CLOUD FUNCTIONS

CONSULTAR CAJA REGISTRADORA BD

Descripció: Aquesta Cloud Function retorna la llista de totes les caixes registradores del supermercat amb la seva posició dins la graella.

Endpoint: `https://europe-west1-compacompraapp.cloudfunctions.net/consultarCajaRegistradoraBD`

Mètode: `GET`

Paràmetres d'entrada: No cal enviar cap paràmetre.

Resposta correcta:

```
{
  "cajas_registradoras": [
    { "x": 2, "y": 1 },
    { "x": 7, "y": 9 }
  ]
}
```

Errors habituals:

- `405` – Mètode no permès (només es permet GET)
- `404` – No s'ha trobat cap caixa registradora
- `500` – Error intern del servidor

Comentaris addicionals: Si no existeix cap caixa registradora, la funció retorna un missatge d'error amb codi 404. És recomanable consultar aquesta funció per saber on han d'acabar les rutes de recollida del robot.

Codi de la Cloud Function

```
import firebase_admin
from firebase_admin import firestore
import functions_framework
from flask import jsonify, request

# Inicialitzar Firebase
firebase_admin.initialize_app()
db = firestore.client()

@functions_framework.http
def consultarCajaRegistradoraBD(request):
    if request.method != 'GET':
        return jsonify({'error': 'Método no permitido, usa GET'}), 405

    try:
        # Obtenir totes les caixes registradores (pot haver-n'hi diverses)
        cajas_ref = db.collection('caja_registradora')
        cajas_docs = cajas_ref.stream()

        posiciones = []
        for doc in cajas_docs:
            data = doc.to_dict()
            x = data.get('x')
            y = data.get('y')
            if x is not None and y is not None:
                posiciones.append({'x': x, 'y': y})

        if not posiciones:
            return jsonify({'error': 'No se encontró ninguna caja registradora.'}), 404

        return jsonify({'cajas_registradoras': posiciones})

    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

CONSULTAR MAPA BD

Descripció: Aquesta Cloud Function retorna les dimensions del supermercat (amplada i llargada) i la llista d'obstacles (posicions x, y).

Endpoint: `https://europe-west1-compacompraapp.cloudfunctions.net/consultarMapaBD`

Mètode: `GET`

Paràmetres d'entrada: No cal enviar cap paràmetre.

Resposta correcta:

```
{
  "ancho": 8,
  "largo": 10,
  "obstaculos": [
    { "x": 1, "y": 0 },
    { "x": 4, "y": 3 }
    // ...
  ]
}
```

Errors habituals:

- `405` – Mètode no permès (només es permet GET)
- `404` – No s'ha trobat el document de dimensions del mapa
- `400` – El document no conté els camps 'ancho' o 'largo'
- `500` – Error intern del servidor

Comentaris addicionals: Aquesta funció és essencial per visualitzar la mida del supermercat i dibuixar els obstacles abans de calcular qualsevol ruta.

Codi de la Cloud Function

```
 import firebase_admin
from firebase_admin import firestore
import functions_framework
from flask import jsonify, request

# Inicialitzar Firebase
firebase_admin.initialize_app()
db = firestore.client()

@functions_framework.http
def consultarMapaBD(request):
    if request.method != 'GET':
        return jsonify({'error': 'Método no permitido, usa GET'}), 405

    try:
        # Obtenir les dimensions del mapa
        dimensiones_ref = db.collection('mapa').document('dimensiones')
        dimensiones_doc = dimensiones_ref.get()

        if not dimensiones_doc.exists:
            return jsonify({'error': "No se encontró el documento de dimensiones del mapa."}), 404

        dimensiones = dimensiones_doc.to_dict()
        ancho = dimensiones.get('ancho')
        largo = dimensiones.get('largo')

        if ancho is None or largo is None:
            return jsonify({'error': "El documento de dimensiones no contiene 'ancho' o 'largo'."}), 400

