A Mobile Application to Assist Alzheimer's Patients and Caregivers

By

Keh Yi Qian

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

June 2025

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my gratitude and appreciation to my supervisor, Dr Kh'ng Xin Yi, who provided her invaluable suggestions, guidance, and comments throughout the Final Year Project. The ideas given by her in every supervisory meeting led me to complete this project and motivated me greatly. Dr Kh'ng is very dedicated and responsible for her duty as a supervisor; she would correct me when I was wrong and guide me when I needed help.

Also, I wish to thank my friends who were willing to hear my problems and provide ideas for me; both support and encouragement gave me the power to complete the entire project. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

Alzheimer's disease (AD) is a progressive neurological disorder that severely impacts memory, cognitive functions, and daily living abilities, posing significant challenges not only for patients but also for caregivers and society. With no known cure, there is a critical need for supportive solutions to alleviate these difficulties. This project aims to develop a mobile application that assists Alzheimer's patients and their caregivers. Therefore, MemoraCare was developed, and it is a Flutter-based mobile application that helps caregivers and patients stay connected, safe, and recognized in real time by integrating functionality such as real-time face detection and recognition, safe-distance navigation, communication channels, AI assistance, diaries, and NFC info cards, with Firebase as a backend service. There are three models, which are the custom CNN model, Siamese Network, and MobileFaceNet.. The fine-tuned MobileFaceNet achieved 80.5% testing accuracy, which clearly outperformed the other models and emerged as the most suitable for this project. Compared to the custom CNN, which suffered from overfitting, and the Siamese network yielded modest results. Firstly, the app delivers an end-to-end, on-device face pipeline designed specifically for caregiver–patient workflows: a lightweight embedding-based recognizer by using the MobileFaceNet to enable open-set identification, indicating that new people can be added without retraining, while the ML Kit is used for face detection. Secondly, by coupling recognition with a person-linked memory repository, MemoraCare enables the recall of the patient's memory. Thirdly, the app provides a safety stack combining geolocation by allowing the caregiver to set a customized safe zone, Haversine distance for geofencing to calculate the shortest distance between the patients and caregiver, and walking-route retrieval to guide the patient return to the designated locations for real-world caregiving scenarios by integrating the Google Cloud APIs: Maps API, Directions API, and Places API to display the locations on the Google Maps, suggest a route between two points, and search for places. Fourthly, the app increases the relationship between the patient and caregivers by integrating a communication chat room that employs deterministic pairwise chat IDs and voice-note peak compaction for achieving 1 to 1 chat room for the users and supports voice messaging. Also, a role-conditioned OpenAI assistant that is driven by carefully designed prompts to respond

with an expected reply. Fifthly, the app writes a minimal public-profile URL to an NFC NTAG213 tag to enable tap-to-view access so bystanders can quickly contact caregivers if the patient becomes lost. Sixthly, the app encourages the patient to write and record a diary about their valuable and interesting memories by storing it in the cloud instead of their brain, so that they can review and recall the memories if needed. Hence, the application attempts to enhance daily living, promote memory recall, improve communication, ensure patient safety, and reduce caregiver burden.

Area of Study: Deep Learning, Mobile and Embedded Model Systems

Keywords: Flutter-based Mobile Application Development, Real-time Face Recognition, MobileFaceNet Model, Firebase, Alzheimer's disease, OpenAI, Google Cloud APIs, NFC, Extreme Programming

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

@	at
#	Octothorpe

LIST OF ABBREVIATIONS

AD	Alzheimer's Disease
MCI	Mild Cognitive Impairment
YOLO	You Only Look Once
OpenCV	Open Source Computer Vision Library
GPS	Global Positioning System
LPI	Lumosity Performance Index
NMS	Non-Maximum Suppression
mAP	Mean Average Precision
IDE	Integrated Development Environment
CNN	Convolutional neural networks
MVC	Model-View-Controller

CHAPTER 1 Introduction

1.1 Background Information

Alzheimer's disease (AD) is a degenerative disease of the brain, and it is also an irreversible, progressive brain disorder, meaning it will gradually destroy the patient's memory, thinking skills, and cognitive ability, eventually resulting in loss of the ability to perform simple tasks in daily lives even cannot maintain their lives independently if lack the assistance of the caregivers. The disease was discovered more than 100 years ago [1]. Unfortunately, no interventions, treatments, or medicines can be used to cure and reverse AD completely. Aging is a process that we must experience in our lives; the probability of suffering from AD will increase with age. Almost all Alzheimer's patients have Alzheimer's symptoms around the age of 60, with the first appearing [2]. Disease is the common cause of dementia in older people. It is a huge health challenge faced in the medical field, and many scientists and therapists are making efforts to overcome it. This is because it will affect not only the patient himself but also the families, communities, and even the government.

However, until now, scientists have not fully understood what Alzheimer’s disease causes among people, with sufficient evidence and research. The causes are most likely a combination of brain changes with age, as well as genetic, environmental, and acquired factors. But the most important thing is that any one of these factors in raising or lowering the risk of suffering AD depends on different people from person to person. First, the factor can be a genetic risk factor, which may be caused by the family history of dementia. Research [2] suggests that a family history of dementia is a risk factor for the development of Alzheimer's disease, depending on the type of family link, the age of onset, and race. If the person has a first-degree relative with the AD patient, the risk will increase by around 10% to 30%. Furthermore, acquired factors also factor in developing AD. These acquired factors include hypertension, lipoproteins, cerebrovascular disease, altered glucose metabolism, and brain trauma, which appear to be more relevant in middle life [2]. Therefore, the most important thing is probably to manage these factors in or before middle life to reach the purpose of decreasing the risk, progression, and severity of suffering from AD. Lastly, the factor is the

environmental factor; the environments and toxic exposures are the potential risk factors that need to be investigated [2]. Environmental risk factors include secondhand smoke, air pollution, and pesticides.

Memory problems are one of the preliminary symptoms related to Alzheimer's disease. Some of the people who have memory problems normally suffer from mild cognitive impairment (MCI) to a certain degree [1]. Older people who suffer from the MIC have a higher risk of suffering from Alzheimer's disease, but not everyone does. There are three stages for the disease, which are mild, moderate, and severe. In the mild stage, the patient will experience mild memory loss and other cognitive troubles that can affect their ability to function independently, where symptoms become noticeable enough to influence daily life, personality, and behaviors. Although individuals still function independently, they start to show noticeable signs of cognitive decline, and this may gradually worsen over time. In the middle stage, the area of controlling language, reasoning, sensory processing, and conscious thought will be damaged [1], and memory loss and confusion will become more severe. The patients will find it hard to recognize their familiar people, such as relatives, families, friends, and so on; they have difficulty learning new things and memorizing them in their minds, have to perform multi-step tasks, and lag in response. People at this stage may also exhibit behavior such as hallucinations, delusions, and paranoia, as well as exhibiting impulsive [1]. Last is the severe stage, where the brain of the patients will shrink obviously, and the brain will be covered with plaques and tangles [1]. The patients cannot communicate with others, cannot maintain their daily lives, and are completely dependent on caregivers for care. Ultimately, most of the time they will be in bed. In summary, one can observe that the main symptoms of AD are memory loss, cognitive difficulty, and changes in behavior.

With the gradual advancement and development of our era, almost all things have evolved; one of the things that has led to the forefront of the era is technology. Many technologies have been improved over time in many areas. Developing a mobile application is supposed to be a not-bad way to face the problems brought on by Alzheimer's disease. In this era, almost all people have a mobile, even more than one, so the development of mobile applications can assist patients' and caregivers' daily lives. The application can help the patients by using games to train their brains, fostering

communication between patients and caregivers, face detection and recognition, GPS tracking, and so on, while also helping the caregivers care for the patients in easy and convenient ways. The additional features can also be developed for the application if needed in the future. Therefore, the project aims to develop a mobile application that can help AD patients and caregivers improve or maintain their daily lives and expects to enhance or maintain patients' cognitive or other areas' ability and delay the time of the stage of AD moving from the current stage to the next stage to a certain degree. However, the project is not expected to treat AD completely because it is not possible to reverse AD completely through a mobile application.

1.2 Problem Statement and Motivation

Alzheimer's disease (AD) is a progressive, degenerative brain disorder that impairs memory, cognitive function, and behavior, eventually leading to the inability of patients to live independently without constant support. As AD seriously develops to impact both patients and caregivers, there is an increasing need for accessible, holistic technological solutions to assist them in covering daily tasks. However, this kind of mobile application is quite rare in the community, and current mobile applications in this domain tend to focus on only one group, which is either the patient or the caregiver, resulting in inefficiencies and gaps in care. Furthermore, some mobile applications always require users to pay some money as a premium to obtain the full or advanced functions, features, and settings. Although this is the channel through which the company gains profit, it may become a burden to some poor families. Moreover, the memory loss and cognitive difficulty symptoms from AD patients prevent them from using the more complex applications without the streamlined navigation and step-by-step guidelines. Therefore, mobile applications are supposed to be designed for easy and convenient use.

A major symptom of Alzheimer's is memory loss, which is not adequately addressed in the world. Most mobile applications focus on training the patient's brain by utilizing games. However, they do not pay more attention to another one of the main symptoms, which is memory loss, that will cause the patients to forget themselves and their relatives, families, and friends. The patients will forget or cannot clearly memorize the

related information about their familiar person, such as names, faces, and valuable memories. This neglect contributes to patient confusion and distress.

Simultaneously, caregivers face high levels of emotional and physical stress because they need to juggle constant attention, monitoring, and support duties. The burden of the caregivers may be heavy and stressful when caring for an AD patient; they need to care carefully for the patients and always pay attention to them. Caregivers often experience high-impact levels of emotional, physical, and mental strain while caring for Alzheimer's patients who require continuous attention and support. However, mobile applications are typically designed for patients, not caregivers. So, how can caregivers care for patients if they are always under high stress? Communication is also a crucial factor that can establish a friendly relationship between the patients and caregivers, so that they can communicate well. Unfortunately, most mobile applications ignore communication between patients and caregivers, which may cause the caregivers to be unable to follow up with the patients in time.

The motivation for developing a mobile application specifically tailored for Alzheimer's disease patients and their caregivers is essential because of the challenges faced by both parties due to the progressive characteristics of the disease. The application aims to use existing Face detection and recognition algorithms, such as ML Kit, MobileFaceNet, deep-learning-based algorithms, and any efficient algorithms for developing the application. The application was proposed in my project to have main functions such as real-time face detection and recognition, safe-distance navigation, communication channels, AI assistance, diaries, and NFC info cards, with Firebase as a backend service. The development of such applications for patients and caregivers should be encouraged so that the daily lives of the patients and the caregivers can be improved and reduce the burden on the caregivers.

The development of the applications is essential because Alzheimer’s disease is gradually increasing. If Alzheimer's disease is not controlled, the overall burden will be enormous with an increasing prevalence among the aging population. In 2016, research [2] indicated that an estimated 47.5 million people suffer from AD globally, with the disease burden in low and middle-income countries accounting for 62%, where the country has the opportunity to get social protection, services, support, and care much

less. The number of individuals suffering from AD in 2016 is already quite high. How to imagine the number of AD patients until now? Statistics [3] have predicted that the number will almost double every 20 years; it is estimated that the number will reach 78 million in 2030, and 139 million in 2050. This is an alarming number! Therefore, if we can delay the prevalence of AD or postpone the AD becoming more severe, this will be a step toward reducing the burden of AD. Not only that, but A. P. Leone also [11] stated that around 60% of patients with dementia, including AD, will experience wandering, which most commonly happens when they are in the middle or last stages of dementia. Although the application cannot inhibit the wandering of the patients, it can monitor the location on the smartphone to track patients' location and increase the probability of finding those patients back to avoid the wandering.

1.3 Project Objectives

1.3.1 To Develop a Mobile Application Designed to Assist Alzheimer's Patients and Their Caregivers

The application is supposed to provide functions, features, and services that can help improve or make both patients' and caregivers' daily lives easier. The app is supposed to implement face recognition so that it can recognize the familiar people of the patients, such as their relatives, friends, colleagues, and so on, and connect to certain memories so that the patient can recall the related memories about the recognized person. The people's enrollment is also needed so that the patient can add the new person to the app and recognize them. The enrolment should not only store personal information about the person but also require the recording and storing of some meaningful and valuable information related to the patient, so that they can recall the memories when the patients review their memories. The management of the enrolled people also needs to be included in the app to allow the patient to manage their memories. Not only can face recognition and people management can recall the memories from the patients, but also diary features should be designed to recall the patient's memories. The diary feature is supposed to enable the patient to record the incident that they feel is meaningful or valuable. This will encourage the patients to write out their memories and remember using modern technologies such as the cloud instead of their brains, ensuring the patients can review the matters even if they forgot. Moreover, the safety problem should

also be considered in this project. The feature, such as GPS tracking, can be developed by using the Maps API to track and monitor the patient's location to avoid the wandering-related problem. The return route feature that displays the route for the patients and the designated places can also help to improve the safety of the patients by providing the route for the patient to return, which may be useful during the early stage of Alzheimer's disease. But, this may be useless when the middle or severe stage of AD, because the memory and cognition of patients have been seriously affected. Therefore, the information card feature that can write the NDEF URL to the NFC tag can help in this case. The caregiver can write the URL to the NFC tag that will display the patient's personal information and emergency contact on the web, then the bystander who encounters the patient wandering can contact the emergency contact based on the information card that is published on the web. Lastly, the app is expected to support a communication channel that can promote and encourage the communication and information exchanged between the patients and caregivers, so that the information between both parties can be followed up on and maintain the relationship.

1.3.2 To Implement Real-Time Facial Detection and Recognition to Recognize Familiar People of Alzheimer's Patients

The project aims to develop a mobile application for Alzheimer's patients that integrates efficient human detection and recognition capabilities. It enables real-time facial recognition using various algorithms to recognize the face and show the related information, so that it can trigger memory associations of patients with familiar people. The purpose of this function is to hope that it can awaken the memories of the patients when the output of the recognition is shown. The functions try to use the information stored in the application to arouse the memories that the patients cannot remember. Furthermore, the application implements a face detection and recognition system that utilizes the mobile phone's camera to detect and identify individuals in real time. Models such as MobileFaceNet and the ML Kit Flutter package will be used in this project to support real-time face detection and recognition. The models are also expected to recognize the newly enrolled people that are stored in memory by the application's users without retraining the model.

1.3.3 To Establish Communication between Alzheimer's Patients and Caregivers and Integrate with AI-driven Assistance

The application aims to alleviate the stress and burden of caring for the patient on the caregivers. The proposed mobile application provides tools and features that simplify caregiving tasks and improve efficiency between the patients and caregivers, such as voice assistants or other AI-driven assistants. The integration with AI-driven assistants to accompany the patients in chatting reduces the feelings of loneliness of the patients because people in this era are always busy working. Therefore, caregivers can be more powerful and energetic in taking care of patients. The communication channel can also be integrated into the proposed application to solve communication problems among the patients and caregivers. Therefore, caregivers can follow up with the patients and deal with the problems on time. Patients can reach out to caregivers whenever and wherever needed. The use of the communication channel in the application can be safe and prevent personal data leaks compared to other social media platforms.

1.4 Project Scope

The delivery of the project is a Flutter-based mobile application that can assist Alzheimer's patients and caregivers in their daily lives. The main functions of the application are expected to be Face detection and recognition, people enrollment, Gallery, communication channels, GPS tracking, navigation route, diary, and information card establishment.

Within scope are on-device face detection by using Google ML Kit and face recognition using a fine-tuned MobileFaceNet pipeline that crops, normalizes, embeds, and stores features in Firebase during a guided enrollment, and links the stored memories so that when the app recognizes the person, the patient can recall the related memory from the app. Therefore, the app is supposed to recognize the face that is stored in the Firebase. Not only that, but the app also supports real-time recognition by capturing the frame from the camera. Furthermore, the app supports the enrollment of the person and their face with the memories, so that the new person can be added into the Firebase and achieve the recognition without retraining the model. The Gallery module is developed to manage those enrolled people in the Firebase via CRUD actions. The patient can

review the stored valuable memories and personal information, which may delay their memory degeneration, or update the latest memories and information about the person. It also provides a seamless communication channel between patients and caregivers, including text and voice messaging, attaching media, and OpenAI-driven assistants. This will ensure the connection between both parties and facilitate timely support.

Safety features include live GPS tracking with two modes: first is tracking the locations between the caregiver and patients, and the second is tracking the locations between the fixed location and patients, which support the settings such as adjustable safe-distance radius, audible alarms, an emergency countdown with auto call to prioritized contacts so that the caregiver can receive the alert notifications if the patients out of the range of sent safe zone or avoid the emergency scenario such as patients wander. Also, a "Return Route" that draws a walking path and ETA via Google Directions is also a safety feature that will guide the patients back to the designated destination by providing the route. Moreover, the diary feature is developed to enable patients to record the meaningful and valuable incidents or memories of daily happenings, so that the patients can recall the memories via the diary. This is also a feature of the app that tries to delay the memory degeneration in the patient and encourages the patients to write their memories out, not only to remember them in the brain. Lastly, the information card features that support the NFC writing allow the app to write a minimal, public URL to tags and sync a privacy-filtered profile, such as name, phone number, address, and emergency details, to a web endpoint, while keeping a richer private card in Firestore. If it is said that the GPS tracking is used to avoid the wandering of the patients, then the publishing of NFC URL is the solution to overcome the wandering because it is most useful, and the bystander can scan the NFC tag to contact the emergency contact of the patient.

The project is expected to be done individually within the given limited time and resources, such as time, cost, and knowledge, are needed to focus on this project. The stages, such as planning, designing, developing, testing, deploying, and launching, are estimated to be completed within 2 Semesters, and the development attempts to develop the application iteratively by adding new features. Therefore, the application will become more powerful and useful after each iteration.

1.5 Impact, Significance, and Contribution

The significance of this project lies in its holistic and dual-target design, which equally prioritizes the needs of both patients and caregivers, a gap that many existing solutions fail to address. With the number of Alzheimer's patients projected to almost double by 2050 [3], the application offers a timely and proactive approach to managing the increasing societal burden of the disease. By promoting memory recall, improving patient safety, supporting communications between patients and caregivers, and decreasing the caregivers' burden, the application not only enhances day-to-day living but also contributes to delaying disease progression. The project's meaningful use of deep learning technologies, including MobileFaceNet, makes it both innovative and socially impactful.

The project aims to improve the quality of life among Alzheimer's disease patients and caregivers in efficient ways by developing a mobile application that combines modern technology with user-centered design. Firstly, the application proposed can help patients recognize familiar people, such as relatives, families, and friends, by using face detection and recognition functions that are integrated into the application. The related information will be shown when the person is recognized, and this may arouse the memories of the patients in the mild or moderate stage. This function provides personalized support to enhance patients' memories of their acknowledged person and important events in their lives. Furthermore, the communication channel built into the application is useful for maintaining the relationship between the patients and caregivers, letting the caregivers understand the requirements of the patients better, and eventually increasing the quality of care. The communication channel is also established with the OpenAI-driven assistant, so the patients and caregivers can also chat with OpenAI. The caregiver can ask for a suggestion when encountering an emergency incident, or the patients can ask what the object is, because the AD will seriously affect the patient's cognitive ability, causing the patient to not know what the object is. Additionally, the Google Cloud APIs, such as Maps API, Directions API, and Places API, are used in the project to support the GPS tracking and return route features to achieve the tracking and monitoring of the locations of the patients to avoid the patients wandering. But what if the patient has already wandered? The app also supports

the feature to publish the personal information and emergency contact of the patients by using the app to write the NDEF URL to the NFC tag, so that the bystander who encounters the wandering patient can contact the emergency contact on the web. Moreover, the application is supposed to have caregiving support features to help reduce the burden on caregivers, as it acknowledges the challenges they face. This caregiver support is an important contribution because it promotes a balanced approach to consider the well-being of both the patients and the caregivers. Lastly, the application is designed with a simple, intuitive interface, streamlined navigation, and step-by-step guidance that is user-friendly for users with cognitive difficulties. This is very crucial to let the patients access digital tools such as mobile phones smoothly without confusion.

CHAPTER 2 Literature Review

In this chapter, we will review the features of the existing mobile applications designed for Alzheimer's patients and caregivers. Additionally, we will explore current mobile applications that focus on human detection and recognition and analyze them. This comprehensive review may help in knowing the fundamentals and referencing the existing related applications for identifying methods or opportunities that are possible to help enhance our proposed mobile application.

2.1 Existing Mobile Applications that Assist Alzheimer's Patients and Caregivers

2.1.1 Constant Therapy

Brief

Constant Therapy is a speech and therapy application designed to help people who have various neurological conditions, such as aphasia, dementia, stroke, and so on by using at-home speech, cognitive, and language therapy. It is available on the Android and IOS platforms for premium after the free trial which is around two weeks. The exercise in Constant Therapy aimed to help practice speech and cognitive recovery and improvement.

Strengths

First, Constant Therapy provides various exercises not only the games to the user, which cover aspects such as memory, analytics, spelling, reading comprehension, speaking, phonological processing, auditory comprehension, attention, arithmetic, and so on. There are more than 60 types with different contents that can train the users' different abilities, especially cognitive ability, speaking, listening, and reading. Besides, Constant Therapy also provides a report about the status and schedule of the users in the exercises. This is the best way to let the user and their Caregivers understand better the situation of the user. The report provides a summary of the exercises done by the users and assigns the level while it also provides the progress report for all tasks that the users have done until the current. The performances of each task have been recorded and done the comparison with the baseline and the latest so that the overviews of the

performance can be seen intuitively and the user can see their improvements. In addition, each exercise has its own levels independently, only the users who achieve some achievements or standard just can upgrade to the next level for that exercise. With the increasing level, the difficulty of the exercise also will rise to become more challenging which can further stimulate the brain of the users and improve their abilities. Furthermore, it provides therapist settings to let the users set the therapist that can help to guide their therapy. Constant Therapy has a clinician version of the application which allows the therapist to build a customized program for clients to practice at home to eliminate the requirements of paper handouts and worksheets. The therapist must install the clinician version of the application so that can establish a connection with the client accounts. It also has some preferences settings provided to the user such as audio settings.

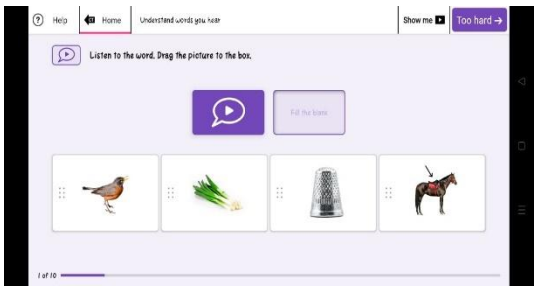


Figure 2.1.1.1 Games of Training Listening Skill

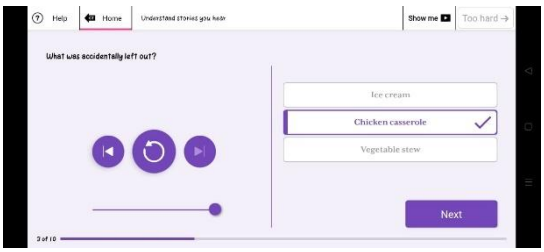


Figure 2.1.1.2 Games of Training Listening Skill

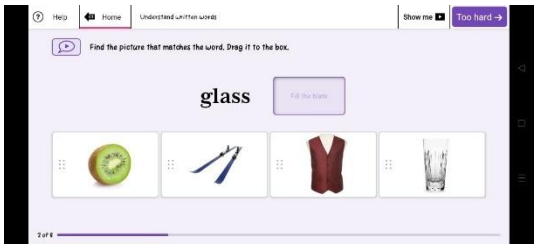


Figure 2.1.1.3 Games of Training Cognitive Ability

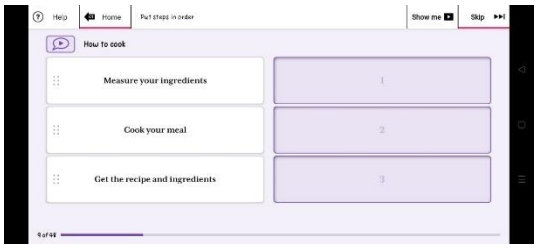


Figure 2.1.1.4 Games of Training Cognitive Ability

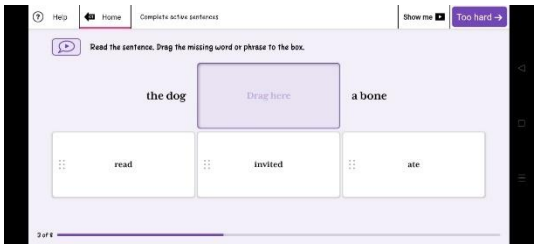


Figure 2.1.1.5 Games of Training Reading Skill

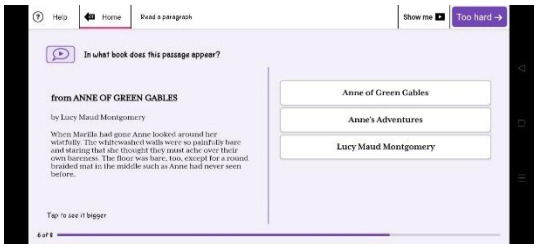


Figure 2.1.1.6 Games of Training Reading Skill

Figure 2.1.1.7 Games of Training Speaking Skill

Figure 2.1.1.8 Games of Training Speaking Skill

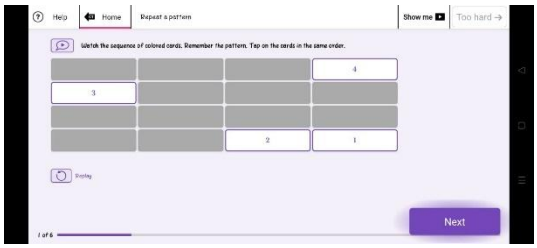


Figure 2.1.1.9 Games of Training Memory

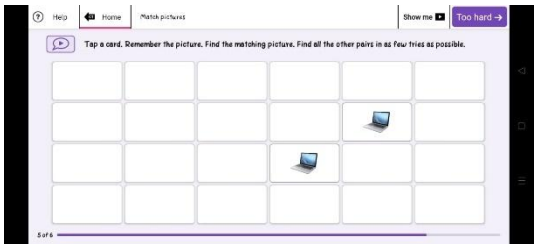


Figure 2.1.1.10 Games of Training Memory

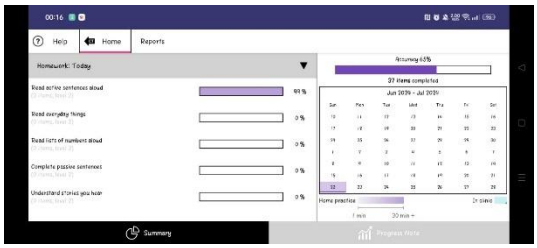


Figure 2.1.1.11 Summary Report of Games

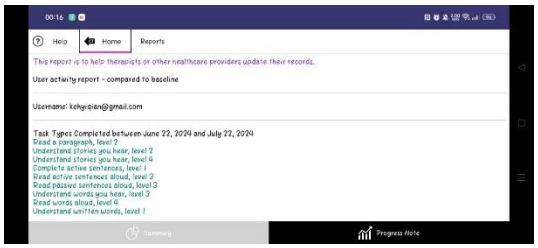


Figure 2.1.1.12 Progress Report of Games

Figure 2.1.1.13 Details of Progress Report

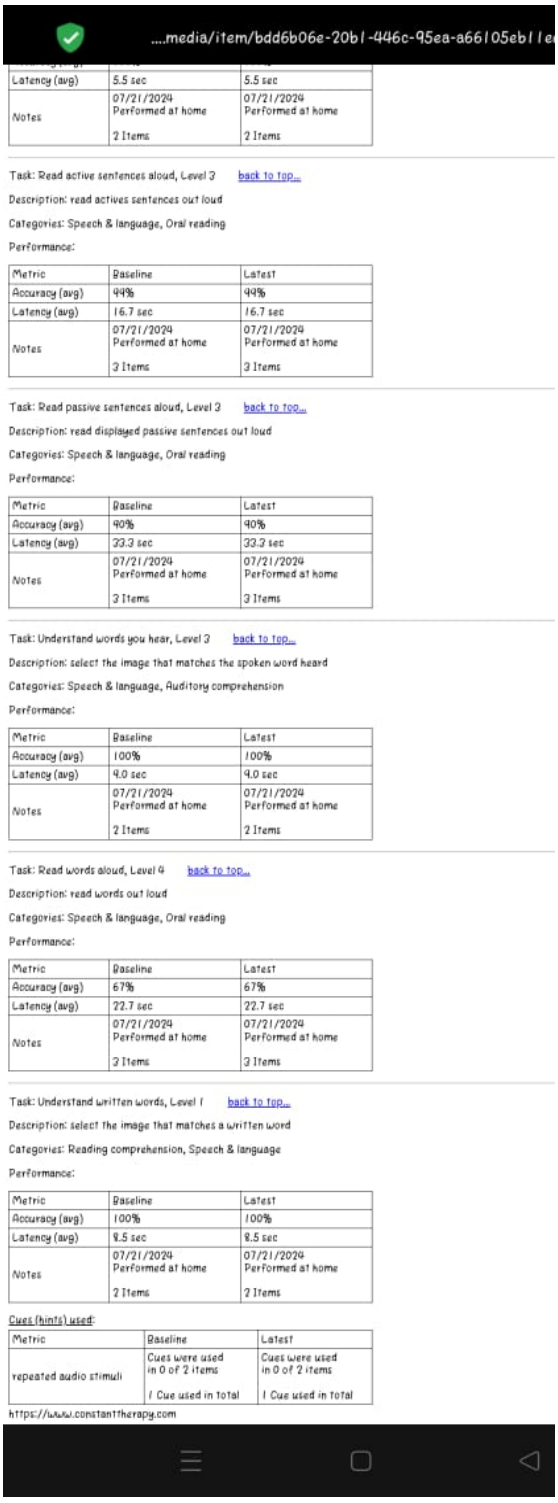


Figure 2.1.1.14 Details of Progress Report

Figure 2.1.1.15 Add Therapist Settings

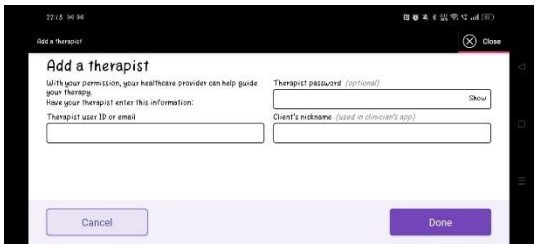


Figure 2.1.1.16 Adding Therapist

C. A. Des Roches et al. [4] demonstrated the research on the effectiveness of language and cognitive skills by using therapy and the positive results with improvements can be seen. The participants were categorized into the control group and the experimental group. The control group only had one hour per week to receive the therapy from the therapist by using Constant Therapy while the experimental group same as the control group but also had practiced with Constant Therapy at home. The evaluation which is Revised-Western Aphasia Battery, Cognitive Linguistic Quick Test, Boston Naming Test, and Pyramids and Palm Trees Test is used to measure whether the participants are improved. The therapy is classified as assess, assign, and then the therapy is individualized so that each patient will receive a different type of therapy based on their language and cognitive ability.

