Funciones implementadas en Google Cloud

advertFunctions.py

```python
from google.cloud import firestore

# permite agregar un anuncio en firestore y actualizar los datos del usuario facilmente
def hello_world(request):
    """Responds to any HTTP request.
    Args:
        request (flask.Request): HTTP request object.
    Returns:
        The response text or any set of values that can be turned into a
        Response object using
        `make_response <http://flask.pocoo.org/docs/1.0/api/#flask.Flask.make_response>`.
    """

    request_json = request.get_json()  # Obtiene el cuerpo de la solicitud como JSON

    if request_json and 'userId' in request_json and 'advert' in request_json:
        # Inicializa el cliente de Firestore
        db = firestore.Client()

        # Verifica si el usuario existe
        user_ref = db.collection(u'users').document(request_json["userId"])
        user_doc = user_ref.get()
        if not user_doc.exists:
            return 'User does not exist', 418

        # Agrega un nuevo anuncio
        col_ref = db.collection(u'adverts')

        _, advert_ref = col_ref.add(request_json["advert"])

        # Agrega el anuncio al documento de usuario
        user_adverts = []
        if ('adverts' in user_doc.to_dict()):
            user_adverts = user_doc.get('adverts')
```

```
        user_adverts.append(advert_ref)
        user_ref.update({"adverts": user_adverts})  # Actualiza el
campo "adverts" en el documento del usuario

        return f'{advert_ref.id}', 200  # Devuelve el ID del nuevo
anuncio

    return 'Invalid request', 418  # Respuesta de solicitud inválida
```

createChat.py

```
from datetime import datetime
from google.cloud import firestore

# permite crear chats y actualizar los documentos de los usuarios
para que se guarden los chats como propios
def hello_world(request):
    """Responds to any HTTP request.
    Args:
        request (flask.Request): HTTP request object.
    Returns:
        The response text or any set of values that can be turned
into a
        Response object using
        `make_response
<http://flask.pocoo.org/docs/1.0/api/#flask.Flask.make_response>`.
    """

    request_json = request.get_json()  # Obtiene el cuerpo de la
solicitud como JSON

    if request_json and 'userId' in request_json and 'advertId' in
request_json:
        # Inicializa el cliente de Firestore
        db = firestore.Client()

        # Verifica si el usuario existe
        user_ref =
db.collection(u'users').document(request_json["userId"])
        user_doc = user_ref.get()
        if not user_doc.exists:
```

```
            return 'User does not exist', 418

        # Obtiene el propietario del anuncio
        advert_ref =
db.collection(u'adverts').document(request_json["advertId"])
        advert_doc = advert_ref.get()
        ownerId = advert_doc.get('owner')

        # Verifica si ya existe un chat para el anuncio y devuelve
su ID
        existingChats = db.collection(u'chats').where('advert',
'==', advert_ref).where('members', 'array_contains',
request_json["userId"]).stream()
        for ch in existingChats:
            return f'{ch.id}', 200

        _, chat_ref = db.collection(u'chats').add({
          "advert": advert_ref,
          "members": [ request_json["userId"], ownerId ],
          "recentMessageSentAt": datetime.now(),
          "recentMessageSentBy": request_json["userId"],
          "recentMessageText": "Hi, I'm interested"
        })

        chat_ref.update({
          "chatID": chat_ref
        })

        add_chat_to_user(db, chat_ref, request_json["userId"])
        add_chat_to_user(db, chat_ref, ownerId)

        return f'{chat_ref.id}', 200  # Devuelve el ID del nuevo
chat

    return 'Invalid request', 418  # Respuesta de solicitud inválida


def add_chat_to_user(db, chat_ref, userId):
    chats_ref =
db.collection(u'users').document(userId).collection(u'private').docu
ment(u'chats')
    chats_doc = chats_ref.get()

    # Agrega el chat a los chats del usuario
    user_chats = []
    if chats_doc.to_dict() is not None:
```

```
        if ('chats' in chats_doc.to_dict()):
            user_chats = chats_doc.get('chats')
    user_chats.append(chat_ref)
    chats_ref.update({"chats": user_chats})  # Actualiza el campo
"chats" en el documento del usuario
```

