



Computação na Nuvem



Google Cloud Platform

Grupo 08

- 47283 Ricardo Bernardino
- 47249 Miguel Almeida

Licenciatura em Engenharia Informática
e de Computadores

Trabalho Final
2022/2023

Contrato

Submissão de um ficheiro imagem (foto) para deteção de monumentos. Esta operação recebe o conteúdo de um ficheiro em *stream* de blocos, guardando o mesmo como um *blob* no serviço Cloud Storage. No final, a operação retorna um identificador do pedido (por exemplo, uma composição única entre o nome do *bucket* e do *blob*) que será usado posteriormente para obter o resultado da submissão;

A partir de um identificador retornado na chamada à operação anterior deve ser possível obter:

- o a lista de resultados que inclui nomes de *landmarks* identificados na imagem, a localização geográfica (latitude e longitude em formato de graus decimais) e o respetivo nível de certeza associado à identificação (valor entre 0 e 1);
- o a imagem com o mapa estático de uma das localizações identificadas;

Obter os nomes de todas as fotos onde houve identificação de monumento com um grau de certeza acima de *t* (por exemplo, acima de 0,6) e o respetivo nome do local identificado.

```
service CN2223TFService {
  rpc uploadImage(stream Block) returns (ImageUploadResponse);
  rpc getLandmarks(Identifier) returns (SubmissionResponse);
  rpc getImage(Identifier) returns (stream ImageResponse);
  rpc getAllImages(Parameters) returns (ImagesResponse);
}

message Block {
  Image image=1;
}

message Image {
  Metadata metadata=1;
  bytes content = 2;
}

message Metadata {
  string name=1;
  string type=2;
}

message ImageUploadResponse {
  Identifier identifier = 1;
  Status status = 2;
}

message Identifier {
  string uuid=1;
}

message SubmissionResponse {
  repeated DetectedLandmark landmarks = 1;
}

message DetectedLandmark {
  string name=1;
  double latitude =2;
  double longitude =3;
  double confidence =4;
}
```

```
message ImageResponse {
  Image image = 1;
}

message Parameters {
  double certainty=1;
}

message ImagesResponse {
  repeated IdentifiedImage identified_image = 1;
}

message IdentifiedImage {
  string name = 1;
  string location=2;
}

enum Status {
  SUCCESS = 0;
  FAILURE = 1;
}
```



Client Cloud Function



```
System.out.println("Searching for available servers...");
String[] serversIP = getAvailableServers();
String serverIP = "";
if (serversIP != null && serversIP.length > 0) {
    serverIP = selectAvailableServer(serversIP);
    serverIP = serverIP.equals("") ? "localhost" : serverIP;
} else {
    serverIP = "localhost";
}
System.out.println("Server IP selected: " + serverIP);

setupServerConnection(serverIP, svcPort);
```

```
private static String[] getAvailableServers() throws IOException, InterruptedException {
    String cfURL = "https://europe-west1-cn2223-t1-g08.cloudfunctions.net/funcLookup?";
    cfURL += "projectId=cn2223-t1-g08&";
    cfURL += "europe-west1-b&";
    cfURL += "instance-group-servers";

    HttpClient client = HttpClient.newBuilder().build();
    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create(cfURL))
        .GET()
        .build();
    HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
    if (response.statusCode() == 200) {
        System.out.println(response.body());
        return response.body().split(",");
    } else {
        System.out.println "[" + response.statusCode() + "] There was a problem! Server's IP couldn't be accessed";
    }
    return null;
}
```

```
static void setupServerConnection(String svcIP, Integer svcPort) {
    channel = ManagedChannelBuilder.forAddress(svcIP, svcPort)
        .usePlaintext()
        .build();

    noBlockStub = CN2223TFServiceGrpc.newStub(channel);
}
```

```
private static String selectAvailableServer(String[] serversIP) {
    int rnd = new Random().nextInt(serversIP.length);
    return serversIP[rnd];
}
```



