AIB.

ครู่มือ Code JS

```
var channelToken =
"r8dp0F1CUXM2fwS/5sYfPB+vqQ++SUSArZsTuIzTchtdfb8ndKLlvTpDAV7iAAN7YFzqLn2mFD
wYtzCJrl+mfYbtOHQd5EnIDBFFHLs4lf2YgrHSO9OvJPTX4kWOghY24Ces0hO2HLTVnxtK153Aa
AdB04t89/1O/w1cDnyilFU="; //Token Line
var apiKey = "AIzaSyDNTF2xmCln4XKz911UwusydJJytU1DmAE"; //Key Gemini
```

โค้ดที่คุณให้มามีค่าตัวแปรสองตัวที่น่าสนใจ ซึ่งเกี่ยวข้องกับการใช้งาน API ต่าง ๆ ดังนี้:

1. channelToken:
    - ค่านี้น่าจะเป็น Token ที่ใช้สำหรับเชื่อมต่อกับบริการของ LINE (LINE Messaging API) โดยเฉพาะเพื่อส่งและรับข้อความผ่านแอปพลิเคชัน LINE
    - ในกรณีที่คุณใช้ channelToken นี้ คุณจะใช้ในการพิสูจน์ตัวตนสำหรับการทำงานกับ LINE Messaging API
2. apiKey:
    - ตัวแปรนี้เก็บค่า API Key ที่เป็นของ Google API ซึ่งในที่นี้คือ Gemini API ที่อาจจะเป็นบริการของ Google Cloud หรือบริการอื่น ๆ ที่เกี่ยวข้อง
    - API Key นี้ใช้เพื่อยืนยันและระบุแอปพลิเคชันหรือผู้ใช้ที่ร้องขอการเชื่อมต่อกับ API ของ Google

การใช้ข้อมูลแบบนี้ต้องระมัดระวังเรื่องความปลอดภัย ไม่ควรเผยแพร่ค่า Token หรือ Key ของ API สู่สาธารณะ เพราะอาจทำให้เกิดปัญหาในเรื่องของการเข้าถึงระบบจากบุคคลที่ไม่พึงประสงค์ได้

AIB.

```
function replyMsg(replyToken, mess, channelToken) {
  var url = 'https://api.line.me/v2/bot/message/reply';
  var opt = {
    'headers': {
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': 'Bearer ' + channelToken,
    },
    'method': 'post',
    'payload': JSON.stringify({
      'replyToken': replyToken,
      'messages': mess
    })
  };
  UrlFetchApp.fetch(url, opt);
}
```

โค้ดฟังก์ชัน replyMsg ที่คุณให้มาใช้สำหรับการส่งข้อความตอบกลับผ่าน LINE Messaging API ซึ่งเป็นการตอบกลับผู้ใช้เมื่อมีการโต้ตอบกับบอทบนแอป LINE โดยใช้ฟังก์ชัน UrlFetchApp.fetch ของ Google Apps Script เพื่อส่งคำขอ HTTP แบบ POST ไปยังเซิร์ฟเวอร์ LINE API ตาม URL ที่กำหนด

การอธิบายฟังก์ชัน:

1. พารามิเตอร์ของฟังก์ชัน:
    - replyToken: เป็นค่าโทเค็นที่ได้รับเมื่อมีข้อความส่งเข้ามาในบอท LINE และใช้ในการระบุข้อความที่ต้องการตอบกลับ
    - mess: เป็นอาร์เรย์ของข้อความที่ต้องการส่งกลับ ซึ่งต้องเป็นไปตามโครงสร้างที่ LINE Messaging API รองรับ
    - channelToken: เป็น Channel Access Token ที่ใช้เพื่อยืนยันตัวตนบอท LINE กับเซิร์ฟเวอร์ LINE API

AIB.