        # Obtenir tots els obstacles
        obstaculos_ref = db.collection('obstaculos')
        obstaculos_docs = obstaculos_ref.stream()

        posiciones_obstaculos = []
        for doc in obstaculos_docs:
            data = doc.to_dict()
            x = data.get('x')
            y = data.get('y')
            if x is not None and y is not None:
                posiciones_obstaculos.append({'x': x, 'y': y})

        return jsonify({
            'ancho': ancho,
            'largo': largo,
            'obstaculos': posiciones_obstaculos
        })

    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

CALCULAR RUTA

Descripció: Aquesta Cloud Function rep una posició d'inici, una llista de productes/objectius i calcula la ruta òptima (visitant tots els productes en l'ordre ideal i acabant a la caixa registradora), utilitzant el mapa real i obstacles del supermercat.

Endpoint: `https://europe-west1-compacompraapp.cloudfunctions.net/calcular_ruta_http`

Mètode: `POST`

Paràmetres d'entrada (JSON):

```
{
  "inicio": [7, 8],
  "objetivos": [[0, 1], [0, 7], [5, 2]],
  "lista_de_la_compra": ["ou", "llet", "pa"]
}
```

- `inicio` : Coordenada `[x, y]` del punt d'inici (habitualment la posició inicial del robot).
- `objetivos` : Llista de coordenades `[x, y]` de cada producte a recollir.
- `lista_de_la_compra` : Noms dels productes a recollir (en el mateix ordre que `objetivos`).

Resposta correcta:

```
{
  "ruta_optima": [
    [[7,8],[7,7],[6,7]],         // Primer tram: de l'inici al primer producte
    [[6,7],[0,7]],               // Segon tram: del primer al segon producte
    [[0,7],[5,2]],               // Tercer tram: del segon al tercer producte
    [[5,2],[2,1]]                // Darrer tram: de l'últim producte fins a la caixa
  ],
  "orden_objetivos": [
    [6, 7],
    [0, 7],
    [5, 2]
  ],
  "compra_ordenada": [
    "ou",
    "llet",
    "pa"
  ]
}
```

- `ruta_optima` : Llista de trams, cada tram és una llista de coordenades `[x, y]` .
- `orden_objetivos` : L'ordre òptim en què es visiten els objectius (coordenades).
- `compra_ordenada` : L'ordre òptim en què s'han de recollir els productes.

Errors habituals:

- `400` – Paràmetres d'entrada incorrectes o absents
- `500` – Error intern del servidor (ex. si algun objectiu no és accessible)

Comentaris addicionals: La funció consulta les Cloud Functions `consultarMapaBD` i `consultarCajaRegistradoraBD` per obtenir la graella del supermercat i la posició de la caixa registradora, respectivament.

Codi principal de la Cloud Function (`main.py`)

```
import json
import requests
from ruta import ruta_optima_super
from mapa import construir_mapa, obtener_posicion_caja

def calcular_ruta_http(request):
    # Espera un POST amb JSON amb les entrades
    datos = request.get_json()

    # Exemple de dades esperades:
    # {
    #   "inicio": [7, 8],
    #   "objetivos": [[0, 1], [0, 7], ...],
    #   "lista_de_la_compra": ["ou", "llet", ...]
    # }
    inicio = tuple(datos["inicio"])
    objetivos = [tuple(obj) for obj in datos["objetivos"]]
    lista_de_la_compra = datos["lista_de_la_compra"]

    # Recuperar la quadrícula del mapa
    url_consultar_mapa = "https://europe-west1-compacompraapp.cloudfunctions.net/consultarMapaBD"
    datos_mapa = requests.get(url_consultar_mapa)
    mapa = construir_mapa(datos_mapa.json())

    # Recuperar la posició de la caixa registradora
    url_consultar_caja_registradora = "https://europe-west1-compacompraapp.cloudfunctions.net/consultarCajaRegistradoraBD"
    datos_caja = requests.get(url_consultar_caja_registradora)
    caja_ubi = obtener_posicion_caja(datos_caja.json())

    fin = caja_ubi
    cuadricula = mapa
    resultado = ruta_optima_super(inicio, objetivos, lista_de_la_compra, fin, cuadricula)

    # Serialitzar llistes de tuples per JSON
    def serializar(o):
        if isinstance(o, str):
            return o
        if isinstance(o, (list, tuple)):
            return [serializar(x) for x in o]
        return o

    resultado_serializado = {k: serializar(v) for k, v in resultado.items()}
    return json.dumps(resultado_serializado), 200, {"Content-Type": "application/json"}
```