Weaknesses

Nevertheless, Constant Therapy needs a premium, it only provides a 14-day free trial, and after a free trial USD 300 around RM 1331.71 is needed to be paid to proceed to use the application. Therefore, it is unfriendly to the family is poor and becomes a burden to them. Furthermore, internet access is required to use Constant Therapy. This is because it requires an internet connection to load the exercise and sync the data. Users with limited internet access or those who do not have a stable internet connection find it hard to access Constant Therapy. Moreover, Constant Therapy causes a lack of human interaction because cognitive rehabilitation always requires human interaction such as encouragement, guidance, and interaction from the therapist. It does not have a real-life connection with the therapist so the therapist may hard to evaluate the status of the users and give more suitable feedback.

Recommendations

Firstly, Constant Therapy can design some exercises without an internet connection so the users can enjoy the application anywhere and anytime. This is because if not an internet connection the users cannot enter Constant Therapy even the interface of it. After that, I propose to establish a real-life and real-time platform in Constant Therapy that allows the therapist can connect with clients to communicate well and the clients can state their status clearly than the therapist can respond with the feedback at the time. So it can provide a service like a video call end by end among the therapist and users.

2.1.2 Lumosity

Lumosity is an application that offers various brain training games aimed at training the core cognitive abilities of the person in fun and interactive ways. Lumosity is available on Android, IOS platforms and a web interface for free but the premium is required to be paid to unlock some advanced features and the number of games that can be played. The Lumosity games attempt to train the brain in speed, memory, attention, flexibility, problem-solving, languages, and math.

Strengths

Firstly, Lumosity provides flexible and user-friendly interfaces that are easy to access. The website and mobile application use colorful graphics and simple navigations but are attractive. All exercises are designed to seem like games and provide sound effects that maintain the user's attention in the exercise. Lumosity provides a platform consisted a fun context for cognitive training. Secondly, Lumosity provides flexible content that is most appropriate to the users according to the user preferences. It provides the setting for training preferences, the users can arrange and order the cognitive areas and skills that they most likely need to train such as speed, problem-solving, flexibility, attention, and memory. The contents and games will be changed and pop out based on the set orders. Users also can choose the training type which is the normal level or the more challenging one. The contents also will be altered based

on some special situations that are appropriate to some special users such as color-blind and not fluent English speakers. Thirdly, Lumosity provides a sequence of practices of cognitive skills that is more flexible. The step-by-step tutorials are provided to the user when the users are in the initial level or just started to use the Lumosity, users can explore and familiarize the games quickly through these guidelines. The level of the games will keep up and rise if the users perform better performances in the games. The more challenging cognitive practices will be faced by the users! Unfortunately, the game level might fall or remain unchanged if the users' performances are poor so that can ensure the exercise is more suitable and not too difficult for the user in the current situation. Furthermore, Lumosity has provided an online "Brain Performance test" that is quite reliable cognitive performance evaluation, it can capture various cognitive abilities and make sensitive to the changes in cognitive training. This evaluation provides a baseline and it is useful so that the users can track their changing cognitive abilities over time. There is preliminary evidence[5] stating that conducting Lumosity training regularly can impact the cognitive abilities of healthy adults, elderly patients with mild cognitive impairment, individuals who have received chemotherapy, and individuals with cognitive impairment-related diseases. Lastly, Lumosity provides the overview of the scores that the users obtained in various games so that the users can see the changing performance themselves intuitively. The overview of the scores is presented based on an evaluation called the Lumosity Performance Index (LPI) which is a standardized scale by measures all the games' scores[6]. The comparison of strengths and shortcomings in games that challenge different cognitive abilities can be conducted through LPI.

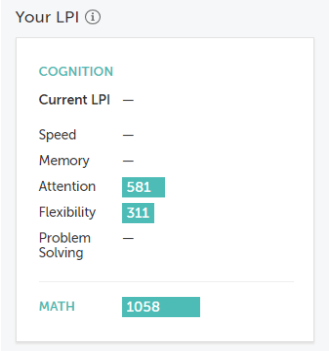


Figure 2.1.2.1 LPI Results

Since Lumosity has consisted of more than 50 games I only use three games as samples for simple briefing. These are some examples of games on Lumosity:

The first game called Word Bubbles attempts to improve the user's verbal fluency which tries to train the user to retrieve the words from their mental vocabulary. The user should try to use the alphabet that the application gives to create the words within the given periods, the longer the words that are created the higher the scores.

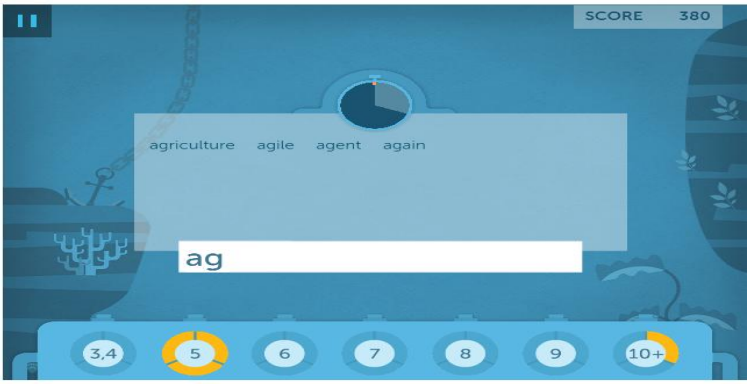


Figure 2.1.2.2 Games in Lumosity

The second game Pet Detective tried to improve the user's problem-solving ability in thinking ahead, evaluating options, and choosing the best course of action. The user needs to deliver the pet to their houses by picking them up and delivering them with a path as short as possible.

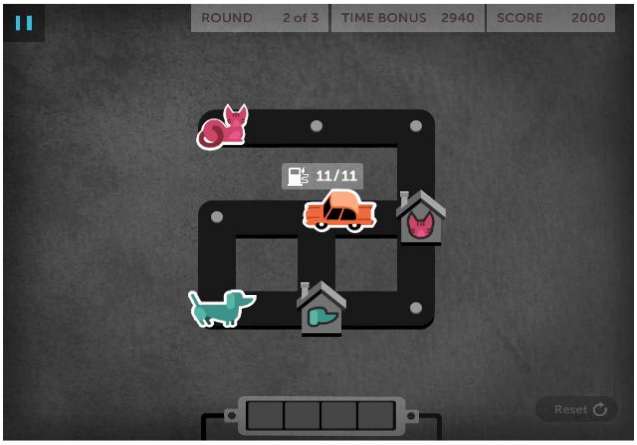


Figure 2.1.2.3 Games in Lumosity

The third game Star Search tried to improve the attention ability of the user through selective attention which needs to focus on the related information meanwhile ignoring the unrelated distractions. The user needs to click the unique objects in the given scene by observing to distinguish the objects.

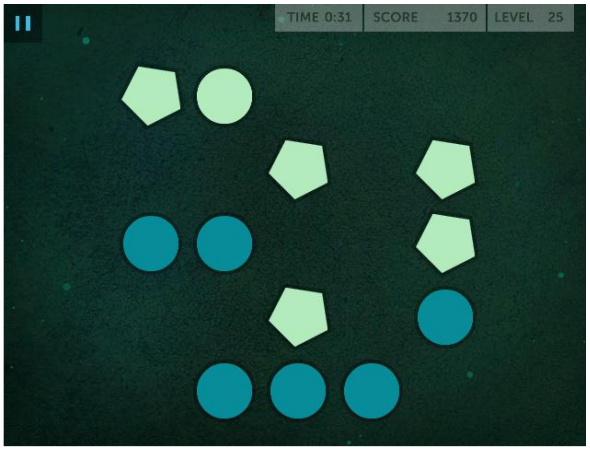


Figure 2.1.2.4 Games in Lumosity

Weaknesses

However, the weakness of Lumosity is also explicitly because the improvements in the cognitive abilities and skills obtained from the game environment may have only small effects on the applicability of cognitive abilities in the real world. K. Bainbridge and R. E. Mayer [7] proposed that Lumosity can improve cognitive skills effectively within the game context, however, the improvements are limited, it may not transfer widely out of the games. These results are based on the experiments that they conducted. In the first experiment, the participants trained with Lumosity for 3 hours spread over four sessions resulting in no strong shreds of evidence and output obviously to prove the positive improvements in cognitive ability. Even though the second experiment conducted a longer period of training which is around an average of 18.5 hours over an average of 18.3 weeks, the results still cannot be proven with strong evidence that the improvements in cognitive ability after using Lumosity. Therefore, the user needs to be alert whether required to put in too much time and money and the user shall persist to use Lumosity in regular times to see the tangible improvements. Secondly, the training of the Lumosity provided is too systematic, and both are based on the games. When

using the Lumosity, It lacks help in real-time from a professional therapist who experienced the training and learned the related knowledge, the more appropriate and proper recommendation can be suggested by the therapist so that may limit the generalization of improving cognitive abilities. Moreover, Lumosity is conducting a study [5] inspecting the ability of it to affect the cognitive abilities of adults with schizophrenia and other mental diseases. Unfortunately, no evidence can support that Lumosity is useful and efficient and can be used to face or change the cognition of patients with moderate to severe neurocognitive impairment.

Recommendations

Games in Lumosity although has various types and all the contents are different, it is still can cause the user to be bored since it may not support the user to play for a long lifetime. If the users wish to improve cognition through Lumosity, then they shall spend too much time in the games, and although more games are provided, what if the users are already done with all the provided games and play many times? How can Lumosity promise that users will use it for a long period? So, I proposed that Lumosity add some games with cultivating lines such as the role can grow day-to-day or give more freedom to the user in the game.

2.1.3 Mindmate

Mindmate is an application aimed to help care for those with Alzheimer's and Dementia even anyone with memory problems meanwhile keeping caregivers sane. Mindmate is only provided on IOS and Desktop/PC platforms. It provides three majority services which are games, workouts, and recipes while in the mobile application, it even offers a daily activities plan.

Strengths

Firstly, Mindmate provides a daily activities plan that allows the user to plan the daily activities so that can keep the mind and body fit. It can remind the users what are the schedules of the activities that had been conducted by he/she on that day. Secondly, Mindmate also provides some games like to other existing applications. They hope that the fun and interactive games can help stimulate the brain and then improve cognition. The games are designed following four majority cognitive areas which are problem-solving, speed, memory, and attention. These areas are great for individuals who have memory problems in priority. Furthermore, Mindmate provides some workout videos and recipes to the user to keep fit although this may not improve cognitive abilities, it helps to enhance the physical and mental health of the users. Besides that, Mindmate is fully free to all users so that the user can enjoy the completed applications without paying any other fees. It is friendly and fair to all people. Last and foremost, Mindmate has opened the clinical trial to the person who is required so that the users can participate in the trial and enjoy the newest and advanced knowledge, tools, medicine, and so on. Although the participants in clinical trials may have obtained a series of advantages and all things enjoyed are the newest, it also has risks because the method or medicine used for treatment consists the unknown effects even if the happens of side effects are quite small. However, it is undeniable that is a good opportunity for the range of people that need this chance.

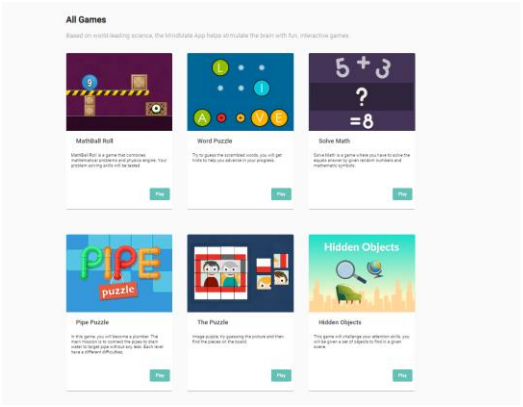


Figure 2.1.3.1 Games in Mindmate

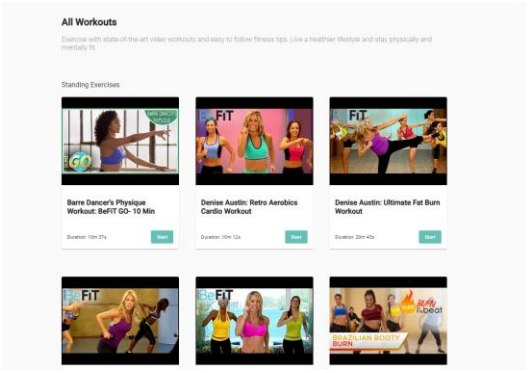


Figure 2.1.3.2 Workouts in Mindmate

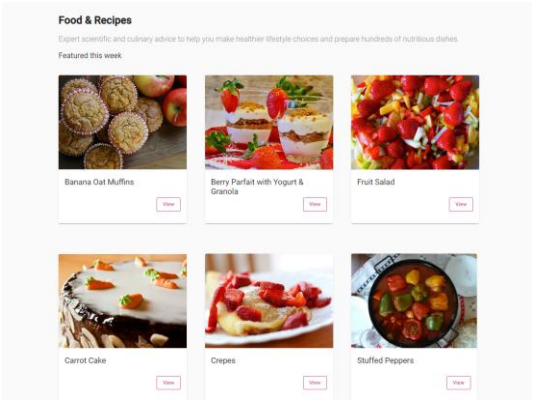


Figure 2.1.3.3 Recipes in Mindmate

C. McGoldrick, S. Crawford, and J. J. Evans [8] experimented with three Alzheimer patients to observe the effectiveness and usability of the Mindmate. There are three patients were recruited for this multiple baseline single case experimental design study. Their daily performance without using Mindmate is recorded by their spouse according to the everyday tasks on a weekly monitoring form during the baseline phase which is the range of five and seven weeks while during the intervention phase, the participants are allowed to use Mindmate last around 5 weeks. Although one participant exited halfway, the other two participants used Mindmate and responded with positive usability ratings and the use of Mindmate is effective and flexible in supporting to remember daily tasks.

Weaknesses

Mindmate is too simple and quite useless to improve the users' cognition. Although it can help the daily life of the user in certain situations, it only makes a bit of function to improve the users, this is because games only the sole way in which Mindmate can be used to improve cognitive abilities, from an individual perspective. However, the games provided by Mindmate are only 6 based on entry with the website! Furthermore, games in Mindmate lack step-by-step tutorials, only simple and quite short guidelines provided on a page and some games do not even provide guidelines! This is quite unfriendly to the person who has memory problems such as Alzheimer's. How do they know the games to play and how do they understand the games? Moreover, the setting on the website is quite simple, with only an option which is to delete the account provided. This is hard to user set their favorite settings based on their situations. Lastly, Mindmate is incompatible with Android, it is only available to IOS and websites resulting in constraining the users of Android use Mindmate.

Recommendations

I consider that Mindmate could add some games to itself in which those games are regarded to gain and improve the person's cognitive abilities. This also can provide more options to the users and let them choose the games that they are interested in while reducing their boredom when using Mindmate. The more the game, the more the aspects that can be covered result in improving the users' cognitive abilities and others efficiently. Secondly, Mindmate shall design the step-by-step tutorial for each game that they provided as clearly as possible to increase the readability and understanding of the games. So, the users can start the games based on the guidelines and understand how to play the games. In addition, I propose increasing some personalized settings to the settings on the website so that the users can adjust the website interface such as adjusting the size of the contents, providing light and dark mode, and so on. These adjustments can establish a quite good user interface to the user and let them have a good experience. Then, I suggest developing an Android platform for Mindmate to let Android users also enjoy this application. This way also helps to promote the Mindmate so that more people have the opportunity to try using this application. Lastly, Mindmate

is too simple and not unique if is designed for those the person who have memory problems, I present that can add other services or features to help train and improve the ability of the patient such as adding relative's pictures into the games, plug-in AI assistant and so on.

2.1.4 Medisafe

Medisafe is an application designed for sending personalized daily medicine reminders and alerts. It makes medicine management easier and ensures that users can take medicine based on their schedules. Medisafe is available on IOS and Android platforms for free for fundamental functions, the extra fee can be paid to obtain extra features and functions. It is commented [9] as one of the top applications in medical management. Therefore, Medisafe can be a good application to improve the daily lives of Alzheimer's patients and caregivers by reminding and monitoring the time for taking medicine so that can ensure the patients will not forget to take medicines thereby the symptoms can be controlled.

Strengths

Firstly, Medisafe can save reminders with more details and be more personalized compared with other applications. The medicines can be added through a search in the Medisafe database or by directly typing in. If the users do not remember or know what the medicines are, can scan and take photos with the medicines then Medisafe will help to detect and recognize the medicines. It does not just record what medicines need to be taken, it also asks users about what form the medicines, the type of measurement of medication, the frequency of taking medicines, how often to take medicine, and the time slot to take the medicines. All of these details can help the Alzheimer's patient understand well how to take the medicines. Secondly, Medisafe provides many medication interaction alerts. This is because some medicines cannot be taken at the same time, there may occur the side effects even threaten life if taken together. Therefore, Medisafe helps to prevent such things happens if the users use it for scheduling to take medicines. Thirdly, Medisafe has a strong feature which is it can send messages to Medfriend if the users miss the time to take medicines. This is the backup if the users miss the time to take the medicines so Medfriend can alert the users

again. Medfriend is the person who shares the data with the patients, and he/she shall install Medisafe and need to sync with the patient's account under an internet connection. Furthermore, Medisafe also provides for recording the appointment with the therapist. It allows the users to add information about the therapist, the medicines that the therapist prescribed, and any appointments with the therapist into the application. This is an effective and strong function when the patients need to take many medicines, and those medicines are prescribed by various therapists. Besides, Medisafe also will generate a report to show the overviews of the taken medicines. Therefore, the users can know how to schedule the period for taking medicines better.

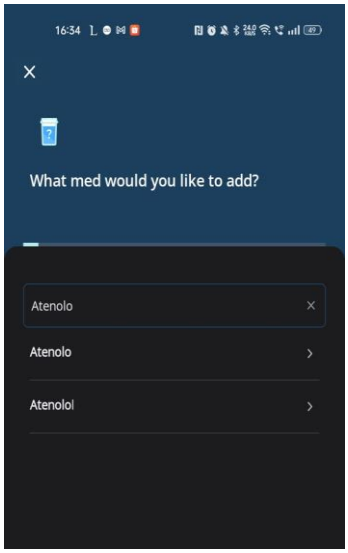


Figure 2.1.4.1 Steps to Adding Medicines

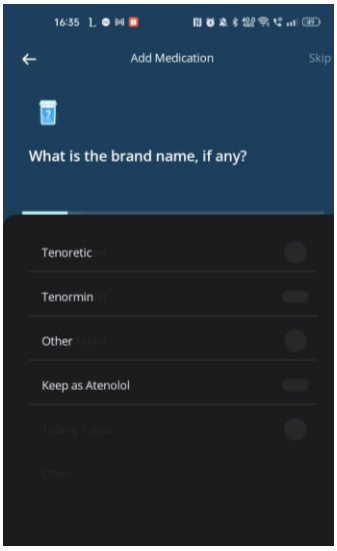


Figure 2.1.4.2 Steps to Adding Medicines

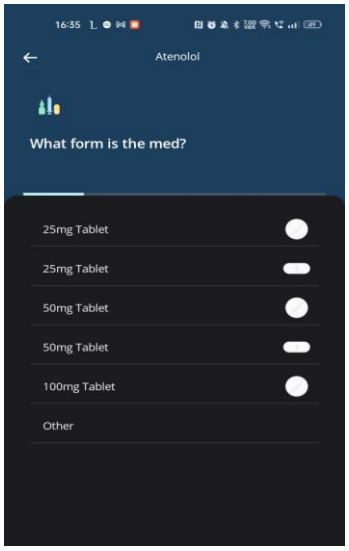


Figure 2.1.4.3 Steps to Adding Medicines

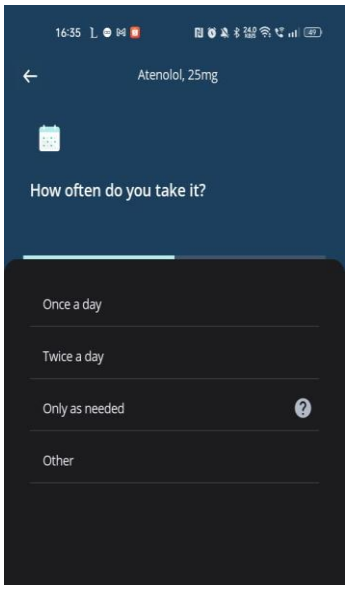


Figure 2.1.4.4 Steps to Adding Medicines

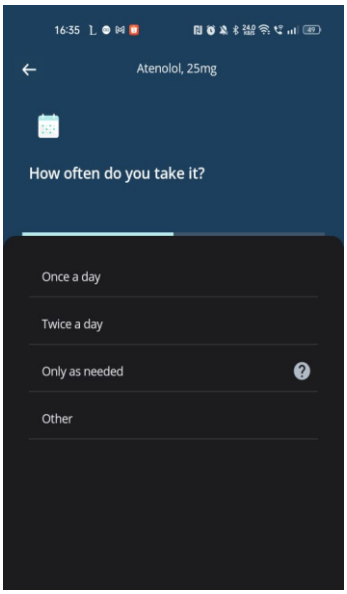


Figure 2.1.4.5 Steps to Adding Medicines

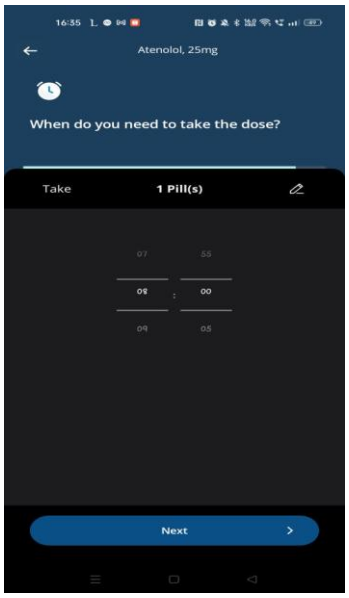


Figure 2.1.4.6 Steps to Adding Medicines

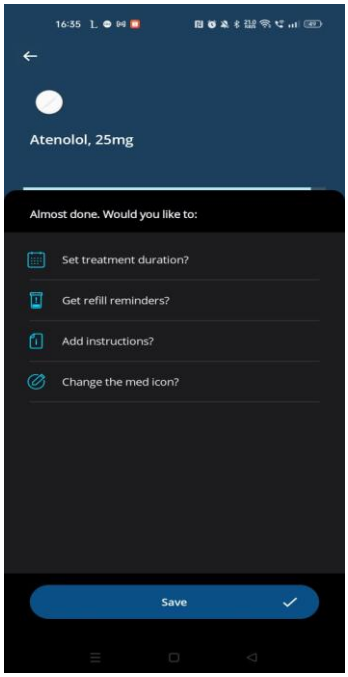


Figure 2.1.4.7 Steps to Adding Medicines

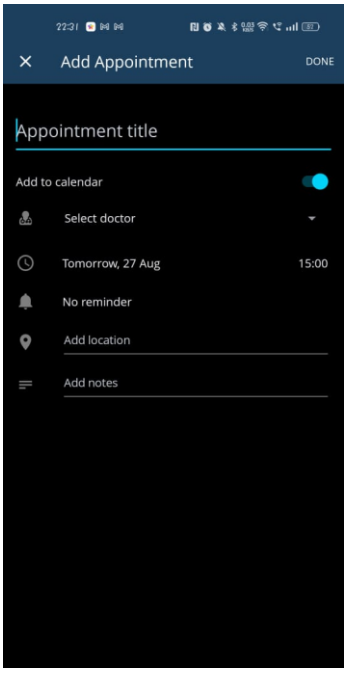


Figure 2.1.4.8 Adding Appointments

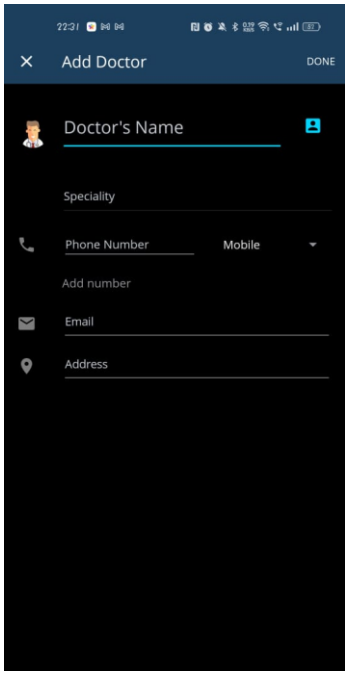


Figure 2.1.4.9 Adding Doctor

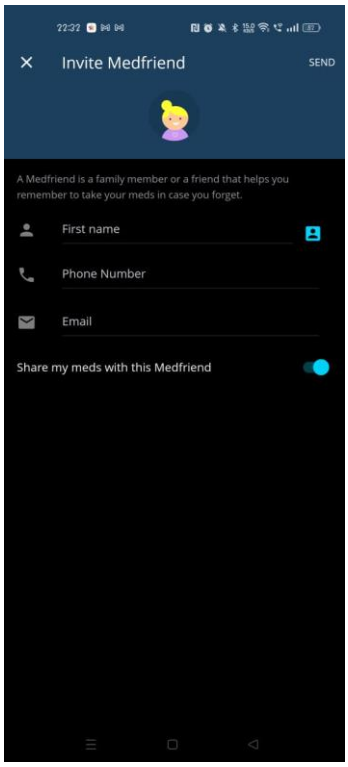


Figure 2.1.4.10 Adding Medfriend

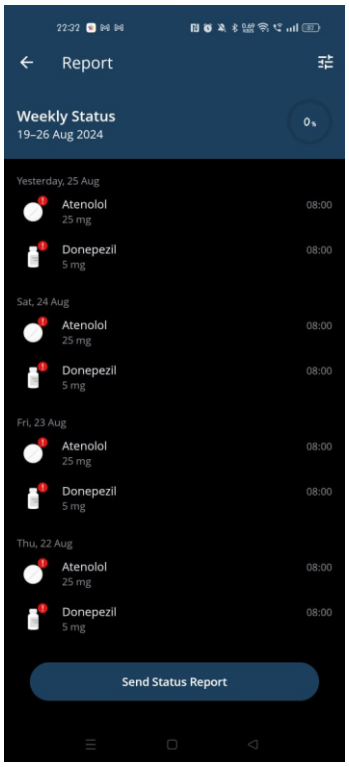


Figure 2.1.4.11 Report Generation

There has been research stating that Medisafe has increased medication adherence [10]. The study included 406 individuals receiving antihypertensive medicine, 150 taking a cholesterol-lowering medication, and a matched control group. After six months and doing the comparison with the test group and control group, found that users with hyperlipidemia had a 10.7% (65.3% with the test group vs. 54.7% with the control group,) lift in medication persistence, and users with hypertension had a 5.4% (69.7% with test group vs. 64.3% with control group) lift in medication persistence. Users with diabetes had a 7.7% (84.6% with the test group vs 76.9% with the control group) lift in medication persistence over three months. The observations prove that Medisafe is useful in reminding users to take medicines.