2. URL:
    - url: คือ URL ของ API สำหรับการตอบกลับข้อความใน LINE ซึ่งในที่นี้คือ https://api.line.me/v2/bot/message/reply
3. opt (Options):
    - headers: หัวข้อของการร้องขอ HTTP ที่ระบุว่าเนื้อหาของคำขอนี้เป็น JSON (Content-Type: application/json) และใช้ Authorization แบบ Bearer เพื่อส่ง channelToken สำหรับการยืนยันตัวตน
    - method: เป็นการกำหนดวิธีการ HTTP ที่จะใช้ ซึ่งในที่นี้เป็น POST
    - payload: คือข้อมูลที่ต้องการส่งในคำขอ HTTP ซึ่งในที่นี้จะเป็น JSON ของข้อความที่ต้องการตอบกลับ (replyToken และ messages)

4. การส่งคำขอ:
    - UrlFetchApp.fetch(url, opt): เป็นคำสั่งของ Google Apps Script ที่ใช้เพื่อส่งคำขอ HTTP ไปยัง URL ที่กำหนดไว้ โดยใช้ opt เป็นตัวกำหนดค่าในการร้องขอ

ตัวอย่างการทำงาน: เมื่อผู้ใช้ส่งข้อความเข้ามาใน LINE บอทจะได้รับ replyToken จากนั้นบอทจะใช้ฟังก์ชัน replyMsg ในการส่งข้อความตอบกลับโดยส่ง replyToken และข้อความ (mess) กลับไปยังผู้ใช้ผ่าน LINE API

AIB.

```
function doPost(e) {
        var value = JSON.parse(e.postData.contents);
        var events = value.events;
        var event = events[0];
        var type = event.type;
        var replyToken = event.replyToken;
        switch (type) {
          case 'follow':
            replyMsg(replyToken, mess, channelToken);
            break;
          case 'message':
            var messageType = event.message.type;
            if(messageType == "text"){
              gemini_pro(event);
            }
            else if(messageType == "image"){
              gemini_provision(event);
            }
            else{
            var mess = [{"type":"text","text":"Hello world"}];
            replyMsg(replyToken, mess, channelToken);
            }
            break;
          default:
            break;
        }
}
```

AIB.

โค้ดฟังก์ชัน doPost(e) นี้เป็นฟังก์ชันที่ใช้ในการประมวลผลข้อมูลที่ได้รับจากการ POST เข้ามาจาก LINE API เมื่อผู้ใช้โต้ตอบกับบอท (ผ่านข้อความ หรือการกระทำอื่นๆ) โดยจะทำการส่งข้อความหรือดำเนินการตามประเภทของเหตุการณ์ที่เกิดขึ้น เช่น เมื่อผู้ใช้ติดตามบอทหรือส่งข้อความถึงบอท

การทำงานของโค้ด:

1. การรับข้อมูลจากการ POST:
    - e.postData.contents: ใช้ในการดึงข้อมูลที่ถูกส่งมาผ่าน HTTP POST (ซึ่งเป็น JSON) แล้วทำการแปลงข้อมูลด้วย JSON.parse เพื่อให้สามารถใช้งานได้ในรูปแบบของออบเจ็กต์ JavaScript
    - var value = JSON.parse(e.postData.contents);: ข้อมูลนี้จะเก็บค่าทั้งหมดที่ถูกส่งมาโดย LINE API
2. การแยกเหตุการณ์ที่เกิดขึ้น:
    - var events = value.events;: ดึงค่าเหตุการณ์ทั้งหมดที่ถูกส่งมาจาก LINE API (มักจะอยู่ในรูปของอาร์เรย์)
    - var event = events[0];: ดึงเหตุการณ์แรกในอาร์เรย์ (ในกรณีที่มีหลายเหตุการณ์)
    - var type = event.type;: ดึงประเภทของเหตุการณ์ (เช่น message หรือ follow)
3. การแยกประเภทเหตุการณ์:
    - ใช้ switch เพื่อแยกประเภทของเหตุการณ์ที่เกิดขึ้น โดยดูจากค่าของ type

กรณีของ type:

- follow: เป็นเหตุการณ์เมื่อผู้ใช้กดติดตามบอท ในกรณีนี้บอทจะส่งข้อความตอบกลับด้วยฟังก์ชัน replyMsg(replyToken, mess, channelToken)
- message: เป็นเหตุการณ์เมื่อผู้ใช้ส่งข้อความถึงบอท
    1. var messageType = event.message.type;: ตรวจสอบประเภทของข้อความที่ส่งมา (เช่น ข้อความธรรมดาหรือรูปภาพ)
    2. ข้อความแบบ text: ถ้าเป็นข้อความแบบ text ฟังก์ชัน gemini_pro(event) จะถูกเรียกใช้งาน (แต่ฟังก์ชันนี้ไม่ได้แสดงในโค้ดที่คุณให้มา)
    3. ข้อความแบบ image: ถ้าเป็นรูปภาพ ฟังก์ชัน gemini_provision(event) จะถูกเรียกใช้งาน (ซึ่งฟังก์ชันนี้ก็ไม่ได้แสดงเช่นกัน)
    4. ข้อความแบบอื่น: ถ้าเป็นประเภทอื่นๆ บอทจะตอบกลับด้วยข้อความ "Hello world"

AIB.

