Project Proposal: CS361 KinKornNao

ข้อมูลทีม

- ชื่อโปรเจกต์: KinKornNao (กินก่อนเน่า)
- สมาชิกทีม: 6 คน (รหัสนักศึกษา-ชื่อนักศึกษา-ชื่อบัญชี GitHub)
    1. 6609650350 ณัฐธิดา บุญเสือ / NatthidaBunsuea
    2. 6609650491 ปิยธิดา ฤกษ์ดี / Muayminly
    3. 6609650335 ณัฏฐ์ เพิ่มกิตติกุล / Nat Pemkittikul - 6609650335
    4. 6609650756 อานุภาพ อนุรักษ์สยาม / ArnuparpAnuraksiam
    5. 6609650137 กชพร หาวิรส
    6. 6609650574 ภูมิภัทร แสนทองแก้ว

Problem Statement

ในชีวิตประจำวัน หลายครัวเรือนประสบปัญหาการจัดเก็บและติดตามวัตถุดิบในบ้าน เช่น ลืมว่าวัตถุดิบใดมีอยู่แล้ว ซื้อซ้ำโดยไม่จำเป็น หรือลืมใช้จนหมดอายุ ส่งผลให้เกิดความสิ้นเปลือง ค่าใช้จ่ายที่เพิ่มขึ้น และยังส่งผลต่อการเกิดขยะอาหาร (Food Waste) ซึ่งเป็นปัญหาสำคัญทั้งในระดับครัวเรือนและสังคมโดยรวม

ผู้ใช้และผู้มีส่วนเกี่ยวข้อง (Stakeholders):

- ผู้ใช้ทั่วไป/เจ้าของบ้าน (Primary Users): ต้องการระบบที่ช่วยจัดการวัตถุดิบในบ้านให้เป็นระเบียบ และลดการสูญเสียจากวัตถุดิบที่หมดอายุ
- สมาชิกครอบครัว (Household Members): ต้องการสามารถใช้งานระบบร่วมกัน เช่น เพิ่ม/ลบวัตถุดิบ ดูรายการที่มีอยู่ และรับการแจ้งเตือนร่วมกัน
- ผู้พัฒนาและดูแลระบบ (Developers/Operators): ต้องการสถาปัตยกรรมที่ปลอดภัย มีประสิทธิภาพ และสามารถขยายระบบได้ง่ายในอนาคต

Objectives

-สำหรับผู้ใช้ทั่วไป / เจ้าของบัญชี:

1. สมัครสมาชิกและเข้าสู่ระบบ (Login/Logout)
2. จัดการวัตถุดิบ (เพิ่ม, แก้ไข, ลบ)
3. ดูรายการวัตถุดิบพร้อมรายละเอียด เช่น ปริมาณ, หน่วย, วันหมดอายุ

4. ระบบแจ้งเตือนอัตโนมัติผ่านอีเมล เมื่อวัตถุดิบใกล้หมดอายุภายใน 3 วัน หรือหมดอายุแล้ว (เวลา 7:00 น. ทุกวัน)
5. จัดเรียงและค้นหาวัตถุดิบตามเงื่อนไขต่าง ๆ เช่น วันหมดอายุ, ปริมาณ

