามารถของระบบด้วยการเพิ่มการแจ้งเตือนผ่านบริการ Amazon SNS เพื่อให้ระบบตอบสนองต่อสถานการณ์สำคัญได้อย่างอัตโนมัติและมีประสิทธิภาพยิ่งขึ้น

กรณีการใช้งาน (Use Cases) และประโยชน์ที่ผู้ใช้ได้รับ

Use Case 1: พนักงานขายค้นหาข้อมูลสินค้าและตรวจสอบ stock

- **เหตุการณ์:** พนักงานขายกำลังนำเสนอสินค้าให้ลูกค้าที่หน้างาน แต่ไม่แน่ใจว่าสินค้าชิ้นนั้นยังมีในคลังหรือไม่
- **ขั้นตอนการใช้งาน:**
    1. เข้าสู่ระบบผ่านหน้า login
    2. เข้าสู่หน้า index.html

3. ใช้ช่องค้นหาเพื่อกรองสินค้าตามชื่อ
4. ดูจำนวน stock คงเหลือในทันที
5. หากจำเป็น สามารถดาวน์โหลดเอกสารสเปกสินค้า (PDF) ส่งให้ลูกค้าได้

- **ประโยชน์:**
  - ลดเวลาในการตรวจสอบข้อมูลสินค้า
  - ลดความผิดพลาดจากการเสนอสินค้าที่หมด stock
  - เพิ่มความน่าเชื่อถือให้กับพนักงานในสายตาลูกค้า

Use Case 2: พนักงานอัปเดต stock สินค้า

- **เหตุการณ์:** เมื่อมีการรับสินค้าล็อตใหม่ หรือสินค้าถูกเบิกออกจากคลัง พนักงานต้องอัปเดตจำนวน stock
- **ขั้นตอนการใช้งาน:**
  1. เข้าสู่ระบบ
  2. คลิกสินค้าในหน้า index.html เพื่อเข้าสู่ index2.html
  3. กรอกจำนวนสินค้าที่ต้องการเพิ่มหรือลด
  4. กดปุ่ม “Update Stock”

กรณีพิเศษ:

หากผลรวม stock จะติดลบ ระบบจะ แสดงข้อความเตือนและไม่อัปเดต

ประโยชน์:

- เพิ่มความแม่นยำในการจัดการคลัง
- ป้องกันการทำข้อมูลผิดพลาด
- ลดการใช้ระบบหลังบ้านแบบ manual

Use Case 3: ผู้บริหารตรวจสอบภาพรวมผ่าน Dashboard

- **เหตุการณ์:** ฝ่ายบริหารต้องการวิเคราะห์ stock สินค้าเพื่อวางแผนจัดซื้อ
- **ขั้นตอนการใช้งาน:**

1.เข้าสู่ระบบด้วย role = admin

2.เข้าหน้า dashboard.html

3.ระบบแสดง:

- จำนวนสินค้าทั้งหมด
- ยอดรวม stock
- สินค้าที่ใกล้หมด
- กราฟแสดงจำนวน stock ตามชื่อสินค้าประโยชน์:
  - เพิ่มความสะดวกและความน่าเชื่อถือในการขาย
  - ประหยัดเวลาในการหาข้อมูลจากระบบอื่น

ประโยชน์:

ตัดสินใจได้จากข้อมูลจริง

เห็นแนวโน้มของสินค้าใกล้หมด

ใช้เป็นเครื่องมือวางแผนเชิงกลยุทธ์

Use Case 4: ระบบแจ้งเตือนเมื่อ stock ต่ำกว่า 3

- **เหตุการณ์:** สินค้าชิ้นหนึ่ง stock ใกล้หมด แต่ยังไม่มีใครทราบ
- **การทำงานของระบบ:**

  เมื่อมีการอัปเดต stock แล้วเหลือน้อยกว่า 3 ชิ้น

  Lambda จะ trigger AWS SNS

  SNS ส่ง email ไปยังผู้ดูแลระบบ

- **ประโยชน์:**

  ป้องกันการขายสินค้าที่ไม่มีใน stock

  ช่วยทีมจัดซื้อวางแผนเติมสินค้าได้ล่วงหน้า

  ลดความเสี่ยงของการเสียลูกค้า

Use Case 5: พนักงานติดตามสถานะลูกค้า

- **เหตุการณ์:** ฝ่ายขายต้องการตรวจสอบว่าลูกค้ารายใดบ้างที่มีการติดต่อไปแล้ว และอยู่ในขั้นตอนไหนของกระบวนการขาย

- **ขั้นตอนการใช้งาน:**

  1.เข้าสู่ customertracking.html

  2.ระบบแสดงรายการลูกค้า, สินค้าที่สนใจ, พนักงานดูแล, สถานะ

  3.พนักงานสามารถกด “แก้ไข” เพื่ออัปเดตสถานะ เช่น A ⟶ B ⟶ C

  4.ระบบเก็บบันทึกวันที่ล่าสุดที่มีการอัปเดต

- **ประโยชน์:**
  - วางแผน follow-up ลูกค้าได้ถูกจังหวะ
  - ป้องกันการลืมหรือซ้ำซ้อนกับทีมอื่น
  - สร้างระบบ CRM ขนาดย่อมในองค์กร