Pub/Sub Server App

```
@Override
public StreamObserver<Block> uploadImage(StreamObserver<ImageUploadResponse> responseObserver) {
    System.out.println();
    System.out.println("Uploading image...");
    return new StreamObserverImage(responseObserver, storage, IMAGES_BUCKET_NAME);
}
```

```
public static void publishMessage(String pubTopicName, String id) throws Exception {
    TopicName topic = TopicName.ofProjectTopicName("cn2223-t1-g08", pubTopicName);
    Publisher publisher = Publisher.newBuilder(topic).build();

    PubsubMessage pubsubMessage = PubsubMessage.newBuilder()
        .putAttributes("id", id)
        .build();
    ApiFuture<String> future = publisher.publish(pubsubMessage);
    String msgID = future.get();
    System.out.println("Message Published with ID=" + msgID);
    publisher.shutdown();
}
```

```
public class StreamObserverImage implements StreamObserver<Block> {

    @Override
    public void onCompleted() {
        try {
            writer.close();
            String uuId = uploadImageToCloudStorage(new ByteArrayInputStream(writer.toByteArray()));
            publishMessage("detectionworkers", uuId);

            Identifier identifier = Identifier.newBuilder()
                .setUuid(uuId)
                .build();

            ImageUploadResponse response = ImageUploadResponse.newBuilder()
                .setIdentifier(identifier)
                .setStatus(status)
                .build();

            replies.onNext(response);
            replies.onCompleted();
            System.out.println("Finished request");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



Pub/Sub Landmarks App

```
public static void main(String[] args){  
  
    initStorage();  
  
    String subscriptionName = "detectionworkers-sub";  
    String projectID = PROJECT_ID;  
    Subscriber subscriber = subscribeMessages(projectID, subscriptionName);  
    System.out.println("Started listening...");  
    subscriber.awaitTerminated();  
  
    System.out.println("Terminating...");  
}
```

```
public static Subscriber subscribeMessages(String projectID, String subscriptionName) {  
  
    ProjectSubscriptionName projSubscriptionName = ProjectSubscriptionName.of(  
        projectID, subscriptionName);  
    Subscriber subscriber =  
        Subscriber.newBuilder(projSubscriptionName, new MessageReceiveHandler(key))  
            .build();  
    subscriber.startAsync().awaitRunning();  
    return subscriber;  
}
```

```
@Override  
public void receiveMessage(PubsubMessage pubsubMessage, AckReplyConsumer ackReplyConsumer) {  
    System.out.println("Message Id:" + pubsubMessage.getMessageId() + "\nData:" + pubsubMessage.getData().toStringUtf8() + "");  
    Map<String, String> atributes = pubsubMessage.getAttributesMap();  
    String imageID = null;  
    for (String key : atributes.keySet()) {  
        System.out.println("Msg Attribute:(" + key + ", " + atributes.get(key) + ")");  
        imageID = atributes.get(key);  
    }  
}
```

Detecção e armazenamento dos mapas

```
@Override
public void receiveMessage(PubsubMessage pubsubMessage, AckReplyConsumer ackReplyConsumer) {
    System.out.println("Message Id:" + pubsubMessage.getMessageId() + "\nData:" + pubsubMessage.getData().toStringUtf8());
    Map<String, String> atribs = pubsubMessage.getAttributesMap();
    String imageID = null;
    for (String key : atribs.keySet()) {
        System.out.println("Map Attribute:{" + key + ", " + atribs.get(key) + "}");
        imageID = atribs.get(key);
    }

    // First: Initiate storage
    if (storage == null) {
        initStorage();
    }
    try {
        operations.init(key);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    // Second: Process image
    List<DetectedLandmark> landmarkList = null;
    try {
        landmarkList = LandmarkDetector.detectLandmarksGcs(imageID);
    } catch (IOException e) {
        try {
            operations.close();
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }
        throw new RuntimeException(e);
    }

    // Third: Create Document in Firestore with all information
    operations.createDocument(imageID, landmarkList);
}
```

```
public void createDocument(String requestID, List<DetectedLandmark> detectedLandmarks) {
    CollectionReference docs = db.collection(currentCollection);
    DocumentReference newDoc = docs.document(requestID);