Mòdul de càlcul de rutes (`ruta.py`)

```
import numpy as np
import heapq
import math
from itertools import permutations

def calcular_ruta(inicio, objetivo, cuadricula):
    def heuristica(a, b):
        return math.sqrt((a[0] - b[0])**2 + (a[1] - b[1])**2)
    def vecinos(nodo):
        x, y = nodo
        for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
            nx, ny = x + dx, y + dy
            if 0 <= nx < len(cuadricula[0]) and 0 <= ny < len(cuadricula):
                if cuadricula[ny][nx] == 0:
                    yield (nx, ny)
    open_set = []
    heapq.heappush(open_set, (heuristica(inicio, objetivo), 0, inicio))
    came_from = {}
```

```
    g = {inicio: 0}
    while open_set:
        _, coste, actual = heapq.heappop(open_set)
        if actual == objetivo:
            ruta = [actual]
            while actual in came_from:
                actual = came_from[actual]
                ruta.append(actual)
            return ruta[::-1]
        for vecino in vecinos(actual):
            tentative_g = g[actual] + 1
            if vecino not in g or tentative_g < g[vecino]:
                came_from[vecino] = actual
                g[vecino] = tentative_g
                f = tentative_g + heuristica(vecino, objetivo)
                heapq.heappush(open_set, (f, tentative_g, vecino))
    return None

def matriz_distancias(nodos, cuadricula):
    n = len(nodos)
    dist = [[0]*n for _ in range(n)]
    rutas = {}
    for i in range(n):
        for j in range(i+1, n):
            ruta = calcular_ruta(nodos[i], nodos[j], cuadricula)
            if ruta is None:
                return None, None
            distancia = len(ruta)-1
            dist[i][j] = dist[j][i] = distancia
            rutas[(i, j)] = rutas[(j, i)] = ruta
    return dist, rutas

def tsp_inicio_objetivos_fin(dist, inicio_idx, objetivos_idx, fin_idx):
    mejor_orden = None
    mejor_coste = float('inf')
    for perm in permutations(objetivos_idx):
        indices = [inicio_idx] + list(perm) + [fin_idx]
        coste = sum(dist[indices[i]][indices[i+1]] for i in range(len(indices)-1))
        if coste < mejor_coste:
            mejor_coste = coste
            mejor_orden = indices
    return mejor_orden, mejor_coste

def ruta_optima_super(inicio, objetivos, lista_de_la_compra, fin, cuadricula):
    nodos = [inicio] + objetivos + [fin]
    inicio_idx = 0
    fin_idx = len(nodos) - 1
    objetivos_idx = list(range(1, len(nodos)-1))
    dist, rutas = matriz_distancias(nodos, cuadricula)
    if dist is None:
        raise ValueError("Hay objetivos sin ruta posible.")
    mejor_orden, mejor_coste = tsp_inicio_objetivos_fin(dist, inicio_idx, objetivos_idx, fin_idx)
    rutas_por_tramo = []
    orden_objetivos = []
    compra_ordenada = []
    for idx in range(1, len(mejor_orden)-1):
        orden_objetivos.append(nodos[mejor_orden[idx]])
        compra_ordenada.append(lista_de_la_compra[mejor_orden[idx]-1])
    for i in range(len(mejor_orden)-1):
        a, b = mejor_orden[i], mejor_orden[i+1]
        tramo = rutas[(a, b)]
        rutas_por_tramo.append(tramo)
    return {
        "ruta_optima": rutas_por_tramo,
        "orden_objetivos": orden_objetivos,
```

```
        "compra_ordenada": compra_ordenada,
    }
```