4. ค่าที่ใช้ในฟังก์ชัน:
    - replyToken: โทเค็นที่ได้จาก LINE API สำหรับใช้ในการตอบกลับข้อความของผู้ใช้
    - mess: อาร์เรย์ข้อความที่ต้องการตอบกลับในรูปแบบ JSON ซึ่งสามารถเป็นข้อความ, รูปภาพ, หรือข้อมูลอื่นที่ API รองรับ
    - channelToken: เป็นค่าที่ใช้สำหรับการยืนยันตัวตนของบอท LINE ในการสื่อสารกับ LINE API

สรุปการทำงาน:
ฟังก์ชันนี้จะถูกเรียกใช้ทุกครั้งที่มีเหตุการณ์ POST มาจาก LINE เมื่อผู้ใช้โต้ตอบกับบอท โดยฟังก์ชันจะตรวจสอบประเภทของเหตุการณ์และข้อความที่ส่งมา และทำการตอบกลับผู้ใช้ตามประเภทของเหตุการณ์นั้นๆ (เช่น ติดตามบอท, ส่งข้อความ, ส่งรูปภาพ)

AIB.

```
function gemini_pro(event) {
var userMessage = event.message.text;
var replyToken = event.replyToken;
var apiUrl = "https://generativelanguage.googleapis.com/v1beta/models/gemini-
pro:generateContent?key=" + apiKey;
var requestBody = {
    "contents": [
      {
        "role": "user",
        "parts": [
          {
            "text": userMessage
          }
        ]
      }
    ],
    "generationConfig": {
      "temperature": 0.9,
      "topK": 1,
      "topP": 1,
      "maxOutputTokens": 2048,
      "stopSequences": []
    }
  };


   var options = {
    "method": "post",
    "contentType": "application/json",
    "payload": JSON.stringify(requestBody)
  };
```

AIB.

ฟังก์ชัน gemini_pro(event) ที่คุณให้มานี้มีการเรียกใช้ API ของ Google's Gemini Language Model เพื่อสร้างเนื้อหาตามข้อความที่ผู้ใช้ส่งมา โดยผ่านทาง Google Generative Language API นี่คือคำอธิบายโค้ดทีละส่วน

<u>การทำงานของโค้ด</u>

1. รับข้อความจากผู้ใช้

```javascript
var userMessage = event.message.text;
var replyToken = event.replyToken;
```

- userMessage: ข้อความที่ผู้ใช้ส่งมาจะถูกดึงออกมาจากวัตถุ event ซึ่งเป็นพารามิเตอร์ที่ส่งเข้ามาในฟังก์ชัน
- replyToken: เป็นโทเค็นที่ใช้ในการตอบกลับผู้ใช้ในภายหลัง ซึ่งดึงมาจากวัตถุ event

2. URL ของ API

```javascript
var apiUrl = "https://generativelanguage.googleapis.com/v1beta/models/gem
```

- apiUrl: URL ที่ใช้เรียก Google Generative Language API โดยในที่นี้ใช้โมเดล gemini-pro เพื่อสร้างเนื้อหา (generateContent)
- apiKey: เป็นคีย์ API ที่ใช้เพื่อยืนยันตัวตนสำหรับเข้าถึงบริการของ Google API ซึ่งต้องแนบไปใน URL ด้วย

3. สร้างคำขอ (Request Body)

```javascript
var requestBody = {
 "contents": [
   {
     "role": "user",
     "parts": [
       {
         "text": userMessage
       }
     ]
   }
 ],
 "generationConfig": {
   "temperature": 0.9,
   "topK": 1,
   "topP": 1,
   "maxOutputTokens": 2048,
   "stopSequences": []
 }
```

