FacEntry

Group ID: 9-10

Group Members:

1. Zhao Jianxiang (2P129) [Leader]
2. David Luo Kecheng (2P114)
3. Jason Wang (2P123)

Table of Contents

1. Introduction

1.1 Abstract

During the current COVID-19 global pandemic, 4.3 million people have died, with another 203 million people infected with COVID-19 as of 10 August 2021.[1] Even with the introduction of vaccines to slow down the spread of COVID-19, one of the most crucial ways to prevent the spread of the virus is a robust and efficient contact-tracing system. Contact-tracing singles out potential infected individuals and action can then be taken to isolate and quarantine them, preventing further spread of the virus.

1.2 Rationale and Objectives

Our app will allow the user to sign in to locations using Face Recognition Technology and Artificial Learning. Our app will also be able to merge the process of marking attendance and contact tracing. This would also make it easier and more convenient to take attendance in situations that require such attendance taking, such as in schools and workplaces.

1.3 Significance of project

This project aims to make it easier for users to participate in contact-tracing activities, and to take attendance.

1.4 Target Audience and Project Scope

The target audience of this project is schools, offices, and other settings that require attendance taking or contact tracing.

2. Literature Review

2.1 SafeEntry and Tracetogether

Contact tracing apps and tokens such as SafeEntry and TraceTogether tokens have already been in use in Singapore during the pandemic. SafeEntry allows visitors to check-in by scanning their official photo ID or by allowing them to scan a unique QR code at their location.[2] The TraceTogether apps and tokens utilise a custom protocol, BlueTrace, which allows for a distributed approach whereby participating devices exchange proximity information whenever an app detects another device with the TraceTogether app installed.[3] However, both of them are unable to use face recognition to enable a person to check-in or mark attendance.

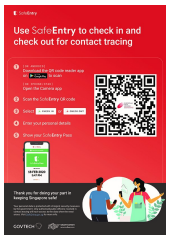

Figure 2.1: SafeEntry

Figure 2.2: Tracetogether token

2.2 FaceEntry - XS

We found a product also called “FaceEntry - XS”. According to its official website, “It allows the biometric verification of cardholders. As a pure access terminal, it is equipped with a 2D face detection and PIN code as standard.”[4] It is a physical product, unlike our app. It does not have any form of contact tracing purpose at all.