Use Case 6: ระบบ Login เพื่อกำหนดสิทธิ์การเข้าถึง

- **เหตุการณ์:** พนักงาน A และ B ต้องเห็นแค่ข้อมูลของตัวเอง ขณะที่ Admin ต้องเห็นทุกอย่าง
- **ขั้นตอนการใช้งาน:**
  1. ระบบตรวจสอบ role และ userId จาก localStorage
  2. ถ้าเป็น Admin ⟶ เห็นข้อมูลทั้งหมด
  3. ถ้าเป็น Staff ⟶ Filter ตาม staffId
  4. ถ้าไม่ login ⟶ redirect ไป login.html
- **ประโยชน์:**
  - ป้องกันข้อมูลรั่วไหล
  - แสดงข้อมูลตามความเกี่ยวข้องของผู้ใช้
  - ปรับสิทธิ์ได้ยืดหยุ่นในอนาคต

สรุปภาพรวม

ระบบ RaPunSale ครอบคลุม Use Case ที่หลากหลาย โดยใช้โครงสร้างแบบ Role-Based Access + Serverless Architecture ทำให้สามารถรองรับการทำงานของฝ่ายต่าง ๆ ได้ครบวงจร ตั้งแต่การเสนอขาย ⟶ การเติม stock ⟶ การวิเคราะห์ข้อมูล และการดูแลลูกค้าอย่างมีระบบ

สถานการณ์ที่ระบบตอบโจทย์

สถานการณ์ที่ 1: พนักงานขายต้องเปิดหลายระบบเพื่อหาข้อมูลสินค้า

- สร้างหน้าเว็บเดียวที่รวมข้อมูลสินค้า, สเปก, และสต๊อกไว้ในจุดเดียว
- ใช้ฐานข้อมูลรวมศูนย์ผ่าน **Amazon DynamoDB** และเชื่อมต่อผ่าน API/Lambda
- ลดเวลาในการค้นหา เพิ่มประสิทธิภาพในการเสนอขาย

สถานการณ์ที่ 2: ข้อมูลไม่อัปเดตตรงกัน (เสนอขายสินค้าที่ไม่มีในคลัง)

- เช็คสต๊อกสินค้าแบบเรียลไทม์จากฐานข้อมูล
- ใช้ **AWS Lambda** เพื่อดึงข้อมูลสต๊อกปัจจุบันจาก DynamoDB
- ป้องกันการเสนอสินค้าที่หมดแล้ว ลดความผิดพลาดในการขาย

สถานการณ์ที่ 3: พนักงานขายหาเอกสารสเปกสินค้าไม่เจอ หรือไม่มีให้ส่งลูกค้า

- จัดเก็บเอกสาร PDF ของสเปกสินค้าไว้ใน **Amazon S3**
- ระบบสามารถค้นหาและดาวน์โหลดได้จากหน้าเดียว
- เพิ่มความสะดวกและความน่าเชื่อถือในการให้ข้อมูลแก่ลูกค้า

สถานการณ์ที่ 4: ไม่มีระบบติดตามลูกค้า (ไม่รู้ว่าลูกค้ารายใดอยู่ในสถานะใด)

- เพิ่มโมดูล CRM เบื้องต้นเพื่อบันทึกข้อมูลลูกค้าและสถานะ
- จัดเก็บลง **Amazon DynamoDB** โดยออกแบบให้รองรับการ Query ข้อมูลลูกค้าและสถานะได้อย่างมีประสิทธิภาพ
- ช่วยให้พนักงานเห็นภาพรวมลูกค้าทุกรายในมือ และไม่หลุดการติดตาม

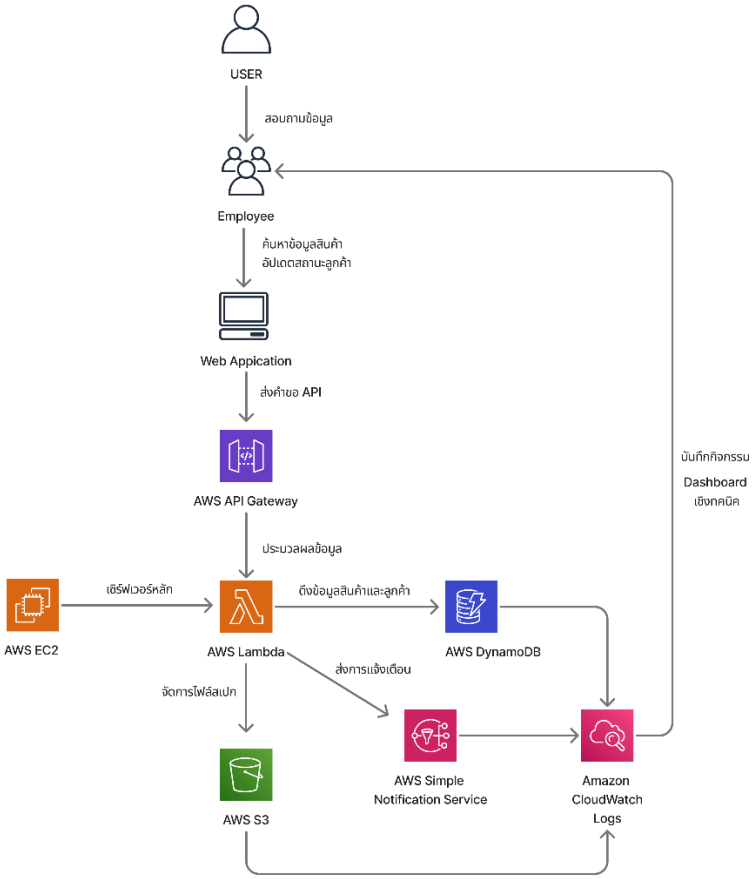
สถานการณ์ที่ 5: ผู้บริหารไม่สามารถวิเคราะห์ยอดขายหรือสินค้า Top Hit ได้