AIB.

```markdown
- **`contents`**: โครงสร้างข้อมูลที่ส่งไปยัง Gemini API มีองค์ประกอบดังนี้:
  - `"role": "user"`: ระบุว่าข้อความนี้มาจากผู้ใช้
  - `"parts"`: เป็นข้อมูลส่วนที่ส่งข้อความของผู้ใช้ (`userMessage`) เพื่อให้โมเดล Gemini
- **`generationConfig`**: ตั้งค่าการสร้างเนื้อหา เช่น:
  - `temperature`: ค่าความแปรปรวน (ยิ่งสูงยิ่งสร้างผลลัพธ์ที่มีความหลากหลาย)
  - `topK` และ `topP`: ใช้ควบคุมการสุ่มข้อความที่สร้างขึ้น
  - `maxOutputTokens`: จำนวนสูงสุดของโทเค็นที่โมเดลจะสร้าง (2048 โทเค็นในกรณีนี้)
  - `stopSequences`: ไม่ได้กำหนด ซึ่งหมายความว่าไม่มีลำดับคำสั่งที่หยุดการสร้างเนื้อหา

4. **ตัวเลือกของการส่งคำขอ (Options)**:
```javascript
var options = {
 "method": "post",
 "contentType": "application/json",
 "payload": JSON.stringify(requestBody)
};
```

- method: กำหนดวิธีการร้องขอเป็น POST
- contentType: ระบุว่าเนื้อหาที่ส่งไปเป็น JSON (application/json)
- payload: แปลง requestBody ให้เป็นสตริง JSON ก่อนส่งไปยัง API

<u>การทำงานโดยรวม</u>

ฟังก์ชันนี้ดึงข้อความที่ผู้ใช้ส่งเข้ามา (userMessage) และใช้ API ของ Google Gemini ในการสร้างข้อความตอบกลับ โดยใช้โมเดลภาษาขั้นสูง

1. การตั้งค่าการสร้างเนื้อหาผ่าน generationConfig ช่วยกำหนดวิธีที่โมเดลจะสร้างผลลัพธ์
2. เมื่อสร้างคำขอเสร็จแล้ว ระบบจะส่งคำขอ POST ไปยัง URL ของ Google API (apiUrl) โดยใช้ options ที่กำหนด
```

AIB.

```
// เลือกเฉพาะข้อมูลที่คุณต้องการ
  var response = UrlFetchApp.fetch(apiUrl, options);
  var responseData = JSON.parse(response.getContentText());
  var textResult = responseData.candidates[0].content.parts[0].text;
  var mess = [{"type":"text","text": textResult.toString()}];
  replyMsg(replyToken, mess, channelToken);
}
```

ส่วนนี้ของโค้ดต่อจากฟังก์ชัน gemini_pro(event) คือการส่งคำขอไปยัง Google Gemini API และจัดการผลลัพธ์ที่ได้เพื่อตอบกลับผู้ใช้ผ่าน LINE Messaging API โดยทำงานดังนี้:

<u>การทำงานของโค้ด:</u>

1. ส่งคำขอไปยัง API:

```javascript
var response = UrlFetchApp.fetch(apiUrl, options);
```

- ใช้ฟังก์ชัน UrlFetchApp.fetch ใน Google Apps Script เพื่อส่งคำขอ POST ไปยัง apiUrl (ซึ่งในที่นี้คือ Google Gemini API) โดยใช้ตัวเลือก options ที่กำหนดไว้ก่อนหน้านี้
- response จะเก็บผลลัพธ์ที่ได้จากคำขอนี้ ซึ่งเป็นข้อมูลจาก API

2. แปลงผลลัพธ์จาก API:

```javascript
var responseData = JSON.parse(response.getContentText());
```

- ผลลัพธ์ที่ได้จาก API จะมาในรูปแบบของข้อความ JSON ซึ่งถูกแปลงเป็นออบเจ็กต์ JavaScript ด้วย JSON.parse เพื่อให้ง่ายต่อการเข้าถึงข้อมูล

3. ดึงข้อความที่สร้างจากโมเดล Gemini:

```javascript
var textResult = responseData.candidates[0].content.parts[0].text;
```

- responseData.candidates[0]: ในโครงสร้างผลลัพธ์ของ Google Gemini API, candidates เป็นรายการของผลลัพธ์ที่โมเดลสร้างขึ้น

AIB.

