รายงานโปรเจควิชา

DE331: Introduction to Cloud Computing And Data Engineering

เรื่อง ระบบติดตามและบันทึกผลการวิเคราะห์ AI ด้วย Gemini และ AWS S3 Log Storage

เสนอ

ผศ.ดร.วีรยุทธ เจริญเรืองกิจ

จัดทำโดย

นายปัณณธร สุวรรณาศรัย รหัสนิสิต 66102010175

นายพลวัต พงศ์ทิพย์พนัส รหัสนิสิต 66102010249

นายพลวิชญ์ วนิดา รหัสนิสิต 66102010584

รายงานนี้เป็นส่วนหนึ่งของรายวิชา DE331 Introduction to Cloud Computing And Data Engineering

ภาคเรียนที่ 1 ปีการศึกษาพุทธศักราช 2568

มหาวิทยาลัยศรีนครินทรวิโรฒ

Executive Summary

1.1 การทำงานของระบบ

ระบบติดตามและบันทึกผลการวิเคราะห์ AI ด้วย Gemini และ AWS S3 Log Storage คือแพลตฟอร์มสำหรับตรวจสอบและรับรองคุณภาพของ AI แบบอัตโนมัติ โดยใช้ Gemini เป็นเครื่องมือหลักในการวิเคราะห์ ประเมินผล และตรวจสอบผลลัพธ์ที่สร้างโดยระบบ AI อื่น ๆ (เช่น โมเดล Machine Learning หรือ Generative AI) ซึ่งโดยตัวระบบนี้ถูกติดตั้งและทำงานบนโครงสร้างพื้นฐานคลาวด์ของ Amazon Web Services (AWS) เพื่อรับรองเสถียรภาพ ความสามารถในการปรับขนาด และประสิทธิภาพสูงสำหรับการดำเนินงานได้อย่างต่อเนื่อง

1.2 ระบบแก้ไขปัญหาอะไร

1.2.1 การตรวจสอบคุณภาพ AI ที่ไม่มีมาตรฐานและใช้เวลานาน

แก้ไขโดย: การตรวจสอบอัตโนมัติและความแม่นยำและใช้ความสามารถในการวิเคราะห์ที่เหนือชั้นของตัว Gemini ในการให้คะแนนความเชื่อมั่น (Confidence Score) และยืนยันความถูกต้องของผลลัพธ์ AI ได้ทันทีในการวิเคราะห์ผลลัพธ์ซึ่งเป็นการเน้นย้ำถึงค่า Confidence Score หรือความมั่นใจในการทำนายค่าซึ่งเป็น Output หลักของระบบ

1.2.2 ความเสี่ยงด้านการกำกับดูแลและความน่าเชื่อถือ

แก้ไขโดย: การติดตามและตรวจสอบที่เชื่อถือได้บนระบบของคลาวด์ที่ระบบเป็นของ AWS เพื่อสร้างความมั่นใจและความน่าเชื่อถือรวมถึงความพร้อมในการใช้งานและกระบวนการตรวจสอบที่เข้มงวดของระบบ

1.2.3 ปัญหาการจัดเก็บ Log และ Audit Trail ขนาดใหญ่

แก้ไขโดย: การเก็บ Log ที่ยั่งยืนด้วย Amazon S3 ใช้ Amazon S3 ในการจัดเก็บ Log การตรวจสอบและข้อมูลดิบทั้งหมดอย่างถาวรและปรับขนาดได้ไม่จำกัด ช่วยให้การตรวจสอบย้อนหลังและการปฏิบัติตามข้อกำหนดด้านการกำกับดูแลเป็นไปอย่างง่ายดายและประหยัดต้นทุน

สถานการณ์ทางธุรกิจและปัญหา (Business Scenario & Problem)

2.1 ตัวอย่างสถานการณ์ทางธุรกิจ (Business Scenario)

บริษัท A ซึ่งเป็นผู้ให้บริการแพลตฟอร์มดิจิทัลขนาดใหญ่ มีความจำเป็นต้องเสริมสร้างความมั่นใจในคุณภาพและความถูกต้องตามกฎหมายของเนื้อหาที่ผู้ใช้สร้างขึ้น (User-Generated Content: UGC) ที่ส่งเข้าสู่ระบบ ดังนั้น บริษัทจึงต้องการแพลตฟอร์มสำหรับตรวจสอบและคัดกรองเนื้อหา UGC อย่างเร่งด่วน เพื่อแยกแยะว่าเนื้อหานั้นถูกสร้างหรือปรับแต่งโดย AI (Generative AI) หรือไม่ ทั้งนี้ เพื่อลดความเสี่ยงสำคัญสองประการ คือ

2.1.1 เพื่อป้องกันการละเมิดลิขสิทธิ์ – ป้องกันไม่ให้เนื้อหาที่มีแหล่งที่มาไม่ชัดเจนหรืออาจละเมิดลิขสิทธิ์ถูกเผยแพร่ในระบบ

2.1.2 เพื่อควบคุมคุณภาพข้อมูล - ป้องกันไม่ให้มีเนื้อหาคุณภาพต่ำที่สร้างโดย AI จำนวนมากเข้าสู่แพลตฟอร์ม ซึ่งอาจส่งผลกระทบต่อประสบการณ์ของผู้ใช้และความน่าเชื่อถือของบริการได้

ความต้องการของข้อมูล

เพื่อให้การตรวจสอบและจัดการเนื้อหา AI เป็นไปตามมาตรฐานระดับองค์กร ข้อมูลและผลลัพธ์จากการวิเคราะห์ทั้งหมดจึงต้องได้รับการดูแลอย่างเข้มงวด โดยมีข้อกำหนดหลักดังนี้

1. ความปลอดภัยและความคงทน (Durability): ข้อมูลบันทึกการตรวจสอบ (Audit Logs) ต้องถูกเก็บรักษาไว้ในบริการจัดเก็บข้อมูลของ AWS ที่มีคุณสมบัติด้านความปลอดภัยและความทนทาน เช่น Amazon S3 เพื่อป้องกันการสูญหายและรองรับการใช้งานในระยะยาว
2. ความสามารถในการตรวจสอบย้อนหลัง (Auditability): ข้อมูลต้องสามารถเรียกค้นและตรวจสอบย้อนหลังได้อย่างสะดวก เพื่อรองรับกระบวนการกำกับดูแลภายในและการปฏิบัติตามข้อกำหนดที่เกี่ยวข้อง

2.2 ปัญหาและความท้าทายทางเทคนิค (Technical Challenges)

ในการเปลี่ยนแปลงจากการพัฒนาบนเครื่องท้องถิ่นไปสู่การใช้งานจริงบน AWS Cloud โปรเจกต์นี้พบอุปสรรคสำคัญ ดังนี้