- สร้าง Dashboard การขายจากข้อมูลที่เก็บใน DynamoDB
- ใช้ **Amazon CloudWatch Logs Insights**ในการสร้างรายงาน เช่น ยอดขายตามเดือน
- ช่วยให้การวางแผนและตัดสินใจของผู้บริหารแม่นยำยิ่งขึ้น

สถานการณ์ที่ 6: ต้องการการแจ้งเตือนเมื่อเกิดเหตุการณ์สำคัญ เช่น สินค้าใกล้หมด

- ใช้ **Amazon SNS** แจ้งเตือนผ่านอีเมลหรือช่องทางอื่น เมื่อสินค้าต่ำกว่าเกณฑ์
- รองรับเหตุการณ์อื่น เช่น ลูกค้าใหม่, อัปโหลดไฟล์ผิด, การเปลี่ยนสถานะการขาย
- ทำให้ระบบตอบสนองแบบ proactive และลดความเสี่ยงจากความล่าช้า

สถาปัตยกรรมระบบ


USER
สอบถามข้อมูล
Employee
ค้นหาข้อมูลสินค้า
อัปเดตสถานะลูกค้า
Web Appication
ส่งคำขอ API
AWS API Gateway
ประมวลผลข้อมูล
เซิร์ฟเวอร์หลัก
AWS EC2
AWS Lambda
ดึงข้อมูลสินค้าและลูกค้า
AWS DynamoDB
ส่งการแจ้งเตือน
จัดการไฟล์สเปก
AWS S3
AWS Simple
Notification Service
Amazon
CloudWatch
Logs
บันทึกกิจกรรม
Dashboard
เชิงเทคนิค

อธิบายการทำงานขององค์ประกอบแต่ละส่วน

1. Amazon DynamoDB

ใช้เก็บข้อมูลหลักของระบบ เช่น:

1. ตาราง Products: ชื่อสินค้า, ราคา, stock, รูป, เอกสาร
2. ตาราง CustomerTracking: สถานะลูกค้าแต่ละราย
3. ตาราง Users: ข้อมูลผู้ใช้งาน + role (admin / staff)

เหตุผลที่เลือก:

- NoSQL ⟶ ยืดหยุ่น, ขยาย scale ได้อัตโนมัติ
- มีความเร็วสูงและ latency ต่ำ
- ไม่ต้องบริหารจัดการเซิร์ฟเวอร์

2. AWS Lambda

1. ฟังก์ชันที่ทำหน้าที่เป็น Backend ของระบบ (ไม่มี server)
2. เขียนด้วย Python ใช้เชื่อมกับ DynamoDB และ S3

มีหลายฟังก์ชัน เช่น:

- get-products: ดึงรายการสินค้า
- update-stock: เพิ่ม/ลด stock
- get-tracking: ดึงสถานะลูกค้า
- update-status: แก้ไขสถานะลูกค้า
- login-user: ตรวจสอบการเข้าสู่ระบบ
- generate-signed-url (ถ้าใช้ S3 แบบ private)

ข้อดี:

- ไม่เสียเงินเมื่อไม่ได้ใช้งาน

- scale ตาม traffic ได้อัตโนมัติ
- แยกความรับผิดชอบตามหน้าที่

3. Amazon API Gateway

ทำหน้าที่เป็น “ประตู” รับคำขอจาก frontend (ผ่าน HTTP)

Route ที่ใช้งาน เช่น:

- GET /products
- PUT /products (อัปเดต stock)
- POST /login
- GET /tracking
- PUT /tracking

ข้อดี:

- รองรับ HTTPS โดยไม่ต้องตั้งค่า SSL เอง
- ทำงานร่วมกับ Lambda ได้สะดวก
- มีระบบ Auth และ Rate limit ในตัว

4. Amazon S3 (Simple Storage Service)

ใช้เก็บ:

1. ไฟล์ PDF ของเอกสารสินค้า (product.doc)
2. รูปภาพสินค้า (product.picture)
3. ไฟล์ frontend (index.html, dashboard.html)