Figure 2.3: FacEntry - XS

3. Study and Methodology

3.1 Timeline

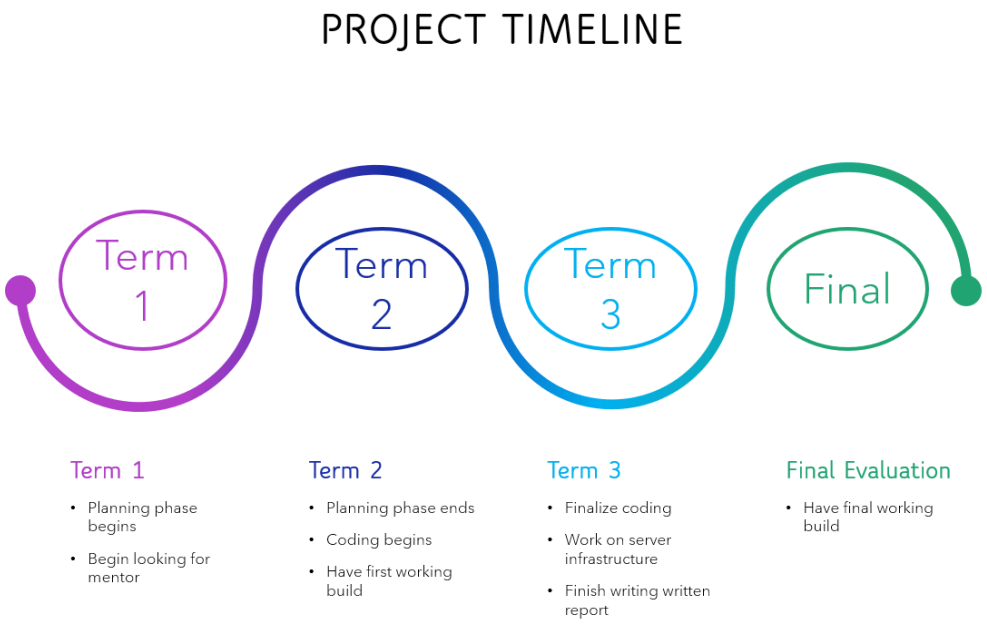

Figure 3.1: Timeline

3.2 Work distribution

Jianxiang: Programmer

Jason: Assistant programmer

David: Assistant programmer

3.3 Software Used

- Flutter - Interface
- Android Studio
- Visual Studio Code - IDE
- MySQL - Database
- OpenCV - Face Recognition
- Python - Server
- Xampp - Server
- Flask - web framework

Figure 3.2: Flutter

Figure 3.3: Python

Figure 3.4: Android Studio

Figure 3.5: MySQL

Figure 3.6: Open CV

Figure 3.7: Visual Studio Code

4. Outcomes, Analysis & Discussion

4.1 System Design

4.1.1 Modules

There are 4 modules in our system, The input from the user is passed from the app to the server. The server then passes the data to the database to be stored. When logging in, the server also uses OpenCV to check if the photo uploaded matches with the BC number.

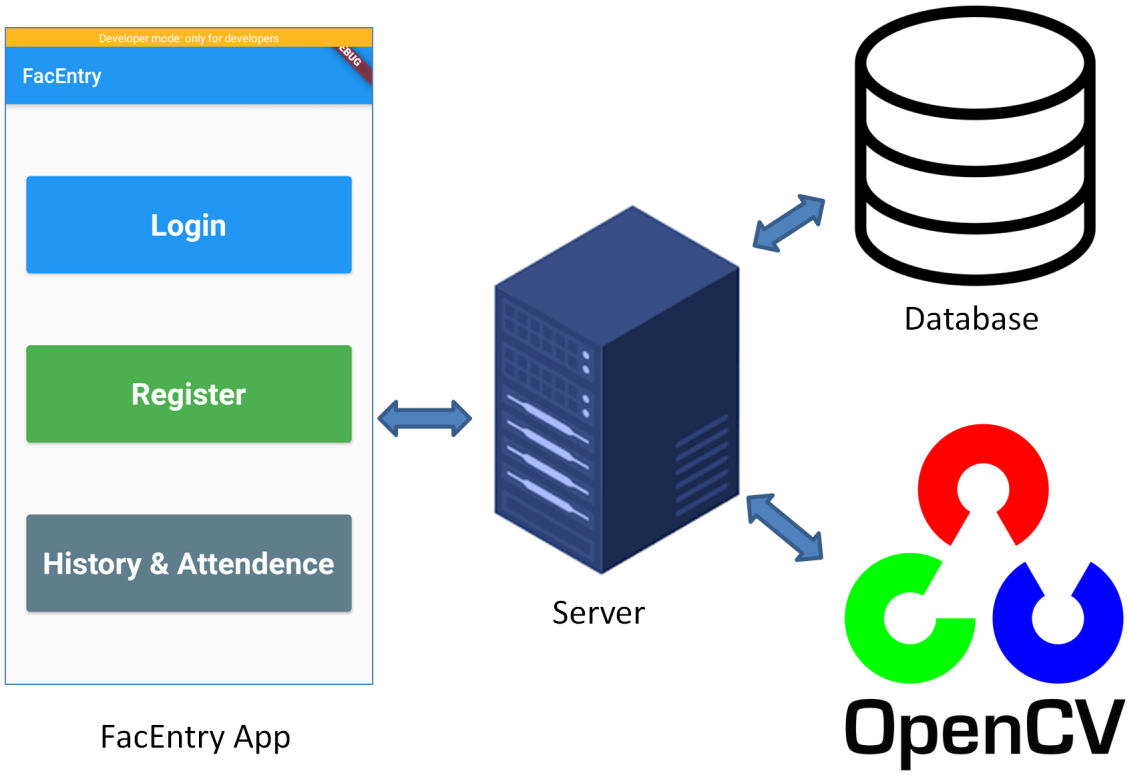

Figure 4.1: System Design

4.1.2 App server communication API design

There are 3 APIs for the communication between the app and server: Register, Login, History.