Weaknesses

However, Medisafe cannot add real medication pictures and create descriptions of the medications independently when adding the medicines. Therefore, it may be a bit confusing to the users who have Alzheimer's disease because they may not remember

what kinds of the medicines they should be taking if many medicines want to be taken at the same time. Furthermore, it lacks a tutorial or presentation about how to use Medisafe. This is because the operations of Medisafe may be hard for some special populations such as Alzheimer's patients or some elderly people, they do not understand how to use it and what the correct things should be typed.

Recommendations

I suggest changing the icon of the medicines to real pictures of the medicines so that the users can classify the medications easily. This also prevents the taking of medicines with mistakes. Besides that, the tutorial with step-by-step may be designed and put into Medisafe so that users can familiarize the procedures quickly and have the best understanding of how to use the application. Furthermore, enhancements to data security and privacy should be performed as strongly as possible to reduce the possibility of leaking data. This is because the data is always shared with others, especially Medfriend, the terms and conditions should be stated clearly, and other security services should be enhanced to protect the personal data well.

2.1.5 Dementia Emergency

Dementia Emergency is an application designed for relatives, caregivers, and emergency workers to teach them how to interact with a person with memory problems such as dementia. The application is available on Android and IOS platforms for fully free. It provides solutions to help people deal with the emergency cases of the person who living with dementia.

Strengths

Dementia Emergency provides clear guidelines and provides many solutions to the relatives, and caregivers to deal with the emergency cases that occur in those people with memory problems such as dementia. There are many aspects covered in this application to teach how to interact well with dementia people in some cases such as communication problems, missing from home, and case-life-threatening life. In an emergency scenario, the patient is likely to be extremely nervous, disoriented, and

doubted. It does not take much to aggravate a problem, causing the patient to become even more afraid, worried, and violent. Therefore, this is the way the guidelines are important so the caregivers can interact with them and console them. Furthermore, Dementia Emergency helps to improve the emergency response for the caregivers because crucial information is provided, and caregivers can follow the information to respond. Moreover, Dementia Emergency allows access without an internet connection. An Internet connection is not required to access the information in the application so caregivers still can get the information without an Internet connection and avoid rely some Internet explorers such as Google Chrome, Firefox, and so on for searching information.

Figure 2.1.5.1 Interface of Dementia Emergency

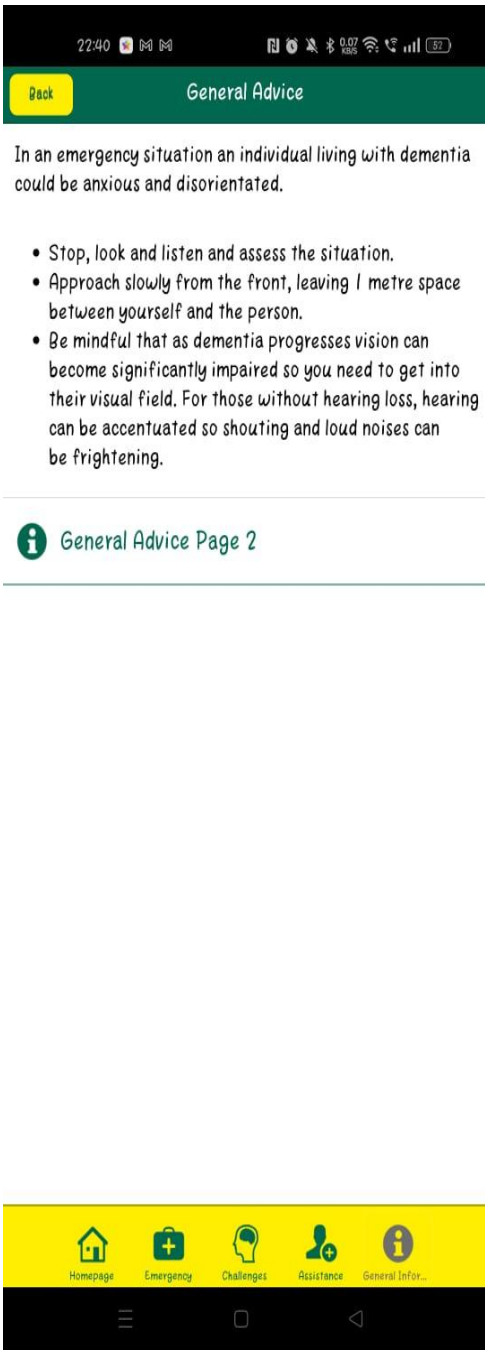


Figure 2.1.5.2 Guidelines on Dementia Emergency for Teaching How to Interact with the Patients

Weaknesses

However, Dementia Emergency is too unitary, with only the information on how to interact with dementia patients and nothing else provided. Furthermore, Dementia Emergency has quite a lot of documentation most are words which makes it too boring when users want to read it. The application with many words will make the users lose interest in it. Besides, there are no settings designed for Dementia Emergency so users cannot change any components in the applications such as language, size of words, and so on. This is quite user-friendly to the users.

Recommendations

Firstly, it can be improved by adding some videos to teach how to interact with such people as dementia patients since the words are quite boring. The addition of videos can make the application more interesting and not just limited in words. Videos also make it easy to access and readable to users so they can learn effectively and quickly. Furthermore, some settings can be added to Dementia Emergency to make it quite personalized such as language, size of words, and so on. The change in languages based on their preferred language can let the users easier to read and understand the information in the application while the adjustment of the size of the words is suitable for people who are short-sighted and presbyopia. Moreover, Dementia Emergency can be enhanced by adding customer service so the users can ask questions to the staff or experts when facing doubts. This is a way to let the users understand the contents of the application in well.

2.1.6 Compare the Existing Applications

Table 2.1.6.1 Comparison Analysis Between 5 Existing Mobile Applications

Services, Features, or Functions Provided	**Existing Mobile Applications**				
	Constant Therapy	**Lumosity**	**Mindmate**	**Medisafe**	**Dementia Emergency**
Designed for Alzheimer's Patient	✓	✓	✓	✓	
Designed for Caregivers					✓
Games Provided and Training Oriented	✓	✓	✓		
Report Generalization	✓	✓		✓	

Help Daily Lives Oriented and Remind-Oriented				✓	✓
Fully Free			✓		✓
Build Connections with Therapists	✓			✓	

Based on the streamlined information in the above table, the comparison of the different existing mobile applications can be observed as straightforward. A total of 5 applications were researched that can assist Alzheimer's patients and caregivers in their lives wherein four applications were designed for patients, and one was designed for caregivers. The table reflects that almost all the applications designed for patients contain games for training purposes. Furthermore, three of five applications provide the report function. This is sufficient to indicate the crucial of the report generation because the report can show the utilization of the applications and can obtain some useful information for analysis. Medisafe and Dementia Emergency are the applications that are more focused on assisting the patients' and caregivers' daily lives as well as the reminder function. Unfortunately, only two applications, which are Mindmate and Dementia Emergency were executed as fully free otherwise need to pay for a subscription to get the full functions. Besides that, some applications such as Constant Therapy and Medisafe are also concerned about the connection and communication among the patients and therapists. In conclusion, many applications provide some functions and features that can help the patients and caregivers but still have additional

features that can be appended to help them as large as possible such as human detection and recognition to help the patients know the specific person, plug-in with AI and so on

2.2 Existing Mobile Applications for Human Detection and Recognition

There are many mobile applications designed for Alzheimer's patients and caregivers, we can observe that almost all are games for training their brains but the improvements in the games are hard to bring into real life and the patients need to spend too much time and effort to focus on the games so that we can conclude that the improvements and assistant bring by those applications is quite negligible and no efficiency. Therefore, human detection and recognition techniques are supposed to be integrated into my proposed application for improving to assist patients and caregivers. The technique can be used to help the patients recognize the visitors, relatives, families, and friends so that the patients can know who is coming and may help in practicing their memory. In order to understand human detection and recognition applications more, some applications are tested and researched with their features.

2.2.1 KBY-AI Face Recognition

Face Recognition is an application that demonstrates the functionalities that can enroll and recognize the person in real-time while also detecting the liveness of the face. This application is offered by KBY-AI and it is fully free. The algorithm used is designed to be lightweight yet highly effective.

Strengths

Face Recognition provides a streamlined user interface to the user while only providing two main features which are enrolled and recognized so that easy and convenient for the user to use. Users only need to enroll the face by inserting that person's face image, the application will automatically enroll the face and assign the label as the name. Then, the user can recognize the face by using the camera, the face is supposed to be detected if the face is already enrolled in the application. Moreover, the application offers face liveness detection to detect whether the face is real to prevent in detecting the pictures online and AI-generated pictures. The threshold value of the liveness is assigned to 0.7

to balance the recall and precision according to this application. It also can adjust the liveness level following the high accuracy and low accuracy options and show the probability of the liveness score to enhance credibility. Furthermore, the application can recognize those faces that are already enrolled, once the face is detected successfully the application performs the comparison with the enrolled pictures and will show the simple output about recognition results. The threshold value of the liveness is assigned to 0.8 to balance the recall and precision according to this application. It also provides the probability of the similarity between the enrolled face and the detected face.

Weakness

Firstly, Face Recognition labels the enrolled face with the format Personxxxxx where the x is the number so that it is not efficient for the recognition. It only can identify whether the face is the enrolled face but it does not recognize who is in detection. The application is supposed to show the name of the face at least. Furthermore, a face can be identified in two different labels, there are two pictures with the same face enrolled with two different labels, and the face can be recognized as one of these two labels in detection sometimes the output is the first label and sometimes is the second label. This may make the users confused about whether the person who has been recognized is correct because the output can be one of two labels.

Recommendations

I proposed to allow the user can rename the label assigned by the application so that the face can be recognized resulting in the output having the name so that the user can know who is detected. This is useful for the Alzheimer's patient to recognize the person and awaken their memories. In addition, I suggested that the application let the user set the label of the enrolled faces with the same name if the faces are the same because the output of the detection face in real-time will have the same name whatever the enrolled face that has been used to identify as long as the face is matched. However, the same pictures are hard to assign with the same name in the normal case so some skills can be used such as storing the same faces in a folder with the name and then the application

can obtain the name of the folder as the output or catch the prefix of the pictures name and take the prefix as the name for output. Besides, the information from the output is quite less, I considered that can store more information for the users such as email, contact number, the relationship between the detected person and user, and so on. This is efficient for the Alzheimer's patient to know the person who is detecting because the application will show more details from that person.

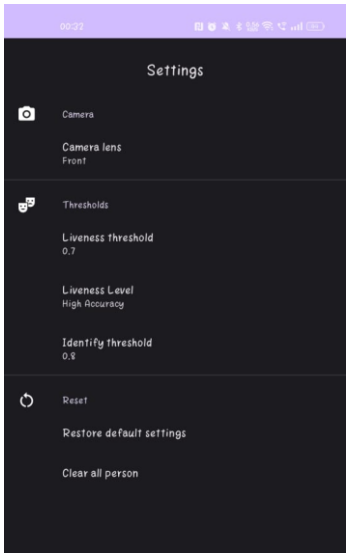


Figure 2.2.1.1 Threshold and Liveness detection are used in the application

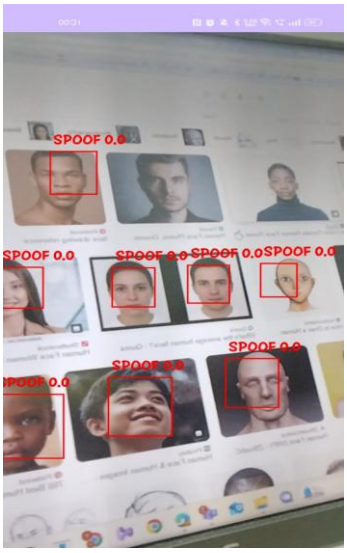


Figure 2.2.1.2 Results of Liveness Detection when the Faces are Spoofed

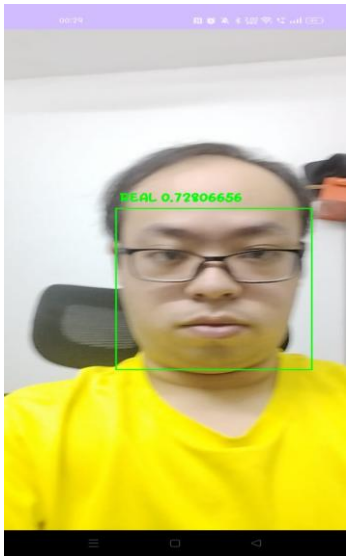


Figure 2.2.1.3 Results of Liveness Detection when the Faces are Real

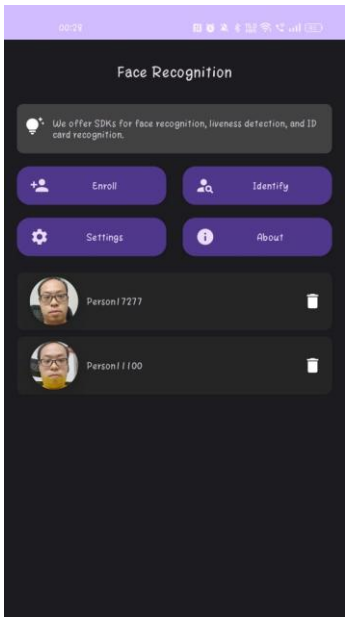


Figure 2.2.1.4 Stored Faces in the Application

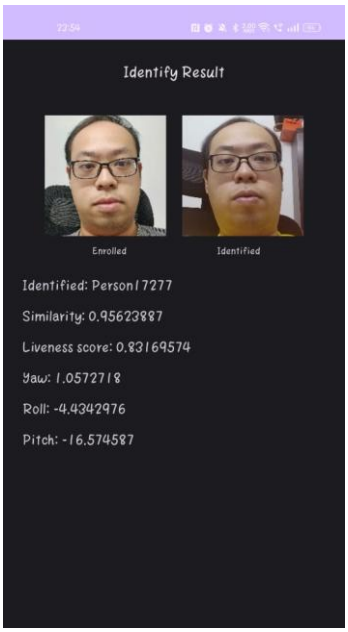


Figure 2.2.1.5 Results of the Recognition

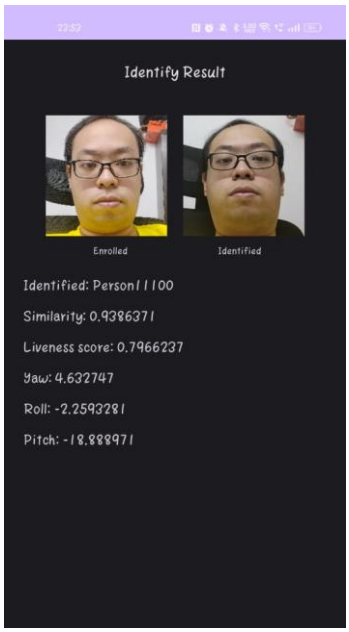


Figure 2.2.1.6 Results of the Recognition, Face is Used Same with Figure 2.2.1.5 but Results of the Label are Different

2.2.2 Luxand Face Recognition

Luxand Face Recognition is an application designed to recognize faces with the user camera and then give their names. This application is fully free. The application just needs any detected face and recognizes it by giving the name.

Strengths

Luxand Face Recognition offers a quite straightforward way to memorize the face. The user just needs to let the application memorize the person who wants to be recognized by taking a camera scan of the face at multiple views. Then, tap to give the face a name and the application will remember the face automatically. The simple and shorter procedures can make Alzheimer's patients easily utilize it in their lives. Furthermore,

the application also provides liveness detection to detect whether the face that is detected is authentic. Moreover, the application also provides a streamlined navigation to the user. For the example, only one step is needed to change the camera lens to either back or front or open the liveness detection while two steps are needed to clear the memories stored in the application.

Weakness

Firstly, Luxand Face Recognition only provides the information which is the name as the output of the recognition because it merely allows the user to insert the name, and the other information is not allowed to be inserted. Therefore, it is hard if the user needs more details to know the person who is recognized. Secondly, the memorization in the application will clear completely if the user wants to remove the specific memory. This is because the sole way the application provides for clearing the memorizes is to delete all the memory in the application without removing the specific memory or name. In addition, the application only serves by providing the name and liveness when detection and no probability of credibility is provided for the recognition and liveness. Unfortunately, it is difficult to let the user believe the reliability of the application. Lastly, the camera cannot detect the side of the face and this can be observed when the camera only takes the side of the face the bounding box does not appear.

Recommendations

I recommended that to allow the user store the different faces independently in different places so that when the user wants to remove the specific face the other faces that are stored will not affected. This can let the user only delete the specific face that he/she wants to delete. Furthermore, I advised the application can store more details of the person who wants to be recognized so that this information can be used as the output of the recognition. The details of the person can be shown when recognizing successfully. Moreover, side face detection techniques such as Haar Cascade classifiers may be used and integrated with the application to detect the side of the face. Therefore, the flexibility and efficiency of the detection can be improved to make the user use the

application easily. Lastly, the application can calculate the probability of face recognition and liveness detection to the user to increase the credibility of the results.

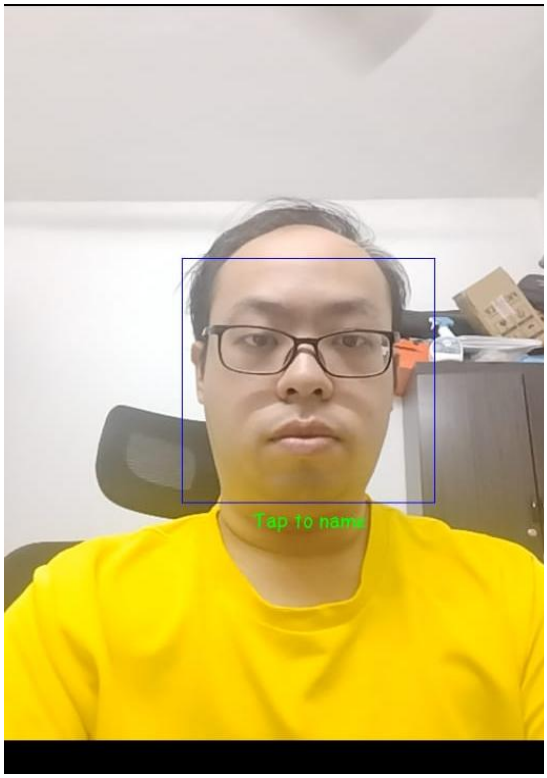


Figure 2.2.2.1 Tracked the Unrecognized Face and Request to Tap the Name

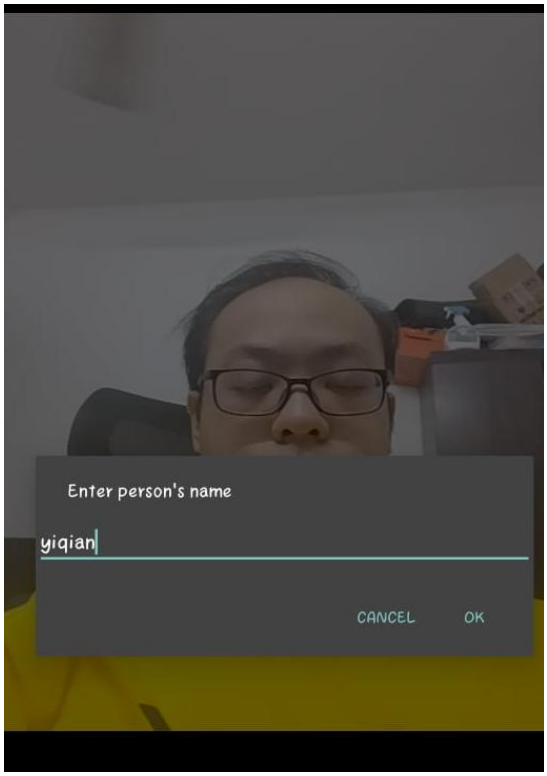


Figure 2.2.2.2 Enter the Name to Store it

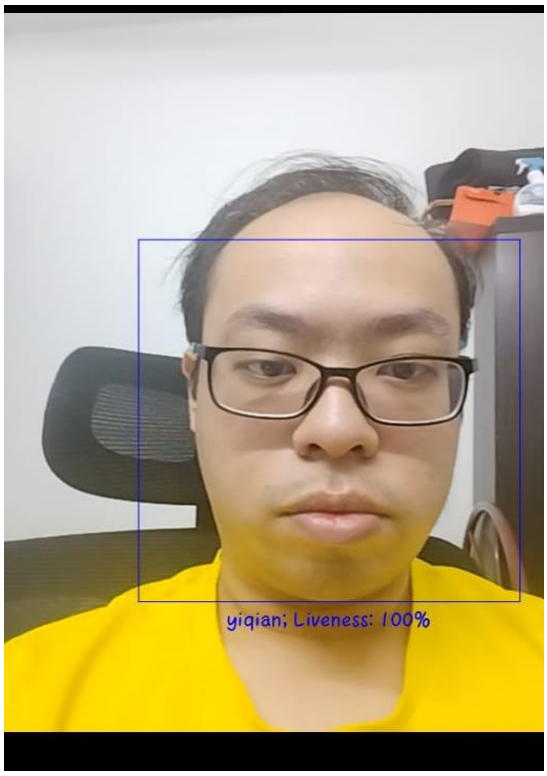


Figure 2.2.2.3 Result of the Recognition when the Face is Recognized

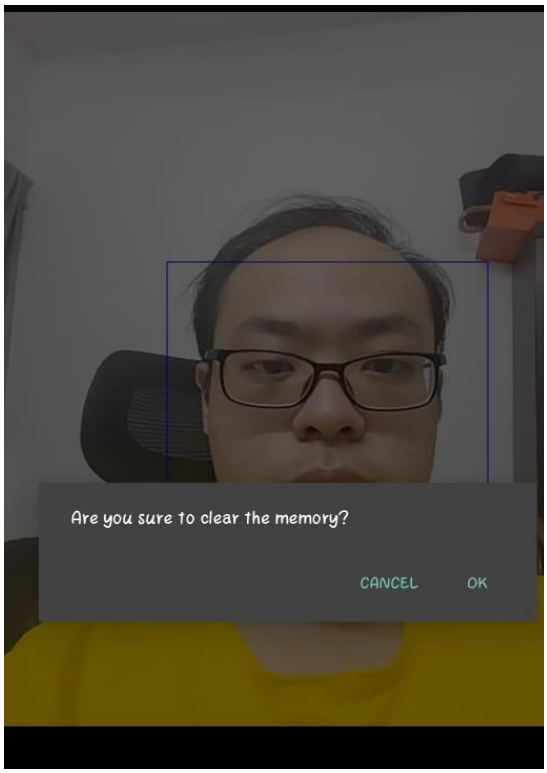


Figure 2.2.2.4 Unable to Choose the Specific Memory to Delete

2.2.3 Comparison of Existing Human Detection and Recognition Applications

Table 2.2.3.1 Comparison Analysis Between KBY-AI Face Recognition and Luxand Face Recognition

Services, Features, or Functions Provided	**Existing Mobile Applications**	
	KBY-AI Face Recognition	**Luxand Face Recognition**
Face Detection and Recognition	✓	✓
Recognize with the Specific Name		✓
Liveness Detection and Real-time Application	✓	✓
Need to Store the Face for Recognition	✓	
Delete the Specific Record	✓	

KBY-AI Face and Luxand Face Recognition are real-time applications that can perform face detection and recognition as well as provide liveness detection. However, Luxand can recognize the face by showing the recorded specific name, whereas KBY-AI cannot. KBY-AI requires the user to insert the image that they want to use to recognize first, and then recognize the face, while Luxand does not require. Furthermore, KBY-AI allows the user to delete the specific face that has already been stored in the memory, but Luxand cannot do such things; if the deletion is needed, the user needs to delete all the records in the memory. A lot of functions can be researched and added to perform the well-being performances compared to these two applications, such as algorithms or methods that can recognize the side of the face.

CHAPTER 3 Methodology

This chapter is crucial, especially in the design phase, as it lets developers demonstrate a clear plan to develop the system. It outlines the design specifications, such as methodologies, tools used, and the system design, such as model design, flow chart diagram, use case diagram, activity diagram, and system architecture pattern. Without these things, the readers cannot understand how the system is executed and how the solution is performed. It reassures that the provided idea is not just theoretical, and also proves the project is doable and can be implemented within the given time and resources. In Chapter1, the problem statement is identified, linking with the proposed solution so that it can assist in addressing the problems.

3.1 Design Specifications

The design specification is essential because it describes how the system should be built, how it should behave, what the requirements of the system are, and what it should achieve. This also helps in making testing and evaluation for the future because it already describes what the system is supposed to do, so that the testing can be easy and fast with this scheme.

3.1.1 Methodologies and General Work Procedures

Software (Application) Development Approach

The project is supposed to use Agile methodologies in the software development which are the programming-centric methodologies. It eliminates the need for much of the modeling, documentation, and time spent when developing the application. This is because it emphasizes simple and iterative application development. Although the traditional Agile methodologies often are team-oriented, there are a few adaptations and practices that can be used and referenced by sole developers. The specific type of Agile methodologies that will be used in this project is Extreme Programming (XP), which is estimated to be quite suited for sole developers.

Extreme Programming supports building the system very quickly and delivering the deliverables sooner because it does not require much documentation, is productive, and focuses more on testing and efficient coding practices. It can also change requirements

at any point during the project's life. The developer will be required to attempt to define all requirements at the beginning (Planning phase) of a project and then expend effort to control changes to the requirements if there are any new requirements needed when developing. In this case, our project is supposed to have 6 phases, which are:

1. Planning

2. Designing

3. Developing

4. Testing

5. Deploying

6. Launching

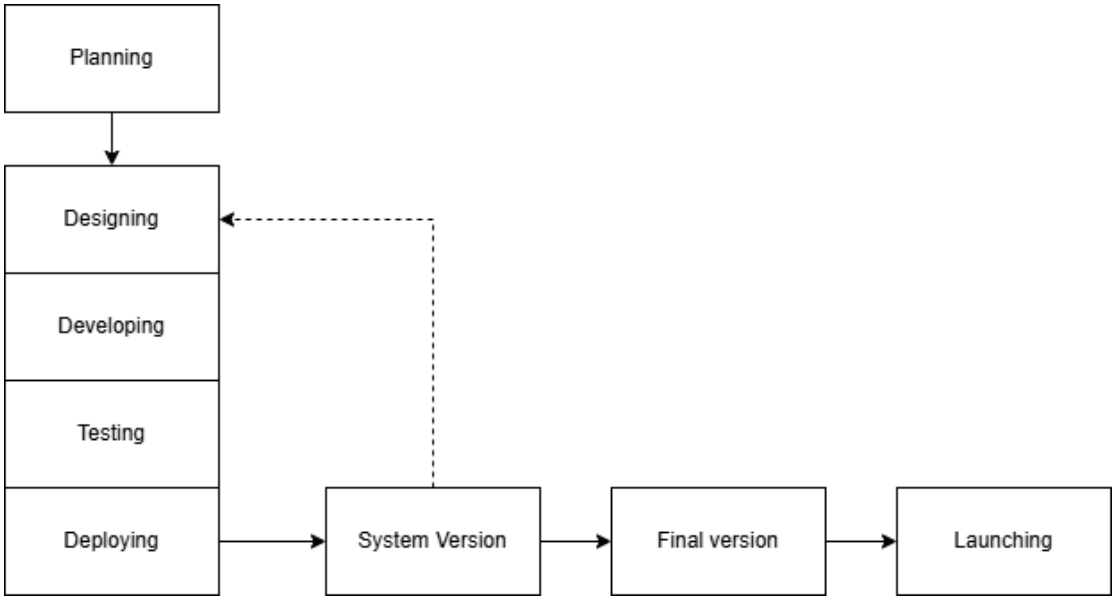


Figure 3.1.1.1 Development Life Cycle of the Proposed Application

Normally, the developer will possess clear and well-defined requirements in the planning phase for the application. After planning, the other 4 phases, which are designing, developing, testing, and deploying, will be conducted iteratively. The

increase of additional features or functions can be added after the end of the current iteration and then start to develop the features and functions with the design phase for the next iteration. Each iteration will take around 2 weeks to complete some specific and small functions so that we can obtain continuous feedback or new requirements.