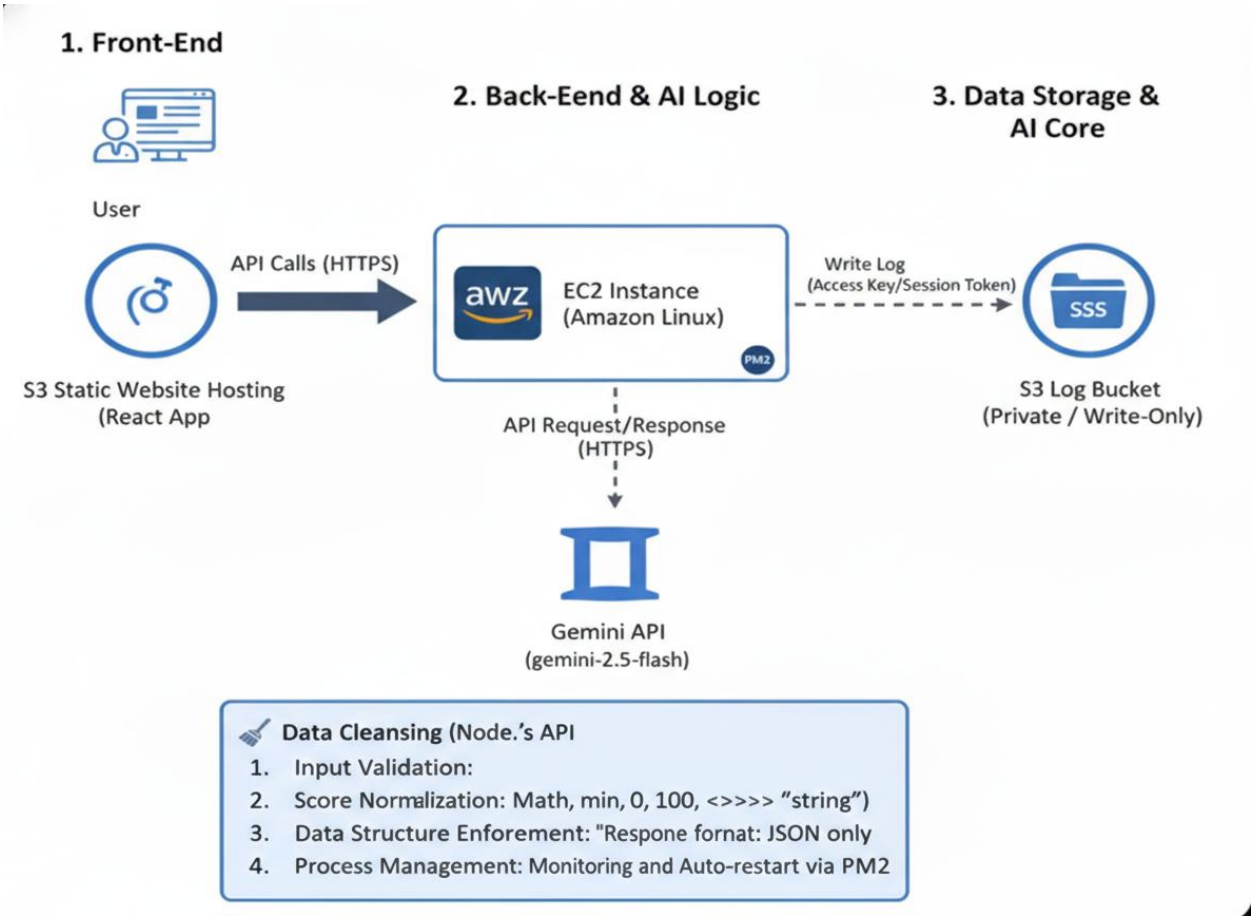
2.2.1 **Deployment Issues -** ระบบไม่สามารถใช้บริการจำลองบนเครื่อง Local (เช่น localhost database) ได้เมื่อย้ายไป Deploy บน AWS ทำให้ต้องเปลี่ยนไปใช้บริการของ AWS ที่รองรับการใช้งานจริง โดยในสถาปัตยกรรมสุดท้ายได้เลือกใช้ Amazon S3 เป็น Log Storage หลัก 2. AWS Lab Environment **Limitations**

2.2.2 **ข้อจำกัดของสภาพแวดล้อม AWS Lab -** ผู้พัฒนาขาดอิสระในการสร้างหรือปรับแต่ง IAM Role ส่งผลให้การกำหนดสิทธิ์การเข้าถึงของแต่ละบริการมีข้อจำกัด แก้ไขโดยใช้ Access Keys หรือ Session Token แทนการกำหนด Role โดยตรง

2.2.3 **สิทธิ์การเข้าถึง DynamoDB -** พบปัญหาการถูกปฏิเสธสิทธิ์ (Access Denied) โดยเฉพาะในฟังก์ชัน dynamodb:Query จึงจำเป็นต้องเปลี่ยนแปลงสถาปัตยกรรมจากการใช้ DynamoDB มาเป็นการจัดเก็บ Log ด้วย Amazon S3 แทน ซึ่งตอบโจทย์ความต้องการใช้ Log Storage ได้อย่างมีประสิทธิภาพ

การออกแบบสถาปัตยกรรมและแนวทางแก้ไข (Final Architecture)

3.1 แผนภาพสถาปัตยกรรม



คำอธิบายแผนภาพ:

User (ผู้ใช้งาน): ผู้ที่เข้ามาใช้งานเว็บไซต์ของคุณ

AWS S3 (Front-End Website): Bucket S3 ที่ Public สำหรับ Host ไฟล์ index.html และ Assets ต่างๆ ของ React App

Internet: เส้นทางที่ User เชื่อมต่อ AWS

EC2 (Back-End API): EC2 Instance ที่รัน Node.js API (Express) บน Port 5005 server.js (PM2)

Google Gemini API: บริการ AI ภายนอกที่ EC2 เรียกใช้เพื่อวิเคราะห์ข้อความ

AWS S3 (Log Storage): Bucket S3 ที่ Private สำหรับเก็บไฟล์ Log JSON (report-timestamp.json) ที่ Back-End สร้างขึ้น

3.2 สถาปัตยกรรมที่ใช้งานจริง (Final Deployed Solution)

3.2.1 Front-End (Amazon S3 Static Website Hosting) : ทำหน้าที่ Host React Application ให้ผู้ใช้งานเข้าถึงได้อย่างรวดเร็วและประหยัดค่าใช้จ่าย