    List<String> locationNames = new ArrayList<String>();
    List<String> locationPositions = new ArrayList<String>();
    List<String> mapIds = new ArrayList<String>();
    List<Double> confidences = new ArrayList<Double>();

    for (DetectedLandmark obj : detectedLandmarks) {
        locationNames.add(obj.name);
        locationPositions.add(obj.latitude + ", " + obj.longitude);
        mapIds.add("landmark_detection_map_" + obj.id);
        confidences.add(obj.confidence);
    }

    HashMap<String, Object> map = new HashMap<String, Object>();
    {
        put("imageId", "landmark_detection_map_" + requestID);
        put("locationName", locationNames);
    }
}
```

```
// Detects landmarks in the specified remote image on Google Cloud Storage.
public static List<DetectedLandmark> detectLandmarksGcs(String blobGPath) throws IOException {
    System.out.println("Detecting landmarks for: " + blobGPath);
    List<AnnotateImageRequest> requests = new ArrayList<>();

    ImageSource imgSource = ImageSource.newBuilder().setGcsImageUri("gs://landmark_detection/").build();
    Image img = Image.newBuilder().setSource(imgSource).build();
    Feature feat = Feature.newBuilder().setType(Feature.Type.LANDMARK_DETECTION).build();
    AnnotateImageRequest request =
        AnnotateImageRequest.newBuilder().addFeatures(feat).setImage(img).build();
    requests.add(request);

    List<DetectedLandmark> detectedLandmarks = new ArrayList<>();

    // Initialize client that will be used to send requests. This client only needs to be created
    // once, and can be reused for multiple requests. After completing all of your requests,
    // the "close" method on the client to safely clean up any remaining background resources.
    try (ImageAnnotatorClient client = ImageAnnotatorClient.create()) {
        BatchAnnotateImagesResponse response = client.batchAnnotateImages(requests);
        List<AnnotateImageResponse> responses = response.getResponsesList();

        for (AnnotateImageResponse res : responses) {
            if (res.hasError()) {
                System.out.format("Error: %s\n", res.getError().getMessage());
                return Collections.emptyList();
            }

            System.out.println("Landmarks list size: " + res.getLandmarkAnnotationsList().size());
            // For full list of available annotations, see http://g.co/cloud/vision/docs
            boolean first = true;
            for (EntityAnnotation annotation : res.getLandmarkAnnotationsList()) {
                LocationInfo info = annotation.getLocationList().iterator().next();
                String mapUUID = UUID.randomUUID().toString();
                System.out.format("Landmark: %s(%f)\n %s\n",
                    annotation.getDescription(),
                    annotation.getScore(),
                    info.getLatitude());

                getStaticMapSaveImage(info.getLatitude(), mapUUID);
                detectedLandmarks.add(new DetectedLandmark(annotation.getDescription(), info.getLatitude(), info.getLongitude(), mapUUID));
            }
        }
    }
    return detectedLandmarks;
}
```

```
private static void getStaticMapSaveImage(LatLng latLng, String uuid) {
    String mapUrl = "https://maps.googleapis.com/maps/api/staticmap?"
        + "center=" + latLng.getLatitude() + "," + latLng.getLongitude()
        + "&zoom=2000"
        + "&size=512x512"
        + "&key=AIzaSyAkgW5k8rFm5vQ9Ww-1bF52ab3bW";
    System.out.println(mapUrl);
    try {
        URL url = new URL(mapUrl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        InputStream in = conn.getInputStream();
        BufferedInputStream buffIn = new BufferedInputStream(in);
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        FileOutputStream out = new FileOutputStream(uuid + ".png");
        byte[] buffer = new byte[8 * 1024];
        int bytesRead;
        while ((bytesRead = buffIn.read(buffer)) != -1) {
            out.write(buffer, 0, bytesRead);
            outputStream.write(buffer, 0, bytesRead);
        }
        buffIn.close();
        in.close();
        out.close();

        byte[] imageBytes = outputStream.toByteArray();
        ByteArrayInputStream inputStream = new ByteArrayInputStream(imageBytes);
        uploadMapToCloudStorage(inputStream, uuid);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
public static String uploadMapToCloudStorage(ByteArrayInputStream inputStream, String uuid) {
    String bucketName = MAPS_BUCKET_NAME;
    String fileName = uuid + ".png";