- content.parts[0].text: เป็นส่วนของข้อความที่โมเดลสร้างขึ้นจากอินพุตที่ผู้ใช้ส่งมา (userMessage)

4. เตรียมข้อความสำหรับตอบกลับ:

```javascript
var mess = [{"type":"text","text": textResult.toString()}];
```

- mess เป็นอาร์เรย์ที่ประกอบด้วยวัตถุ JSON ซึ่งบอท LINE จะใช้เพื่อส่งข้อความตอบกลับ
- ข้อความนี้อยู่ในรูปแบบที่กำหนดโดย LINE API ({"type": "text", "text": <ข้อความ>}) โดยที่ textResult เป็นข้อความที่โมเดล Gemini สร้างขึ้นมา

5. ส่งข้อความตอบกลับผู้ใช้:

```javascript
replyMsg(replyToken, mess, channelToken);
```

- เรียกใช้ฟังก์ชัน replyMsg ที่เคยอธิบายไว้ก่อนหน้านี้ โดยใช้ replyToken เพื่อระบุว่าข้อความนี้เป็นการตอบกลับไปยังข้อความที่ผู้ใช้ส่งเข้ามา
- mess คือข้อความที่เตรียมไว้เพื่อส่งกลับไป และ channelToken คือโทเค็นที่ใช้ในการยืนยันตัวตนกับ LINE API

AIB.

```
function gemini_provision(event) {
  var messageId = event.message.id;
  var replyToken = event.replyToken;
  var url = "https://api-data.line.me/v2/bot/message/"+messageId+"/content";
  var headers = {
    "headers":{"Authorization": "Bearer "+channelToken}
  };
  var getcontent = UrlFetchApp.fetch(url, headers);
  var imageBlob = getcontent.getBlob();
  var encodedImage = Utilities.base64Encode(imageBlob.getBytes());
  var apiUrl = 'https://generativelanguage.googleapis.com/v1beta/models/gemini-pro-
vision:generateContent?key=' + apiKey;
  var payload = {
    "contents": [
      {
        "parts": [
          {
            "text": "ช่วยบรรยายภาพนี้ให้หน่อย"
          },
          {
            "inlineData": {
              "mimeType": "image/jpeg",
              "data": encodedImage
            }
          }
        ]
      }
    ],
    "generationConfig": {
      "temperature": 0.4,                                    // มี Code ต่อหน้าถัดไป //
```

```
    "topK": 32,
    "topP": 1,
    "maxOutputTokens": 4096,
    "stopSequences": []
  },
  "safetySettings": [
    {
      "category": "HARM_CATEGORY_HARASSMENT",
      "threshold": "BLOCK_MEDIUM_AND_ABOVE"
    },
    {
      "category": "HARM_CATEGORY_HATE_SPEECH",
      "threshold": "BLOCK_MEDIUM_AND_ABOVE"
    },
    {
      "category": "HARM_CATEGORY_SEXUALLY_EXPLICIT",
      "threshold": "BLOCK_MEDIUM_AND_ABOVE"
    },
    {
      "category": "HARM_CATEGORY_DANGEROUS_CONTENT",
      "threshold": "BLOCK_MEDIUM_AND_ABOVE"
    }
  ]
};
var options = {
  'method': 'post',
  'contentType': 'application/json',
  'payload': JSON.stringify(payload)
};
```

AIB.

ฟังก์ชัน gemini_provision(event) ที่คุณให้มา เป็นการใช้ API ของ LINE และ Google Gemini Vision Model เพื่อวิเคราะห์และบรรยายภาพที่ผู้ใช้ส่งมาในแชท LINE บอท โดยผ่านทาง Google Generative Language API (Vision Model) สำหรับภาพ

การทำงานของโค้ด:

1. รับข้อมูลเกี่ยวกับภาพจาก LINE API:

```javascript
var messageId = event.message.id;
var replyToken = event.replyToken;
```

- messageId: ดึง ID ของภาพที่ผู้ใช้ส่งมา ซึ่งจะใช้ในการดาวน์โหลดภาพจาก LINE API
- replyToken: ใช้ในการตอบกลับผู้ใช้ภายหลังเมื่อสร้างคำบรรยายภาพเรียบร้อยแล้ว

2. สร้าง URL เพื่อดึงภาพจาก LINE API:

```javascript
var url = "https://api-data.line.me/v2/bot/message/"+messageId+"/content";
var headers = {
  "headers":{"Authorization": "Bearer "+channelToken}
};
```

- url: เป็น URL ของ LINE API ที่ใช้ดึงเนื้อหาของข้อความที่มีรูปภาพโดยใช้ messageId
- headers: กำหนดหัวข้อที่ใช้ในการพิสูจน์ตัวตนกับ LINE API ผ่าน Bearer token (channelToken)

3. ดึงภาพจาก LINE API:

```javascript
var getcontent = UrlFetchApp.fetch(url, headers);
var imageBlob = getcontent.getBlob();
```

- UrlFetchApp.fetch(url, headers): ส่งคำขอ GET เพื่อดึงภาพจาก LINE API
- getcontent.getBlob(): แปลงข้อมูลที่ดึงมาเป็น Blob (Binary Large Object) เพื่อเก็บข้อมูลของภาพในรูปแบบที่ใช้งานได้ในภายหลัง

AIB.

