```
    # Get the information about face recognition algorithm
    encoding_method =
events_data.document(event_id).get().to_dict()

    try:
        # Start Face Representation and save to array value
        if encoding_method['face_recognition_method'] == 'basic':
            for img in result[0]:
                img_read = cv2.imread(img)
                img = cv2.cvtColor(img_read, cv2.COLOR_BGR2RGB)
                encode = represent(img, model_name="ArcFace",
detector_backend="mtcnn",
                                   enforce_detection=False)
                encode_list.append(encode)

        # Start Face Representation and save to array value
        else:
            for img in result[0]:
                img_read = cv2.imread(img)
                img = cv2.cvtColor(img_read, cv2.COLOR_BGR2RGB)
                encode = represent(img, model_name="ArcFace",
detector_backend="retinaface",
                                   enforce_detection=False)
                encode_list.append(encode)

    except ValueError:
        return
flask.abort(flask.make_response(jsonify(message='Pictures must
contain face',

status=403), 403))

    # Save encoding value to Data Json for saving to database and
return response
    for x in range(len(encode_list)):
        data = {
            "event_id": event_id,
            "encoding_images": encode_list[x],
            "class_names": result[1][x],
        }

        # Add Encoding data to database
        encoding_data.add(data)

    # Delete the img of converting in server Storage
    for img in image_list:
        img = img.to_dict()
        os.remove(str(img['participant_id'] + ".png"))

    return flask.make_response(jsonify(message="success",
status="200"), 200)
```

Attachment 2. Source Code Face Recognition

```
# Recognizing Attendance Image
@check_session
def recognition(event_id):
```

```
    try:
        # Get session cookie of user
        session_cookie = request.cookies.get('user')

        session_cookie = session_cookie.split(".")

        session_cookie = session_cookie[0]

        # Get the recognition algorithm of event
        recognition_method =
events_data.document(event_id).get().to_dict()
        # Cek the threshold
        threshold = dst.findThreshold('ArcFace', 'cosine')

        # Get Image request from POST Request
        img_request = request.files['image']
        filename = 'face' + session_cookie + '.jpg'
        img_request.save(filename)
        img_read = cv2.imread(filename)

        # Get the encoding data training of selected event
        known_encode = encoding_data.where("event_id", "==",
event_id).get()

        norm = np.zeros((800, 800))
        norm_image = cv2.normalize(img_read, norm, 0, 255,
cv2.NORM_MINMAX)

        # Start Face Representation of Attendance Image (Request
File from POST Request)
        if recognition_method['face_recognition_method'] ==
'basic':
            img_request_encode = represent(norm_image,
model_name="ArcFace",

detector_backend="mtcnn",

enforce_detection=False)

        else:
            img_request_encode = represent(norm_image,
model_name="ArcFace",

detector_backend="retinaface",

enforce_detection=False)

        temp_distance = []
        class_names = []

        # Comparing attendance image with encoding data training
image
        for comparison in known_encode:
            data_enc = comparison.to_dict()
            class_names.append(data_enc["class_names"])
            distance =
dst.findCosineDistance(data_enc["encoding_images"],
img_request_encode)
            temp_distance.append(distance)
```

```
        # Get the Min value of distance in comparison
        match_index = argmin(temp_distance)

        # Get the class names of smallest distance for recognition
        recognized_as = class_names[match_index]

        # Input all required data for response
        check = {
            "all_temp": temp_distance,
            "min_distance": temp_distance[match_index],
            "threshold": threshold,
            "recognized": recognized_as,
            "class_names": class_names,
        }

        # Check the smallest distance with threshold
        if temp_distance[match_index] > (threshold + 0.22):
            os.remove('face' + session_cookie + '.jpg')

            return
flask.abort(flask.make_response(jsonify(checker=check,
message='Face Not Recognized',

status=403), 403))
        # Find the data of recognized guest
        data = \
            guest_details_data.where("event_id", "==",
event_id).where("participant_id", "==", recognized_as).get()[
                0].to_dict()

        return flask.make_response(jsonify(message="Success",
status=200, data=data, cheker=check), 200)

    except ValueError:
        return flask.make_response(jsonify(message="Face could not
be detected", status=400), 400)
```