Mòdul de mapa (`mapa.py`)

```
import numpy as np

def construir_mapa(mapa_json):
    ancho = mapa_json['ancho']
    largo = mapa_json['largo']
    obstaculos = mapa_json['obstaculos']
    cuadricula = np.zeros((largo, ancho), dtype=int)
    for obs in obstaculos:
        x, y = obs['x'], obs['y']
        cuadricula[y, x] = 1
    return cuadricula

def obtener_posicion_caja(caja_json):
    if not caja_json["cajas_registradoras"]:
        raise ValueError("No hay cajas registradoras en el mapa.")
    caja = caja_json["cajas_registradoras"][0]
    return (caja["x"], caja["y"])
```

CONSULTAR NOMBRES BD

Descripció:
Aquesta Cloud Function comprova quins productes d'una llista existeixen realment a la base de dades (serveix per validar la llista de l'usuari).

Endpoint:
`https://europe-west1-compacompraapp.cloudfunctions.net/consultarNombresBD`

Mètode:
`POST`

Paràmetres d'entrada (JSON):

```
{
  "lista_nombres": ["Llet", "Pa", "Tomàquet"]
}
```

- `lista_nombres` : Llista de noms de productes que vols validar.

Resposta correcta:

```
{
  "coincidencias": ["Llet", "Tomàquet"]
}
```

- `coincidencias` : Llista dels productes de l'entrada que existeixen a la base de dades.

Errors habituals:

- `405` – Mètode no permès (només es permet POST)
- `400` – Falta la clau `lista_nombres` al cos de la petició

Comentaris addicionals: Serveix per netejar o validar la llista de la compra segons els productes reals existents.

Codi de la Cloud Function:

```
import firebase_admin
from firebase_admin import firestore
import functions_framework
from flask import jsonify, request

# Inicialitzar Firebase (assegura't de tenir GOOGLE_APPLICATION_CREDENTIALS configurada)
firebase_admin.initialize_app()
db = firestore.client()

@functions_framework.http
def consultarNombresBD(request):
    # Només accepta POST amb JSON
    if request.method != 'POST':
        return jsonify({'error': 'Método no permitido, usa POST'}), 405

    # Obtenir JSON de la request
    request_json = request.get_json(silent=True)
    if not request_json or 'lista_nombres' not in request_json:
        return jsonify({'error': 'Falta lista_nombres en el cuerpo de la petición'}), 400

    lista_nombres = request_json['lista_nombres']

    # Referència a la col·lecció 'productos'
    productos_ref = db.collection('productos')

    # Obtenir tots els documents de la col·lecció
    docs = productos_ref.stream()

    # Extreure noms des de la base de dades
    nombres_en_bd = set()
    for doc in docs:
        data = doc.to_dict()
        if "nombre" in data:
            nombres_en_bd.add(data["nombre"])

    # Filtrar només els noms que estan a tots dos conjunts
    coincidencias = [nombre for nombre in lista_nombres if nombre in nombres_en_bd]

    # Retorna JSON amb la llista de coincidències
    return jsonify({'coincidencias': coincidencias})
```

GET NOMBRESPRODUCTOS BD

Descripció:
Aquesta Cloud Function retorna la llista de tots els noms de productes existents a la base de dades.

Endpoint:
`https://europe-west1-compacompraapp.cloudfunctions.net/getNombresProductosBD`

Mètode:
`GET`

Paràmetres d'entrada:
No cal enviar cap paràmetre.