เหตุผลที่เลือก:

- รองรับ static website hosting ได้ง่าย
- โหลดไฟล์เร็วทั่วโลกด้วย CloudFront (ต่อยอดได้)

- ราคาถูกมาก และขยายได้ไม่จำกัด

5. Amazon SNS (Simple Notification Service)

1. ใช้ส่ง email แจ้งเตือนอัตโนมัติเมื่อ:
2. stock ของสินค้าต่ำกว่า 3 ชิ้นหลังอัปเดต

การทำงาน:

- Lambda update-stock ตรวจสอบ stock ใหม่
- ถ้า stock < 3 ⟶ sns.publish(...) ไปยัง topic
- ผู้ดูแลได้รับ email เตือนให้เติมสินค้า

ข้อดี:

- ทำงานแบบ async ไม่ส่งผลกับ latency
- สามารถต่อยอดไป SMS, Lambda อื่น หรือ webhook ได้ในอนาคต

6. IAM Roles and Policies

กำหนดสิทธิ์อย่างปลอดภัย:

- Lambda ใช้ role ที่สามารถอ่าน/เขียน DynamoDB และเรียก SNS ได้
- S3 เปิดเฉพาะ public read เฉพาะไฟล์ PDF
- ไม่มีการ hardcode credentials ในโค้ด

การทำงานร่วมกัน

กระบวนการ	การเชื่อมโยง
ผู้ใช้เปิดหน้าเว็บ	Browser ⟶ S3
โหลดสินค้า	frontend ⟶ API Gateway ⟶ Lambda ⟶ DynamoDB

อัปเดต stock	frontend ⟶ API Gateway ⟶ Lambda ⟶ DynamoDB ⟶ (SNS)
ดาวน์โหลด PDF	frontend ⟶ product.doc (URL จาก S3)
ดู dashboard	frontend ⟶ API ⟶ วิเคราะห์ข้อมูล + Chart
เปลี่ยนสถานะลูกค้า	frontend ⟶ API Gateway ⟶ Lambda ⟶ DynamoDB

ขั้นตอนหลักในการสร้างและตั้งค่าระบบ

ภาพรวมของกระบวนการพัฒนา

ระบบนี้ถูกสร้างขึ้นบนแนวคิด Serverless Web Application ซึ่งประกอบด้วย 3 ส่วนหลัก:

- Frontend (HTML/JS/CSS) – แสดงผลบนเบราว์เซอร์
- API Backend (AWS Lambda + API Gateway) – เชื่อมกับฐานข้อมูล
- Data Storage (DynamoDB, S3) – เก็บข้อมูลและเอกสา

ขั้นตอนที่ 1: สร้าง DynamoDB Table

1.1 ตาราง Products

- เปิด AWS Console ⟶ ไปที่ DynamoDB ⟶ Create Table
- Table name: Products
- Partition key: productId (String)

1.2 ตาราง CustomerTracking

- Table name: CustomerTracking
- Partition key: trackingId (String)
- Attributes: customerName, productName, staffId, status, updatedDate

1.3 ตาราง Users (สำหรับ Login)

- Table name: Users
- Partition key: userId (String)
- Attributes: name, email, password, role (admin / staff)

ขั้นตอนที่ 2: อัปโหลดไฟล์ PDF / รูปสินค้าไปยัง S3

1. สร้าง S3 Bucket เช่น product-spec-bucket
2. เปิดสิทธิ์ public read สำหรับไฟล์ .pdf ที่จะให้ดาวน์โหลด
3. อัปโหลดรูปภาพสินค้า (ใช้ productId ตั้งชื่อไฟล์ เช่น P001.png)
4. Copy URL มาใช้ใน picture, doc field ของตาราง Products

ขั้นตอนที่ 3: สร้าง Lambda Functions

ฟังก์ชันที่ต้องมี:

ชื่อฟังก์ชัน	หน้าที่
get-products	ดึงสินค้าทั้งหมด
update-stock	อัปเดต stock สินค้า
get-tracking	ดึงรายการติดตามลูกค้า
update-status	อัปเดตสถานะลูกค้า
login-user	ตรวจสอบการ login
add-tracking	เพิ่มข้อมูลการติดตาม
(option) signed-url	(option) signed-url gen URL สำหรับไฟล์ PDF

ขั้นตอน:

1.ไปที่ AWS Lambda > Create Function > Author from scratch

2.Runtime: Python 3.9

3.Permissions: สร้าง IAM role ใหม่หรือใช้ labRole

4.อัปโหลด zip ที่มี lambda_function.py และ requirements.txt (ถ้ามี)

5.ตั้ง Environment Variable (เช่น DB Table Name)

6.ทดสอบฟังก์ชันด้วย Test event

ขั้นตอนที่ 4: สร้าง API Gateway

- ไปที่ API Gateway > Create API
- เลือก REST API > Regional
- สร้าง resource เช่น /products, /tracking
- สร้าง method เช่น GET, PUT, POST แล้วเชื่อมกับ Lambda
- เปิด CORS (สำหรับให้เบราว์เซอร์ frontend เรียกได้)