3.2.2 Back-End (EC2 Instance): รัน Node.js API ซึ่งเป็น Logic หลักของระบบ ควบคุมการทำงานให้ต่อเนื่องด้วย PM2

3.2.3 AI Core (Gemini API) : ทำหน้าที่ตรวจสอบเนื้อหา (Core AI Logic)

3.2.4 Storage (Log, S3 Bucket) : จัดเก็บ Audit Logs และผลการตรวจสอบทั้งหมด โดยถูกกำหนดให้เป็น Write-Only Log Storage เพื่อความปลอดภัย

3.3 Data Cleansing และ Input Robustness

เพื่อเพิ่มความเสถียรและความแข็งแกร่ง (Robustness) ให้กับระบบ API ก่อนการส่งคำขอไปยัง Gemini API จึงมีการ Implement โค้ดสำหรับทำความสะอาดและตรวจสอบข้อมูลในไฟล์ server.js ดังนี้

3.3.1 Input Validation — ตรวจสอบข้อมูลที่รับเข้ามา

```
if (!text || typeof text !== "string") {
    return res.status(400).json({ error: "Invalid input" });
}
```

ใช้ตรวจสอบว่าค่าที่ส่งเข้ามาต้องเป็นข้อความจริงๆ เพื่อป้องกัน Error หากผู้ใช้ส่งข้อมูลว่าง หรือส่งข้อมูลเป็นชนิดอื่น เช่น ตัวเลข, JSON

3.3.2 Normalization — ปรับค่าความเชื่อมั่น (confidence score) ให้ถูกต้อง บีบค่าความเชื่อมั่นให้อยู่ในช่วง 0 - 100% เสมอป้องกันไม่ให้ค่าหลุดกรอบ เช่น ติดลบ หรือเกิน 100

3.3.3 Data Structure Enforcement — บังคับให้ Gemini ตอบเป็น JSON เสมอใช้ฟีเจอร์ของ Gemini API เพื่อกำหนดให้ Output ต้องเป็น JSON Structure เท่านั้น ซึ่งช่วยป้องกันไม่ให้ระบบ Back-End (Node.js) เกิดข้อผิดพลาดเมื่อพยายาม Parse ข้อความที่ไม่ใช่ JSON และเพิ่มความน่าเชื่อถือในการจัดการข้อมูลและป้องกันการ crash

ผลการดำเนินงานโครงงาน

4.1 ผลการพัฒนาโปรแกรม

จากการดำเนินโครงการ "ระบบติดตามและบันทึกผลการวิเคราะห์ AI ด้วย Gemini และ AWS S3 Log Storage" ระบบที่พัฒนาได้มีคุณสมบัติและองค์ประกอบหลักดังนี้:

4.1.1 ระบบตรวจสอบเนื้อหา (AI Verification Core): พัฒนาด้วย Node.js เชื่อมต่อกับ Gemini API (Model: gemini-2.5-flash) ที่สามารถรับข้อความ input และวิเคราะห์ค่าความน่าจะเป็น (Confidence Score) ว่าข้อความนั้นถูกสร้างโดย AI หรือมนุษย์ พร้อมทั้งทำ Data Cleansing เพื่อป้องกันข้อมูลผิดพลาดก่อนประมวลผลได้

4.1.2 ระบบบันทึกข้อมูลตรวจสอบ (Audit Logging on AWS S3): ระบบสามารถบันทึกผลการตรวจสอบทุกครั้งลงใน Amazon S3 ในรูปแบบไฟล์ JSON (Log File) เพื่อความทนทาน (Durability) และสามารถตรวจสอบย้อนหลังได้ (Auditable) โดยมีการจัดการความปลอดภัยผ่าน Environment Variables

4.1.3 ระบบบริหารจัดการกระบวนการ (Process Management): ใช้ PM2 ในการควบคุมการทำงานของ Server บน EC2 Instance ให้สามารถทำงานได้ต่อเนื่อง (Keep-alive) และ Restart อัตโนมัติเมื่อเกิดข้อผิดพลาด

4.1.4 ส่วนติดต่อผู้ใช้งาน (User Interface): พัฒนา Dashboard ด้วย React แสดงสถานะการเชื่อมต่อผ่านตัว Server และกราฟสถิติความมั่นใจ (Recharts) และประวัติการตรวจสอบล่าสุดของผู้ใช้

4.2 ผลการทดสอบระบบ

จากการทดสอบระบบทั้งในสภาพแวดล้อมจำลอง (Localhost) และบน Cloud (AWS EC2) พบผลสำคัญดังนี้

4.2.1 ความถูกต้องของข้อมูล (Data Integrity): ระบบสามารถบันทึกไฟล์ Log ลงใน Amazon S3 ได้ครบถ้วน 100% โดยไม่มีการสูญหายของข้อมูล แม้มีการส่ง Request ในจำนวนมากและถี่

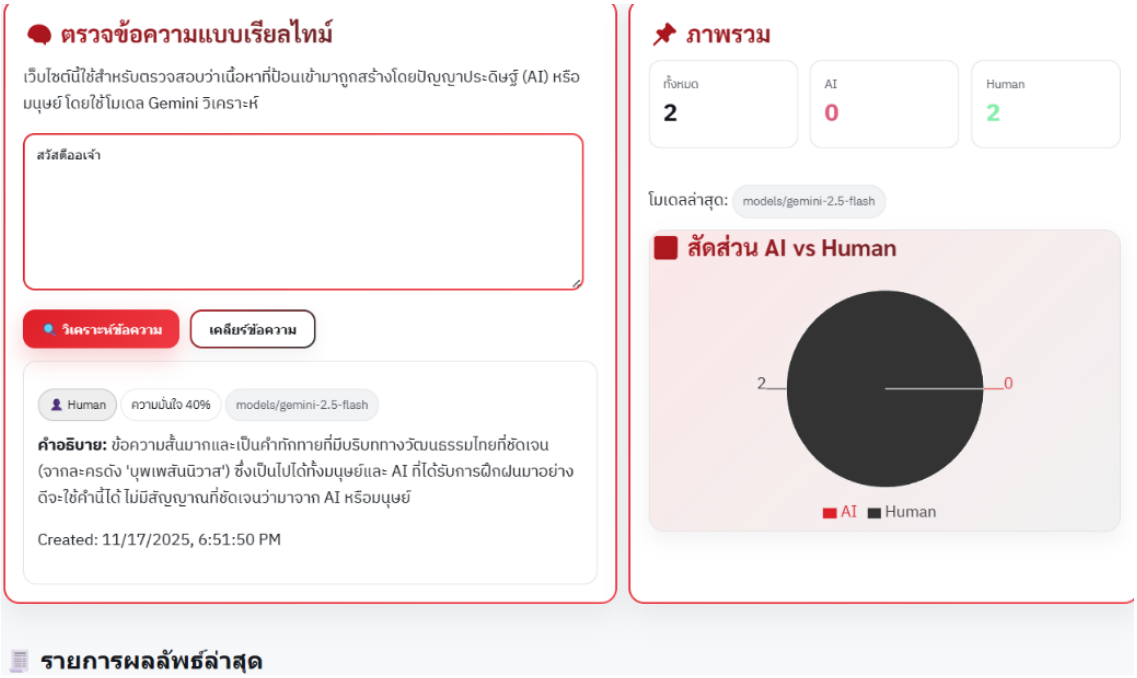
4.2.2 การจัดการข้อผิดพลาด (Error Handling): เมื่อรับข้อมูลที่ไม่ถูกต้อง เช่น ค่าว่างหรือข้อมูลไม่ใช่ประเภทข้อความ (String) ระบบ Data Cleansing สามารถจัดการได้อย่างเหมาะสม โดยไม่ส่งผลกระทบทำให้ Server หยุดทำงาน