Register

Description	Register all personal details: First name, Last name, BC no., 1 face photo
URL structure	http://127.0.0.1:5000/register
Method	POST
Example	http://127.0.0.1:5000/register first_name=<First name> last_name=<Last name> bc_num=<BC no.> photo=<face photo>
Parameters	first_name *string* first name of person last_name *string* last name of person bc_num *string* birth cert. number of person photo *.jpg file* face photo of person
Returns	true: all personal details match & no account of same person exists yet false: personal details missing **or** account already exists
Errors	

Login

Description	Scan location(qr code), enter BC no., scan face
URL structure	http://127.0.0.1:5000/login
Method	POST
Example	http://127.0.0.1:5000/login location=<location> bc_num=<BC no.> photo=<face photo>
Parameters	location *string* location bc_num *string* birth cert. number of person photo *.jpg file* face photo of person
Returns	true: face matches BC no. false: face does not match BC no. **or** bc no. not found
Errors	

History

Description	Scan location(qr code), enter BC no., scan face
URL structure	http://127.0.0.1:5000/history
Method	POST
Example	http://127.0.0.1:5000/history bc_num=<BC no.> photo=<face photo>
Parameters	bc_num *string* birth cert. number of person photo *.jpg file* face photo of person
Returns	returns list of places visited and timestamp
Errors	Face does not match **or** BC no. not found

4.2 System Implementation

4.2.1 FacEntry App

At the homepage of FacEntry, the user will choose to register, login, or view history and attendance. The register page will require the user to enter their name, BC number and scan their face. The login page requires the user to scan a QR code, enter their BC number, and scan their face for verification. The History & Attendance page will require the user to enter their BC number and scan their face for verification.