The project goes through the recycle phase for designing, developing, testing, and deploying to create a newer version of the application so that the functions of the application can be improved and enhanced from version to version, resulting in creating a comprehensive application. The requirements are researched and studied to get a clear understanding of planning the project. After planning, the designing, developing, testing, and deploying processes are supposed to be conducted iteratively to develop the specific functions of the application until the final version of the application is done and can be launched.

Classification Pipeline

In this project, face recognition functionality is being developed by self-training a CNN model, a Siamese Network, and MobileFaceNet so that it can recognize faces based on the provided dataset. The pipeline is divided into two main phases: the training phase and the testing phase. Both models are conducted using almost identical pipelines. The process begins with collecting a dataset that consists of facial images of 32 different people. The dataset is separated into a training dataset and a testing dataset with a ratio of 8:2. In data processing, the training dataset is further split into training data and validation data with a ratio of 8:2. Therefore, the training data occupies around 65%, the testing data occupies 20%, and the validation data occupies around 15%. In the training phase, a Convolutional Neural Network (CNN) architecture is applied to both the training data and validation data to train the model, and the hold-out validation is used in this case. After that, the validation data is used to evaluate the performance of the data. The metrics, such as loss, accuracy, precision, recall, and F1-score, are used to monitor progress and prevent overfitting. Once the model has been trained and validated, it is tested and evaluated using the previously unseen testing data.

3.1.2 Tools to use

Hardware Involved

The hardware involved in this project is a laptop and Android mobile devices. A laptop was issued to develop the applications. At the same time, mobile devices will be used to test and launch the application to ensure the application can be performed on the mobile successfully. However, at least two mobile devices will be used in this project because one is used to log in as a patient and one is used to log in as a caregiver, so the interaction between two different roles can be observed clearly. The NTAG213, which is an NFC tag, is also needed in this project to write and read the NDEF URL.

Table 3.1.2.1 Specifications of Laptop

Description	**Specifications**
Model	Aspire A515-56
Processor	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
Operating System	Windows 11 Home 64-bit
Graphic	Intel® Iris® Xᵉ Graphics
Memory	8 GB DDR4 SDRAM
Storage	512 GB PCI Express SSD

Table 3.1.2.2 Specifications of First Mobile Device

Description	**Specifications**
Model	OPPO Reno8z 5G (CPH2547)
Processor	Qualcomm Snapdragon™ 695 (SDM 695)
Operating System	Android 14
Graphic	Adreno™ 619 @ 840MHz
Memory	16GB LPDDR4X
Storage	256GB UFS2.2

Table 3.1.2.3 Specifications of Second Mobile Device

Description	**Specifications**
Model	OPPO F9 (CPH1823)
Processor	Octa-core
Operating System	Android 10
Graphic	Mali-G72 MP3
Memory	6GB RAM
Storage	64GB eMMC 5.1

Table 3.1.2.4 Specifications of NTag213

Description	**Specifications**
Standard	ISO/IEC 14443 Type A, NFC Forum Type 2 Tag
Frequency	13.56 MHz
Memory (EEPROM)	144 bytes of user memory
Size	21*11mm
Security	32-bit password + 16-bit PACK, and lock bits
Extra Features	24-bit counter, UID & ASCII mirroring, fast read
Endurance	100,000 times of reading and writing
Data Retention	Around 10 years
Operating Temp.	–25 °C to +70 °C
Operating Range	Up to 100 mm (typically 2–5 cm with smartphones)

Software and Programming Language Involved

The utilization of the software is also a crucial component that we are supposed to consider in our project. In preliminary, the software most possibly used in this project includes Android Studio, Flutter, Dart, Firebase, OpenCV, Python, and TensorFlow.

Android Studio is an important integrated development environment (IDE) that is supposed to be used in this project, especially for Flutter app development. It is the official IDE for Android development provided by Google, and it comes with comprehensive tools that make building, testing, and deploying mobile applications more efficient. Unlike lightweight editors such as VS Code, Android Studio is a full-fledged IDE based on IntelliJ IDEA, which offers advanced code editing, debugging, and profiling features tailored for mobile app development.

For Flutter development, Android Studio provides dedicated support through the Flutter and Dart plugins. These plugins enable features such as syntax highlighting, intelligent

code completion, widget editing assistance, hot reload, and visual layout previews. The IDE also integrates seamlessly with the Android SDK to allow it to build, test, and run Flutter apps directly on Android emulators or connected physical devices. Moreover, it offers tools like Layout Inspector, Profiler, and Device Manager, which are essential for analyzing app performance, monitoring resource usage, and debugging UI issues.

Figure 3.1.2.1 Android Studio

Flutter is a framework and open-source UI software designed by Google to develop applications. The purpose of using Flutter in this project has few reasons. Firstly, Flutter supports cross-platform development because it allows the user to develop multiple platforms, such as Android, iOS, Linux, Mac, Windows, and so on, by using the same codebase and deploying to multiple devices. It also increases the developer experience by transforming the workflow, it can take control of the user's codebase with testing automatically, developer tooling, and otherwise, which is used to build the applications. Besides, it can integrate with various Google services such as Firebase, Google Ads, Google Play, Google Maps, and so on seamlessly to streamline the development. Furthermore, it supports hot-reload, which enables the user to see the changes in real-time when they are developing the application, which speeds up the development process.

Dart Language is a main component and language used in Flutter; it is designed for building optimized UI on multiple platforms. Utilizing Dart can let Flutter avoid using separate declarative layout languages like XML, and it can create highly responsive and attractive interfaces to increase the user experience.

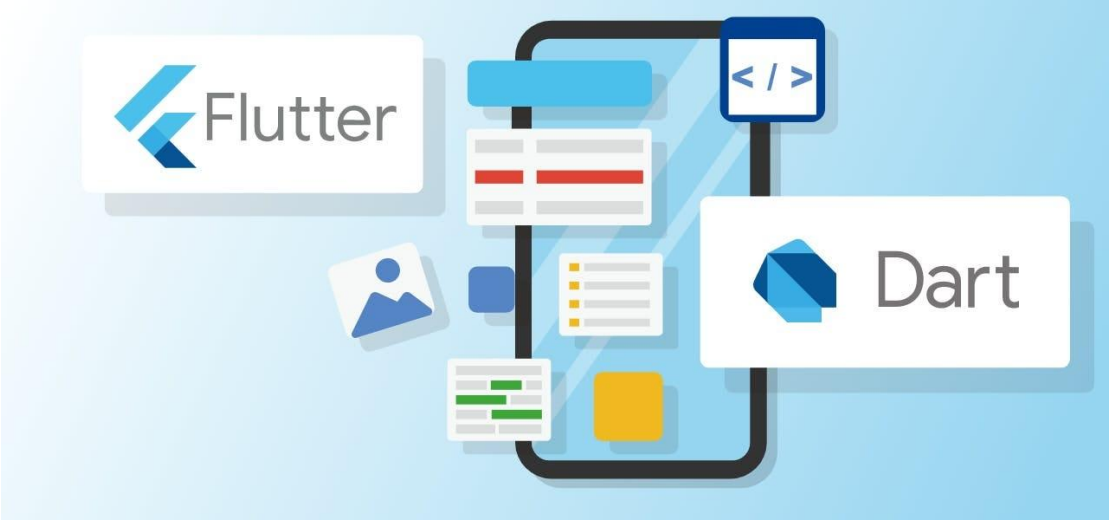


Figure 3.1.2.2 Flutter and Dart

Firebase is a powerful platform that can be integrated easily with Flutter to improve the performance of the application. It provides backend functionality for building and running mobile and web applications, making it more suitable for use in this project. It provides various services such as Firebase Authentication, Firebase Cloud Messaging, Firebase Cloud Functions, Firebase Analytics, and so on. These services help the user to manage data, store data, analyze data, and authenticate better. The Firebase used in this project not only will accelerate the development of the application by reducing the requirement to write some specific server-side code but also enhance application performance.

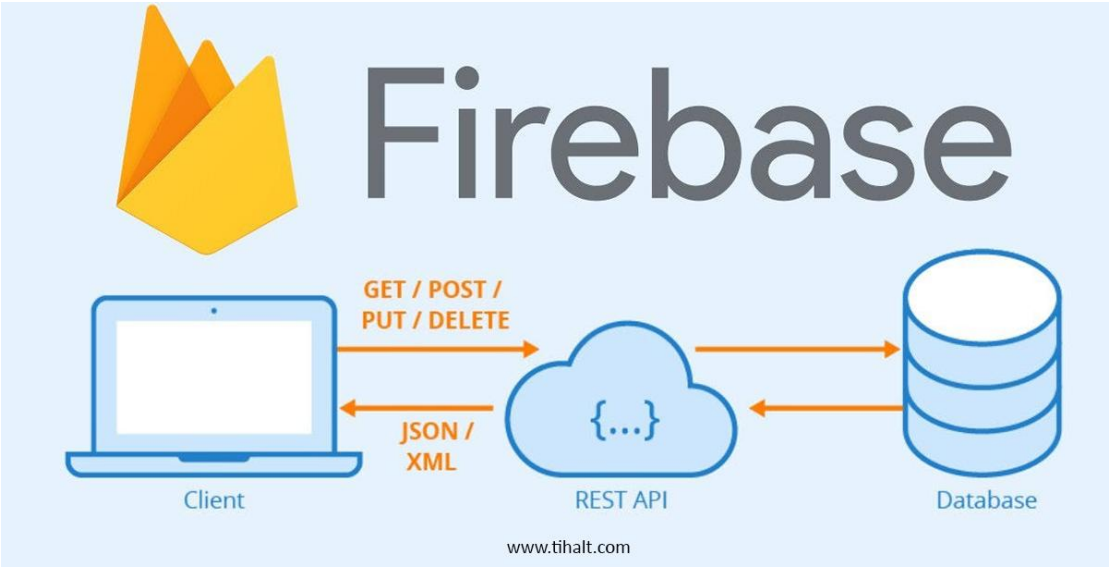


Figure 3.1.2.3 Firebase

TensorFlow is an end-to-end platform for machine learning. It is easy to create machine learning models that can run in any environment. It can be used for building and training models for classification, regression, and prediction. Moreover, it is powerful in computer vision because it is ideal and efficient in handling large data sets such as people images and neural network algorithms, so it is suited for utilization in this project.

Figure 3.1.2.4 TensorFlow

3.2 System Design

The system design is also critical because it translates how the system should be built into how the system will be built. It puts the action to structure the application into components, processes, data flow, and interactions before coding. It acts as a blueprint for developing a system and helps the developer develop the system via predefined ideas or guidelines. Furthermore, it helps to structure the entire system clearly by breaking down the system into smaller parts, like components, so that it allows the developer to build the interaction between the components or design the data flows. In this case, it also helps to improve and maintain the system because the application is developed by using an agile methodology, so that the system design is important for redesigning the application.

3.2.1 Model Design

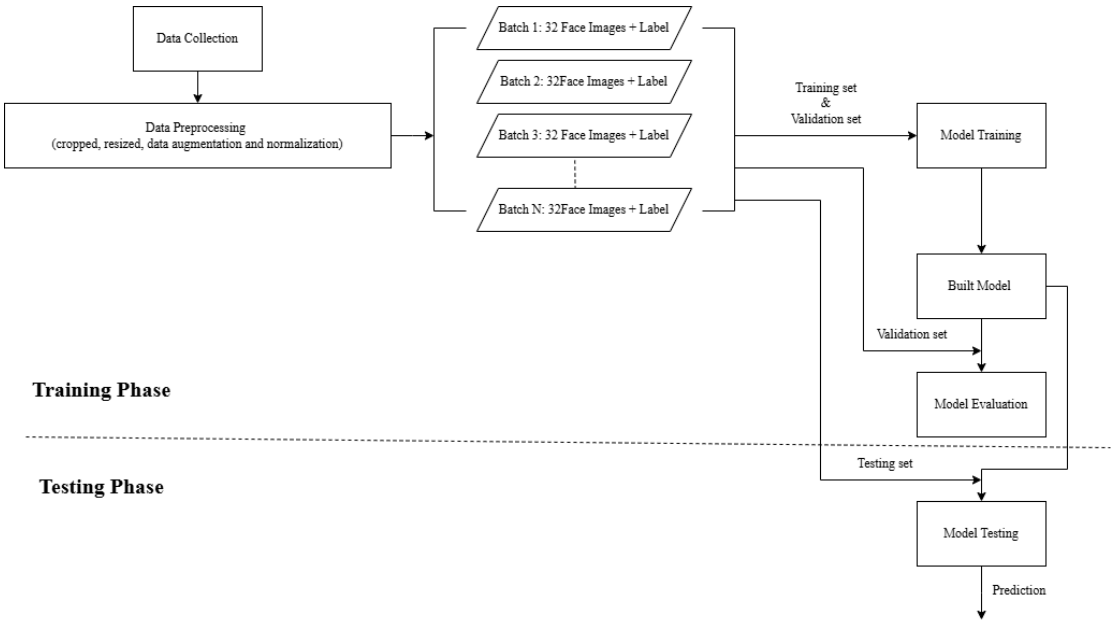


Figure 3.2.1.1 Model Design Diagram

Dataset Used

Before starting the model training, the data collection is implemented so that it can be applied to the model training. The dataset is moderately sized enough for easier observing, faster training time, building and validating the prototype of the model quickly, and saving memory and computational power. Furthermore, the dataset has added the developer images. The dataset should be divided into training and testing datasets, where the training dataset is used for model training (including training data and validation data), and the testing dataset is used for model testing.

Data Loading and Preprocessing

In the data loading and preprocessing phase, the training dataset should be separated into the training data and validation data with a common ratio of 8:2, and do the resizing and batching. Moreover, the process, such as data augmentation, normalization, and pipeline optimization (cached and prefetched), should be applied to improve the performance of the model

Model Training

The Convolutional layer, BatchNormalization, MaxPooling2D, flatten layer, fully connected layer, and so on should be used to build the convolutional neural network. The different values of the hyperparameter should be tested on the layers or activation function, as well as using the optimizer or callbacks, so that good performance can be obtained. This model design is appropriate for either face embedding or classification.

Model Evaluating and Testing

Lastly, the trained model should be evaluated with the validation data and testing data, so that the extent to which the model generalizes to unseen data can be observed. The classification report, such as the accuracy, precision, recall, and f1 score, should also be visualized for interpreting the performance of the model.

3.2.2 System Architecture Diagram

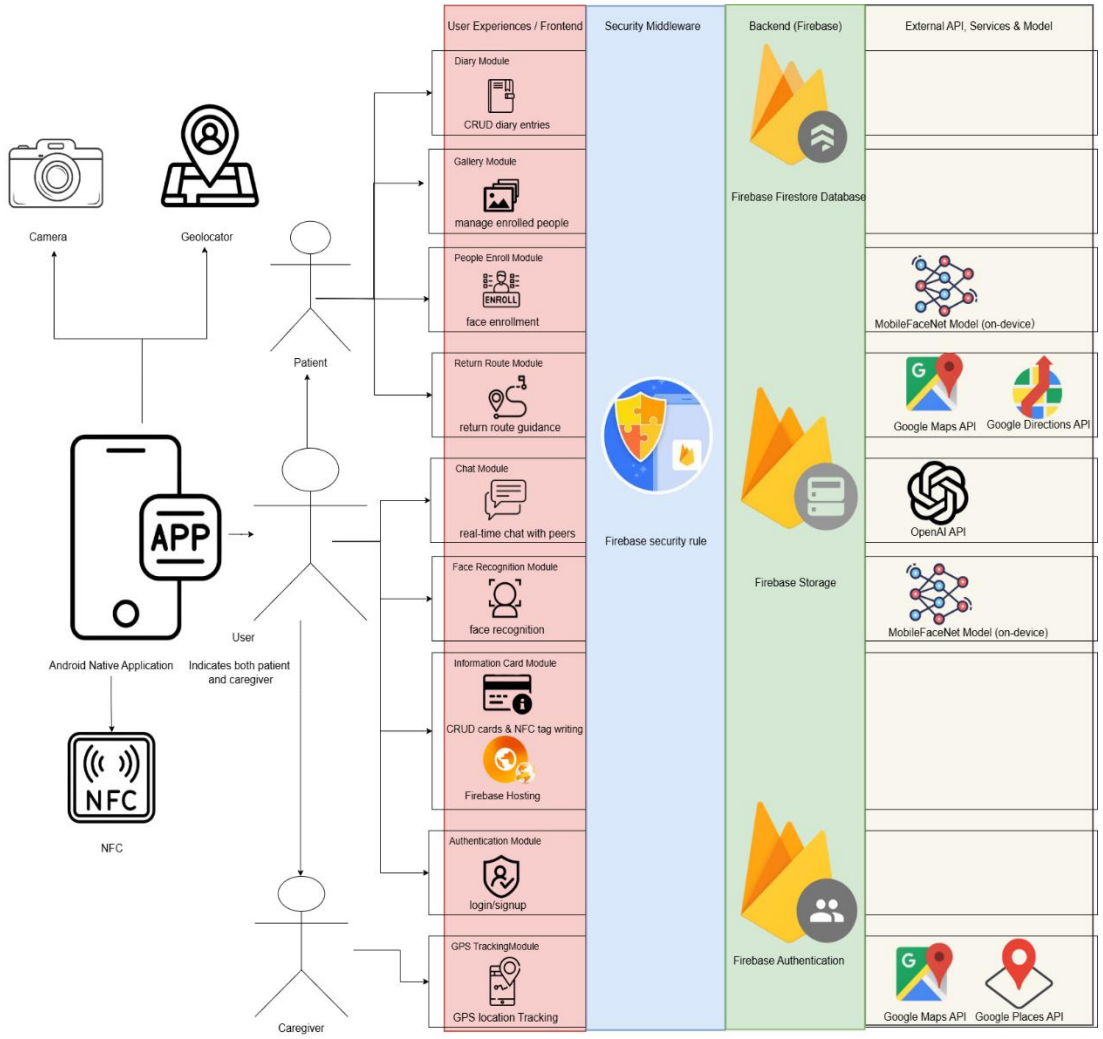


Figure 3.2.2.1 System Architecture Diagram

The figure illustrates the architecture diagram of the proposed application. MemoraCare is a mobile-first, serverless system that is composed of frontend, security policy enforcement, managed backend services, and device sensors or external integrations.

On the front-end, the Android native application delivers the full UI/UX for both patient and caregiver modules. In parallel, a small public web page is used to render the NFC information card that is deployed on Firebase Hosting. Although Hosting is part of the Firebase platform, in this system, it serves static front-end assets such as HTML, CSS,

and JS over a global CDN with SSL. It does not execute any business logic and should therefore be considered a front-end delivery component rather than a backend service.

Moreover, the Firebase Security Rules for Firestore and so on act as the Security middleware to enforce authorization and data shape constraints. This layer does not run custom code, and it is a declarative policy that gates access to the backend services.

Besides that, Firebase is used as a backend that helps to manage services, security, and data stored in the database. For example, Firebase Authentication acts as the identity that handles sign-in, sign-up, and issuing ID tokens for logged-in users. The Firestore Database acts as the database of the app to manage the structured data for each module, whose security is governed by Firestore Security Rules. The Firebase Storage helps to store files or blobs that are larger, such as images, videos, and so on.

The external third-party API, services, and models include Google Maps/Directions/Places APIs that help map rendering, route computation, place details for return guidance, and place searching, MobileFaceNet (on-device) that helps to generate facial embeddings, and OpenAI API that acts as the conversational assistance within the chat module. These components interact with the front-end of the corresponding modules to achieve certain specific and useful functions.

The device sensors used in this app include the camera that helps to capture the faces, the geolocator that helps to obtain the longitude and latitude of the device, and the NFC that helps to read or write the NFC tag.

In conclusion, there are a total of 9 modules designed in the app, and each module performs different functions and tasks. Both patients and caregivers are allowed to access chat, face recognition, and the information card modules with certain limitations based on the role. The return route, diary, and gallery modules are designed for patients to help them recall their memories and also overcome the emergency incident, while the GPS tracking module is designed for the caregiver to help them monitor the location of the patient to avoid the patient wandering.