Resposta correcta:

```
{
  "nombres": ["Llet", "Pa", "Tomàquet"]
}
```

* `nombres` : Llista de tots els productes disponibles a la col·lecció `productos` .

Errors habituals:

- `405` – Mètode no permès (només es permet GET)
- `500` – Error intern del servidor

Comentaris addicionals: Ideal per mostrar tots els productes disponibles a l'usuari, per exemple en el desplegable inicial.

Codi de la Cloud Function:

```
import firebase_admin
from firebase_admin import firestore
import functions_framework
from flask import jsonify

# Inicialitzar Firebase (assegura't de tenir GOOGLE_APPLICATION_CREDENTIALS configurada)
firebase_admin.initialize_app()
db = firestore.client()

@functions_framework.http
def getNombresProductosBD(request):
    # Només accepta GET
    if request.method != 'GET':
        return jsonify({'error': 'Método no permitido, usa GET'}), 405

    try:
        # Referència a la col·lecció 'productos'
        productos_ref = db.collection('productos')
        docs = productos_ref.stream()

        # Extreu noms
        nombres = []
        for doc in docs:
            data = doc.to_dict()
            if "nombre" in data:
                nombres.append(data["nombre"])

        return jsonify({'nombres': nombres})

    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

OBTENER POSICIONES BD

Descripció:
Aquesta Cloud Function retorna la posició (`x` , `y`) de cadascun dels productes sol·licitats, segons la seva ubicació a la base de dades.

Endpoint:
`https://europe-west1-compacompraapp.cloudfunctions.net/obtenerPosicionesBD`

Mètode:
`POST`

Paràmetres d'entrada (JSON):

```
{
  "lista_nombres": ["Llet", "Pa", "Tomàquet"]
}
```

- `lista_nombres` : Llista de noms dels productes de què es vol saber la posició.

Resposta correcta:

```
{
  "posiciones": [
    { "x": 2, "y": 1 },
    { "x": 5, "y": 2 },
    null
  ]
}
```

- `posiciones` : Llista de posicions (objectes `{x, y}`) en el mateix ordre que `lista_nombres` . Si un producte no existeix, retorna `null` a la seva posició.

Errors habituals:

- `405` – Mètode no permès (només es permet POST)
- `400` – Falta la clau `lista_nombres` al cos de la petició

Comentaris addicionals: Aquesta funció és útil per convertir una llista de noms de productes en una llista de coordenades del supermercat, per mostrar-los en un mapa o calcular rutes.

Codi de la Cloud Function:

```
 import firebase_admin
from firebase_admin import firestore
import functions_framework
from flask import jsonify, request

# Inicialitzar Firebase (assegura't de tenir GOOGLE_APPLICATION_CREDENTIALS configurada)
firebase_admin.initialize_app()
db = firestore.client()

@functions_framework.http
def obtenerPosicionesBD(request):
    # Només accepta POST amb JSON
    if request.method != 'POST':
        return jsonify({'error': 'Método no permitido, usa POST'}), 405

    # Obtenir JSON de la request
    request_json = request.get_json(silent=True)
    if not request_json or 'lista_nombres' not in request_json:
        return jsonify({'error': 'Falta lista_nombres en el cuerpo de la petición'}), 400

    lista_nombres = request_json['lista_nombres']

    # Referència a la col·lecció 'productos'
    productos_ref = db.collection('productos')
    docs = productos_ref.stream()

    # Crear diccionari de posicions
    posiciones_dict = {}
    for doc in docs:
        data = doc.to_dict()
        nombre = data.get('nombre')
        posicion = data.get('posicion')
        if nombre and posicion and 'x' in posicion and 'y' in posicion:
            posiciones_dict[nombre] = {'x': posicion['x'], 'y': posicion['y']}