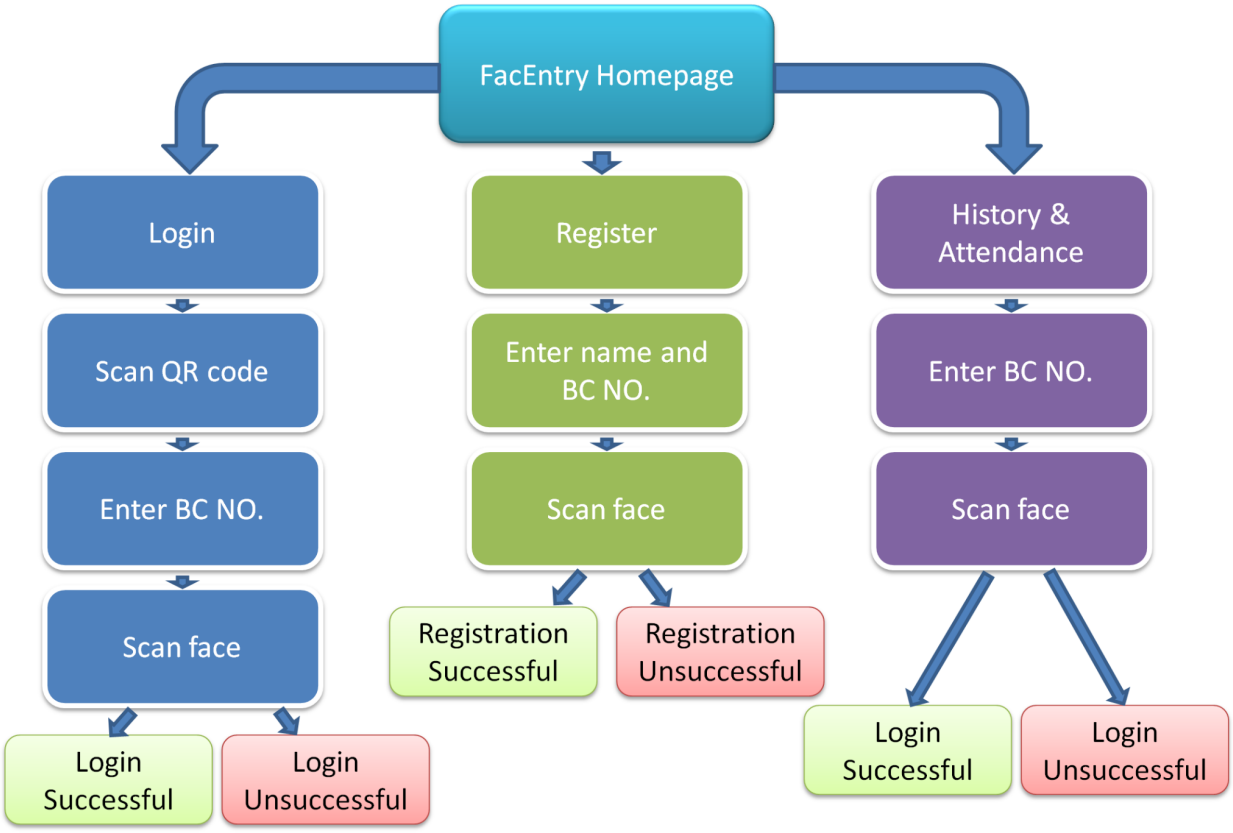

Figure 4.2: Flowchart showing the steps in using the app

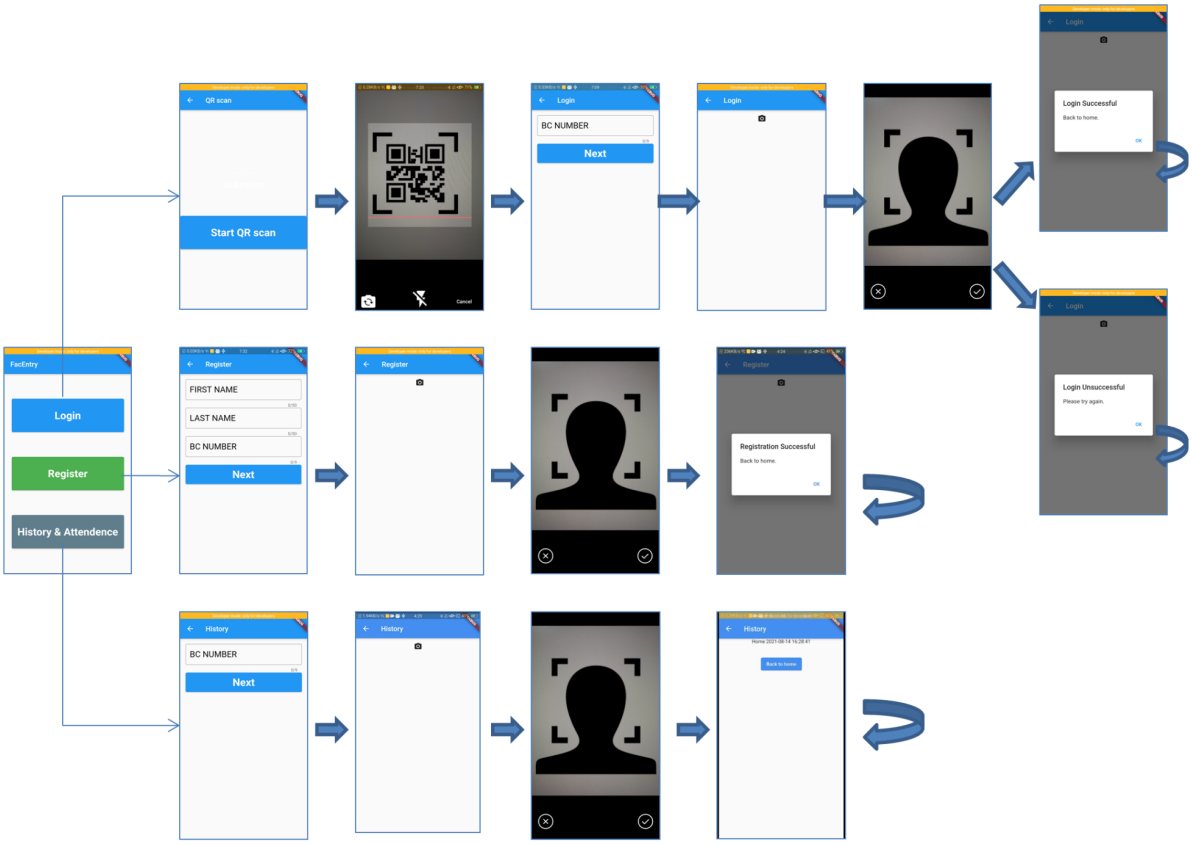

Figure 4.3: Flowchart showing the interface of the app

Figures 4.4 - 4.10 are screenshots of our app:

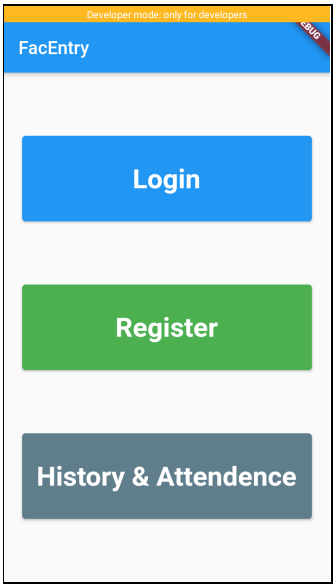

Figure 4.4: Homepage

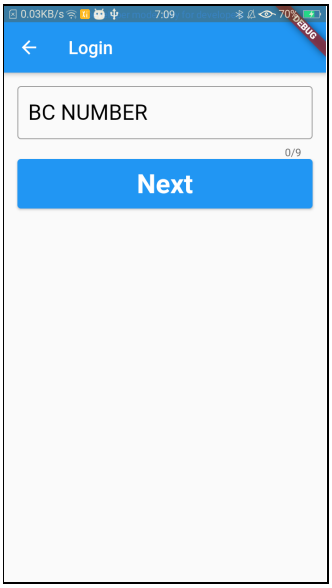

Figure 4.5: Login page

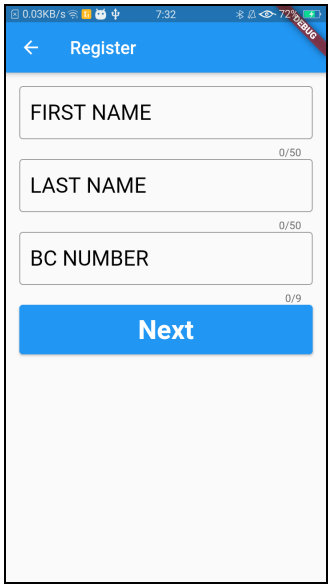

Figure 4.6: Register page

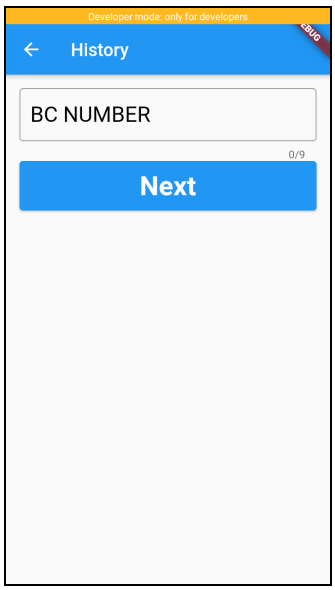

Figure 4.7: Login for History and Attendance

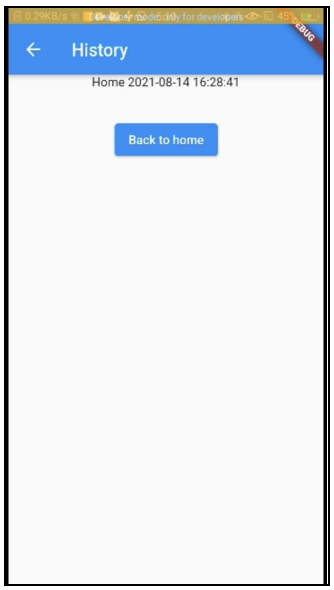

Figure 4.8: History and Attendance page

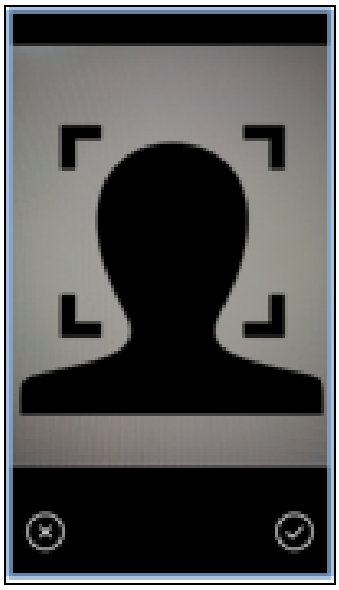

Figure 4.9: Face scanning page

Figure 4.10: QR code scanning page

This code captures a photo of the user's face, and uploads it, together with the user's location and BC number to the server, and shows a pop-up dialog box telling the user if login was successful.

```
254   Future getImage(bool isCamera) async {
255     late XFile? image;
256     if (isCamera) {
257       image = (await _picker.pickImage(source: ImageSource.camera));
258       if (image != null) {
259         DialogBuilder(context).showLoadingIndicator('Preparing image...');
260
261         img.Image resizedImg = await compute(prepareImage, image);
262         final directory = await getApplicationDocumentsDirectory();
263         File imgFile = await File(directory.path + '/login.jpg')
264             .writeAsBytes(img.encodeJpg(resizedImg));
265         var request = http.MultipartRequest(
266             'POST', Uri.parse('http://192.168.1.112:5000/login'));
267         request.fields['bc_num'] = Global.bcNumber;
268         request.fields['location'] = Global.location;
269         request.files.add(http.MultipartFile.fromBytes(
270             'photo', imgFile.readAsBytesSync(),
271             filename: 'login.jpg'));
272         DialogBuilder(context).hideOpenDialog();
273         DialogBuilder(context).showLoadingIndicator('Verifying...');
274         var res = await request.send();
275         DialogBuilder(context).hideOpenDialog();
276
277         final respStr = await res.stream.bytesToString();
278
279         if (respStr == 'true') {
280           showDialog<void>(
281             context: context,
282             barrierDismissible: false, // user must tap button!
283             builder: (BuildContext context) {
284               return AlertDialog(
285                 title: const Text('Login Successful'),
286                 content: SingleChildScrollView(
287                   child: ListBody(
288                     children: const <Widget>[