ตัวอย่าง:

Method	URL path	เชื่อมกับ Lambda
GET	/get-products	get-products
PUT	/get-products	update-stock
GET	/tracking	get-tracking
PUT	/tracking	update-status
POST	/login	login-user

ขั้นตอนที่ 5: สร้าง SNS Topic (แจ้งเตือน)

1. ไปที่ Amazon SNS ⟶ Create topic ⟶ ชนิด Standard
2. ตั้งชื่อเช่น LowStockAlert
3. สร้าง Subscription ⟶ Email ⟶ ใส่อีเมลผู้ดูแล
4. ตรวจสอบอีเมลและกด Confirm

ขั้นตอนที่ 6: พัฒนาและอัปโหลด Frontend (index.html, login.html)

นำsourcecodeของระบบอัปโหลดลง Github

ขั้นตอนที่ 7: ทดสอบ End-to-End

- Login ⟶ ตรวจสอบ localStorage
- ไปหน้า index ⟶ ดึงสินค้า ⟶ แสดงภาพ + stock
- ไปหน้า index2 ⟶ อัปเดต stock ⟶ ทดสอบ SNS trigger
- เข้า dashboard.html ⟶ โหลด Chart.js ⟶ แสดงสรุป
- ไปหน้า customertracking.html ⟶ เปลี่ยนสถานะ ⟶ บันทึกวันที่

สรุป

องค์ประกอบ	บริการ AWS ที่ใช้	วิธีตั้งค่า
ฐานข้อมูลสินค้า	DynamoDB	สร้าง table + ใส่ข้อมูลเริ่มต้น
เอกสารสินค้า	S3	อัปโหลด PDF + เปิด public read
API Backend	Lambda + API Gateway	เขียน function แล้วเชื่อม endpoint
Dashboard	Chart.js + fetch	ดึงข้อมูลมาวิเคราะห์
แจ้งเตือน	SNS	ตั้ง topic แล้ว subscribe email

คาดการณ์ผลการทดสอบการทำงานของระบบ (End-to-End)

การทดสอบแบบ End-to-End (E2E) หมายถึงการจำลองเส้นทางการใช้งานของผู้ใช้จากต้นทางถึงปลายทาง โดยระบบจะต้องสามารถให้บริการได้ครบถ้วน ถูกต้อง และเสถียรในแต่ละกรณี

กรณีทดสอบที่ 1: ผู้ใช้งานทั่วไปเข้าสู่ระบบ และเรียกดูข้อมูลสินค้า

สถานการณ์

ผู้ใช้ (Staff A) ต้องการตรวจสอบข้อมูลสินค้า เพื่อดู stock และรายละเอียดสินค้า

ขั้นตอน:

1. เข้าสู่ระบบผ่าน login.html ด้วยอีเมลและรหัสผ่านที่ถูกต้อง
2. ระบบทำการตรวจสอบกับ DynamoDB ผ่าน Lambda login-user
3. เมื่อ login สำเร็จ ⟶ localStorage ถูกบันทึกค่าผู้ใช้

4. ผู้ใช้ถูก redirect ไปยังหน้า index.html
5. ระบบเรียก Lambda get-products ผ่าน API Gateway
6. แสดงรายการสินค้าในรูปแบบ product-card (รูป, ชื่อ, stock)
7. หาก stock ≤ 3 ⟶ มีแถบเตือนและกรอบเปลี่ยนเป็นสี
8. กด “Update สินค้า” ⟶ ไปหน้า index2.html ของสินค้านั้น

ผลที่คาดว่าจะได้:

- ข้อมูลโหลดสำเร็จทุกอย่าง
- รูป, ชื่อสินค้า, ราคา, stock แสดงถูกต้อง
- รายการที่ใกล้หมด stock แสดงแถบสีแดง/ส้มตามเงื่อนไข
- ประสบการณ์ผู้ใช้ราบรื่น ไม่มี error

กรณีทดสอบที่ 2: การอัปเดต stock สินค้า และระบบแจ้งเตือนผ่าน SNS

สถานการณ์:

Staff ต้องการอัปเดต stock ของสินค้า “เก้าอี้ไม้” จำนวน -5 ชิ้น และต้องดูว่า SNS แจ้งเตือนทำงานหรือไม่

ขั้นตอน:

- เข้าสู่หน้า index2.html?productId=P002
- ระบบเรียก get-products ⟶ แสดงข้อมูลสินค้า P002
- ใส่ค่า -5 ลงใน input และกด “Update Stock”
- ระบบเรียก update-stock ผ่าน API Gateway
- Lambda ตรวจสอบ stock ใหม่ (ถ้าเหลือ < 3)
- ระบบส่ง SNS ไปยัง email ที่สมัครไว้
- บนหน้าเว็บโหลดข้อมูลใหม่อีกครั้งและแสดงค่าที่อัปเดต

ผลที่คาดว่าจะได้:

- stock ลดลงสำเร็จ
- email แจ้งเตือนจาก SNS ถูกส่งภายในไม่เกิน 10 วินาที
- stock แสดงใหม่ถูกต้อง

- หาก stock < 0 ⟶ ระบบป้องกันไว้และแสดงข้อความเตือน

กรณีที่ 3: สถานการณ์ผิดพลาด – อัปเดต stock ติดลบ

สถานการณ์:

Staff ใส่จำนวน -20 ในสินค้าที่มี stock ปัจจุบันแค่ 5

ขั้นตอน:

1. ไปที่ index2.html?productId=P001
2. ระบบแสดง stock ปัจจุบัน = 5
3. กรอก -20 แล้วกด "Update Stock"
4. Lambda ตรวจสอบแล้วพบว่า newStock = -15
5. ระบบไม่อัปเดต ⟶ ส่งกลับ error
6. JS ตรวจจับ error และแสดง alert ให้ผู้ใช้

ข้อความที่แสดง:

"ไม่สามารถอัปเดตได้: stock จะติดลบ (-15)"

ผลที่คาดว่าจะได้:

- ระบบป้องกันไม่ให้ stock ติดลบ
- ไม่เกิดความเสียหายต่อข้อมูลใน DynamoDB
- UX ดี มีข้อความแจ้งเตือนผู้ใช้

กรณีทดสอบที่ 4: Dashboard ผู้บริหาร

สถานการณ์ :

Admin login และต้องการดูภาพรวมสินค้าในระบบ

ลำดับขั้นตอนที่ทดสอบ

1. Login ด้วย role “admin”
2. เข้าสู่หน้า dashboard.html
3. ระบบตรวจ role ⟶ อนุญาตให้ดูทุกข้อมูล
4. เรียก API get-products
5. แสดงผล:

- จำนวนสินค้า
- ยอดรวม stock
- สินค้าใกล้หมด
- กราฟแท่ง stock ต่อสินค้า (Chart.js)

ผลลัพธ์ที่ได้

- Dashboard โหลดได้ภายใน 1–2 วินาที
- กราฟแสดงค่า stock ได้ถูกต้อง
- ไม่มีปัญหาแม้ข้อมูลเกิน 20 รายการ

กรณีทดสอบที่ 5: ติดตามสถานะลูกค้า (Customer Tracking)

สถานการณ์ :

พนักงานต้องการอัปเดตสถานะของลูกค้ารายหนึ่งจากขั้น A ⟶ B

ลำดับขั้นตอนที่ทดสอบ

1. เข้าหน้า customertracking.html
2. ระบบแสดงรายการลูกค้าที่ผูกกับ staffId
3. กดปุ่ม "แก้ไข" ⟶ แสดง dropdown เปลี่ยนสถานะ
4. กด “บันทึก” ⟶ ระบบส่ง PUT /tracking
5. Lambda บันทึกค่าใหม่ใน DynamoDB และอัปเดตวันที่อัตโนมัติ

ผลลัพธ์ที่ได้

- สถานะลูกค้าถูกอัปเดตทันที
- แสดงวันที่อัปเดตล่าสุด
- ถ้า role เป็น Admin ⟶ เห็นทุก record แต่แก้ไขไม่ได้ (read-only)

สรุปการทดสอบ End-to-End

กรณี	ผลลัพธ์	ความเสถียร
ดูสินค้า	สำเร็จ	สูง
อัปเดต stock	สำเร็จ + SNS	สูง
stock ติดลบ	ป้องกันได้	สูง
Dashboard	โหลดเร็ว + กราฟถูกต้อง	สูง
Tracking	อัปเดตได้พร้อมเวลา	สูง

บทวิเคราะห์ข้อดีและข้อเสียของระบบ

1. Operational Excellence (การปฏิบัติงาน)

ข้อดี:

- ใช้ **AWS Lambda** และ **API Gateway** ทำให้ระบบเป็นแบบ Serverless ไม่ต้องดูแลเซิร์ฟเวอร์ ลดภาระ DevOps
- ระบบมี log ผ่าน **CloudWatch Logs** ช่วยให้ติดตาม ปรับปรุง และตรวจสอบเหตุการณ์ได้ง่าย
- รองรับการ deploy แบบ incremental ด้วย Lambda versioning

ข้อเสีย:

- หากไม่มีระบบ CI/CD หรือการจัดการ version อย่างเป็นระบบ อาจเกิดปัญหาจากการ deploy แบบ manual ในหลายทีม
- CloudWatch Insights ต้องใช้ความรู้ด้าน Query เพื่อวิเคราะห์ ทำให้ผู้ใช้ทั่วไปอาจใช้งานไม่คล่อง

2. Security (ความปลอดภัย)

ข้อดี:

- ใช้ **IAM Role/Policy** เพื่อจำกัดสิทธิ์ของแต่ละ service อย่างชัดเจน (เช่น Lambda เข้าถึงเฉพาะ S3 บาง bucket)
- รองรับการเข้ารหัสข้อมูลที่จัดเก็บใน **DynamoDB** และ **S3** ด้วย AWS-managed keys