3.2.3 Flow Chart Diagram

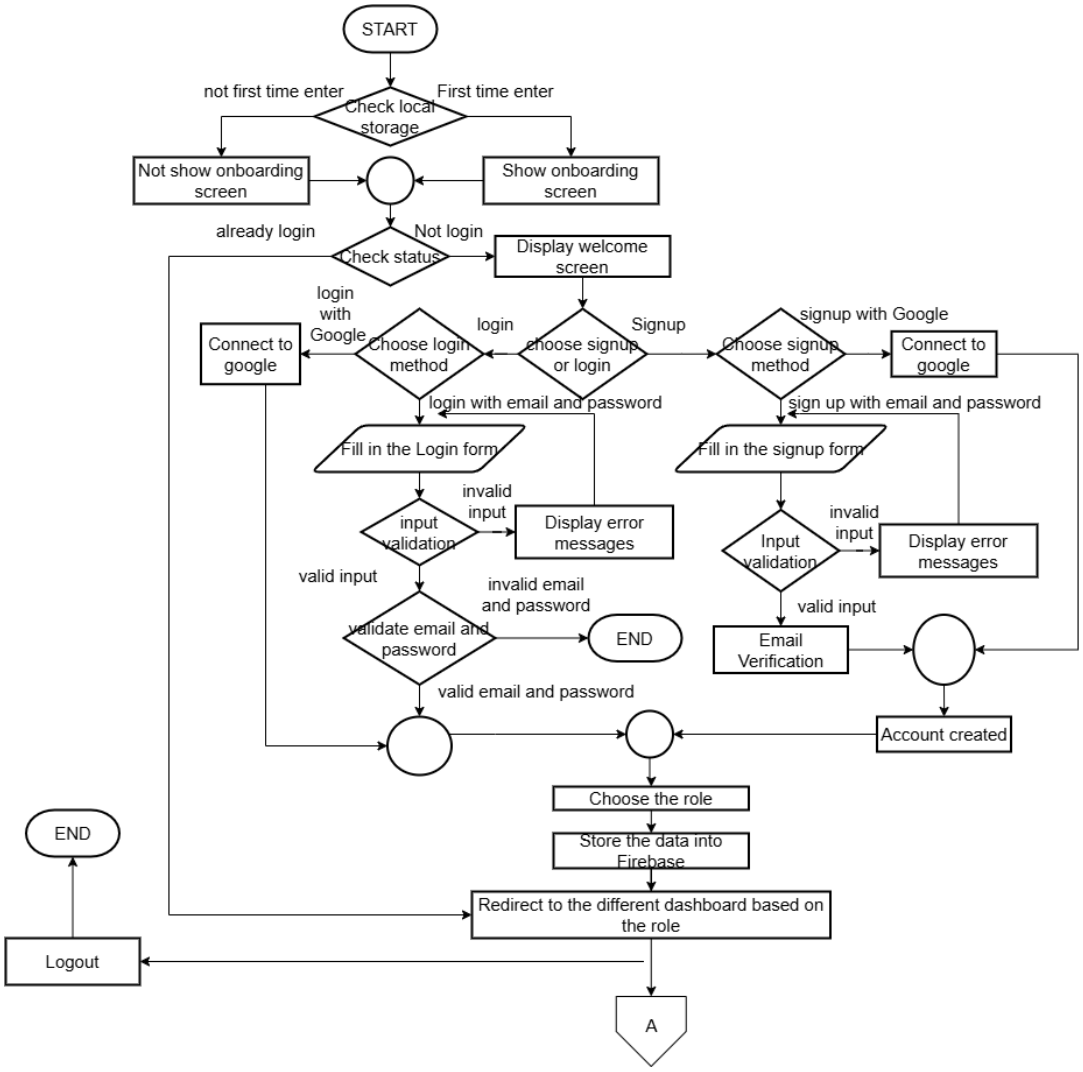


Figure 3.2.2.1 Flow Chart Diagram part 2

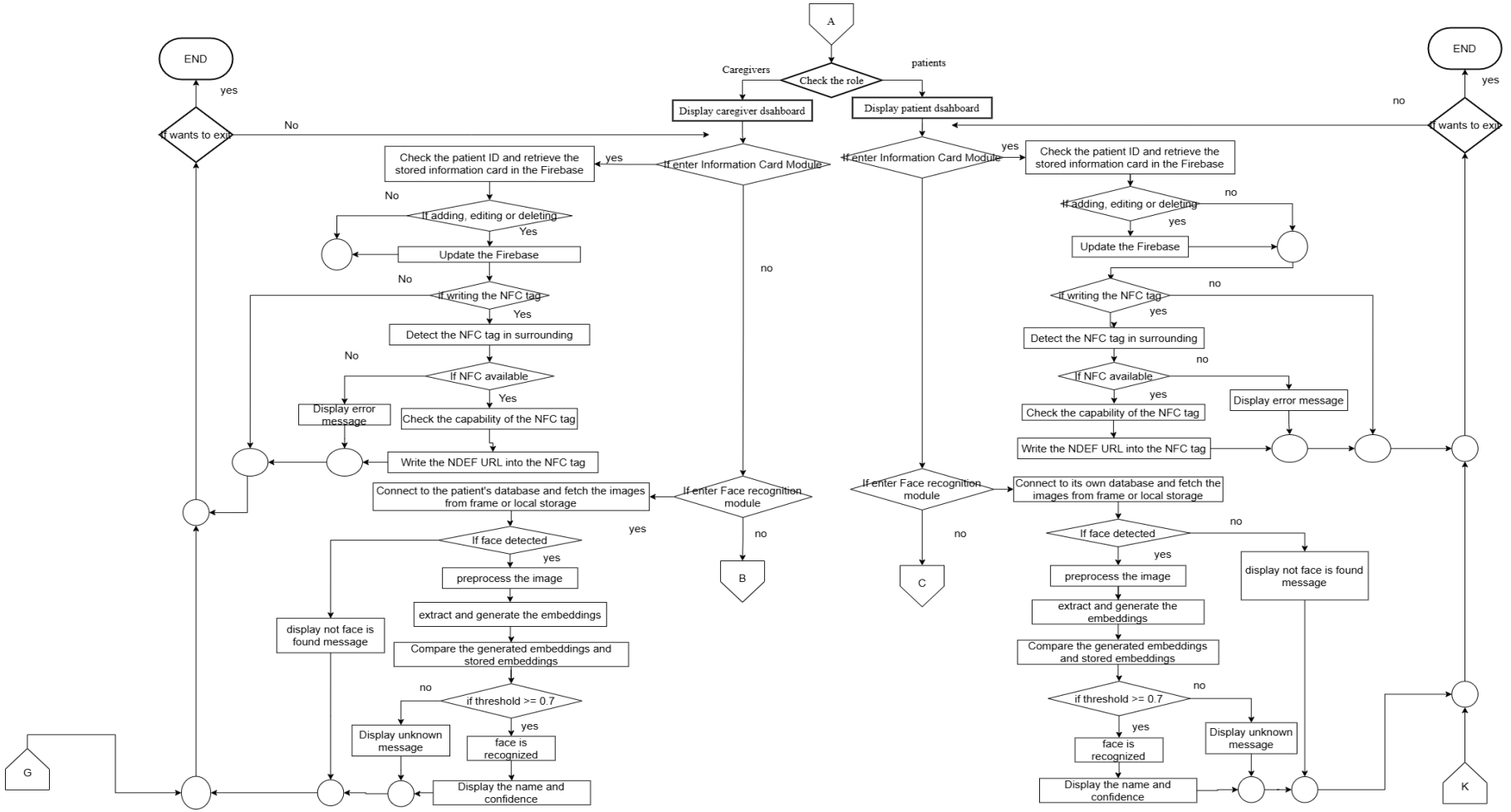


Figure 3.2.2.2 Flow Chart Diagram part 2

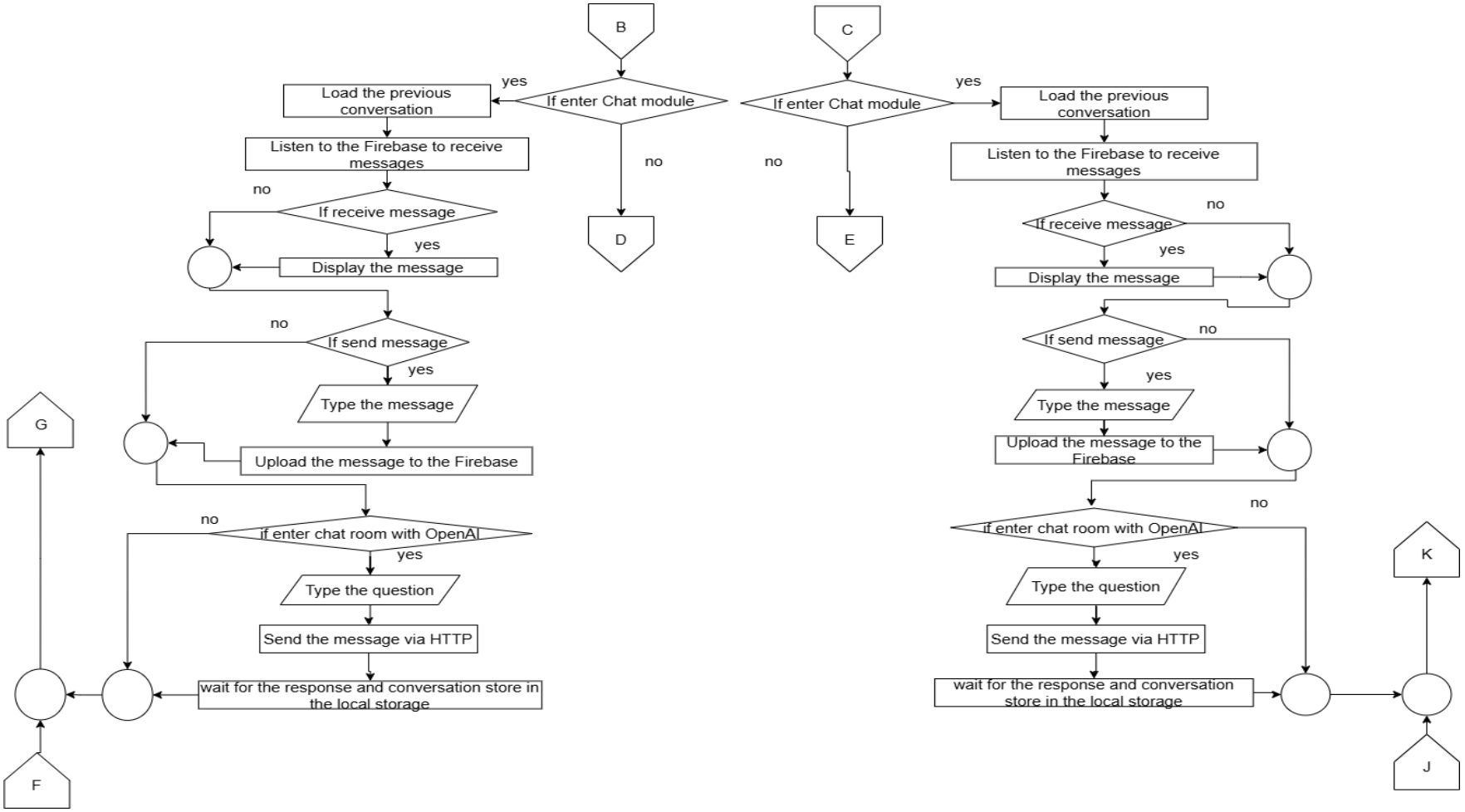


Figure 3.2.2.3 Flow Chart Diagram part 3

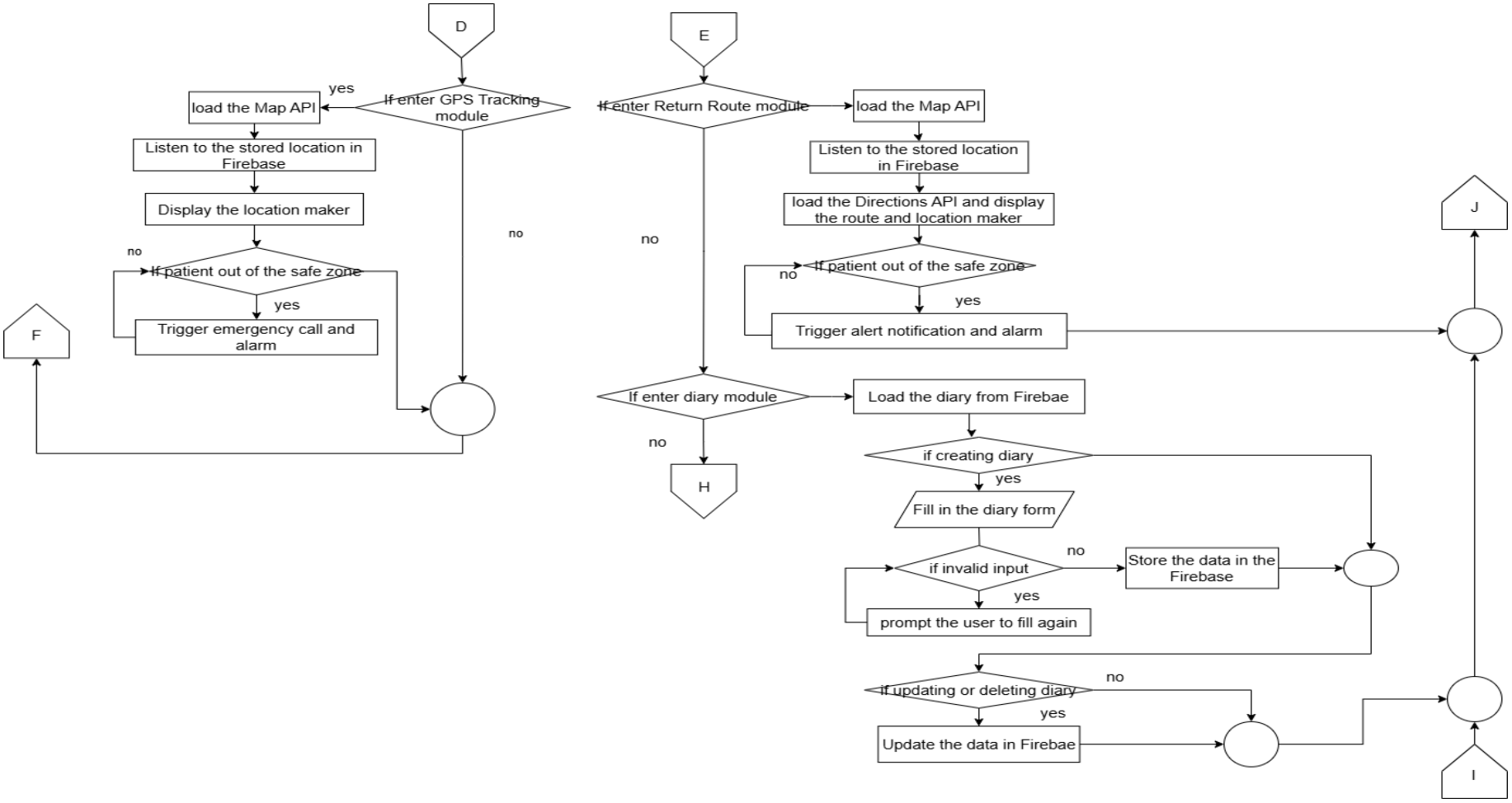


Figure 3.2.2.4 Flow Chart Diagram part 4

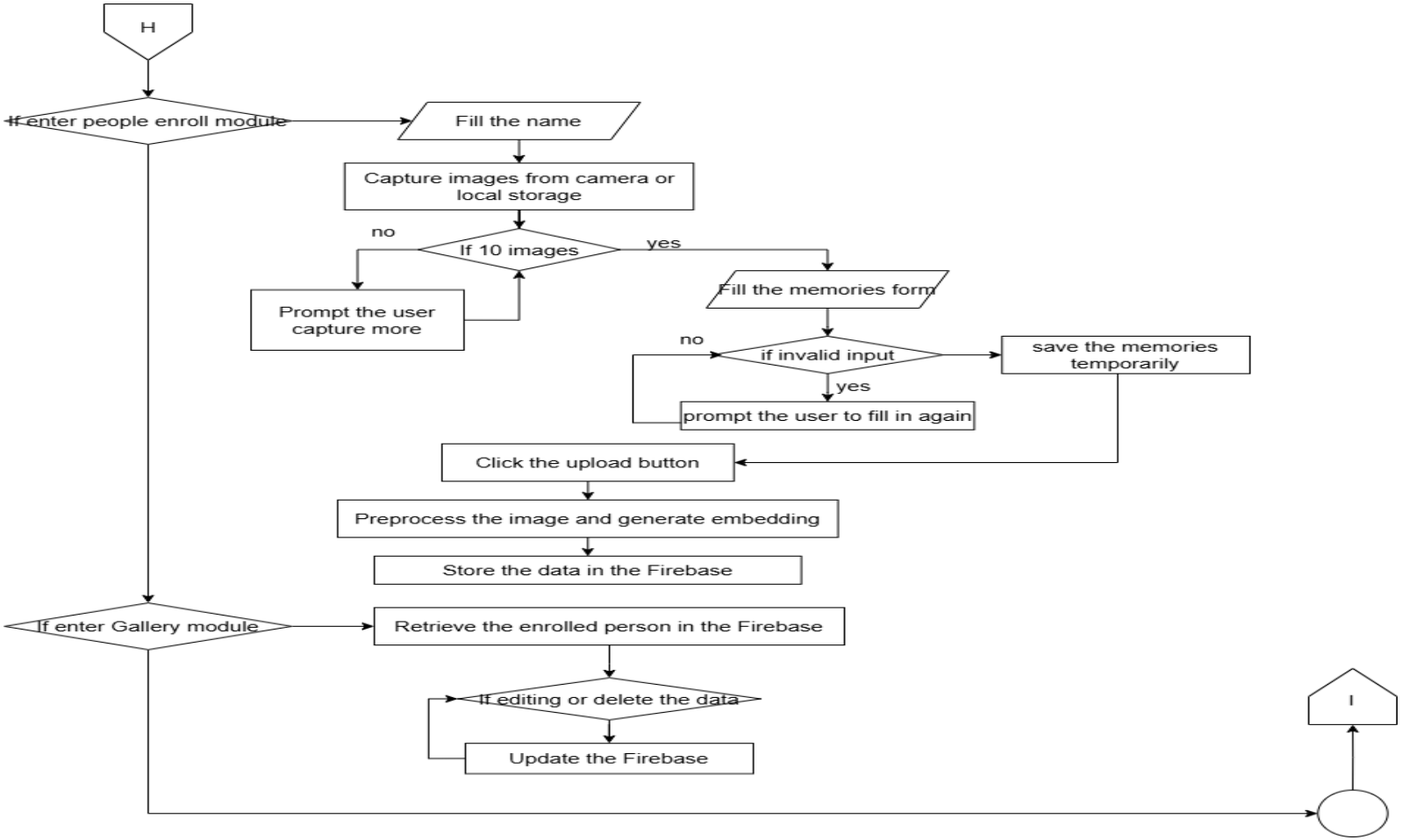


Figure 3.2.2.5 Flow Chart Diagram part 5

3.2.3 Use Case Diagram

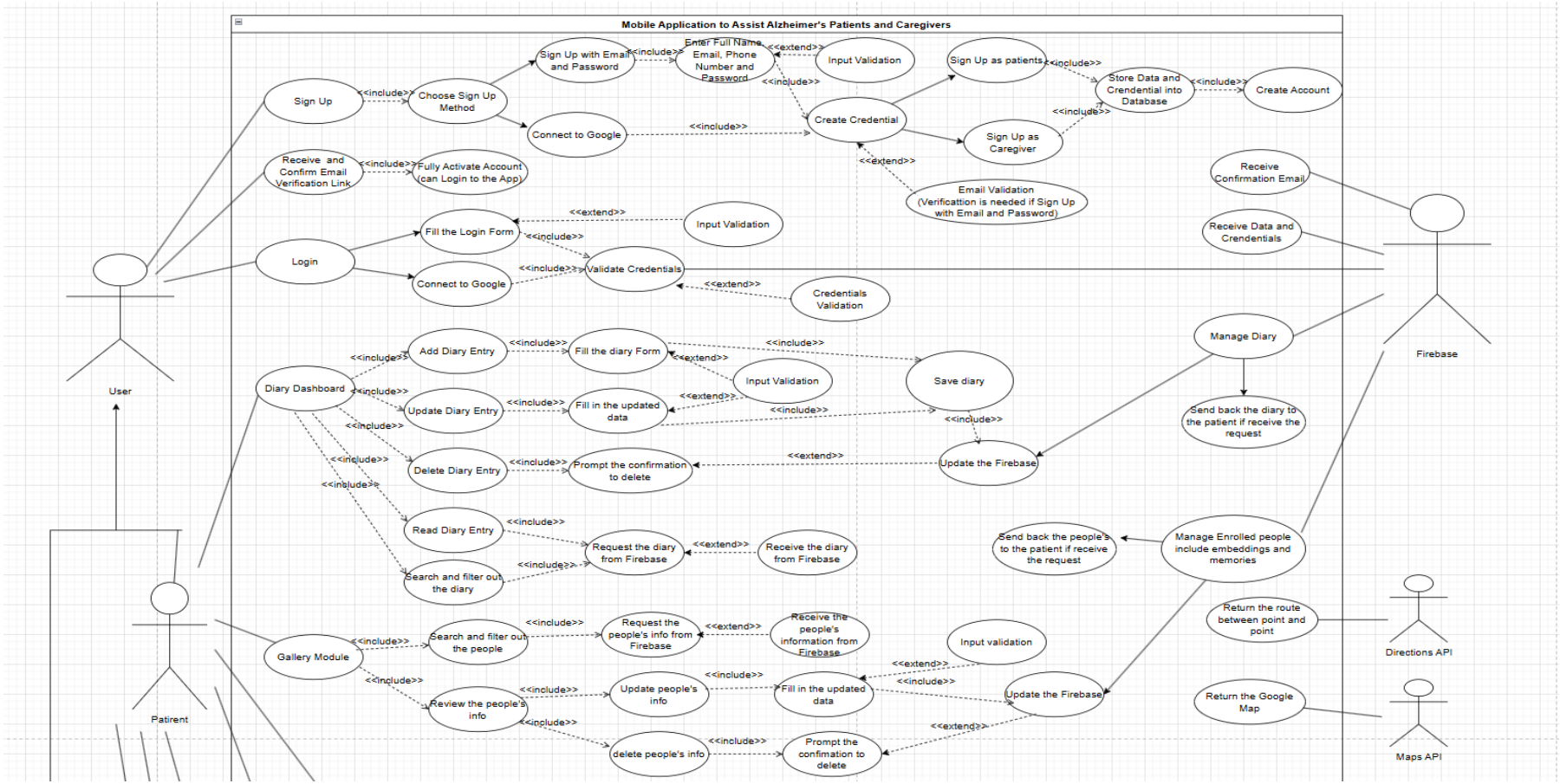


Figure 3.2.3.1 Use Case Diagram Part 1

Figure 3.2.3.2 Use Case Diagram Part 2

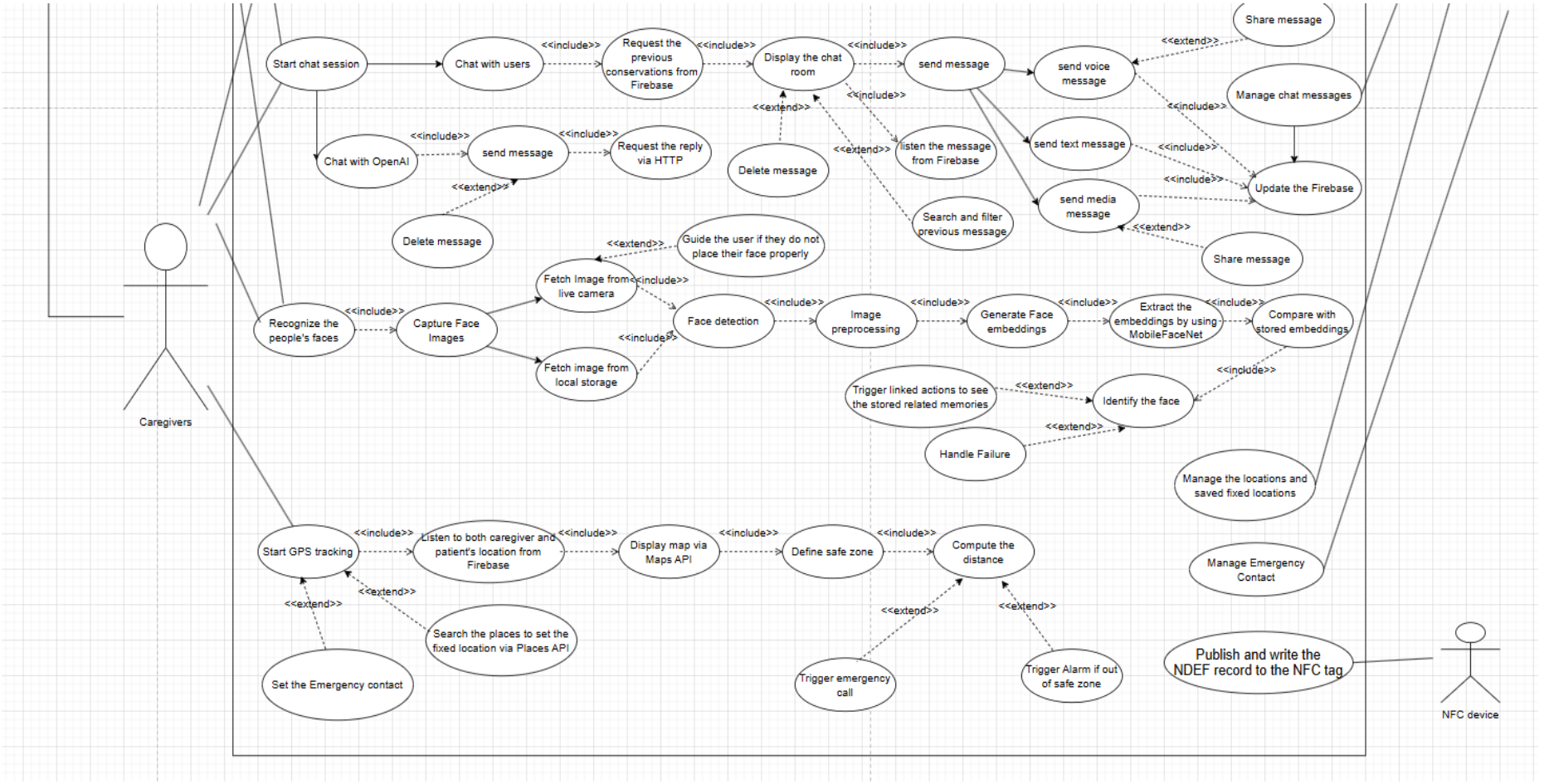


Figure 3.2.3.3 Use Case Diagram Part 3

3.2.4 Use Case Description

Table 3.2.4.1 Sign-up use case description

Use Case Name: Sign Up	**ID:** UC0001	**Importance Level:** High
Primary Actor: User	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • User – Wants to create an account to access the application features • Firebase Authentication – enables the user to sign in with different methods, such as email and Google. • Firebase Firestore Database – allow the user to store their sign-up information in the database		
Brief Description: • User provides required information details such as email, full name, phone number, and password to create an account. The system will validate the input and create the new account.		
Trigger: The user clicks the Sign-Up button on the welcome screen **Type:** External		
Relationships: • Association: User • Include: Choose Sign Up Method, Enter Full Name, Email, Phone Number, and Password, Create Credential, Store Data and Credential into Database, and Create Account. • Extend: Input Validation, Email Validation • Generalization: Sign Up with Email and Password, connect to Google, Sign Up as Patients, Sign Up as Patients		
Normal Flow of Events: 1. User clicks the Sign Up button. 2. System displays two methods for sign-up If the user wants to sign up via email and password, S-1: Sign up via email and password is performed. If the user wants to sign up by connecting to Google S-2: Connect to Google 3. The system will create credentials based on the selected sign-up method. 4. If the user signs up by email and password,		

Execute the email validation use case.

5. Users choose the role that they want to play

   If the user signs up as a patient

   S-3: Sign up as patients is performed

   If the user signs up as a caregiver

   S-4: Sign up as caregivers is performed

6. The system transfers data and credentials to the Firestore database and is recorded in the Firebase.
7. The account was created, and then the system redirected to the home/dashboard.

Sub Flow of Events:

- S-1: Sign up via email and password
  1. Users need to fill in the sign-up form that is on the sign-up screen.
  2. Users need to enter the required information, like full name, email, phone number, and password.

     Execute input validation use case.
- S-2: Connect to Google
  1. Connect to Google services and choose what the accounts are used to log in.
- S-3: Sign up as a patient is performed
  1. If the users sign up as the patient, the record will be stored for further redirecting to the corresponding dashboard for the patient.
- S-4: Sign up as a caregiver is performed
  1. If the users sign up as the caregiver, the record will be stored for further redirecting to the appropriate dashboard for the caregiver.

Alternate / Exceptional Flows:

- S-1, 2a1: The real-time input validation is applied to prevent empty or incorrectly formatted fields by displaying error messages to remind the user. For example, the input fields cannot be left empty if the user wants to submit the form. There are other validations for the input fields. For example, the email input field should contain the symbol @ or a dot, while the password input field only accepts 8 characters that include uppercase, lowercase, numbers, and symbols.
- 4a1: The user is asked to verify their email, and can redirect it to the dashboard.

Table 3.2.4.2 Login use case description

Use Case Name: Login	**ID:** UC0002	**Importance Level:** High
Primary Actor: User	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • User – allows access to the application features • Firebase Authentication – Provide the authentication to different methods, such as email and Google. • Firebase Firestore Database – Manage the authenticated account and help in the validation process.		
Brief Description: • The user logs into the application by choosing one of the available methods: email/password or Google login. Upon successful authentication, the system verifies the user and redirects to the dashboard.		
Trigger: The user clicks the Login button on the welcome screen **Type:** External		
Relationships: • Association: User • Include: Validate Credentials and Connect to Google. • Extend: Input Validation, Credentials Validation • Generalization: Fill the Login Form and Connect to Google.		
Normal Flow of Events: 1. User clicks the Login button**.** 2. System displays two methods for login		

If the user wants to log in via email and password,

S-1: Fill the Login Form is performed.

If the user wants to log in by connecting to Google

S-2: Connect to Google

3. The system will validate credentials from fetching data and credentials from the Firebase.
4. If the credentials are valid, the process will continue; while if the credentials invalid, show error handling.

   Execute Credentials Validation

5. The system will redirect the user to the dashboard.

Sub Flow of Events:

- S-1: Sign up via email and password
  1. Users need to fill in the login form that is on the login screen.
  2. Users need to enter the required information, such as email and password.

     Execute Input Validation use case

- S-2: Connect to Google
  2. Connect to Google services and choose what the accounts are used to log in.

Alternate / Exceptional Flows:

- S-1, 2a1: The real-time input validation is applied to prevent empty or incorrectly formatted fields. For example, the input fields cannot be left empty if the user wants to submit the form. There are other validations to the input fields, such as the email input field should have the symbol @ or a dot.
- 4a1: The credentials validation is performed. Only the email and password recorded in the Firebase can be accepted; otherwise, it displays error messages.

Table 3.2.4.3 Face Recognition Use Case Description

Use Case Name: Face Recognition	**ID:** UC0003	**Importance Level:** High
Primary Actor: User (Patients and Caregivers)	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • User – allows the recognition of the person whose data was stored in the Firebase • Firebase Firestore Database – Manage the enrolled people and help to recognize the people by providing the stored embeddings. • MobileFaceNet model – helps to extract the embeddings.		
Brief Description: • User is enabled to recognize faces through face recognition. The system captures the user's face via the live camera or local storage, preprocesses it, and uses the model to extract the embeddings of the images. If matched with the stored embeddings in the Firebase, it means the person is recognized.		
Trigger: The user clicks the Face Recognition Module on the dashboard **Type:** External		
Relationships: • Association: User • Include: Capture Face Images, Face detection, Image Preprocessing, Generate Face embeddings, Extract the embedding by using MobileFaceNet, compare with stored embeddings, and identify the face • Extend: Trigger linked actions to see the stored related memories, and Handle Failure • Generalization: Fetch Image from live camera, and Fetch image from local storage		

Normal Flow of Events:

1. User clicks the Face Recognition module to start the face recognition
2. The system provided two methods to let the user capture their face images

   If the user wants to conduct real-time face recognition

   S -1: Fetch images from the live camera is performed

   If the user wants to conduct normal face recognition

   S-2: Fetching images from the local storage is performed
3. After the image is captured, face detection via ML Kit will be used to detect whether the face is in the image.
4. If the face is detected, the system would preprocess the image, such as cropping, resizing, and so on.
5. The system attempts to generate face embeddings, and the MobileFaceNet model helps in extracting the face embeddings.
6. The generated embeddings will be compared with the stored embeddings from the Firebase.
7. The best matches of embedding and threshold will be selected
8. By following where the embedding comes from the labelled name stored in the Firebase, the person is recognized.
9. The show more button is presented if the person is recognized, and this button is linked to that person's related stored memories.

Sub Flow of Events:

- S -1: Fetch images from the live camera is performed
  1. User clicks the live camera button to start the real-time recognition
  2. The system prompts the user to place their face in a box
- S-2: Fetch images from the local storage is performed

  1. The system provides the allowed path to the user to choose the image from the local storage

Alternate / Exceptional Flows:

• S-1, 2a1: The system guides the user if they do not place their face properly in the center of the designated box by displaying the message. • 8a1: The system displays unknown if no generated embedding is matched with the stored embedding, and displays " face not found if the face is not detected in the image • 9a1: User can click the show more button to review the related memories with the recognized person

Table 3.2.4.4 Enrol Faces use case description

Use Case Name: Enroll Faces	**ID:** UC0005	**Importance Level:** High
Primary Actor: Patient	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • Patient – wants to register their face so that it can be recognized by the model. • Firebase Firestore Database – helps to store the information or data in the enrolment and manage it. • Firebase Storage – helps to store the media such as images, videos, audio, and so on.		
Brief Description: • Patients are allowed to enrol their face by capturing multiple facial images through the live camera and local storage. The system detects the face in each image, preprocesses it, and stores it under the provided name, as well as stores the memories about the person.		
Trigger: Patient clicks the People enrol module at the patient dashboard. **Type:** External		
Relationships: • Association: Patient • Include: Label the names of the people, Capture Face Images, Face detection, fill in the form to record memories, attempt to upload both data, Image Preprocessing, Generate Face embeddings, extract the embedding by using MobileFaceNet, store the embeddings, faces, and memories into Firebase • Extend: cancel enrolment • Generalization: Fetch Image from live camera, Fetch image from local storage, and validate the limitations of uploaded items		

Normal Flow of Events:

1. Patient navigates to the People enrol module.
2. The patient can type the name to label the faces that they want to enroll.
3. System prompts the user to capture 10 face images.
4. The system provides two methods for face enrolment

   If the patient wants to capture images from a live camera

   S-1: Fetch Face Images from live camera is performed

   If the patient wants to obtain images from the Local Browser

   S-2: Obtain Images from Local Browser is performed
5. The system performs face detection via ML Kit to detect the existence of the face in the images.
6. The patient can continue the enrolment process by adding some valuable memories with the person, such as relationships, notes, images, videos, audio, any files, and so on.
7. The patient can click the upload all images and memories button to upload the face images and memories to the Firebase.
8. System preprocesses each image, such as resizing and normalizing.
9. The system will generate face embedding with the help of the MobileFaceNet model.
10. The face images, embeddings, and memories will be stored in the Firebase.

Sub Flow of Events:

- S-1: Capture Face Images from live camera
  1. The system opens the camera.
  2. The system detects the face in each frame.
  3. The system offers a box to align the face.
  4. Patient aligns their faces in the box and clicks the button to capture the image
  5. Repeat the procedure from 1 to 4 until 10 images are captured
- S-2: Obtain Images from Local Browser
  1. The system opens the local browser.
  2. Patients choose the 10 facial images for enrolment.

Alternate / Exceptional Flows:

- 6a1: The system will limit the size of the attachments and validate them, such as images cannot exceed 5 MB, videos should be between 20 and 50 MB, audio cannot exceed 10 MB, and files cannot exceed 15 MB.
- 7a1: The system will ask for confirmation from the user whether to upload the images and memories, and the user is allowed to cancel or agree.