289                       Text('Back to home.'),
290                     ], // <Widget>[]
291                   ), // ListBody
292                 ), // SingleChildScrollView
```

Figure 4.11: Flutter code for capturing image and posting to server for login page

4.2.2 Server

The server checks if the face of the user matches with the BC number entered, using face recognition, and returns a value to the app. The server implements 3 APIs: Register, Login, and History. Figure 4.12 shows the code for the login API.

```python
@app.route('/login', methods=['POST'])
def login():
    message = ''
    status = False

    location = request.form.get('location')
    bc_num = request.form.get('bc_num')

    if location == '':
        message += 'Location is missing.\n'
        status = False

    if bc_num == '':
        message += 'BC No. is missing.\n'
        status = False

    if 'photo' not in request.files:
        message += 'No file part\n'
        status = False
    else:
        file = request.files['photo']
        if file.filename == '':
            flash('No image selected for uploading')
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(randomword(10) + '_' + file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

            mycursor = mydb.cursor()
            sql = "SELECT * FROM `users` WHERE `bc_number`=%s"
            val = (bc_num,)
            mycursor.execute(sql, val)

            myresult = mycursor.fetchall()

            if len(myresult) > 0:
                dbfilename = myresult[0][5]
                id = myresult[0][0]

                if facerec(UPLOAD_FOLDER + filename, REGISTER_FOLDER + dbfilename):
                    message = 'Image Matched'

                    sql = "INSERT INTO `history` (`id`, `location`, `time`) VALUES (%s
                    val = (str(id), location, str(int(time.time())))
                    mycursor.execute(sql, val)

                    mydb.commit()

                    status = True
                else:
                    message = 'Image not Matched'
                    status = False

            mycursor.close()

    if status:
        return 'true'
    else:
        return message
```

Figure 4.12: Python code for login with face recognition

4.2.3 Database

The user table stores the name, BC number, and face of the user.