    # Crear llista de posicions en l'ordre rebut
    posiciones = []
    for nombre in lista_nombres:
        if nombre in posiciones_dict:
            posiciones.append(posiciones_dict[nombre])
        else:
            posiciones.append(None)  # o no incloure'l, segons la teva preferència

    # Retorna JSON amb les posicions
    return jsonify({'posiciones': posiciones})
```

PROCESAR LISTA DE LA COMPRA

Descripció:
Aquesta Cloud Function rep un àudio en format base64 (.flac, espanyol), el transcriu a text, analitza el text amb Google NLP i extreu els productes (entities de tipus CONSUMER_GOOD) mencionats a la llista de la compra dictada per veu.

Endpoint:
`https://europe-west1-compacompraapp.cloudfunctions.net/procesar_lista_de_la_compra`

Mètode:
`POST`

Paràmetres d'entrada (JSON):

```
{
  "audio": "BASE64..."
}
```

- `audio`: String base64 amb l'àudio de veu (format .flac, 16000 Hz, mono, idioma espanyol).

Resposta correcta:

```
{
  "transcript": "Quiero comprar huevos, leche y pan",
  "productos": ["huevos", "leche", "pan"],
  "es_lista_valida": true
}
```

- `transcript`: El text transcrit de l'àudio.
- `productos`: Llista de productes detectats.
- `es_lista_valida`: `true` si s'han detectat almenys un producte.

Errors habituals:

- `405` – Només es permet POST
- `400` – Falten dades d'àudio

Comentaris addicionals: Ideal per a funcionalitat de dictar la llista de la compra amb veu i reconèixer automàticament els productes a partir de la transcripció i anàlisi semàntica.

Codi de la Cloud Function:

```
from google.cloud import speech
from google.cloud import language_v1
import base64
import tempfile
import functions_framework

"""
Flujo de Trabajo:
1. Rep un àudio en base64 (.flac, 16000 Hz, mono).
2. El transcriu amb l'API de Speech-to-Text.
3. Processa el text amb la API de Natural Language.
4. Extreu els productes (CONSUMER_GOOD).

Retorna:
- El text transcrit.
- La llista de productes.
- Si és una llista vàlida (es_lista_valida: true/false).
"""

@functions_framework.http
def procesar_lista_de_la_compra(request):
    if request.method != 'POST':
        return ('Only POST allowed', 405)

    data = request.get_json()
    if not data or 'audio' not in data:
        return ('Missing audio data', 400)

    # === PAS 1: Decodificar audio .flac ===
    audio_content = base64.b64decode(data['audio'])

    with tempfile.NamedTemporaryFile(suffix=".flac") as temp_audio:
        temp_audio.write(audio_content)
        temp_audio.seek(0)

        # === PAS 2: Transcriure àudio ===
        speech_client = speech.SpeechClient()
        with open(temp_audio.name, "rb") as f:
            audio = speech.RecognitionAudio(content=f.read())
```

```
    config = speech.RecognitionConfig(
        encoding=speech.RecognitionConfig.AudioEncoding.FLAC,
        language_code="es-ES"
    )

    response = speech_client.recognize(config=config, audio=audio)
    transcript = " ".join([r.alternatives[0].transcript for r in response.results])

if not transcript.strip():
    return {'transcript': '', 'productos': [], 'es_lista_valida': False}

# === PAS 3: Analitzar text amb NLP ===
language_client = language_v1.LanguageServiceClient()
document = language_v1.Document(
    content=transcript,
    type_=language_v1.Document.Type.PLAIN_TEXT,
    language="es"
)

entities_response = language_client.analyze_entities(document=document)
productos = []
for entity in entities_response.entities:
    if language_v1.Entity.Type(entity.type_).name == 'CONSUMER_GOOD':
        productos.append(entity.name.lower())

productos = list(set(productos))  # Eliminar duplicats
es_lista_valida = len(productos) >= 1

return {
    'transcript': transcript,
    'productos': productos,
    'es_lista_valida': es_lista_valida
}
```