• S-1, 4a1: The system guides the user if they do not place their face properly in the center of the designated box by displaying the message.

Table 3.2.4.5 Chat use case description

Use Case Name: Start Chat Session	**ID:** UC0006	**Importance Level:** High
Primary Actor: User (Patients and Caregivers)	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • User – wants to communicate with the other user or OpenAI. • OpenAI API – reply to the message from the user		
Brief Description: • Users are allowed to initiate, participate in, and view chat conversations in real time and also communicate with OpenAI. It supports text or voice messages and attaches attachments such as images, videos, audio, and files. These messages and attachments are supposed to be stored in the Firebase		
Trigger: Users navigate to the chat module and start the conversation **Type:** External		
Relationships: • Association: User • Include: Request the previous conversations from Firebase, Display the chat room, Send a message, Update the Firebase, Request the reply via HTTP • Extend: Receive the message, and Search and filter previous messages • Generalization: Chat with users, Chat with OpenAI, Send text message, Send media message, Send voice message, and share message		
Normal Flow of Events: 1. Use the chat modules to start the conversations; the default is to display the chat room for chatting with the user. If the user wants to chat with OpenAI S -1: chat with OpenAI is performed 2. The system requests the previous conversations from Firebase and displays the chat room. If the system is allowed to receive the message		

S -2: listen to the message from Firebase is performed 3. The user can send messages such as text, voice, and media to the connected user. 4. The system will store both messages in the Firebase
Sub Flow of Events: • S -1: chat with OpenAI 1. The user can send the message and attach the photo to OpenAI. 2. The system will request a reply from OpenAI via HTTP. • S -2: listen to the message from Firebase 1. The system will receive the message and display it in the chat room if it listens to the message from Firebase.
Alternate / Exceptional Flows: • S -1, 2a1: The user can delete the conversations in the chat room. • 2a1: The user can search and filter the messages in the chat room. • 2a2: The user can delete the messages in the chat room. • 2a3: The user can choose to delete for everyone to delete it, or both sides • 2a4: The user can choose to delete for me to delete the sent message by himself or the opposite side, and this deletion is just operated on his side. • 3a1: The user can share the media message and voice message to other platforms.

Table 3.2.4.6 Start GPS Tracking use case description

Use Case Name: Start GPS Tracking	**ID:** UC0007	**Importance Level:** High
Primary Actor: Caregivers	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • Patient - Location is shared and is being tracked. • Caregiver – expect to obtain the real-time location of the patient for supervision. • Maps API – allows the location and map visuals to be displayed on the map and provides a comprehensive map. • Places API – allows the user to search for places, and convert the address to longitude and latitude and vice versa.		

Brief Description:

- Caregivers are allowed to track the real-time location of the patient from the Google Maps API.

Trigger: Caregiver enters the GPS tracking module to track the patient.

Type: External

Relationships:

- Association: Caregivers
- Include: Listen to both caregiver and patient's location from Firebase, Display maps via Maps API, Define a safe zone, and Compute the distance
- Extend: Set the Emergency contact, Search the places to set the fixed location via Places API, Trigger emergency call, and Trigger Alarm if out of the safe zone
- Generalization: None

Normal Flow of Events:

1. Caregivers enter the modules to start the location tracking.
2. The system will listen to both the patient's and the caregiver's location from Firebase.
3. The system will update the location and map visuals on the map that is supported by the Maps API.
4. The caregiver needs to define the safe zone, and this setting is customized by the caregiver.
5. The system will compute the distance between caregiver and patient, or fixed location and patient, by using the haversine formula.

Sub Flow of Events:

- None

Alternate / Exceptional Flows:

- 1a1: The caregiver can search for places to set the fixed location via the Places API, so the caregiver can monitor the patient's location, whether in the safe zone, by measuring the distance between the fixed location and the patient.
- 1a2: The caregiver can set the emergency contact so that they can call other people if the patient wanders.
- 5a1: The system will trigger emergency calls if the patient is out of the safe zone and call the emergency contact, if the emergency contact is not set will call the number 123 as default in this project.
- 5a2: The system will release an alarm if the patient is out of the safe zone and close until the patient has come back the safe zone or the caregiver can close the alarm manually.

Table 3.2.4.7 Diary Dashboard use case description

Use Case Name: Diary Dashboard	**ID:** UC0007	**Importance Level:** High
Primary Actor: Patients	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • Patient – can record the valuable memory on a specific day and recall it if needed. • Firebase Firestore Database – stores the information and data of the diary to manage the diary and helps in filtering and searching for the diary. • Firebase Storage – store the image that is used in the diary		
Brief Description: • The patient can record their valuable memory on a specific day by following the format like a diary, which regulates writing, helping them to record their memories efficiently, and also makes it easier to manage the diary via searching date, hashtag, and participant.		
Trigger: Patients can click the Diary module and perform the CRUD operations via the dashboard, and also filter and search. **Type:** External		
Relationships: • Association: Patients • Include: Add Diary Entry, Update Diary Entry, Delete Diary Entry, Read Diary Entry, Search and filter out the diary, Fill the diary Form, Save diary, Fill in the updated data, Prompt the confirmation to delete, Request the diary from Firebase, and Receive the diary from Firebase • Extend: Input Validation, and Update the Firebase • Generalization: None		
Normal Flow of Events: 1. The patient can click the diary module to enter the diary dashboard. 2. The patient can click the Add Diary Entry button to create a new diary. 3. The system shows a diary form. 4. The patient should fill in the data, such as title, text, date, hashtags, images, and so on. 5. The patient can save the diary if they fill in the form completely and the input is valid and save the data in the Firebase. 6. Furthermore, the patient can update the data of the existing created diaries; the changing data also needs to be validated, and if the changing data is valid, the		

diary can be saved and update the corresponding fields in the Firebase. 7. Moreover, the patient can delete the diary entry, and the system will update the Firebase and remove the entry. 8. Besides that, the patient can enter search text such as #tags or @person, and the system will request diaries from Firebase and receive the diaries, resulting in the matching entries being displayed. 9. The patients can also read the diaries by tapping the diary entry to recall their memories.
Sub Flow of Events: • None
Alternate / Exceptional Flows: • 4a1: If the input validation fails, ask and prompt the user to correct it. For example, the header image, title, at least one hashtag, and contents are necessary, tags must be alphanumeric and extracted with regex such as #tag, participants should mention with like@name, title cannot be longer than 160 characters, and participants can only be added if the people are enrolled • 7a1: The system prompts confirmation to delete the entry for the patient to avoid careless deletion.

Table 3.2.4.8 Find Return Route use case description

Use Case Name: Find Return Route	**ID:** UC0007	**Importance Level:** High
Primary Actor: Patients	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • Patient – can find the way back if they wander by following the provided route. • Caregiver – reduces the burden of the caregiver and helps to monitor the patient's location, reminds the patient timely if out of the range of the safe zone. • Maps API - allows the location and map visuals to be displayed on the map and provides a comprehensive map. • Directions API – provides the route between point to point		
Brief Description: • The patient is allowed to find the route back if they wander, and the destination of the route is followed by the setting of the caregiver's side's GPS Tracking module, either the caregiver's location or a predefined fixed location. It will remind the patient if they are out of the safe zone		

by releasing an alarm.
Trigger: Patients can enter the Return Route module to find the route, and the alarm can be triggered without opening the module. **Type:** External
Relationships: • Association: Patients • Include: Detect safe zone Breach, Notify patient to return, and Show return route, Notify caregiver, Display Google Map API, Find route via Directions API, and Guide the patient back to the designated place • Extend: None • Generalization: None
Normal Flow of Events: 1. The system continuously monitors patient location. 2. The system will detect a safe zone breach to prevent the patient from moving out of the safe zone. 3. The system will notify the patient to return if the patient is out of the safe zone, and the caregiver's side will receive the notifications in the GPS Tracking module. 4. The system will request a route from the Directions API. 5. Directions API returns the optimal path to the safe zone, and the route will be displayed on the Maps API. 6. The patient follows the route to return to the designated place.
Sub Flow of Events: • None
Alternate / Exceptional Flows: • None

Table 3.2.4.9 Find Gallery use case description

Use Case Name: Gallery Module	**ID:** UC0007	**Importance Level:** High
Primary Actor: Patients	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • Patient – can recall the memories or personal information about the enrolled person in the Firebase and manage the person's information. • Firebase Firestore Database – helps to manage the person's information • Firebase Storage – helps to manage the images stored in the storage		
Brief Description: • The patient is allowed to review the previous enrolled person's information that is stored in the Firebase; the patient can perform actions like view, update, delete, sort, and filter to manage the stored information. This allows the patient to modify certain content according to their preferences.		
Trigger: Patients can click the Gallery module **Type:** External		
Relationships: • Association: Patients • Include: Search and filter out the people, Review the people's info, Update people's info, delete people's info, Request the people's info from Firebase, Fill in the updated data, Prompt the confirmation to delete, and Update the Firebase • Extend: Receive the people's information from Firebase, and Input validation • Generalization: None		
Normal Flow of Events: 1. The patient can search by name or filter by relationship in the Gallery module. 2. The system will send the request to the Firebase based on the actions. 3. The patient can browse through the retrieved info and choose to delete or update a record. 4. If the patient wants to update the people's info, he must select a person to edit and fill in the updated data in the desired fields; the updated data will be updated in the Firebase if it is valid. 5. If the patient wants to delete the people's info, he must select a person to delete, and the system should prompt for confirmation to confirm the deletion.		

Sub Flow of Events: • None
Alternate / Exceptional Flows: • 2a1: The Firebase will respond with matching people records and display them if it successfully retrieves the data, while showing an empty list message if no data is found. • 4a1: The system will limit the size of the attachments and validate them, such as images cannot exceed 5 MB, videos should be between 20 and 50 MB, audio cannot exceed 10 MB, and files cannot exceed 15 MB. • 5a1: If the patient confirms, the Firebase will be updated and remove the related document and storage files, otherwise return to browsing.

Table 3.2.4.10 Information Card Module use case description

Use Case Name: Information Card Module	**ID:** UC0007	**Importance Level:** High
Primary Actor: Users	**Use Case Type:** Detail, Essential	
Stakeholders and Interests: • User – allows the creation, editing, deleting, display, and sharing of information cards that store important details about the patient • Patients - can use the information on the information card if needed, like when they wander. • Firebase Firestore Database – helps to manage the information card's data. • NFC device – used to publish and write the NDEF record to the NFC tag		
Brief Description: • The patient is allowed to implement CRUD actions on the information card and share the information card by using the NFC tag that stores some important details like patient info, address, emergency contact, and so on.		
Trigger: Users click the Information Card module **Type:** External		

Relationships:

- Association: Users
- Include: Create the information card, Edit the information card, Delete the information card, Display the information card, Fill in the form, Fill in the changing data, Update the Firebase, Prompt the confirmation to delete, Request the information card's info, Publish public card on the web, and Store the related info in the Firebase
- Extend: Update the Firebase, Receive the information card's info from Firebase, and Input validation
- Generalization: None

Normal Flow of Events:

1. The user can create an information card for the patient and needs to fill in a form. The system will check the input validation.
2. The data will be uploaded to the Firebase if the information is created successfully.
3. Furthermore, the user can edit the information card by filling in the changed data in the form, and the Firebase will be updated if the data is valid.
4. Moreover, the user can delete the information card, and the system will prompt for confirmation to delete the card.
5. Besides that, the user can review the created information card's contents, and the system should retrieve the data from Firebase and display it on the screen if the request is received and the response is obtained successfully.

Sub Flow of Events:

- None

Alternate / Exceptional Flows:

- 1a1: The system should ensure the certain fields are required and prompt the user to fill in such as name, phone, address line 1, state, city, and emergency contact.
- 4a1: If the user confirms, the Firebase will be updated and remove the related document and storage files, otherwise return to browsing.
- 5a1: In the view mode of the information card, the user can choose whether to write the NDEF URL to the NFC tag, this will publish the information card as public on the web to let the other people can access via link and this is helpful if the patient wanders.