4. แปลงภาพเป็น Base64:

```javascript
var encodedImage = Utilities.base64Encode(imageBlob.getBytes());
```

- ใช้ฟังก์ชัน Utilities.base64Encode เพื่อแปลงข้อมูลภาพ (Blob) เป็นสตริง Base64 ซึ่งเป็นรูปแบบที่ API ของ Google สามารถรองรับได้

5. เรียกใช้ Google Gemini Vision Model เพื่อบรรยายภาพ:

```javascript
var apiUrl = 'https://generativelanguage.googleapis.com/v1beta/models/gem
var payload = {
  "contents": [
    {
      "parts": [
        {
          "text": "ช่วยบรรยายภาพนี้ให้หน่อย"
        },
        {
          "inlineData": {
            "mimeType": "image/jpeg",
            "data": encodedImage
          }
        }
      ]
    }
  ],
  "generationConfig": {
    "temperature": 0.4,
    "topK": 32,
    "topP": 1,
    "maxOutputTokens": 4096,
    "stopSequences": []
  },
  "safetySettings": [
    {
      "category": "HARM_CATEGORY_HARASSMENT",
      "threshold": "BLOCK_MEDIUM_AND_ABOVE"
    },
    {
      "category": "HARM_CATEGORY_HATE_SPEECH",
      "threshold": "BLOCK_MEDIUM_AND_ABOVE"
    },
    {
      "category": "HARM_CATEGORY_SEXUALLY_EXPLICIT",
      "threshold": "BLOCK_MEDIUM_AND_ABOVE"
    },
    {
      "category": "HARM_CATEGORY_DANGEROUS_CONTENT",
      "threshold": "BLOCK_MEDIUM_AND_ABOVE"
```

AIB.

- apiUrl: URL ที่ใช้เรียกใช้ Google Gemini Vision Model เพื่อสร้างคำบรรยายภาพ
- payload: ค่าข้อมูลที่ส่งไปยัง API ประกอบด้วย:
    - "contents": โครงสร้างของข้อมูลที่ส่งไปยัง API ประกอบด้วย:
        - "text": ข้อความที่ขอให้ API บรรยายภาพ
        - "inlineData": ข้อมูลภาพในรูปแบบ Base64 ที่ถูกแนบมาด้วย
    - "generationConfig": กำหนดค่าการสร้างเนื้อหา เช่น temperature, topK, และ maxOutputTokens
    - "safetySettings": กำหนดการตั้งค่าความปลอดภัยเพื่อตรวจสอบเนื้อหาที่มีโอกาสจะมีความเสี่ยง เช่น คำหยาบ, การคุกคาม, หรือเนื้อหาที่ไม่เหมาะสม

6. ตั้งค่าคำขอ HTTP:

```javascript
var options = {
  'method': 'post',
  'contentType': 'application/json',
  'payload': JSON.stringify(payload)
};
```

    - options: กำหนดการตั้งค่าการส่งคำขอ HTTP โดยใช้วิธีการ POST และระบุประเภทเนื้อหา (contentType) เป็น application/json พร้อมกับแปลง payload เป็นสตริง JSON

<u>สรุปการทำงาน:</u>

1. ฟังก์ชันนี้ดึงภาพที่ผู้ใช้ส่งจาก LINE API
2. แปลงภาพเป็น Base64 เพื่อนำไปใช้กับ Google Gemini Vision API
3. เรียกใช้ Google API เพื่อขอให้โมเดลบรรยายภาพ โดยส่งภาพและข้อความที่กำหนดไปพร้อมกับการตั้งค่าความปลอดภัย
4. ผลลัพธ์ที่ได้จากโมเดลจะสามารถนำไปตอบกลับผู้ใช้ใน LINE ได้ในภายหลัง

ฟังก์ชันนี้ช่วยให้บอทสามารถรับรูปภาพจากผู้ใช้ วิเคราะห์ และตอบกลับด้วยคำบรรยายภาพที่สร้างโดยโมเดล AI