4.2.3 **ประสิทธิภาพ (Performance):** การใช้ Amazon S3 เป็นที่เก็บข้อมูล Log แทนฐานข้อมูลแบบดั้งเดิม ช่วยให้ระบบมีประสิทธิภาพในการเขียนข้อมูลสูง รองรับไฟล์ขนาดเล็กจำนวนมากได้อย่างเหมาะสมกับลักษณะงาน

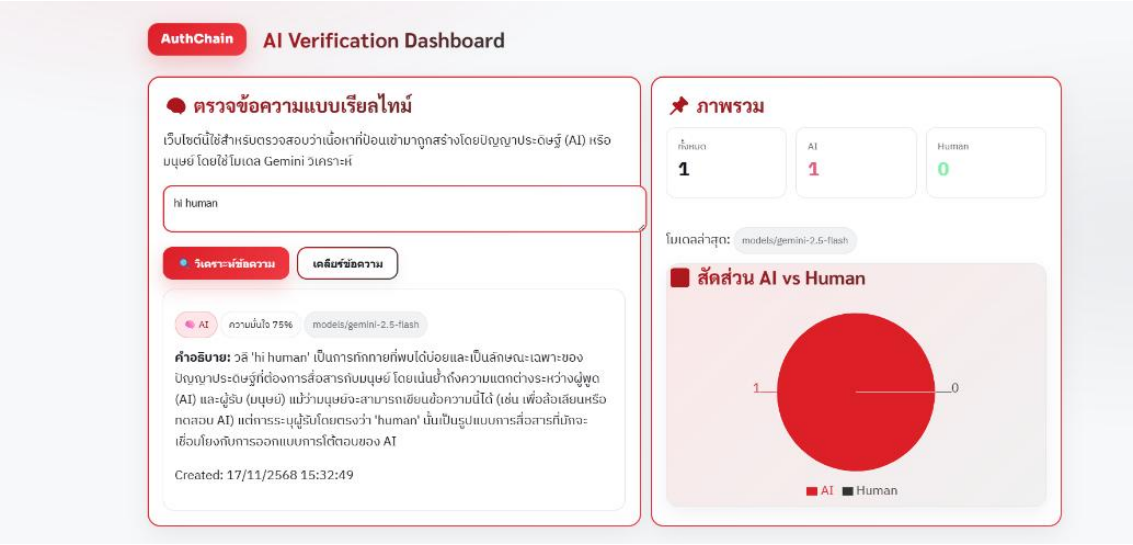
4.3 ตัวอย่างผลลัพธ์ของระบบ

4.3.1 ภาพหน้าจอส่วนติดต่อผู้ใช้ (Dashboard UI)

- หน้าจอ Dashboard แสดงกราฟสถิติและฟอร์มกรอกข้อมูล

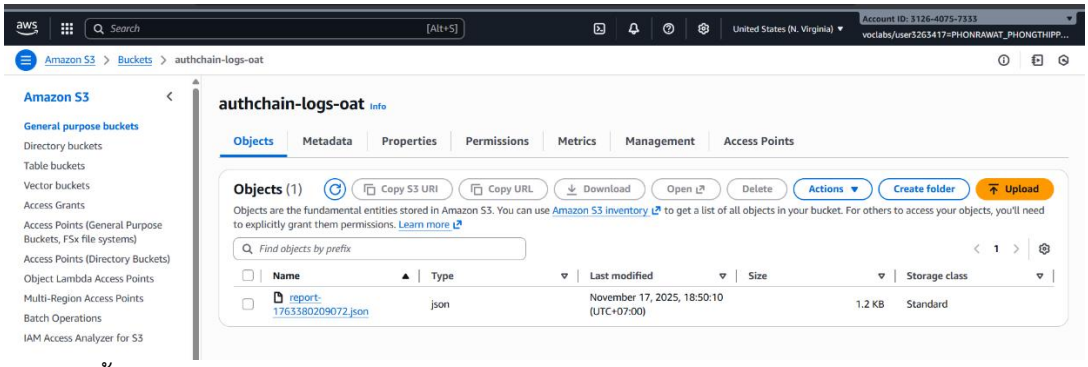


- หน้าจอแสดงผลลัพธ์เมื่อตรวจสอบพบว่าเป็น AI (Confidence Score สูง)

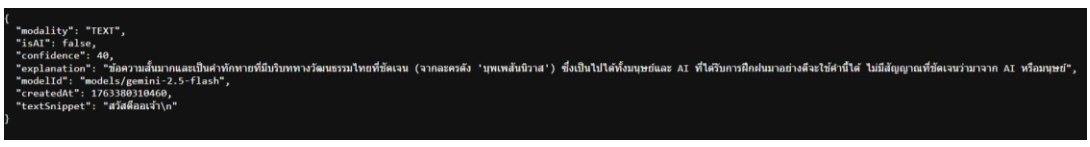


4.3.2 ภาพแสดงการทำงานเบื้องหลัง (Backend & Logs)