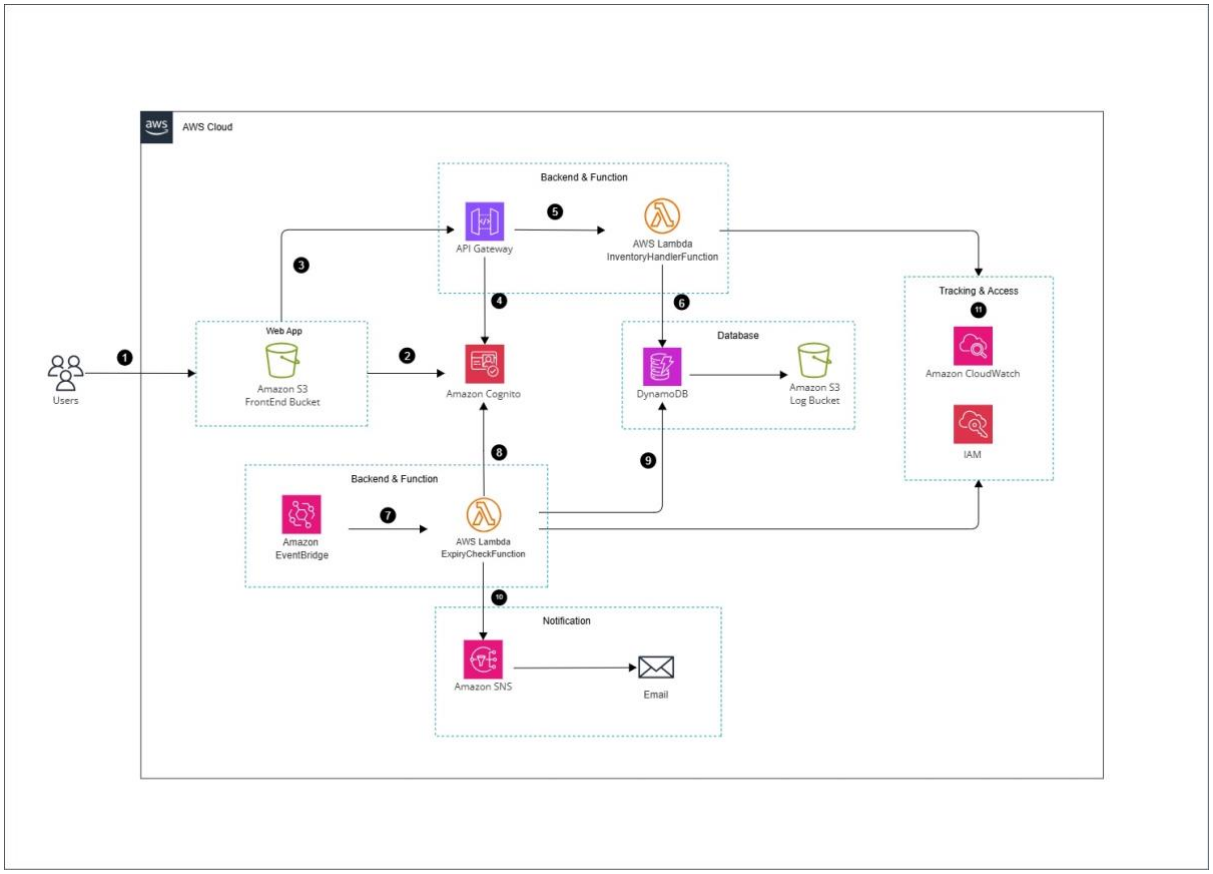
-สำหรับสมาชิกครอบครัว/ผู้ใช้ร่วม:

1. เข้าถึงรายการวัตถุดิบที่แชร์ภายในครอบครัว
2. เพิ่ม/ลบ/แก้ไขวัตถุดิบได้ตามสิทธิ์ที่เจ้าของบัญชีกำหนด
3. รับอีเมลแจ้งเตือนร่วมกัน หากวัตถุดิบใกล้หมดอายุหรือหมดอายุแล้ว

-สำหรับผู้ดูแลระบบ (Backend/Cloud operators):

1. ดูแลความปลอดภัยของข้อมูลและสิทธิ์การเข้าถึง
2. ตรวจสอบและปรับปรุงประสิทธิภาพระบบให้พร้อมใช้งาน
3. พัฒนา/ขยายระบบได้ง่าย เพื่อรองรับจำนวนผู้ใช้ที่เพิ่มขึ้น

Initial Architecture Diagram (**อันเก่า**)



Assumptions

- ผู้ใช้ทุกคนมีอีเมลที่ยืนยัน (verified) สำหรับรับอีเมลแจ้งเตือน
- ระบบจะโฮสต์ทั้งหมดใน us-east-1 (region เดียว) เพื่อความเรียบง่าย
- ปริมาณผู้ใช้ในระยะเริ่มต้น < 1,000 ผู้ใช้ และไม่จำเป็นต้องการ multi-region replication ตอนแรก
- ผู้ใช้ล็อกอินผ่าน Cognito (no social login in v1)
- Frontend เป็น static SPA โฮสต์บน S3/CloudFront