3.2.5 Activity Diagram of Sign-Up Functionality

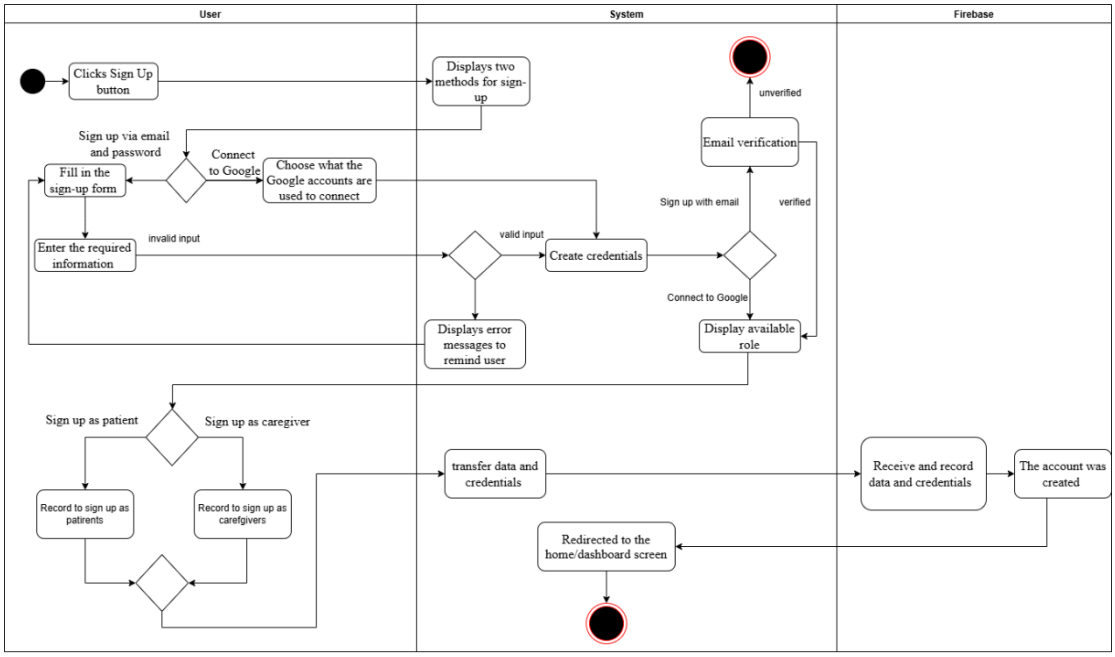


Figure 3.2.5.1 Activity Diagram of Sign-Up Functionality

3.2.6 Activity Diagram of Login Functionality

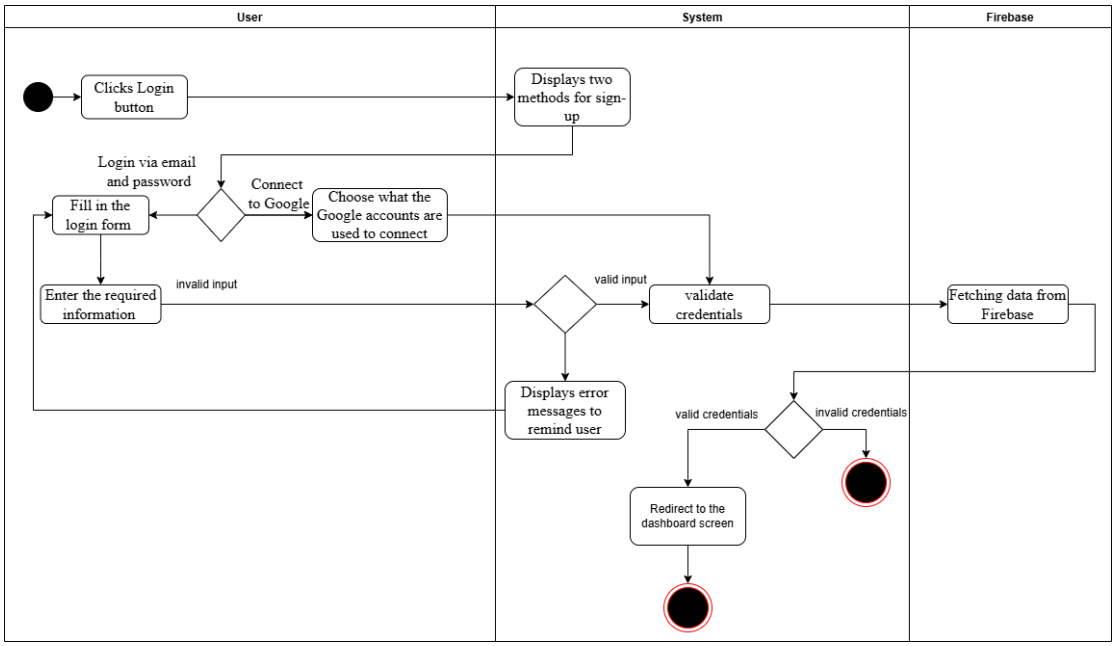


Figure 3.2.6.1 Activity Diagram of Login Functionality

3.2.7 Activity Diagram of Face Recognition Functionality

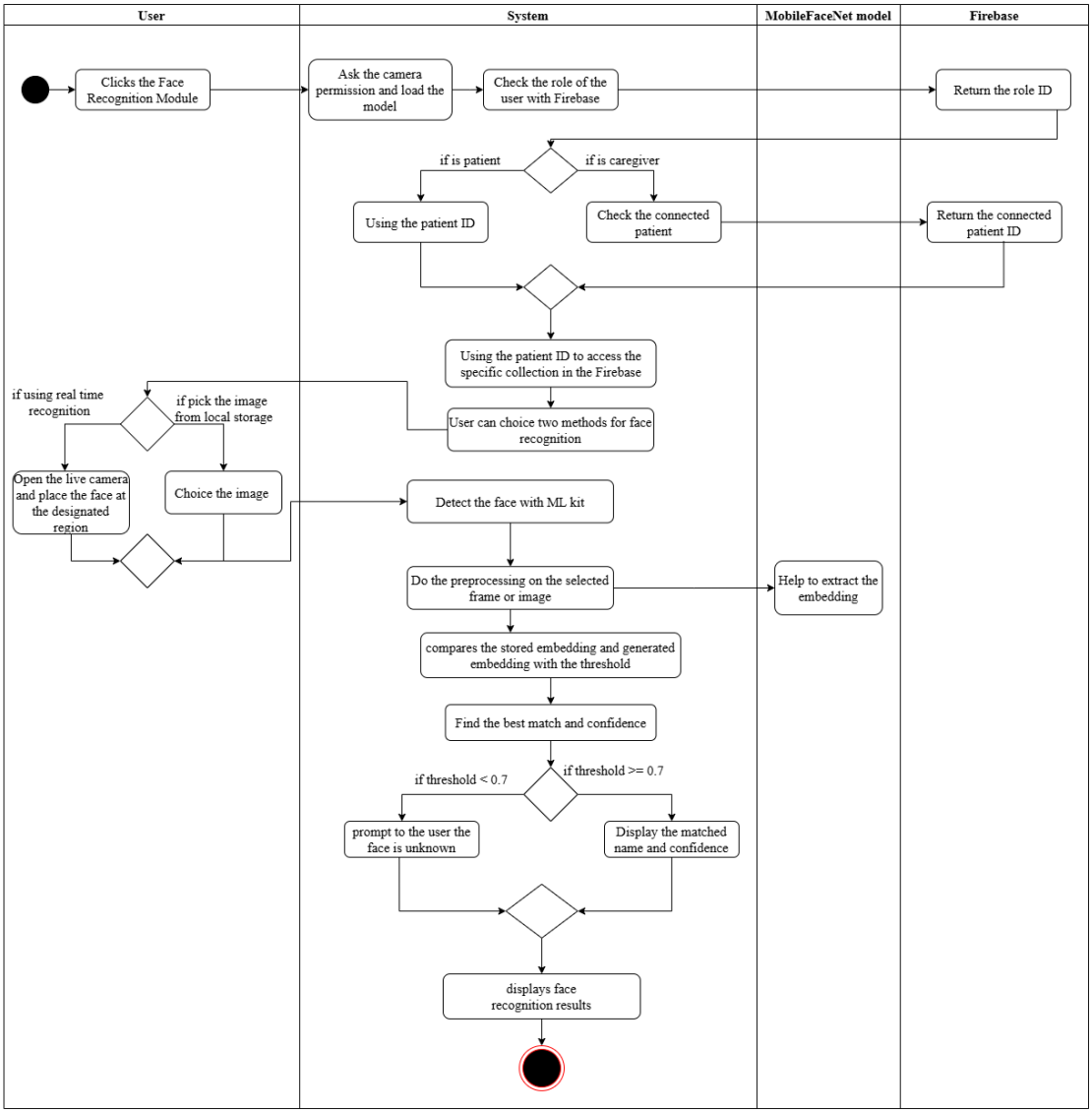


Figure 3.2.7.1 Activity Diagram of Face Recognition Functionality

3.2.8 Activity Diagram of People Enrolment Functionality

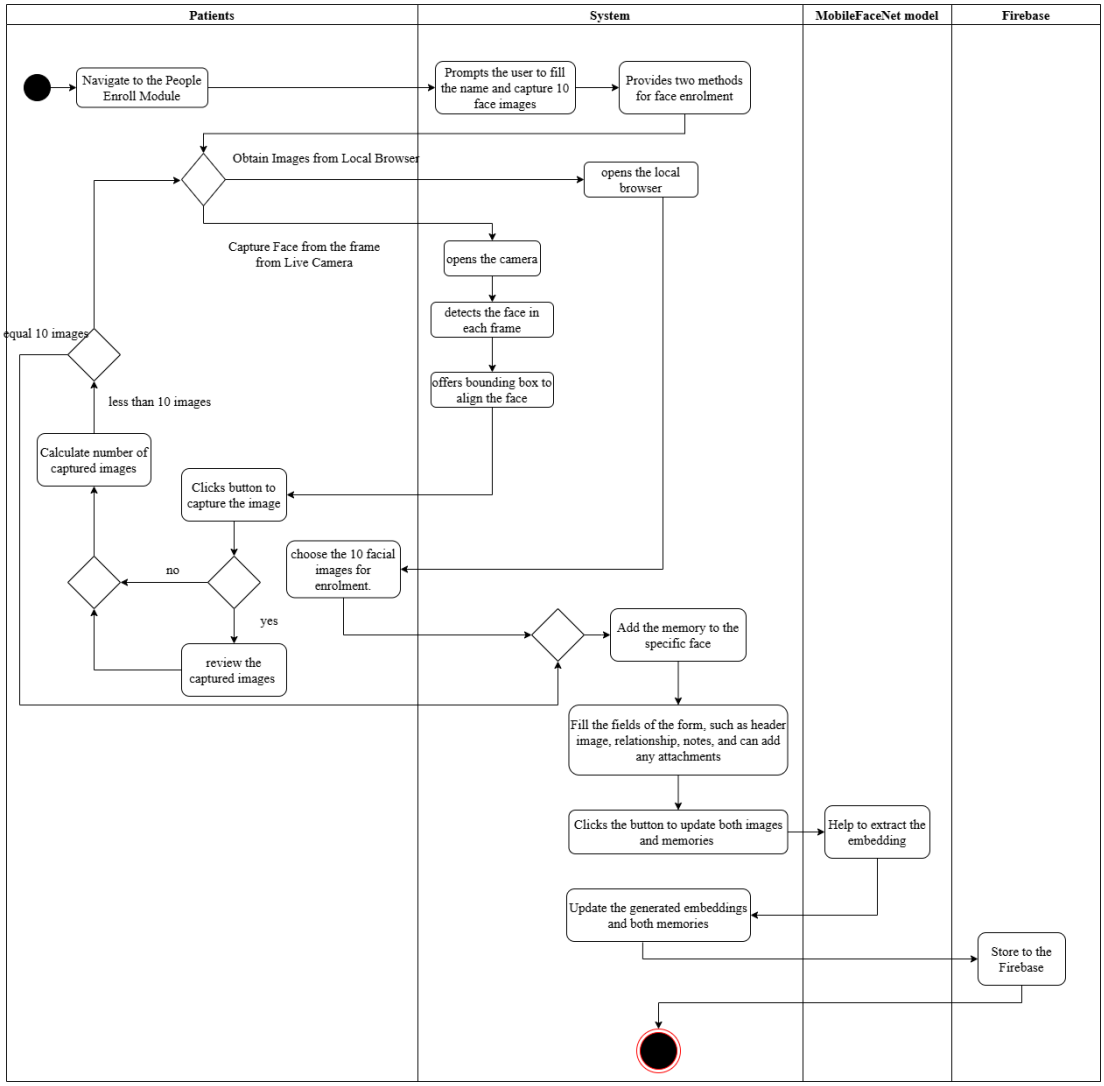


Figure 3.2.8.1 Activity Diagram of People Enrolment Functionality

3.2.9 Activity Diagram of Chat Functionality

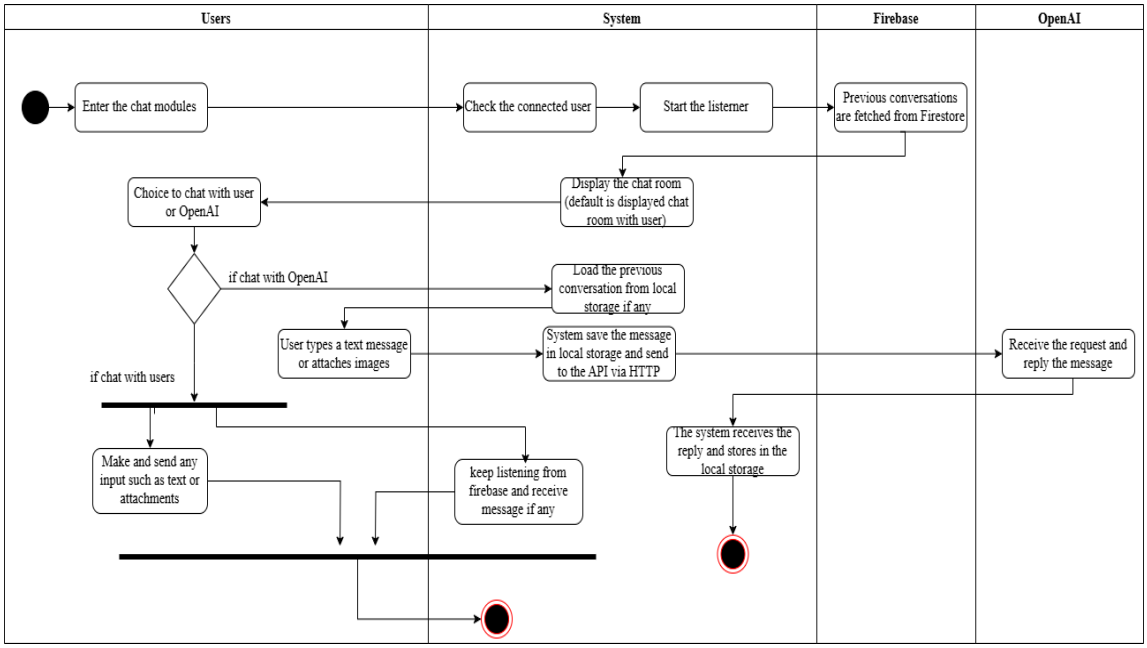


Figure 3.2.9.1 Activity Diagram of Chat Functionality

3.2.10 Activity Diagram of GPS Tracking Functionality

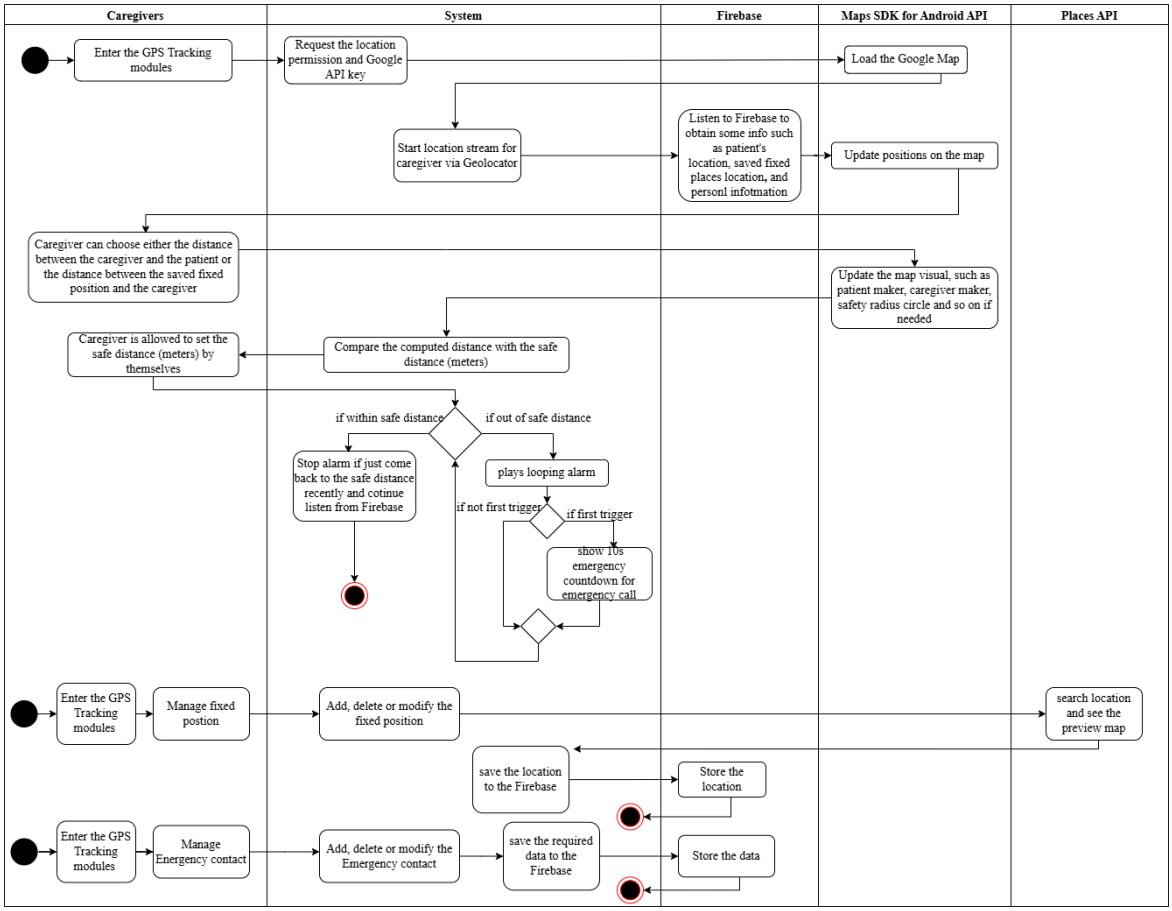


Figure 3.2.10.1 Activity Diagram of GPS Tracking Functionality

3.2.11 Activity Diagram of Return Route Functionality

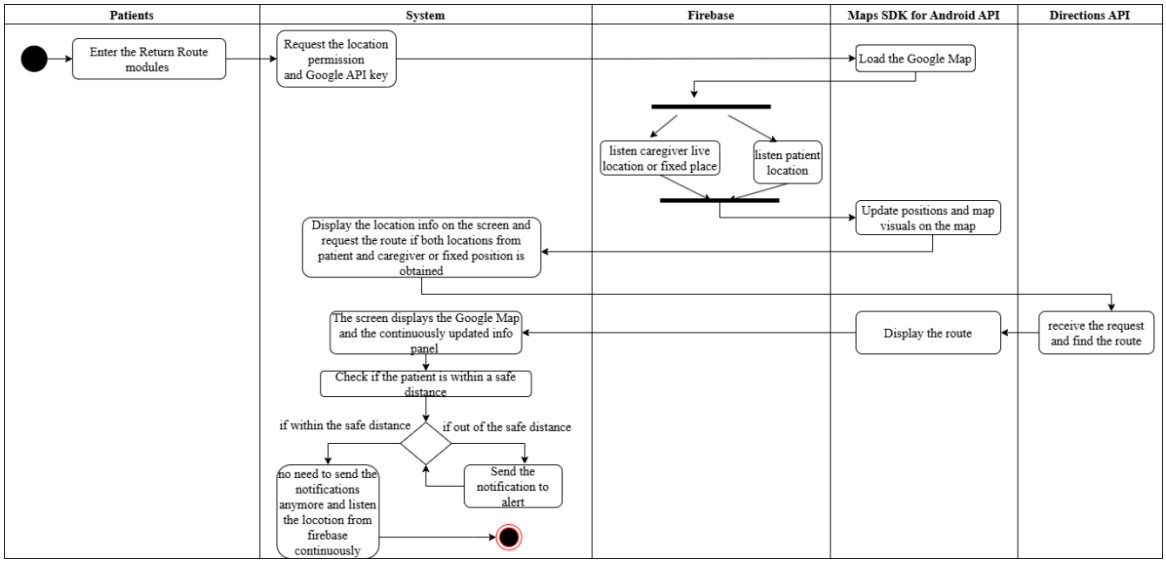


Figure 3.2.11.1 Activity Diagram of Return Route Functionality

3.2.12 Activity Diagram of Information Card Functionality

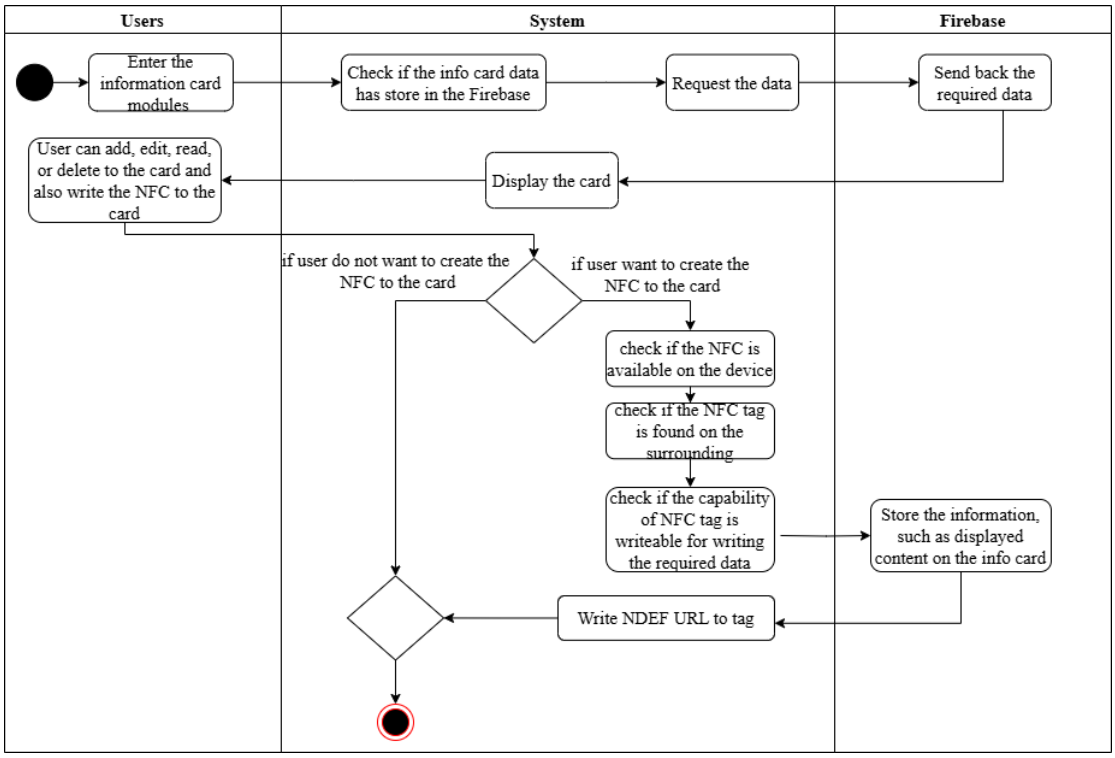


Figure 3.2.12.1 Activity Diagram of Information Card Functionality

3.2.13 Activity Diagram of Gallery Functionality

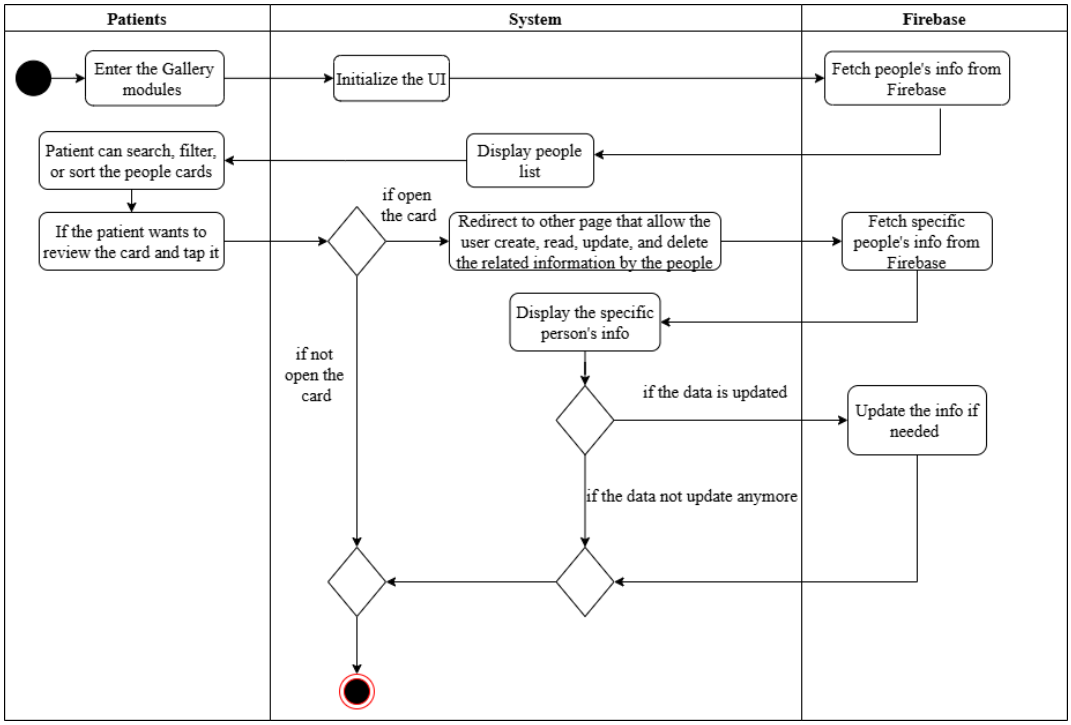


Figure 3.2.13.1 Activity Diagram of Gallery Functionality

3.2.14 Activity Diagram of Diary Functionality

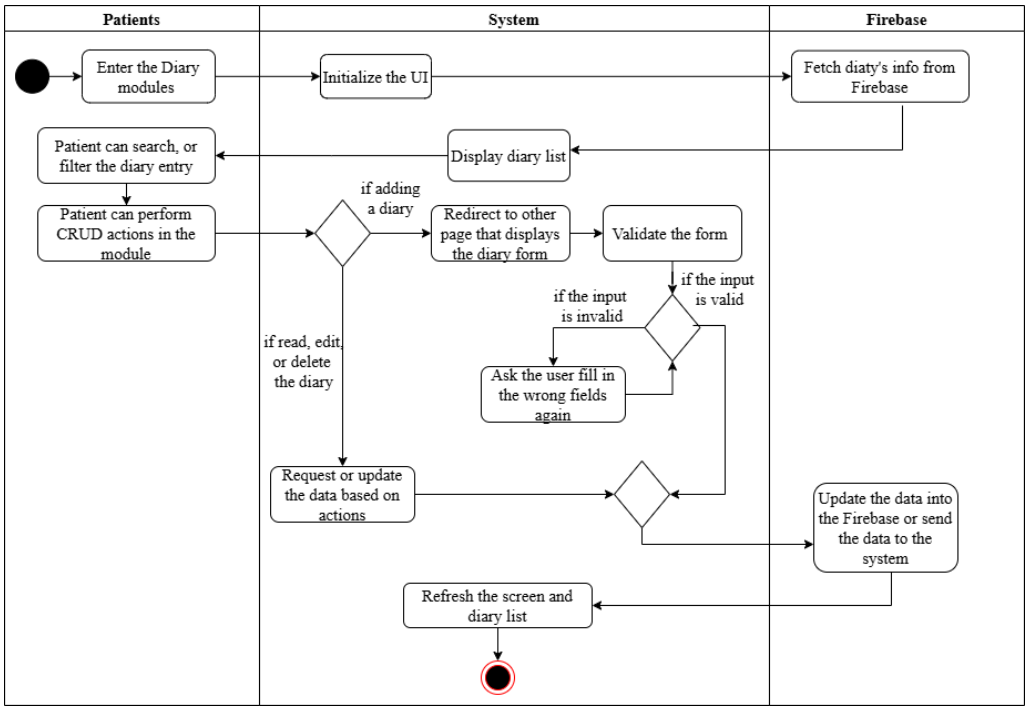


Figure 3.2.14.1 Activity Diagram of Diary Functionality

3.2.15 System Architecture Pattern

Choosing the MVC (Model-View-Controller) architectural pattern for developing the application is a well-considered decision based on several key reasons. First and foremost, the MVC pattern enforces a clear separation of concerns, such that the core functionality of the system is separated into three distinct parts: the Model, which does all the business logic and data management; the View, concerned with UI display; and the Controller, which acts to intercede between the Model and the View in processing user inputs and determining how to update the Model and View. This enhances the code's maintainability and scalability, where a developer can modify each part independently without affecting the others.

Choosing the MVC (Model-View-Controller) architectural pattern for developing the application is a well-considered decision based on several key reasons. First and foremost, the MVC pattern enforces a clear separation of concerns, such that the core functionality of the system is separated into three distinct parts: the Model, which does all the business logic and data management; the View, concerned with UI display; and the Controller, which acts to intercede between the Model and the View in processing user inputs and determining how to update the Model and View. This enhances the code's maintainability and scalability, where a developer can modify each part independently without affecting the others.

The MVC architecture also lends itself to high reusability and modularity. Since the components are loosely linked with each other, every single component is developed, tested, and reused independently according to need. A modular design like this makes it easier to integrate new features or update existing ones without affecting other components. Besides, the clear separation of responsibilities in the structured MVC architecture facilitates unit testing.

Furthermore, MVC architecture provides multiple views to the user so that the user can access the different interfaces independently, that with different functions. In this project, multiple interfaces that are expected, such as login and signup, face detection and recognition, GPS tracking and communication room interfaces, are provided to the user. MVC provides the view separated from the underlying model and controller so

that the various views can be added easily and effectively without affecting the underlying data and business logic.

In terms of scalability and flexibility, it provides great extensibility to the system. The structure of MVC makes it easy to extend or update modules without disrupting existing systems when the application wants to add some new features. Lastly, the last element in the MVC architecture is the View layer for displaying the fetched data to the user. It gives the chance to render a good user experience. The layer is completely separated from the business logic and data processing of the system, hence making the design of the user interface more intuitive and responsive to users' needs. The choice of the MVC architectural pattern provides a structured, modular, and scalable solution for the application; hence, this fits very well in the implementation of this complex system.

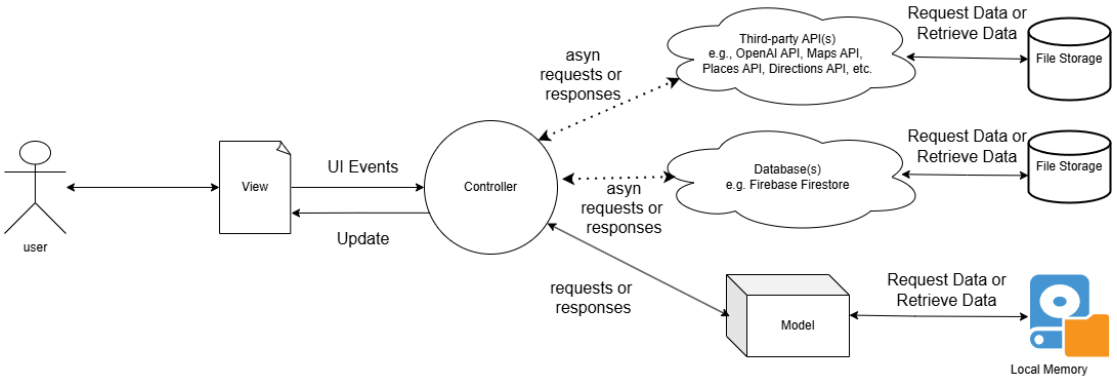


Figure 3.15.1 System Architecture Pattern Diagram

For example, I would like to demonstrate how the MVC architecture is used in the login and signup activities of the mobile application. The application adopts the MVC architecture to structure the login and sign-up functionalities in a modular, maintainable, and testable way. This design pattern separates business logic, UI, and data handling tasks into distinct layers to improve code readability and scalability.

In the view layer, the UI screens, such as the login and sign-up screen, will display to the user first when initializing the application. These views are responsible for displaying input text fields, buttons, and form validation messages for interacting with users.

However, after the UI Events occurred, such as filling in the form and submitting, clicking the button, and so on, the view layer does not process these events directly. Instead, it will forward them to the controller for logic handling. The controller will be triggered and then handle the UI events from the view by requesting the data or information from the model, third-party API, or database, or storing the data in the destination to assist in completing logic handling. This is because the controller is used to handle the logic and interaction between the view and the model. In this case, the signup controller handles tasks like managing sign-up logic, form validation, Firebase authentication, and storing user data in Firestore, while the login controller handles duties like email-password login and error responses.

The Model represents the data structure used during authentication processes. In this case, the model that is used is called the user model, which encapsulates user-related data such as full name, phone number, email and password. The model allows the application to create new user accounts or retrieve user information from Firebase in a consistent and standardized manner. It is also used to transfer the data between layers and store the information in Firestore after successful signup. Once the controller obtains the data, it will handle the tasks and update the data to the view or screen.

In summary, using the MVC patterns looks good and can assist in developing the mobile application. This is because it helps to organize code for each module, separation of concerns, better collaboration between frontend and backend, and easier bug fixing and extension in future phases, like adding GPS tracking and communication functionalities as well as repairing the bug and improving the current developing face recognition function easily

3.3 Timeline

Task Description (FYP1)	Week1	Week2	Week3	Week4	Week5	Week6	Week7	Week8	Week9	Week10	Week11	Week12
Revise the Proposal												
Do Research on the CNN Model Related Topic												
System Specification												
System Design/Overview												
Training the CNN Model												
Optimizing the Model												
Develop Login and Setup Modules												
Integrate the Model into Mobile Application												
Testing and Deploying Current Version												
Update the system design for redesigning and improving current functionality												
Continue developing												
Report Submission												

Figure 3.3.1 Timeline for FYP1

Task Description (FYP2)	Week1	Week2	Week3	Week4	Week5	Week6	Week7	Week8	Week9	Week10	Week11	Week12	Week13	Week14
Refine the Current Version														
Find a better dataset and try new models like FaceNet, mobileNet)														
Building the Model and Testing on the Mobile Application														
Developing Face Detection and Enrolment Functionality														
Developing GPS Tracking and Return Route Functionality														
Developing Chat Functionality														
Developing Diary Functionality														
Developing Information Card Functionality														
Refine the entire app														
Developing for improvement, Testing and Deploying														
Writing Report														
Report Submission														
Presentation														

Figure 3.3.2 Timeline for FYP2

CHAPTER 4 System Implementation

4.1 Software Setup

Installation of Android Studio

First and foremost, Android Studio should be installed as the platform on which we design the Android mobile application. Therefore, go to the Android Studio website and download the version that you want. After the download is completed, run the installer to complete the installation flows, and choose a suitable directory where Android Studio is desired to be installed.

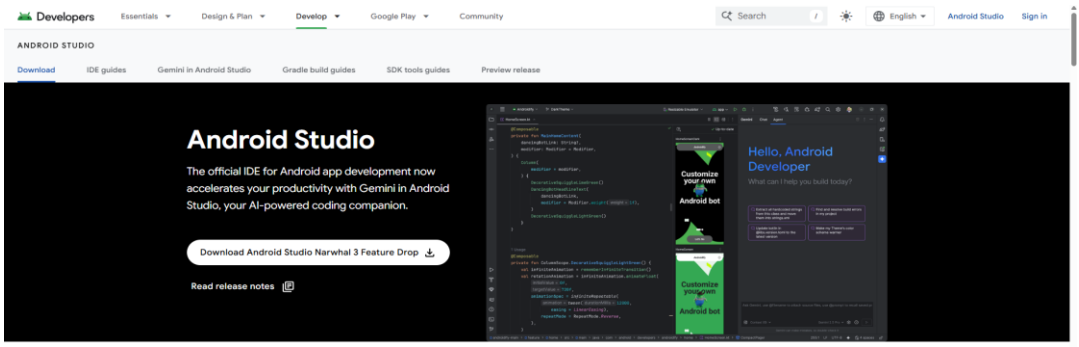


Figure 4.1.1.1 Download Android Studio from the website

Installation of Flutter

After completing the installation of Android Studio, the SDK needs to be downloaded, which is Flutter in this project. Go to the Flutter website to select the suitable Windows version.

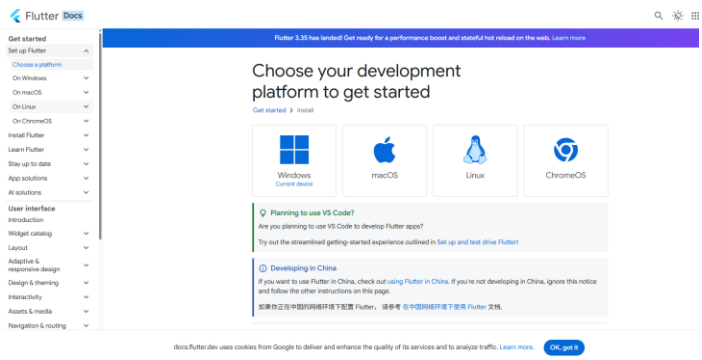


Figure 4.1.1.2 Download Flutter from the website

After extracting the downloaded folder, Flutter needs to be added to the PATH, then open the “Edit the System Environment Variable” which can be found in the laptop’s search bar for quick access, as shown in Figure 4.1.1.3. The System Properties window will pop up and click on the “Environment Variables” as shown in Figure 4.1.1.4. After that, the Environment Variables sub window will pop up and select the Path by following the click of the edit button as shown in Figure 4.1.1.5. The Edit environment variable sub window will pop up and click the New button to add the extracted Flutter folder path into the field as shown in Figure 4.1.1.6.

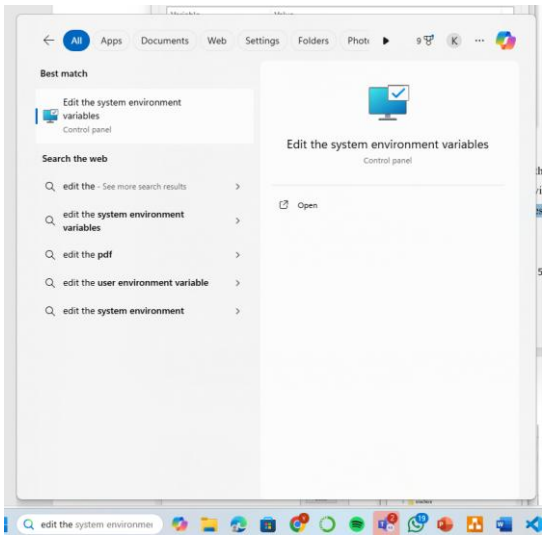


Figure 4.1.1.3 Search “Edit the System Environment Variable”

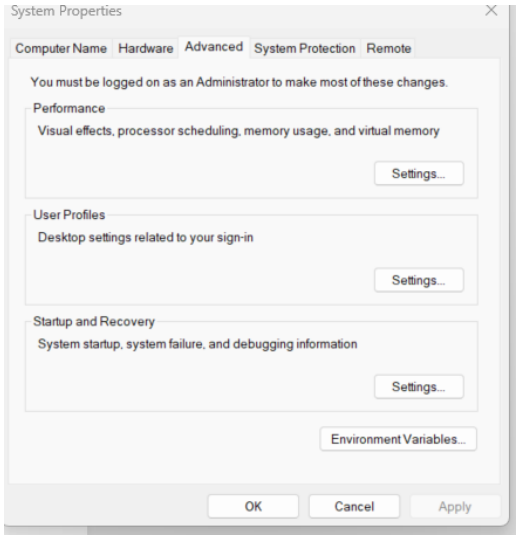


Figure 4.1.1.4 Clicks “Environment Variables”

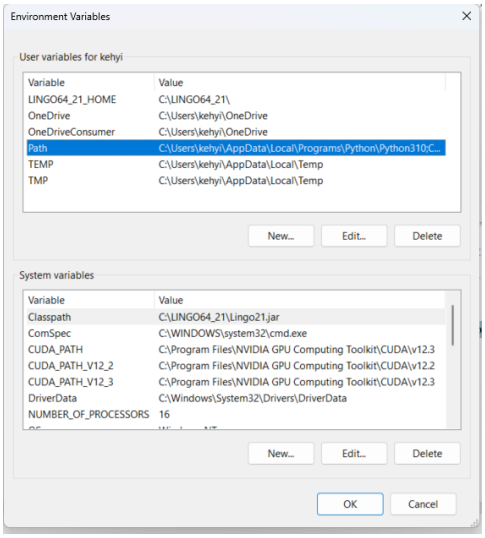


Figure 4.1.1.5 Select the path and click the edit button

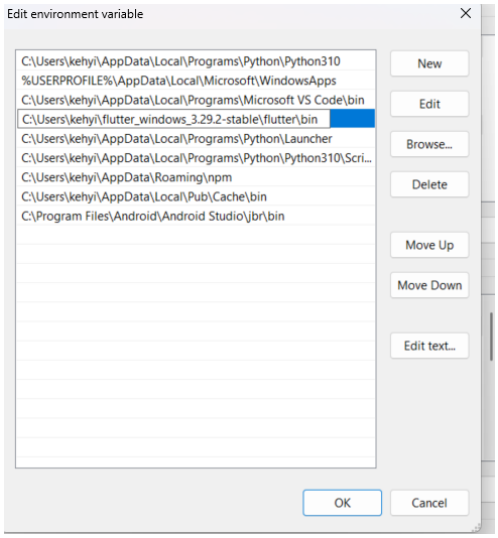


Figure 4.1.1.6 Click the New button to add the Flutter path

To verify the flutter/bin is in PATH, run the command “flutter --version” in Command Prompt as shown in Figure 4.1.1.7.

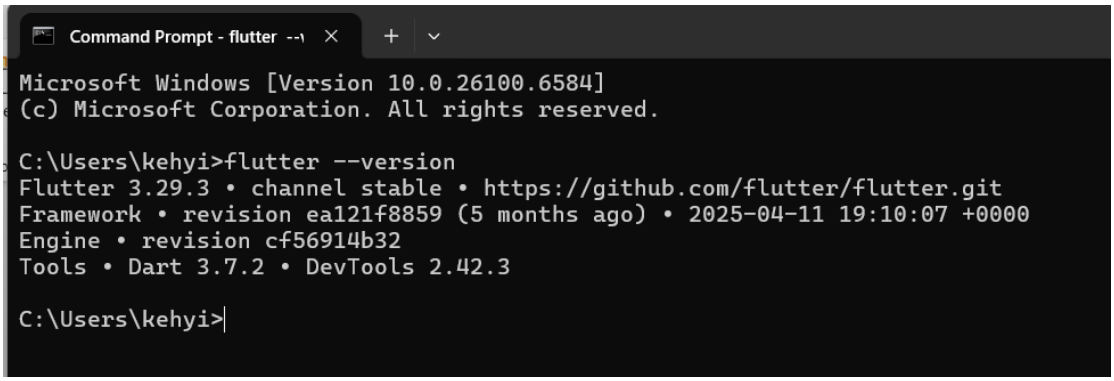


Figure 4.1.1.7 Verify the flutter/bin in PATH

To check all the tools Flutter needs for different platforms, run the command “flutter doctor” in Command Prompt, and ensure as many green check marks as possible in the output. If all green check marks are made, it means Flutter setup is fully correct and ready to use for development, as shown in Figure 4.1.1.8.

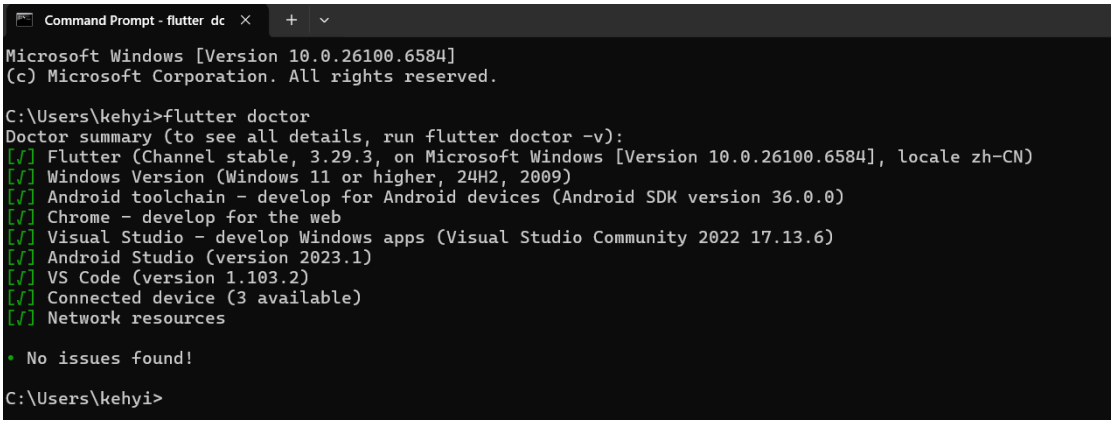


Figure 4.1.1.8 Command's Result

Install Firebase CLI

Install Node.js LTS via the website, as shown in Figure 4.1.1.9, and install the CLI, and then sign in through the command as shown in Figure 4.1.1.10 to ensure the Firebase CLI is installed.