ImageDetect.js

```
// Importa las bibliotecas de cliente de Google Cloud
const vision = require('@google-cloud/vision');

// Crea un cliente
const client = new vision.ImageAnnotatorClient();

exports.labelDetection = async (req, res) => {
  const bucketName = 'exadas';  // Nombre del depósito de Google
Cloud Storage
  const fileName = req.query.filename;  // Obtiene el nombre de
archivo de la consulta de la solicitud

  // Realiza la detección de etiquetas en el archivo de GCS
  const [result] = await
client.labelDetection(`gs://${bucketName}/${fileName}`);
  const labels = result.labelAnnotations;  // Obtiene las
anotaciones de etiquetas del resultado

  let firstNonFurnitureLabel = null;
  for (const label of labels) {
    if (label.description.toLowerCase() !== 'furniture') {
      firstNonFurnitureLabel = label.description;  // Obtiene la
primera etiqueta que no sea "furniture"
      break;
    }
  }

  res.status(200).send(firstNonFurnitureLabel);  // Envía la
etiqueta no relacionada con muebles como respuesta
};
```

upload-image-py.py

```
from flask import make_response, send_file
import hashlib
import os
from io import BytesIO

# Imports the Google Cloud client library
from google.cloud import storage

# permite obtener y guardar imagenes
def entry(request):
    """Responds to any HTTP request.
    Args:
        request (flask.Request): HTTP request object.
    Returns:
        The response text or any set of values that can be turned
into a
        Response object using
        `make_response
<http://flask.pocoo.org/docs/1.0/api/#flask.Flask.make_response>`.
    """
    if request.method == 'POST': #upload
        # Verifica si hay archivos en la solicitud POST y si se
envió el archivo llamado "file"
        if len(request.files) == 0 or not 'file' in request.files:
            return make_response("Missing file from post request",
400)
        file = request.files["file"]  # Obtiene el archivo de la
solicitud
        extension = file.filename.split('.')[-1]  # Obtiene la
extensión del archivo

        m = hashlib.sha256()
        sha256Hash = hashlib.sha256(file.read())  # Calcula el hash
SHA256 del contenido del archivo
        file.seek(0)  # Restablece la lectura del archivo para poder
leerlo nuevamente
        sha256Hashed = sha256Hash.hexdigest()  # Obtiene el hash
SHA256 como una cadena hexadecimal

        newName = f'{sha256Hashed}.{extension}'  # Genera un nuevo
nombre de archivo utilizando el hash y la extensión

        # Instantiates a client
```

```
        storage_client = storage.Client()  # Crea una instancia del
cliente de Google Cloud Storage
        bucket_name = "exadas"  # Nombre del depósito de Google
Cloud Storage
        bucket = storage_client.bucket(bucket_name)  # Obtiene el
depósito especificado
        blob = bucket.blob(newName)  # Obtiene el objeto Blob
correspondiente al nuevo nombre de archivo

        # Verifica si el objeto Blob ya existe en el depósito
        if blob.exists():
            return newName, 299

        # Escribe el contenido del archivo en el objeto Blob
        with blob.open('wb') as f:
            f.write(file.read())

        return newName  # Devuelve el nuevo nombre de archivo
generado

    if request.method == 'GET': #download
        # Obtiene el parámetro "filename" de la solicitud GET
        filename = request.args.get('filename')
        if not filename:
            return 'Missing filename parameter', 400

        # Instantiates a client
        storage_client = storage.Client()  # Crea una instancia del
cliente de Google Cloud Storage

        # El nombre del depósito
        bucket_name = 'exadas'
        bucket = storage_client.get_bucket(bucket_name)  # Obtiene
el depósito especificado

        # Obtiene el objeto Blob correspondiente al nombre de
archivo
        blob = bucket.blob(filename)
        if not blob.exists():
            return 'File not found', 404

        # Crea un objeto de tipo archivo para contener los datos del
blob
        file_object = BytesIO()
        blob.download_to_file(file_object)
        file_object.seek(0)
```

```
        # Envía los datos del archivo como respuesta
        return send_file(file_object, attachment_filename=filename,
as_attachment=True)

    return 'idk'  # Respuesta predeterminada si no se cumple ninguna
condición
```