ข้อเสีย:

- หากไม่ตั้งค่า IAM อย่างถูกต้อง อาจเสี่ยงต่อการเปิดสิทธิ์กว้างเกินไป (เช่น S3 public)
- ยังไม่มีระบบ Authen/Authorize ฝั่งผู้ใช้ในโครงงานนี้ (เช่น Cognito หรือ JWT) อาจเสี่ยงหากระบบขยายเป็น production

3. Reliability (ความน่าเชื่อถือ)

ข้อดี:

- ใช้บริการแบบ Managed (DynamoDB, S3, SNS) ที่มี High Availability โดยดีฟอลต์
- Lambda รองรับการ retry อัตโนมัติเมื่อมี error ชั่วคราว

ข้อเสีย:

- หากไม่ได้ใช้ Multi-AZ ใน DynamoDB (เช่น ใน Learner Lab) อาจมี downtime ได้
- ไม่มี queue buffer เช่น SQS ทำให้หาก Lambda fail แล้ว retry ไม่สำเร็จ ⟶ ข้อมูลสูญหายได้

4. Performance Efficiency (ประสิทธิภาพการทำงาน)

ข้อดี:

- Lambda scale ตามจำนวนผู้ใช้โดยอัตโนมัติ รองรับผู้ใช้หลายคนพร้อมกัน
- สามารถใช้ **DynamoDB** Index และ Query Optimization ปรับให้ระบบตอบสนองเร็ว

ข้อเสีย:

- หากจำนวน concurrent Lambda พุ่งสูง อาจต้องขอเพิ่ม soft limit
- หากออกแบบ query ใน DynamoDB ไม่ดี อาจทำให้ response time ช้าหรือเกิด connection error

5. Cost Optimization (ค่าใช้จ่าย)

ข้อดี:

- ใช้ Lambda และ S3 ที่มีค่าบริการต่ำหรือฟรีในระดับ Free Tier
- ใช้ SNS เฉพาะเมื่อจำเป็น ⟶ ไม่เกิดค่าใช้จ่ายแบบถาวร

ข้อเสีย:

- หากไม่มีการกำหนดเวลาหรือ logic ในการจัดเก็บ log อาจทำให้ค่า CloudWatch Logs พุ่งสูงในระยะยาว
- การเปิดใช้งาน DynamoDB โดยไม่ได้ scale down หรือ set auto stop ⟶ สิ้นเปลืองเครดิต/ค่าใช้จ่าย

6. Sustainability (ความยั่งยืนของระบบ)

ข้อดี:

- โครงสร้าง Serverless ลดทรัพยากร idle computing ⟶ เป็นมิตรต่อพลังงานและสิ่งแวดล้อม
- ใช้บริการที่มีการดูแลความยั่งยืนในดาต้าเซ็นเตอร์ของ AWS อยู่แล้ว

ข้อเสีย:

- หากไม่กำหนดอายุข้อมูล (data lifecycle) เช่น log หรือไฟล์เก่าใน S3 อาจใช้ storage เกินจำเป็น

สรุปภาพรวม

ด้าน	ข้อดี	จุดที่ควรปรับปรุง
การปฏิบัติงาน	Serverless, มี Log ครบ	ควรเพิ่ม CI/CD
ความปลอดภัย	IAM แยกบทบาท, Data Encryption	ยังไม่มีระบบ Authen ผู้ใช้
ความน่าเชื่อถือ	บริการ Managed, มี retry	ไม่มี queue สำรอง (SQS)
ประสิทธิภาพ	Auto-scale, Low latency	ต้อง optimize SQL Query
ค่าใช้จ่าย	Free Tier ใช้งานได้ครบ	ต้องจัดการ log rotation
ความยั่งยืน	ใช้ resource เท่าที่จำเป็น	ควรตั้ง data lifecycle

คาดการณ์สรุปผลการดำเนินโครงงาน และแนวทางพัฒนาต่อในอนาคต

สรุปผลการดำเนินโครงงาน

โครงงาน “RaPunSale” เป็นระบบ Web Application แบบ Serverless บน AWS ที่มุ่งเน้นการช่วยพนักงานขายและผู้บริหารในองค์กรที่มีสินค้าหลากหลาย

เช่น เครื่องใช้ไฟฟ้าและวัสดุก่อสร้าง โดยระบบสามารถ รวบรวม จัดการ และนำเสนอข้อมูลการขาย ได้ในหน้าจอเดียวอย่างมีประสิทธิภาพ

ฟีเจอร์ที่พัฒนาสำเร็จ:

หมวดหมู่	ฟีเจอร์
ระบบสินค้า	ค้นหา, แสดง, และอัปเดต stock แบบ real-time
ระบบติดตามลูกค้า	เปลี่ยนสถานะ + แสดงเวลาอัปเดต
ระบบ login	แยกบทบาท admin, staff และจำกัดสิทธิ์
Dashboard	แสดงสรุปภาพรวมสินค้าด้วย Chart.js
ระบบแจ้งเตือน	ใช้ SNS แจ้งเตือนเมื่อ stock ต่ำกว่าเกณฑ์
เอกสารสินค้า	ดาวน์โหลด PDF สเปกสินค้าจาก S3
ความปลอดภัย	ใช้ IAM และ Lambda role แทน hardcoded credentials
หมวดหมู่	ทดสอบ E2E แล้วสามารถใช้งานได้ครบทุกฟังก์ชัน

ระบบใช้บริการของ AWS ได้อย่างเหมาะสม และสามารถขยายต่อได้ในอนาคต โดยไม่มีข้อจำกัดจากโครงสร้างเดิม

2. จุดเด่นของระบบ

- ไม่มีเซิร์ฟเวอร์ (Serverless) ⟶ ลดภาระการดูแล infrastructure
- ขยายง่ายและคล่องตัว ⟶ เพิ่มตาราง, role, ฟีเจอร์ได้ง่าย
- ค่าบำรุงรักษาต่ำ ⟶ จ่ายเฉพาะเมื่อใช้งานจริง
- รวมศูนย์ข้อมูล ⟶ พนักงานไม่ต้องเปิดหลายระบบ
- เหมาะสำหรับทีมขาย ที่ต้องการใช้งานผ่าน browser และ mobile

3. แนวทางพัฒนาต่อในอนาคต

เพื่อเพิ่มความสมบูรณ์และยกระดับระบบให้ใช้งานได้จริงในองค์กรระดับกลางถึงใหญ่ สามารถต่อยอดได้ตามแนวทางต่อไปนี้:

3.1. ด้านระบบผู้ใช้ (Authentication & Authorization)

สิ่งที่มี	แนวทางพัฒนา
login ด้วย email/password ธรรมดา	เปลี่ยนเป็นระบบ Login ด้วย Amazon
ใช้ localStorage ตรวจ role	ใช้ JWT Token แทน เพื่อเพิ่มความ
ไม่มีระบบ reset password	เพิ่มฟีเจอร์ Forgot

3.2. ด้านการวิเคราะห์และรายงาน

สิ่งที่มี	แนวทางพัฒนา
Dashboard พื้นฐาน	เพิ่ม Filter ตามเดือน/สินค้า
ไม่มีรายงานยอดขายจริง	เชื่อมต่อข้อมูลการขายจริงจาก
ใช้ Chart.js แบบ static	เปลี่ยนเป็น Amazon QuickSight หรือ BI tools อื่น

3.3. ด้านการแจ้งเตือน

สิ่งที่มี	แนวทางพัฒนา
แจ้งเตือน stock ต่ำผ่าน email	เพิ่ม Line Notify หรือ SMS ผ่าน SNS
ไม่สามารถกำหนดเกณฑ์เอง	ให้ admin ตั้งค่า

3.4. ด้านระบบติดตามลูกค้า (Customer CRM)

สิ่งที่มี	แนวทางพัฒนา
สถานะลูกค้าแบบ 3 ขั้นตอน (A⟶B⟶C)	ขยายเป็น workflow เช่น “สนใจ ⟶ ทดลอง ⟶ ปิดการขาย”
บันทึกเหตุผลสั้น ๆ	เพิ่มหน้าต่าง comment history, แนบไฟล์
ไม่มีระบบแจ้งเตือน follow-up	เพิ่ม Reminder หรือ Calendar Integration

3.5. ด้าน UI/UX และระบบหน้าเว็บ

สิ่งที่มี	แนวทางพัฒนา
Static HTML	เปลี่ยนเป็น React หรือ Vue.js เพื่อความยืดหยุ่น
โหลด API ด้วย JavaScript	ใช้ Framework เช่น Next.js + ISR
ไม่มีระบบ responsive เต็มรูปแบบ	ปรับปรุง UI ให้เหมาะกับ mobile และ tablet มากขึ้น

3.6. ด้านการขยายระบบ

สิ่งที่มี	แนวทางพัฒนา
ข้อมูลสินค้าใน DynamoDB	สร้างระบบสั่งซื้อ
สินค้าแบบ manual	เชื่อมต่อ API กับ ERP หรือระบบ stock จริง
Web App แบบ internal	สร้าง Progressive Web App (PWA) รองรับการทำงานแบบ offline

4. สรุปสุดท้าย

โครงการนี้พิสูจน์ให้เห็นว่าองค์กรสามารถพัฒนาระบบรวมศูนย์ข้อมูลได้อย่างง่ายดายและมีประสิทธิภาพ โดยใช้บริการ AWS ที่พร้อมใช้งานและประหยัดงบประมาณ

ระบบ RaPunSale สามารถใช้งานจริงในหลายบริบท เช่น ทีมขาย, ฝ่ายขายหน้าร้าน, ผู้บริหารที่ต้องการภาพรวมสินค้า และทีม IT ที่ต้องการระบบขนาดย่อมแต่ยืดหยุ่นสูง

Link video Demo ตาม 3 สถานการณ์ : https://youtu.be/jcFGj_EPfD0