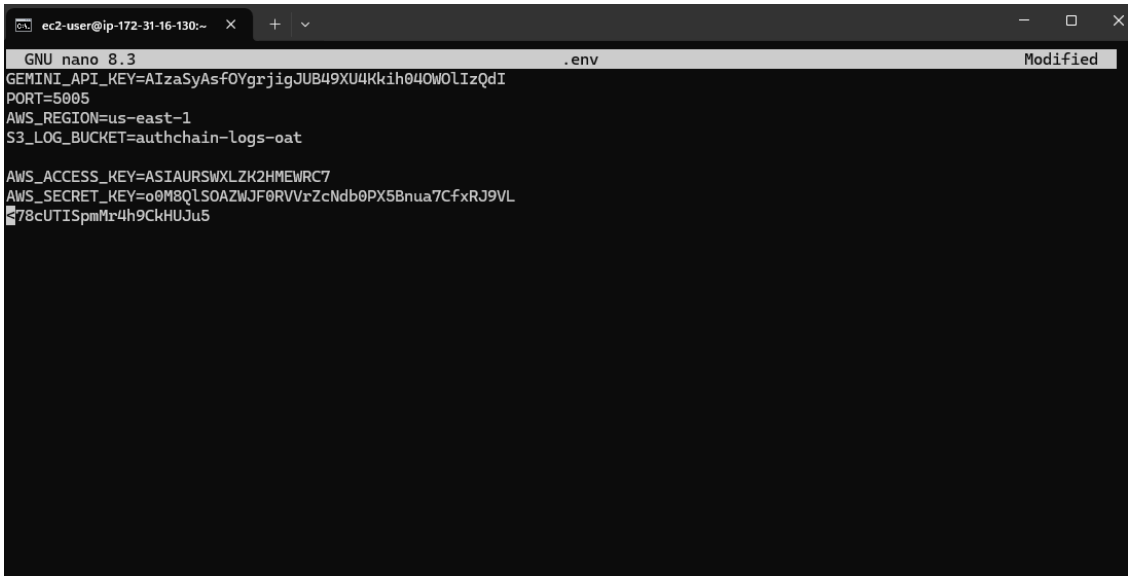
- หน้าจอ AWS Console (S3 Bucket) แสดงรายการไฟล์ JSON Log ที่ถูกบันทึกจริง



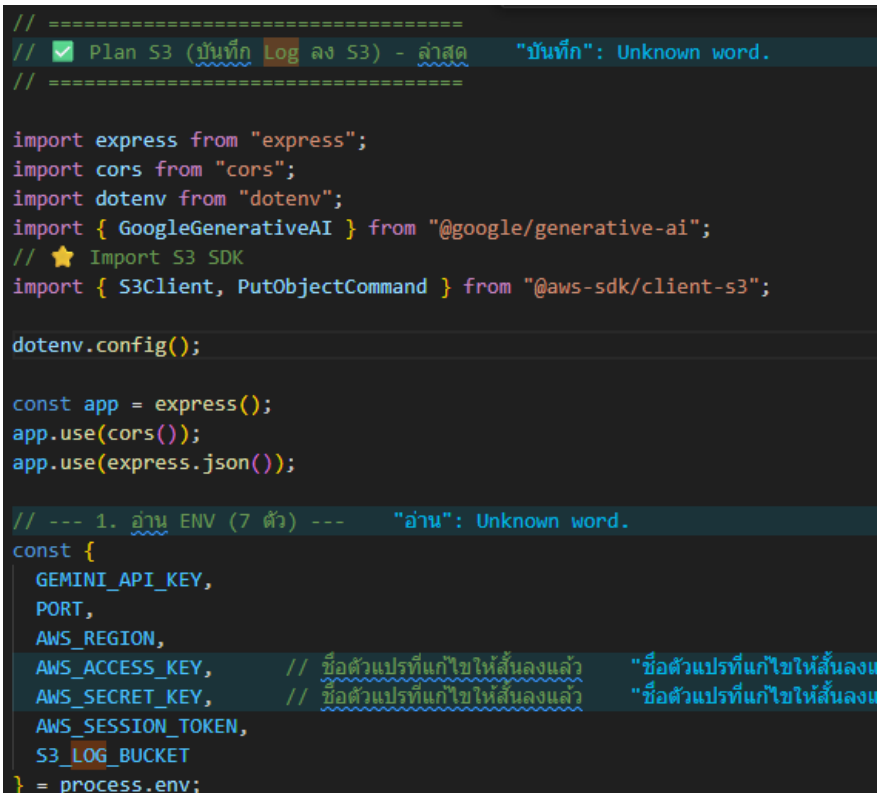
- ตัวอย่างเนื้อหาภายในไฟล์ JSON Log (แสดง Timestamp, Input text, Result)



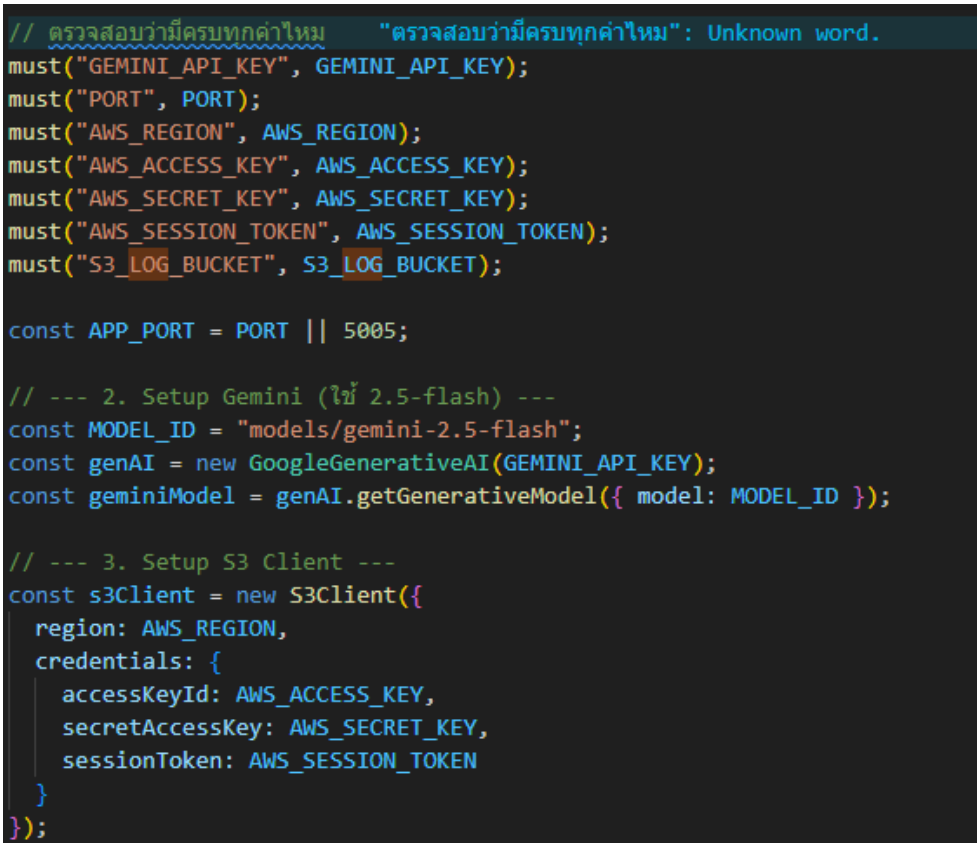
- ตัวอย่างไฟล์ .env ที่เอาไว้เก็บค่าลับของโปรเจคจากภายนอกต่างๆ เช่น Gemini , AWS