AIB.

```
  var response = UrlFetchApp.fetch(apiUrl, options);
  var responseData = JSON.parse(response.getContentText());
  var textResult = responseData.candidates[0].content.parts[0].text;
  var mess = [{"type":"text","text": textResult.toString()}];
  replyMsg(replyToken, mess, channelToken);
}
```

ในส่วนสุดท้ายของฟังก์ชัน gemini_provision(event) นี้เป็นการส่งคำขอไปยัง Google Gemini Vision Model API เพื่อสร้างคำบรรยายภาพที่ได้รับจากผู้ใช้ และจากนั้นนำผลลัพธ์ที่ได้มาส่งกลับไปยังผู้ใช้ผ่าน LINE Messaging API โดยทำงานดังนี้:

การทำงานของโค้ด:

1. ส่งคำขอไปยัง Google Gemini Vision Model:

```javascript
var response = UrlFetchApp.fetch(apiUrl, options);
```

- ใช้ฟังก์ชัน UrlFetchApp.fetch เพื่อส่งคำขอ HTTP POST ไปยัง apiUrl ซึ่งในที่นี้เป็น Google Gemini API โดยใช้ตัวเลือก (options) ที่กำหนดไว้ก่อนหน้า
- response: เก็บผลลัพธ์ที่ได้จาก API หลังจากส่งคำขอไปแล้ว

2. แปลงข้อมูลที่ได้รับจาก API:

```javascript
var responseData = JSON.parse(response.getContentText());
```

- response.getContentText(): แปลงผลลัพธ์ที่ได้รับจาก API ให้อยู่ในรูปแบบข้อความ (JSON string)
- JSON.parse(): แปลงข้อความ JSON ที่ได้มาเป็นออบเจ็กต์ JavaScript เพื่อให้สามารถเข้าถึงข้อมูลได้ง่ายขึ้น

AIB.

3. ดึงคำบรรยายภาพที่โมเดลสร้างขึ้น:

```javascript
var textResult = responseData.candidates[0].content.parts[0].text;
```

- responseData.candidates[0]: ข้อมูลผลลัพธ์จาก Gemini API จะมาในรูปแบบของ candidates ซึ่งเป็นรายการของข้อความที่โมเดลสร้างขึ้น
- content.parts[0].text: ในแต่ละคำตอบที่โมเดลสร้างขึ้น จะมีส่วนของข้อความ (text) ที่เป็นคำบรรยายภาพ ซึ่งดึงออกมาเพื่อใช้งาน

4. เตรียมข้อความตอบกลับ:

```javascript
var mess = [{"type":"text","text": textResult.toString()}];
```

- mess: เป็นอาร์เรย์ของ JSON ที่มีโครงสร้างตามที่ LINE Messaging API กำหนด เพื่อส่งข้อความตอบกลับผู้ใช้
- ในกรณีนี้เป็นข้อความแบบธรรมดา (type: "text") โดย textResult ที่ได้จากโมเดล Gemini จะถูกแปลงเป็นสตริงก่อนส่ง

5. ส่งข้อความตอบกลับไปยังผู้ใช้:

```javascript
replyMsg(replyToken, mess, channelToken);
```

   - ฟังก์ชัน replyMsg ถูกเรียกใช้เพื่อตอบกลับผู้ใช้ โดยใช้ replyToken เพื่อระบุว่าข้อความนี้เป็นการตอบกลับข้อความที่ผู้ใช้ส่งมา และใช้ mess เป็นเนื้อหาที่จะตอบกลับ
   - channelToken: ใช้เพื่อยืนยันตัวตนบอท LINE ในการเชื่อมต่อกับ LINE API

<u>สรุปการทำงาน:</u>

1. ส่งคำขอไปยัง Google Gemini Vision API เพื่อวิเคราะห์และสร้างคำบรรยายภาพจากภาพที่ผู้ใช้ส่งมา
2. รับผลลัพธ์จาก API ซึ่งเป็นข้อความคำบรรยายภาพ
3. เตรียมข้อความตอบกลับในรูปแบบที่ LINE API รองรับ
4. ส่งข้อความตอบกลับไปยังผู้ใช้ผ่าน LINE Messaging API

ฟังก์ชันนี้ช่วยให้บอทสามารถรับภาพจากผู้ใช้ วิเคราะห์ภาพ และตอบกลับด้วยคำบรรยายภาพที่สร้างจากโมเดล AI ของ Google Gemini Vision ได้อย่างอัตโนมัติ.