Table structure | Relation view

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	id	bigint(20)			No	None		AUTO_INCREMENT
☐	2	first_name	varchar(50)	utf8mb4_general_ci		No	None		
☐	3	last_name	varchar(50)	utf8mb4_general_ci		No	None		
☐	4	bc_number	varchar(20)	utf8mb4_general_ci		No	None		
☐	5	status	int(11)			No	None		
☐	6	photo	varchar(256)	utf8mb4_general_ci		No	None		

Figure 4.13: Structure of user table

The history table stores the location and time of logins by users.

Table structure | Relation view

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
☐	1	history_id	bigint(20)			No	None		AUTO_INCREMENT
☐	2	id	bigint(20)			No	None		
☐	3	location	varchar(256)	utf8mb4_general_ci		No	None		
☐	4	time	bigint(20)			No	None		

Figure 4.14: Structure of history table

4.2.4 Face Recognition

According to source [5], there are 3 steps to face recognition: 1) Face Detection, 2) Posing and projecting faces, 3) Encoding faces.

1) Encode a picture using the HOG algorithm to create a simplified version of the image. Using this simplified image, find the part of the image that most looks like a generic HOG encoding of a face.

2) Figure out the pose of the face by finding the main landmarks in the face. Once we find those landmarks, use them to warp the image so that the eyes and mouth are centered.

3) Pass the centered face image through a neural network that knows how to measure features of the face. Save those 128 measurements.

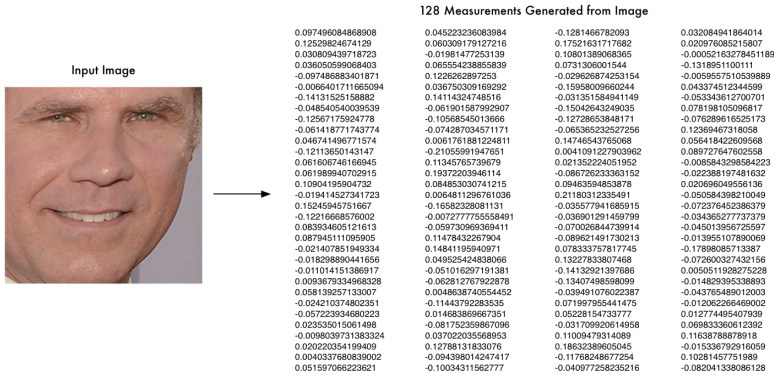

Figure 4.15: Input image and the measurements generated by the algorithm(Source: [5])

5. Future Developments

This section proposed possible improvements to our FacEntry App:

- The app can utilise BlueTrace protocol for contact tracing
- Interface can be improved by having photo of face automatically taken upon face detection
- There can be a separate account for administrators to check the attendance of users
- The app can use JSON data interchange format
- The history page can return a list view instead of text
- More work can be done to find a faster algorithm to verify the face of the user

6. Conclusion

6.1 Reflections and Learning points

We learned that teamwork was very important, as each of us had our own roles and responsibilities, and all of us needed to fulfill it in order to complete the project. Good communication was also essential, as there was a tight timeline and each of us needed to be clear of our roles and deadlines for the process to be efficient.

6.2 Summary

We have successfully managed to create an attendance taking app which allows the user to sign in to locations using Face Recognition technology and Artificial Intelligence. The app is able to transmit the data of the user, such as their name and BC. number, face, as well as their past visited locations to the database and store it. This app is useful for contact-tracing and attendance taking.

Bibliography

[1] “COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University”, https://github.com/CSSEGISandData/COVID-19, 2021

[2] “SafeEntry”(2020-09-06), https://www.safeentry.gov.sg/, 2021

[3] Baharudin, Hariz (2020-04-10), "Coronavirus: S'pore contact tracing app now open-sourced, 1 in 5 here have downloaded", The Straits Times, 2021

[4] “XS - ADATIS”, https://adatis.com/en/products/xs/, 2021

[5] Adam Geitgey(2016-06-24), “Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning”, https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78, 2021