- ภาพตัวอย่างการดำเนินการ server.js
  - ตัวอย่างการ import library ที่จำเป็นและตรวจเช็คอ่านไฟล์ env ที่จำเป็น



- ตัวอย่างการตรวจเช็คค่า API แต่ละค่าในไฟล์ env รวมถึงการติดตั้ง



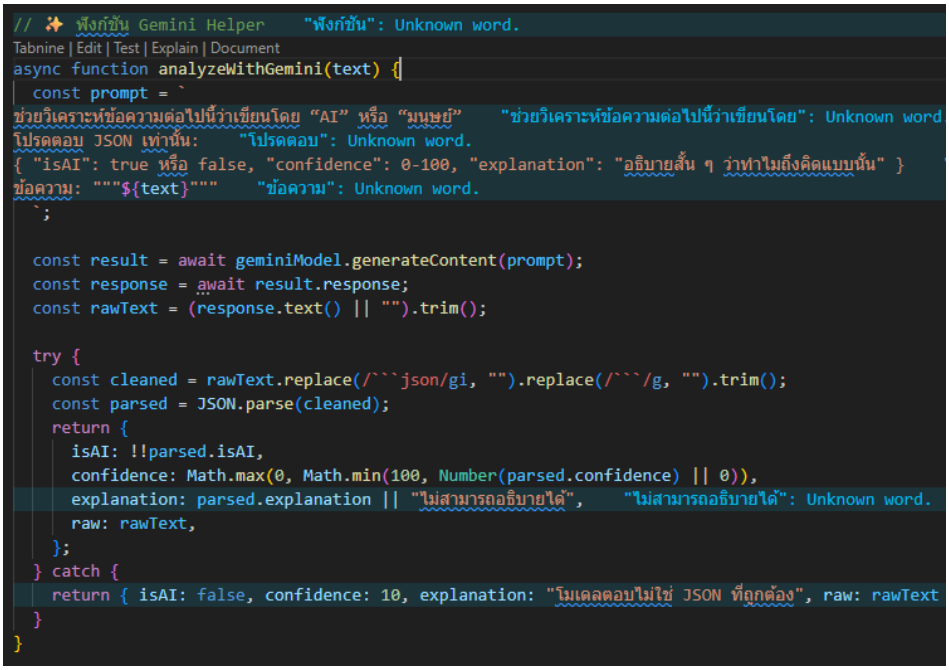
```
// ตรวจสอบว่ามีครบทุกค่าไหม     "ตรวจสอบว่ามีครบทุกค่าไหม": Unknown word.
must("GEMINI_API_KEY", GEMINI_API_KEY);
must("PORT", PORT);
must("AWS_REGION", AWS_REGION);
must("AWS_ACCESS_KEY", AWS_ACCESS_KEY);
must("AWS_SECRET_KEY", AWS_SECRET_KEY);
must("AWS_SESSION_TOKEN", AWS_SESSION_TOKEN);
must("S3_LOG_BUCKET", S3_LOG_BUCKET);

const APP_PORT = PORT || 5005;

// --- 2. Setup Gemini (ใช้ 2.5-flash) ---
const MODEL_ID = "models/gemini-2.5-flash";
const genAI = new GoogleGenerativeAI(GEMINI_API_KEY);
const geminiModel = genAI.getGenerativeModel({ model: MODEL_ID });

// --- 3. Setup S3 Client ---
const s3Client = new S3Client({
  region: AWS_REGION,
  credentials: {
    accessKeyId: AWS_ACCESS_KEY,
    secretAccessKey: AWS_SECRET_KEY,
    sessionToken: AWS_SESSION_TOKEN
  }
});
```

- ตัวอย่างการตรวจเช็คค่า API แต่ละค่าในไฟล์ env รวมถึงการติดตั้ง Gemini และ S3



```
// ✨ ฟังก์ชัน Gemini Helper     "ฟังก์ชัน": Unknown word.
Tabnine | Edit | Test | Explain | Document
async function analyzeWithGemini(text) {
  const prompt = `
ช่วยวิเคราะห์ข้อความต่อไปนี้ว่าเขียนโดย “AI” หรือ “มนุษย์”     "ช่วยวิเคราะห์ข้อความต่อไปนี้ว่าเขียนโดย": Unknown word.
โปรดตอบ JSON เท่านั้น:     "โปรดตอบ": Unknown word.
{ "isAI": true หรือ false, "confidence": 0-100, "explanation": "อธิบายสั้น ๆ ว่าทำไมถึงคิดแบบนั้น" }
ข้อความ: """${text}"""     "ข้อความ": Unknown word.
  `;

  const result = await geminiModel.generateContent(prompt);
  const response = await result.response;
  const rawText = (response.text() || "").trim();

  try {
    const cleaned = rawText.replace(/```json/gi, "").replace(/```/g, "").trim();
    const parsed = JSON.parse(cleaned);
    return {
      isAI: !!parsed.isAI,
      confidence: Math.max(0, Math.min(100, Number(parsed.confidence) || 0)),
      explanation: parsed.explanation || "ไม่สามารถอธิบายได้",     "ไม่สามารถอธิบายได้": Unknown word.
      raw: rawText,
    };
  } catch {
    return { isAI: false, confidence: 10, explanation: "โมเดลตอบไม่ใช่ JSON ที่ถูกต้อง", raw: rawText
  }
}
```

- ตัวอย่าง API Endpoint (Route) ที่กำหนดจุดรับคำขอแบบ POST

```
// --- 4. API Endpoint (POST) ---
Tabnine | Edit | Test | Explain | Document
app.post("/api/verify-text", async (req, res) => {
  try {
    const { text } = req.body ?? {};
    if (!text || typeof text !== "string") {
      return res.status(400).json({ error: "missing text" });
    }
```

- ตัวอย่างการ เรียกใช้งาน Gemini API และ สร้างโครงสร้างข้อมูล Report

```
  // 1. เรียก Gemini    "เรียก": Unknown word.
  const ai = await analyzeWithGemini(text);
  const timestamp = Date.now();

  const report = {
    modality: "TEXT",
    isAI: ai.isAI,
    confidence: ai.confidence,
    explanation: ai.explanation,
    modelId: MODEL_ID,
    createdAt: timestamp,
    textSnippet: text.slice(0, 200),
  };
```