    BlobId blobId = BlobId.of(bucketName, fileName);
    BlobInfo blobInfo = BlobInfo.newBuilder(blobId).build();
    if (inputImage.available() < 1,000,000) {
        // When content is not available or large (1MB or more) it is recommended
        // to write it in chunks via the Blob's channel writer.
        try (WriteChannelWriter writer = storage.writer(blobInfo)) {
            byte[] buffer = new byte[1024];
            int limit;
            while ((limit = inputStream.read(buffer)) != 0) {
                try {
                    writer.write(ByteBuffer.wrap(buffer, 0, limit));
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else if (inputImage.available() > 0) {
        byte[] bytes = inputImage.readAllBytes();
        // create the blob in one request.
        storage.create(blobInfo, bytes);
    }
    System.out.println("Blob " + blobName + " created in bucket " + bucketName);
    return uuid;
}
```




Google Cloud Storage

- ☐ [landmark_detection_app_images](#)
- ☐ [landmark_detection_app_maps](#)

landmark_detection_app_images

Location	Storage class	Public access	Prot
europa-southwest1 (Madrid)	Standard	Not public	Non

OBJECTS CONFIGURATION PERMISSIONS PROTI

Buckets > landmark_detection_app_images

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRAN

Filter by name prefix only Filter objects and folders

<input type="checkbox"/>	Name	Size
<input type="checkbox"/>	193e2c98-cbcc-44f8-b6aa-59da8965d93a.jpg	6.4 KB
<input type="checkbox"/>	4b248b09-3744-4c07-8d82-d811cc18c73d.j_	126.3 KB
<input type="checkbox"/>	d505ff37-1e85-4d68-8628-5cf39d632e40.jpg	4.4 KB

landmark_detection_app_maps

Location	Storage class	Public access	Prote
europa-southwest1 (Madrid)	Standard	Not public	None

OBJECTS CONFIGURATION PERMISSIONS PROTEI

Buckets > landmark_detection_app_maps

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANS

Filter by name prefix only Filter objects and folders

<input type="checkbox"/>	Name	Size
<input type="checkbox"/>	55261602-f9ee-424b-a807-a5399767bba3.png	27.1 KB
<input type="checkbox"/>	87814cee-6af5-4afb-ab35-a504ed3abcd6.png	13.3 KB
<input type="checkbox"/>	92d46ac7-6ba7-45b4-b5b3-56f4d9b43fda.png	13.1 KB
<input type="checkbox"/>	9c0d5374-9e79-4b4a-a9a8-4ab316dd18a9.png	47.7 KB
<input type="checkbox"/>	a3ab95da-d587-491f-a7ec-6ae289642004.png	34.5 KB



Cloud Firestore

Coleção

cn2223-t1-g08

+ START COLLECTION

ImagesAndLandmarks

Documentos

ImagesAndLandmarks

+ ADD DOCUMENT

193e2c98-cbcc-44f8-b6aa-59da8965d93a

4b248b09-3744-4c07-8d82-d811cc18c73d

d505ff37-1e85-4d68-8628-5cf39d632e40

Documento (e os seus respectivos campos)

193e2c98-cbcc-44f8-b6aa-59da8965d93a

+ START COLLECTION

+ ADD FIELD

ImageId: "landmark_detection_app_images/193e2c98-cbcc-44f8-b6aa-59da8965d93a"

LocationName

0: "Belém Tower Garden"

1: "Belém Tower"

LocationPosition

0: "38.6927205,-9.215710399999999"

1: "38.691583699999999,-9.215977299999999"

MapId

0: "landmark_detection_app_maps/a3ab95da-d587-491f-a7ec-6ae289642004"

1: "landmark_detection_app_maps/55261602-f9ee-424b-a807-a5399767bba3"

confidence

0: 0.7065483331680298

1: 0.6941734552383423

FAQ