Constraints

- งบประมาณจำกัด เลือกบริการ managed เพื่อเลี่ยงต้นทุนการดูแลโครงข่ายมากเกินไป
- DynamoDB design constraints ต้องออกแบบ partition key ให้เหมาะสม (userId partition) เพื่อหลีกเลี่ยง hotspot
- CORS & browser security API ต้องตอบ preflight (OPTIONS) และตั้ง header ให้ถูกต้อง
- Latency expectation API response < 500ms สำหรับคำขอ CRUD ปกติ

- Privacy / Data retention เก็บข้อมูลผู้ใช้ตามนโยบาย (ไม่เก็บ sensitive data เกินจำเป็น)
- Third-party limits SNS/SES rate limits, Cognito token expiry

Well-Architected Analysis

สภาพปัจจุบัน: ระบบใช้ S3 สำหรับหน้าเว็บ, API Gateway + JWT/Cognito สำหรับจัดการสิทธิ์, Lambda สำหรับประมวลผล, DynamoDB เก็บข้อมูล, และ SNS ส่งอีเมลแจ้งเตือนเวลา 7:00 น.

1) Operational Excellence (ประสิทธิภาพในการดำเนินงาน)

- จุดแข็ง:
    - โค้ดแยกเป็นฟังก์ชันชัดเจน
    - มี CloudWatch Logs ตรวจสอบการทำงาน
    - การอัพเดตระบบทำได้ง่าย
- จุดอ่อน:
    - ยังไม่มีเอกสารขั้นตอนแก้ปัญหา (Runbook/Playbook)
    - ระบบ CI/CD ยังไม่ครบวงจร
    - การเก็บ Log ยังไม่เป็นระเบียบและติดตามแต่ละคำสั่งยาก
- แนวทางปรับปรุง:
    - ใช้ GitHub Actions ทำ lint, test, และ deploy จาก dev ⟶ prod
    - จัดเก็บ Log เป็น structured พร้อมใส่รหัสติดตาม (Correlation ID)
    - ทำหน้า Health Check / Status Page
    - เขียน Runbook แก้ปัญหาต่าง ๆ

2) Security (ความปลอดภัย)

- จุดแข็ง:
    - ใช้ Cognito + JWT ตรวจสอบผู้ใช้
    - API Gateway มี Authorizer
    - หน้าเว็บ S3 static hosting ปลอดภัย
- จุดอ่อน:
    - สิทธิ์ IAM อาจกว้างเกิน ไม่ได้กำหนดเฉพาะเจาะจง
    - การจัดการ secret หรือ parameter ยังไม่ชัดเจน

    - การตั้งค่า CORS ผิดพลาด อาจทำให้เกิด 401
- แนวทางปรับปรุง:
    - กำหนด IAM แบบ least-privilege ให้ Lambda, DynamoDB, SNS
    - เก็บ secret/parameter ใน SSM Parameter Store
    - ควบคุม CORS ให้อนุญาตเฉพาะ origin ที่จำเป็น
    - จัดการ Token ฝั่งหน้าเว็บให้รองรับการหมดอายุและรีเฟรช

3) Reliability (ความเสถียร)

- จุดแข็ง:
    - ใช้ Serverless ลดจุดล้มเหลว
    - AWS ให้บริการ Multi-AZ อยู่แล้ว
- จุดอ่อน:
    - ระบบแจ้งเตือนยังไม่มี Retry/DLQ
    - การเขียนข้อมูลซ้ำยังไม่ได้ป้องกัน
    - ไม่มี Alarm ติดตาม SLA
- แนวทางปรับปรุง:
    - ตั้ง DLQ + Retry สำหรับ Lambda และ SNS
    - ใช้ Idempotency Key ป้องกันเขียนซ้ำ
    - ทำ CloudWatch Alarm สำหรับ 5xx, latency, throttling
    - Backup และ export ข้อมูลสำคัญเป็นประจำ

4) Performance Efficiency (ประสิทธิภาพ)

- จุดแข็ง:
    - DynamoDB ตอบสนองเร็ว
    - Lambda scale อัตโนมัติ
- จุดอ่อน:
    - Schema หรือ Index อาจยังไม่เหมาะกับ use case หลายมิติ
    - ไม่มี caching
    - Cold start ของ Lambda ยังไม่ได้คำนึงถึง
- แนวทางปรับปรุง:

    - ออกแบบ Primary Key หรือ Sort Key และ GSI ให้เหมาะกับ use case
    - Query เฉพาะ field ที่ต้องใช้
    - เปิด short-lived cache สำหรับอ่านข้อมูลบ่อย
    - ปรับ Lambda memory และ Node version ให้เหมาะสม

5) Cost Optimization (การใช้ค่าใช้จ่ายอย่างเหมาะสม)

- จุดแข็ง:
    - จ่ายตามที่ใช้จริง ไม่มีเซิร์ฟเวอร์ค้าง
- จุดอ่อน:
    - ยังไม่มี Budget/Alert
    - Provisioned RCU/WCU ไม่ถูกคำนึงถึง
    - งาน Batch/Scan เสี่ยงค่าใช้จ่ายบานปลาย
- แนวทางปรับปรุง:
    - ตั้ง AWS Budgets + Alarm
    - ใช้ On-demand / ปรับ RCU/WCU ให้เหมาะสม
    - ลดการ Scan/Batch write-read ขนาดใหญ่

6) Sustainability (ความยั่งยืน)

- จุดแข็ง:
    - Serverless ใช้ทรัพยากรตามจำเป็น
- จุดอ่อน:
    - ไม่มีการตรวจงานที่ซ้ำกัน
    - ไม่มีนโยบายล้างข้อมูลชั่วคราว
- แนวทางปรับปรุง:
    - ลบข้อมูลชั่วคราวด้วย TTL
    - ปิดทรัพยากร dev ที่ไม่ใช้
    - รวม Log/Metric และตั้งนโยบาย retention

Development / Improvement Plan

Checkpoint #1 (สัปดาห์ที่ 5–6)

ช่วงเวลา: หลังส่ง proposal ก่อนสัปดาห์ 6

- Frontend
    - สร้างโครงร่างหน้าเว็บเบื้องต้น (Login, Dashboard แสดงรายการวัตถุดิบ)
    - เชื่อมต่อ API Gateway แบบ mock / Lambda stub เพื่อลอง flow
- Backend
    - ออกแบบ DynamoDB schema (userId, itemId, expiryDate)
    - พัฒนา Lambda สำหรับ CRUD (เพิ่ม/ลบ/แก้ไข/ดึงรายการวัตถุดิบ)
- Cloud / Infra
    - ตั้งค่า Cognito User Pool (สมัครสมาชิก/ล็อกอิน)
    - สร้าง API Gateway (HTTP API) + integrate Lambda
    - ตั้งค่า S3 hosting สำหรับ frontend (Dev environment)
- เป้าหมาย: ให้ระบบสามารถ Login และจัดการข้อมูลใน DynamoDB ผ่าน API ได้ ⟶ มี ฟังก์ชันหลักอย่างน้อย 30%

Checkpoint #2 (สัปดาห์ที่ 10–11)

ช่วงเวลา: หลังส่งงาน #1ก่อนสัปดาห์ 11

- Frontend
    - เพิ่ม UI ดูรายละเอียดสินค้า + วันหมดอายุ
    - เพิ่มการแสดงผลแจ้งเตือน (warning: ใกล้หมดอายุ / expired)
- Backend
    - Lambda Schedule ผ่าน EventBridge ตรวจสอบสินค้าใกล้หมดอายุใน 3 วัน และ expired
    - ส่งแจ้งเตือนผ่าน SNS/SES ไปยัง email ผู้ใช้
- Cloud / Infra
    - เพิ่ม CI/CD (GitHub Actions S3 + Lambda deploy)
    - Monitor ผ่าน CloudWatch (log, metrics, alarms)
- เป้าหมาย: ระบบแจ้งเตือนเริ่มทำงานจริง, รองรับผู้ใช้หลายคน, ฟังก์ชันครบอย่างน้อย 70%

Final (สัปดาห์ที่ 15)

ช่วงเวลา: หลังส่งงาน #2 สัปดาห์ 15

- Frontend
    - เพิ่มหน้าโปรไฟล์ผู้ใช้ (แก้ไขข้อมูลติดต่อ, email, password reset)
    - UI สมบูรณ์และ responsive
- Backend
    - ปรับปรุง query (เช่น index DynamoDB) ให้ทำงานเร็วขึ้น
    - เพิ่มฟังก์ชัน optional เช่น “แนะนำเมนูจากวัตถุดิบที่มี” หรือ “แชร์รายการกับเพื่อนในบ้าน”
- Cloud / Infra
    - Optimize security (CORS allow เฉพาะ domain จริง, SES ออกจาก sandbox)
    - Load testing + tuning DynamoDB/Lambda timeout
- เป้าหมาย: Demo ระบบครบ 100% ตาม objectives + วิเคราะห์ตาม Well-Architected Framework