- ตัวอย่างการบันทึก Log ไฟล์ลง Amazon S3

```
// 2. บันทึกลง S3 (เป็นไฟล์ JSON)    "บันทึกลง": Unkno
const logFileName = `report-${timestamp}.json`;

await s3Client.send(new PutObjectCommand({
  Bucket: S3_LOG_BUCKET,
  Key: logFileName,
  Body: JSON.stringify(report, null, 2),
  ContentType: "application/json"
}));
```

- ตัวอย่างการส่งผลลัพธ์กลับ (Response) ไปยังผู้ใช้งาน (Frontend) และการจัดการข้อผิดพลาด (Error Handling)

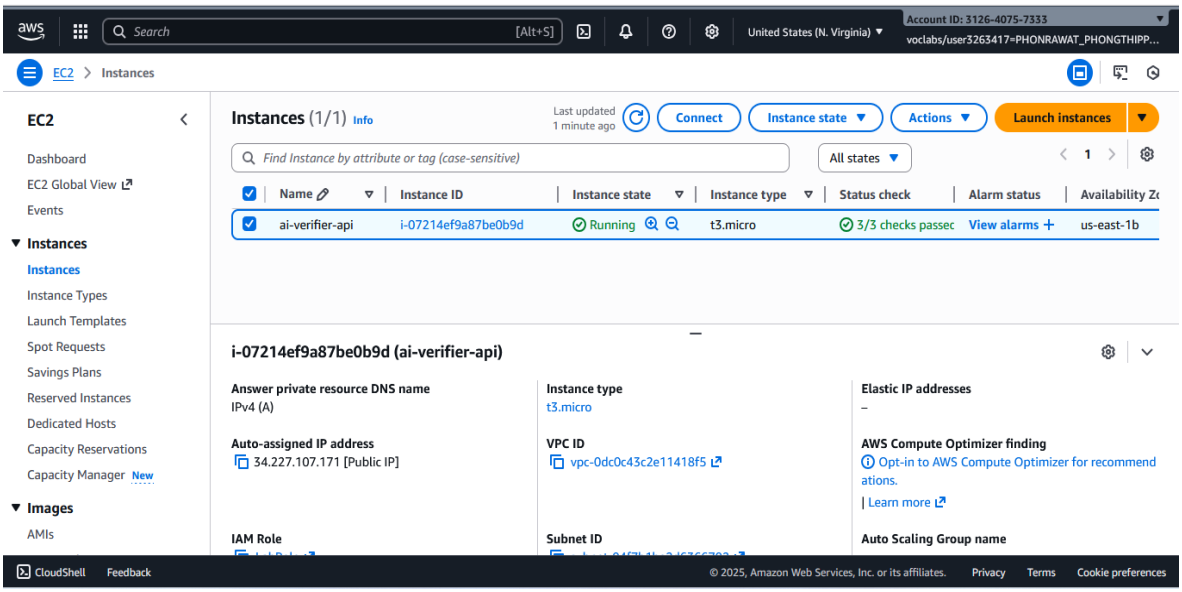
```
    // 3. ส่งผลลัพธ์กลับไปที่หน้าเว็บ     "ส่งผลลัพธ์กลับไปที่หน้าเว็บ": Unknown word.
    return res.json({
      success: true,
      result: report,
    });

  } catch (err) {
    console.error("verify-text error:", err);
    res.status(500).json({ error: err.message || "unknown error" });
  }
});
```

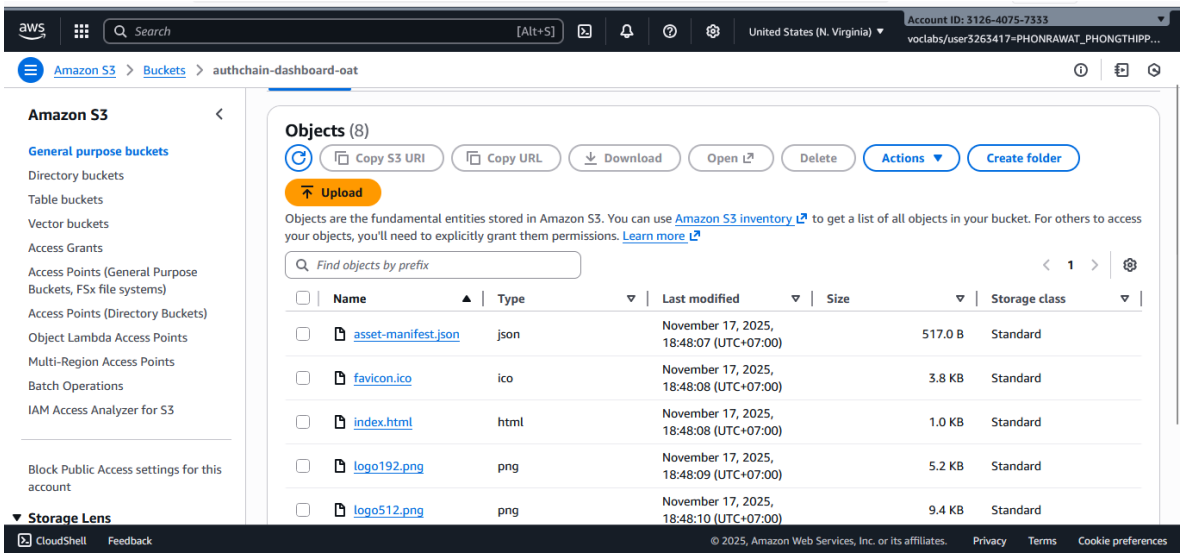
- ตัวอย่างการเริ่มการทำงานของ Server (Start Server)

```
// --- 5. Start Server ---
Tabnine | Edit | Test | Explain | Document
app.listen(APP_PORT, "0.0.0.0", () =>
  console.log(`✅ Gemini (S3 Log) API running on port ${APP_PORT}`)
);
```

- หน้าต่าง AWS EC2 ที่ทำหน้าที่เป็น Backend (Node.js/Express/PM2)



- หน้าต่าง AWS S3 สำหรับเก็บไฟล์ Frontend (S3 Static Website Hosting)