Success Criteria

1. Functional Success
    - ผู้ใช้ ล็อกอิน/ล็อกเอาท์ ได้ token ถูกต้อง, session ทำงาน
    - สามารถ เพิ่ม/ลบ/แก้ไข วัตถุดิบ ข้อมูลเก็บใน DynamoDB อย่างถูกต้อง
    - ดูรายการวัตถุดิบ พร้อมวันหมดอายุ แสดงผลตรงกับ DB
    - ระบบ ส่งอีเมลแจ้งเตือน วัตถุดิบใกล้หมดอายุ/หมดอายุทุกเช้า 07:00 น.
    - ระบบสามารถแนะนำเมนูจากวัตถุดิบที่มี
    - ระบบสามารถดูหน้าแดชบอร์ดสรุปผลรายสัปดาห์
    - ระบบสามารถแชร์วัตถุดิบร่วมกันระหว่างคนในครอบครัวหรือคนใสบ้านเดียวกัน
2. Technical Success
    - ระบบมี Uptime $\geq$ 99% (ตาม CloudWatch logs)
    - Response time ของ API (CRUD) $\leq$ 500ms เฉลี่ย
    - DynamoDB query latency $\leq$ 50ms ต่อ request

3. Team/Process Success
    - สมาชิกทุกคนมี contribution ใน GitHub (commits, issues, PR)
    - ใช้ Wiki อัปเดต progress ชัดเจนในแต่ละ Checkpoint
    - ระบบพัฒนาเสร็จ $\geq$ 90% ของ objectives ที่ตั้งไว้ใน proposal
4. Well-Architected Alignment
    - Security: Authentication ผ่าน Cognito + IAM roles จำกัดสิทธิ
    - Reliability: Lambda stateless, ใช้ retry mechanism, error logs
    - Performance Efficiency: DynamoDB with proper keys, On-demand scaling
    - Operational Excellence: Monitoring ผ่าน CloudWatch + CI/CD pipeline
    - Cost Optimization: ใช้ serverless (จ่ายตามใช้จริง ไม่ต้องมี EC2 ตลอดเวลา)

Functions (สรุปฟีเจอร์หลัก)

1. Authentication: login/logout ผ่าน Cognito
2. Inventory Management: เพิ่ม/แก้ไข/ลบ วัตถุดิบ พร้อมรายละเอียด (ชื่อ, จำนวน, วันหมดอายุ)
3. View Items: ดูรายการทั้งหมด + search/filter/sort
4. Notification: อีเมลแจ้งเตือนทุกเช้า 7 โมง
    - วัตถุดิบที่ใกล้หมดอายุใน 3 วัน
    - วัตถุดิบที่หมดอายุแล้ว
5. User Profile: จัดการข้อมูลส่วนตัว
6. Extra Features (Final):
    - แนะนำเมนูจากวัตถุดิบที่มี
    - Drag & Drop reorder
    - Report ลดการสูญเสีย
7. Dashboard วิเคราะห์การใช้วัตถุดิบ (Ingredient Usage Analytics)

- ผู้ใช้สามารถดูสถิติ เช่น
    - วัตถุดิบที่หมดอายุบ่อยที่สุด
    - วัตถุดิบที่ใช้มากที่สุด / น้อยที่สุด
    - กราฟแนวโน้มการใช้ของแต่ละเดือน
- ประโยชน์: ช่วยให้ผู้ใช้วางแผนซื้อของได้ดีขึ้น ลดการทิ้งของ

8. การแชร์รายการวัตถุดิบในครอบครัว (Family / Group Sharing)

- ผู้ใช้สามารถ แชร์ตู้เย็นเสมือน (inventory) ร่วมกับสมาชิกในครอบครัว/เพื่อนได้
- ทุกคนเห็นรายการเดียวกันแบบเรียลไทม์
- ประโยชน์: ป้องกันซื้อซ้ำ, จัดการวัตถุดิบในบ้านร่วมกันได้สะดวก