สรุปผลการทำโครงการ อภิปรายผล และข้อเสนอแนะ

5.1 สรุปผลการทำโครงการ

โครงการนี้ประสบความสำเร็จในการพัฒนาระบบตรวจสอบ AI (AI Verification System) ที่สามารถทำงานร่วมกับ AWS Cloud ได้ตามเป้าหมายที่กำหนด โดยระบบประกอบด้วย 3 ส่วนหลัก ได้แก่ (1) ส่วนหน้า (Frontend) ที่พัฒนาด้วย React เพื่อให้ผู้ใช้สามารถโต้ตอบได้สะดวก (2) ส่วนกลาง (Backend) ที่ใช้ Node.js ทำงานบน EC2 สำหรับประมวลผล และ (3) การจัดเก็บ Log อย่างถาวรด้วย Amazon S3 ส่งผลให้ระบบมีความเสถียร สามารถแยกแยะข้อความที่สร้างโดย AI ได้อย่างถูกต้อง พร้อมทั้งเก็บหลักฐานการตรวจสอบ (Audit Trail) อย่างปลอดภัยบนระบบ Cloud

5.2 อภิปรายผล (Well-Architected Framework Analysis)

จากกระบวนการพัฒนาและทดสอบระบบ สามารถสรุปผลการดำเนินงานตามหลัก Well-Architected Framework และข้อจำกัดสำคัญที่พบได้ดังนี้:

5.2.1 Operational Excellence & Reliability: การใช้ PM2 แทนการรัน Node.js แบบปกติ ช่วยเพิ่มความน่าเชื่อถือ (High Availability) ให้กับระบบ โดยสามารถกู้คืนสภาวะทำงานโดยอัตโนมัติหากเกิดข้อผิดพลาด (Crash) ซึ่งเหมาะสมกับการประมวลผลบน EC2 ที่ต้องการความต่อเนื่องตลอดเวลา

5.2.2 Security & Access Control: ด้วยข้อจำกัดของสภาพแวดล้อม AWS Lab ที่ไม่สามารถสร้าง IAM Role ได้โดยตรง ทีมผู้พัฒนาจึงเลือกใช้ Session Token และ Access Keys ผ่าน Environment Variables (.env) เป็นทางเลือกสำรอง แม้จะไม่ใช่วิธีที่ดีที่สุด แต่ยังคงรักษาความปลอดภัยของข้อมูล Credential ไม่ให้ปรากฏใน Source Code ได้

5.2.3 Performance Efficiency (Database Decision): การเลือกใช้ Amazon S3 เก็บ Log แทน DynamoDB เป็นการตัดสินใจเชิงเทคนิคที่สำคัญ เนื่องจากข้อจำกัดด้านสิทธิ์ (Permission Denied ใน Lab) แต่ผลลัพธ์ที่ได้กลับพบว่า S3 มีความเหมาะสมสูงมากสำหรับการเก็บข้อมูลดิบ (Raw Data) ที่เน้นการเขียน (Write-heavy) และต้องการความทนทานสูง ในต้นทุนที่ต่ำกว่าการเช่า Database Server

5.3 ตารางสรุป: ปัญหาและอุปสรรค และแนวทางแก้ไข

ปัญหาหรืออุปสรรคที่พบ (Technical Challenges)	วิธีการแก้ไข / แนวทางปรับปรุง (Solutions)
1. ข้อจำกัดด้าน IAM Role: ไม่สามารถสร้าง Role ให้ EC2 เข้าถึง S3 ได้โดยตรงตาม Best Practice	ใช้ AWS Session Token ชั่วคราวจาก Vocareum และกำหนดค่าผ่าน Environment Variables ในไฟล์ .env บน Server
2. การเชื่อมต่อ Database: ไม่สามารถใช้งาน DynamoDB ได้เนื่องจากติดสิทธิ์ Policy (Access Denied)	เปลี่ยนสถาปัตยกรรมไปใช้ Amazon S3 เป็น Log Storage แทน ซึ่งให้ผลลัพธ์เรื่อง Durability ที่ดีกว่าสำหรับงาน Audit
3. ความเสถียรของ API: Gemini API อาจตอบกลับมาในรูปแบบที่ไม่ใช่ JSON หรือเกิด Timeout	พัฒนาฟังก์ชัน Data Cleansing และ Try-Catch Block ใน server.js เพื่อตรวจสอบและแปลงข้อมูลให้ถูกต้องก่อนบันทึก
4. การ Deploy บน Cloud: Environment บน EC2 (Linux) ต่างจาก Localhost (Windows)	ใช้คำสั่ง Linux Command Line และติดตั้ง Dependencies ให้ครบถ้วน พร้อมใช้ PM2 ช่วยบริหารจัดการ Process

5.4 ข้อเสนอแนะและแนวทางพัฒนาในอนาคต

5.4.1 ย้ายระบบไปสู่ Serverless Architecture (AWS Lambda): ปัจจุบันระบบรันบน EC2 ซึ่งต้องเปิดเครื่องตลอดเวลา หากเปลี่ยนไปใช้ AWS Lambda จะช่วยลดต้นทุน (Cost Optimization) โดยจ่ายเงินตามจำนวนครั้งที่มีการเรียกใช้งานจริง และลดภาระการดูแลรักษา Server

5.4.2 เพิ่มระบบแจ้งเตือนอัตโนมัติ (Automated Alerting): ควรนำ AWS SNS (Simple Notification Service) มาประยุกต์ใช้ เพื่อแจ้งเตือนผู้ดูแลระบบผ่านอีเมลทันที หากระบบตรวจพบเนื้อหาที่มีความเสี่ยงสูง หรือเมื่อค่าความมั่นใจ (Confidence Score) ต่ำกว่าเกณฑ์ที่กำหนด

5.4.3 การจัดการวงจรชีวิตข้อมูล (S3 Lifecycle Policy): ในระยะยาว Log ไฟล์อาจจะมีจำนวนมหาศาล ควรตั้งค่า S3 Lifecycle Policy เพื่อย้าย Log เก่าไปยัง S3 Glacier (Archival Storage) โดยอัตโนมัติ เพื่อประหยัดพื้นที่และลดค่าใช้จ่ายในการจัดเก็บข้อมูล

5.4.4 พัฒนาระบบตรวจสอบไฟล์ (File Analysis): ต่อยอดระบบให้รองรับการอัปโหลดไฟล์เอกสาร (PDF, Docx) เพื่อตรวจสอบเนื้อหาทั้งฉบับ แทนการคัดลอกข้อความมาวางเพียงอย่างเดียว เพื่อเพิ่มความสะดวกให้กับผู้ใช้งานในภาคธุรกิจ

เอกสารอ้างอิง

โปรเจกต์ zip

https://drive.google.com/file/d/1QMzMCorJjFtRAtb0LixI10qYkqvLaeIR/view?usp=sharing

วิดีโอนำเสนอ

https://youtu.be/T7XcpvF3Nus?si=OO4RrwmbQBrtc-1e

GitHub

https://github.com/PhonrawatLimfaguang/ai-verification-aws-blockchain.git