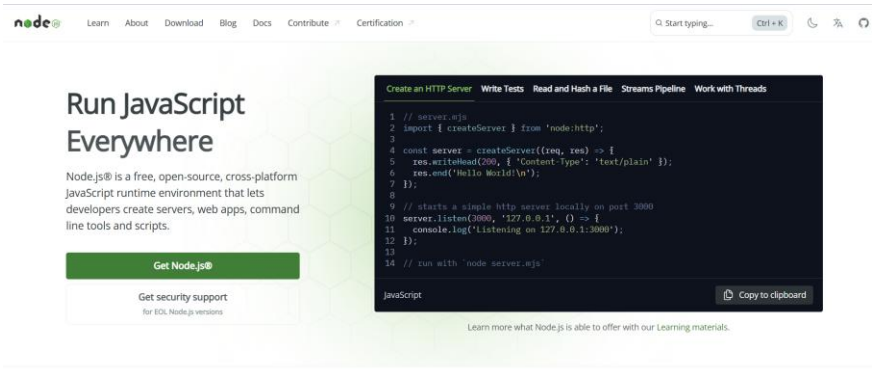


Figure 4.1.1.9 Download the Node.js

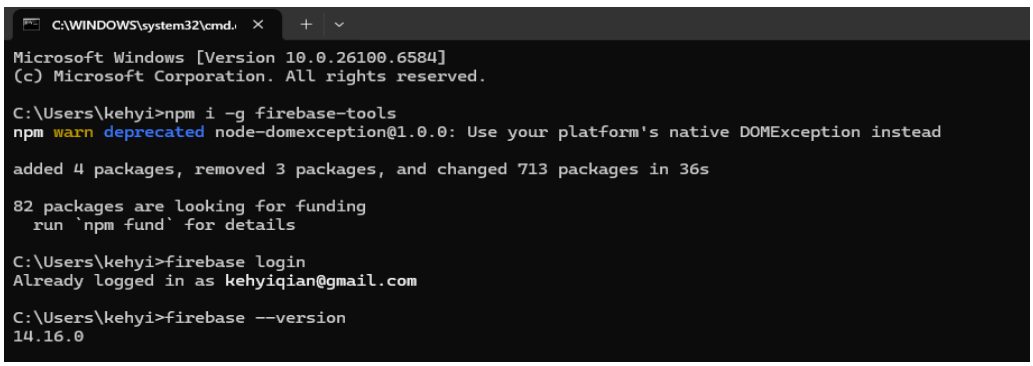


Figure 4.1.1.10 Run the commands

Create the Flutter Project

To create the Flutter project by using Android Studio, the Flutter and Dart plugins must be installed as shown in Figure 4.1.1.11. Then, click the New Flutter Project button to start the creation, select Flutter, and then click the Next button as shown in Figure 4.1.1.12. Then, provide the name of the project, and the Android language must be set as Java, and the Platforms should include Android as shown in Figure 4.1.1.13.

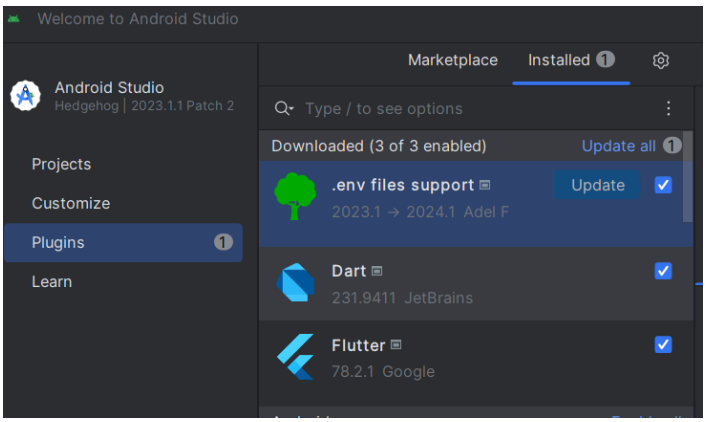


Figure 4.1.1.11 Ensure the Dart and Flutter Plugins are installed

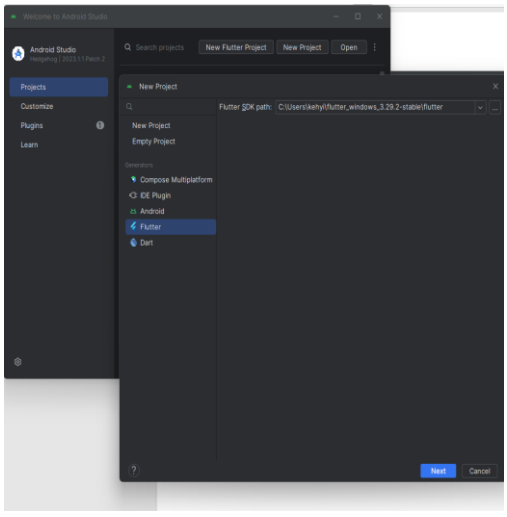


Figure 4.1.1.12 Create the Flutter Project part

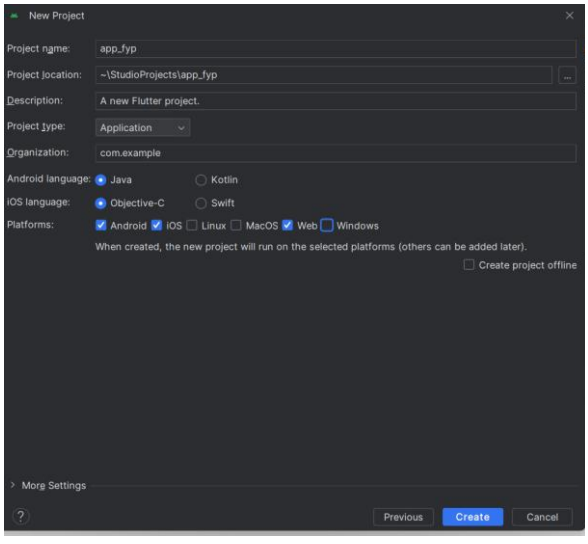


Figure 4.1.1.13 Create the Flutter Project part 2

Connect Flutter to the Firebase Project

Next, go to the Firebase console to create a Firebase project, just follow the provided instructions to complete the creation as shown in Figure 4.1.1.14. Go to the project settings and download the google-service.json as shown in Figure 4.1.1.15 and place it inside the android/app/google-services.json in the Flutter project. Later, edit android/ settings.gradle.kts by adding the Google Services plugin as shown in Figure 4.1.1.16. By following, edit android/app/build.gradle to ensure the com.google.gms.google-services plugin has been added. Both of these make sure Firebase initialization is in the main.dart file can work successfully, so the services such as Firebase Authentication, Firestore Database, Firebase Storage, and so on can perform well.

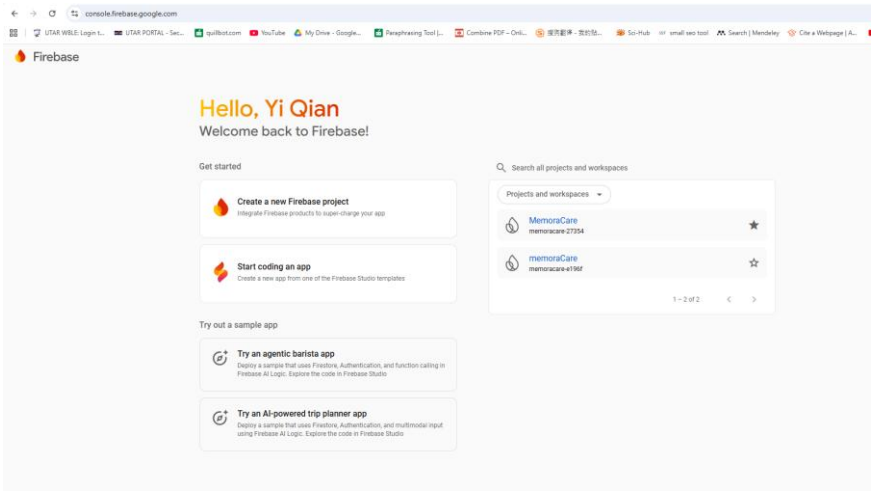


Figure 4.1.1.14 Create a Firebase Project

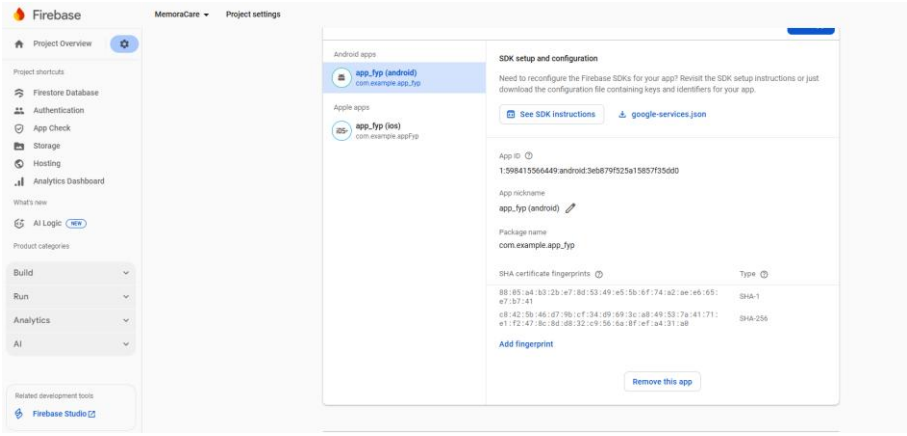


Figure 4.1.1.15 Download google-services.json

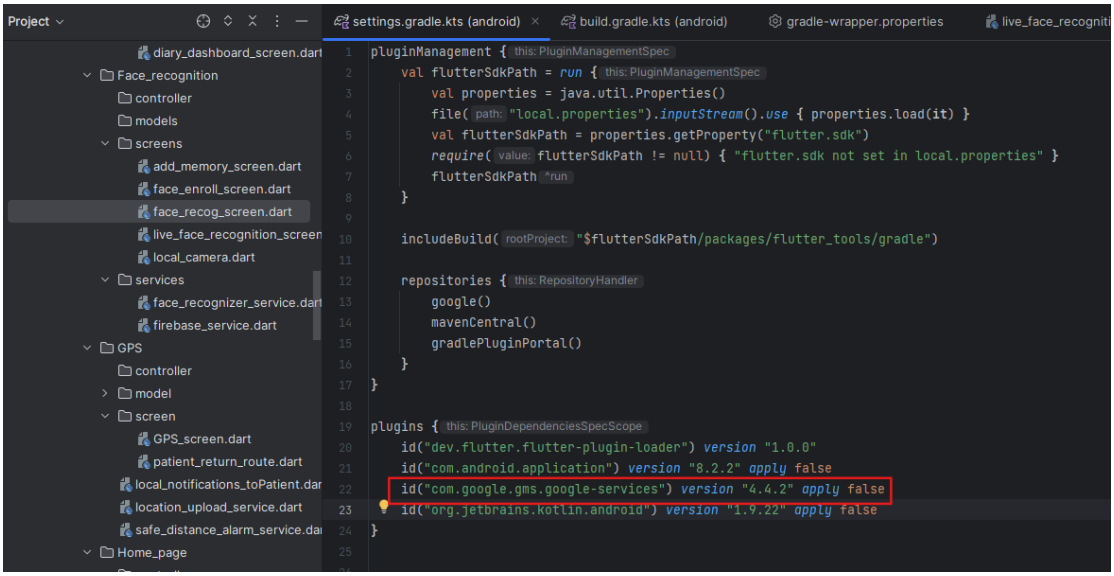


Figure 4.1.1.15 Add the google-services plugin

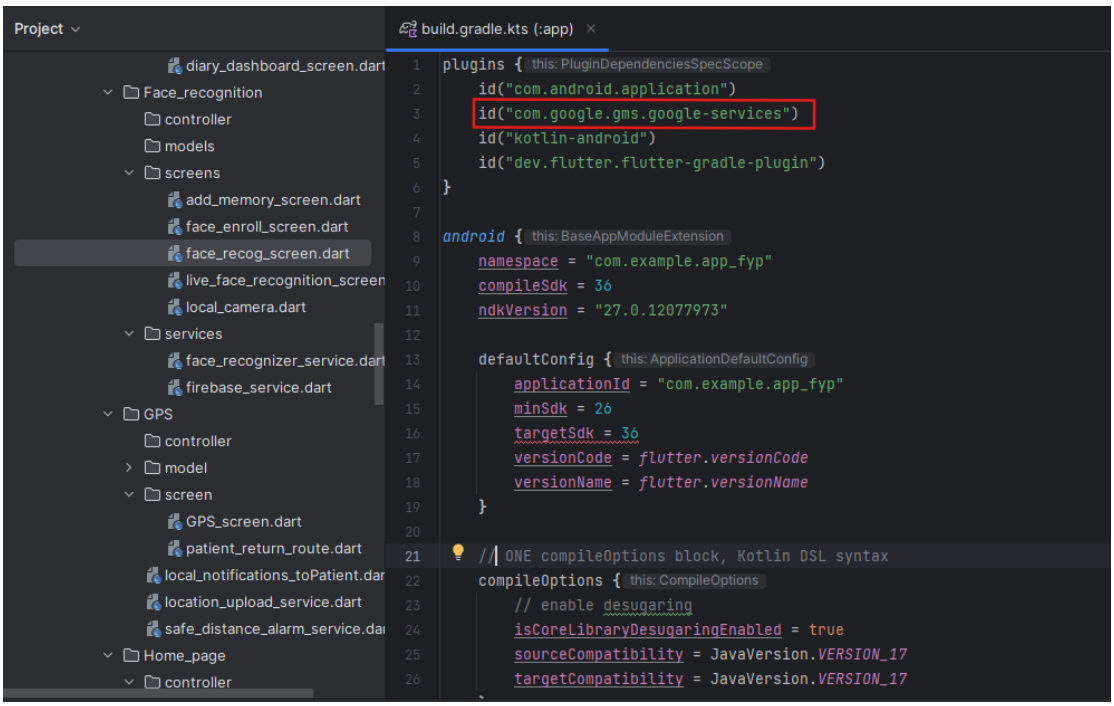


Figure 4.1.1.16 Add the google-services plugin

4.2 Setting and Configuration

Android Manifest Configuration

The AndroidManifest.xml file should define the permissions, features, and configurations required for the MemoraCare Flutter application to operate effectively across its diverse functionalities such as face recognition, chat, GPS tracking, and notifications. Telephony and NFC hardware are declared as optional features to ensure the app can run on devices without SIM support or NFC while still enabling these features if present. The manifest supports the multiple modules of the app includes a comprehensive set of permissions:

Storage Access: READ_EXTERNAL_STORAGE, and WRITE_EXTERNAL_STORAGE

Location Access: ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, and ACCESS_BACKGROUND_LOCATION

Phone and Internet Access: CALL_PHONE, and INTERNET

Notifications: POST_NOTIFICATIONS

NFC: android.permission.NFC

Audio Recording: RECORD_AUDIO

Foreground Services: FOREGROUND_SERVICE, and FOREGROUND_SERVICE_LOCATION

Alarm Scheduling: SCHEDULE_EXACT_ALARM

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.app_fyp">

    <uses-feature
        android:name="android.hardware.telephony"
        android:required="false" />
    <uses-feature android:name="android.hardware.nfc" android:required="false"/>

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
    <uses-permission android:name="android.permission.NFC"/>
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE_LOCATION"/>
    <uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM"/>
```

Figure 4.2.1 Declared Features and Permissions

Intent filters are defined to handle deep links from the app hosted Firebase domain which is memoracare-27354.web.app/tag. This allows seamless navigation into the app from external links.

```xml
    />
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>

    <intent-filter android:autoVerify="true">
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data
            android:scheme="https"
            android:host="memoracare-27354.web.app"
            android:pathPrefix="/tag"/>
    </intent-filter>
```

Figure 4.2.2 Deep Link Configuration with Host and Path Prefix

The manifest also embeds the Google Maps API Key to enable map-related functionalities such as route fetching and geolocation display.

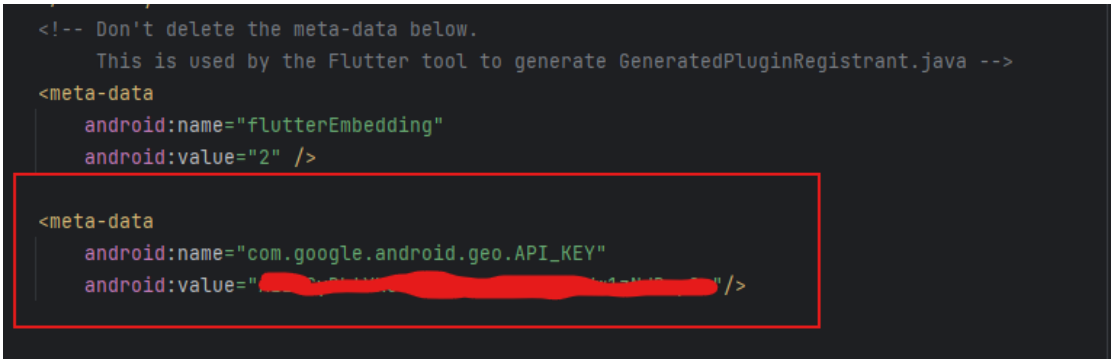


Figure 4.2.3 Google Maps API Key (value is hidden to avoid leak)

Configure OpenAI API Key

The project uses an environment file (.env) to store private keys such as the OpenAI API key to ensure secure handling of sensitive credentials. This practice prevents hardcoding secrets directly into the source code, thereby reducing the risk of accidental leaks when code is shared publicly such as in GitHub repositories. By placing the key inside a separate file, only the environment file needs to be secured or ignored in version control, while the source code remains clean and safe. The flutter_dotenv package is used to load and access environment variables at runtime. The dotenv.load() is used to read the .env file, and dotenv.env['OPENAI_API_KEY'] is used to assign the API key to the OpenAI client to enable safe integration of external services.

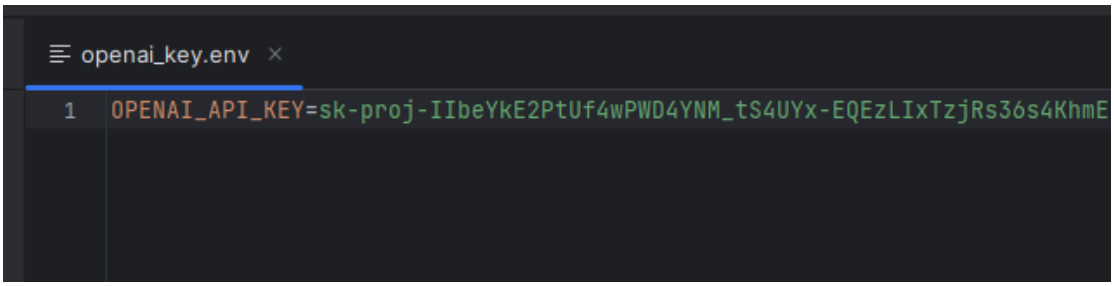


Figure 4.2.4 openai_key_env

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await dotenv.load(fileName: "openai_key.env");
  OpenAI.apiKey = dotenv.env['OPENAI_API_KEY'] ?? "";

  await GetStorage.init();

  FlutterNativeSplash.preserve(widgetsBinding: WidgetsBinding.instance);

  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform)
      .then((_) => Get.put(AuthenticationRepository()));

  await FirebaseAppCheck.instance.activate(
    androidProvider: kReleaseMode ? AndroidProvider.playIntegrity : AndroidProvider.debug,
  );

  FlutterForegroundTask.initCommunicationPort();

  runApp(const App());
  await initLocalNotifications();
  FlutterNativeSplash.remove();
}
```

Figure 4.2.5 Load OpenAI API Key

FireStore Database Security Rule

The Firestore Security Rule is crucial in this project because it will help to define to simplify and organize access control checks. Therefore, it can reduce repetition, improve readability, and centralize logic such as authentication checks, ownership validation, or verifying relationships between users.

First and foremost, there are a set of helper functions such as isSignedIn(), isOwner(), areConnected(), and rolesDiffer() to encapsulate common checks such as authentication, ownership validation, and relationship status. This modular approach improves readability and maintainability.

For the Users collection, each user can only read or write their own profile and subcollections such as People, Embeddings, and Memories. Info cards extend access to linked caregivers, while the locations subcollection allows patients to write their own locations and grants read-only access to caregivers who are explicitly connected. Contacts remain fully private that is accessible only to the owner.

The PublicCards collection is openly readable to all users but restricts write access to authenticated users only to maintain openness while ensuring that data can only be written from inside the application.

The Chats collection enforces strong access control: new chat rooms can only be created if both participants are distinct, authenticated, connected in Firestore, and have different roles which is patient and caregiver. Only participants can read, update, or delete chat documents. For chat messages, participants can read and create new entries, but messages are immutable after creation to prevent tampering. Strict field validation ensures that only whitelisted fields such as senderId, type, text, imageUrl, createdAt, and isRead are written.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    match /{document=**} {
      allow read, write: if
        request.time < timestamp.date(2025, 10, 22);
    }
     /* ---------- Helper Functions ---------- */
    function isSignedIn() {
      return request.auth != null;
    }

    function user(uid) {
      return get(/databases/$(database)/documents/Users/$(uid)).data;
    }

    function isOwner(uid) {
      return isSignedIn() && request.auth.uid == uid;
    }

    function isLinkedCaregiver(patientUid) {
      return isSignedIn()
        && exists(/databases/$(database)/documents/Users/$(patientUid)/connections/$(request.auth.uid))
        && get(/databases/$(database)/documents/Users/$(patientUid)/connections/$(request.auth.uid)).data.status == "accepted";
    }

    function isParticipant(room) {
      return isSignedIn() && (
        room.data.p0 == request.auth.uid ||
        room.data.p1 == request.auth.uid ||
        (room.data.participants != null && room.data.participants[request.auth.uid] == true)
      );
    }

    function areConnected(uid1, uid2) {
      let u1 = user(uid1);
      let u2 = user(uid2);

      let r1 = (u1.Role != null) ? u1.Role : u1.role;
      let r2 = (u2.Role != null) ? u2.Role : u2.role;

      let u1List = (r1 == 'caregiver') ? u1.connectedPatients
                 : (r1 == 'patient')   ? u1.connectedCaregivers
                 : u1.connectedUsers;
      let u2List = (r2 == 'caregiver') ? u2.connectedPatients
                 : (r2 == 'patient')   ? u2.connectedCaregivers
                 : u2.connectedUsers;

      return (u1List != null && u1List.hasAny([uid2])) ||
             (u2List != null && u2List.hasAny([uid1]));
    }

```

Figure 4.2.6 Firestore Security Rules Part 1

```
    function rolesDiffer(uid1, uid2) {
      let a = (user(uid1).Role != null) ? user(uid1).Role : user(uid1).role;
      let b = (user(uid2).Role != null) ? user(uid2).Role : user(uid2).role;
      return a != null && b != null && a != b;
    }

    /* ---------- User Profile and Subcollections ---------- */
    match /Users/{userId} {
      allow read, write: if isOwner(userId);

      match /People/{pid} {
        allow read, write: if isOwner(userId);

        match /Embeddings/{eid} {
          allow read, write: if isOwner(userId);
        }

        match /Memories/{mid} {
          allow read, write: if isOwner(userId);
        }
      }

      match /InfoCards/{cardId} {
        allow read, write: if isOwner(userId) || isLinkedCaregiver(userId);
      }

     match /locations/{locId} {
      allow write: if isOwner(userId);
      allow read: if isOwner(userId) ||
        (isSignedIn() &&
         (
           ((user(request.auth.uid).Role != null)
             ? user(request.auth.uid).Role
             : user(request.auth.uid).role) == 'caregiver'
         ) &&
         user(request.auth.uid).connectedPatients != null &&
         user(request.auth.uid).connectedPatients.hasAny([userId])
        );
    }


      match /Contacts/{peerUid} {
        allow read, write: if isOwner(userId);
      }
    }

    /* ---------- Public Cards ---------- */
    match /PublicCards/{cid} {
      allow read: if true;
      allow write: if isSignedIn();
    }

```

Figure 4.2.7 Firestore Security Rules Part 2

```

    /* ---------- Chats (1:1) ---------- */
    match /chats/{roomId} {
      allow create: if isSignedIn() &&
        (
          (
            request.resource.data.p0 is string &&
            request.resource.data.p1 is string &&
            request.resource.data.p0 != request.resource.data.p1 &&
            (request.auth.uid == request.resource.data.p0 || request.auth.uid == request.resource.data.p1) &&
            areConnected(request.resource.data.p0, request.resource.data.p1) &&
            rolesDiffer(request.resource.data.p0, request.resource.data.p1)
          ) ||
          (
            request.resource.data.participants is map &&
            request.resource.data.participants.size() == 2 &&
            request.resource.data.participants[request.auth.uid] == true &&
            areConnected(
              request.resource.data.participants.keys()[0],
              request.resource.data.participants.keys()[1]
            ) &&
            rolesDiffer(
              request.resource.data.participants.keys()[0],
              request.resource.data.participants.keys()[1]
            )
          )
        );

      allow read, update, delete: if isParticipant(resource);

      match /messages/{msgId} {
        allow read: if isParticipant(get(/databases/$(database)/documents/chats/$(roomId)));
        allow create: if isParticipant(get(/databases/$(database)/documents/chats/$(roomId))) &&
                      request.resource.data.senderId == request.auth.uid &&
                      request.resource.data.keys().hasOnly(
                        ['senderId','type','text','imageUrl','createdAt','isRead']
                      );
        allow update, delete: if false;
      }
    }
  }
}

```

Figure 4.2.8 Firestore Security Rules Part 3

4.3 Code Explanation

4.3.1 Authentication Module

Sign up with an Email and a Password

Figure 4.3.1.1 shows the function that creates a new user account in Firebase with an email and password, and the function sends a verification email to the currently logged-in user. A new user signs up with an email and a password, then the registerWithEmailAndPassword() runs. After successful registration, sendEmailVerification() is called to allow the user to receive an email to verify their account.

```
//--EmailAuthentication - REGISTER
Future<void> registerWithEmailAndPassword(String email, String password) async {
  try {
    await _auth.createUserWithEmailAndPassword(email: email, password: password);
  } on FirebaseAuthException catch (e) {
    final ex = TExceptions.fromCode(e.code);
    throw ex.message;
  } catch (_) {
    const ex = TExceptions();
    throw ex.message;
  }
}

// Email Verification
Future<void> sendEmailVerification() async {
  try {
    await _auth.currentUser?.sendEmailVerification();
  } on FirebaseAuthException catch (e) {
    final ex = TExceptions.fromCode(e.code);
    throw ex.message;
  } catch (_) {
    const ex = TExceptions();
    throw ex.message;
  }
}
```

Figure 4.3.1.1 Script of Email Authentication and Verification

Sign Up with Google

This function allows users to sign in with their Google account. It returns a UserCredential that can be null, which represents the Firebase user information. The system will open Google Sign-In dialog. If the user cancels, googleUser is null, so it stops and returns null. The accessToken and idToken will be retrieved from the Google account. These are needed to prove identity to Firebase. Then, these Google tokens will convert into a Firebase credential object, and Firebase will use this to authenticate the user. Sign the user into Firebase using the generated Google credential. The Firebase will check whether the user is new; if yes, the app takes the user to RoleSelectionScreen, otherwise, the app takes the user to the proper dashboard.

```
Future<UserCredential?> signInWithGoogle() async {
  try {
    final GoogleSignInAccount? googleUser = await GoogleSignIn().signIn();
    if (googleUser == null) return null;

    final GoogleSignInAuthentication? googleAuth = await googleUser.authentication;

    final credential = GoogleAuthProvider.credential(
      accessToken: googleAuth?.accessToken,
      idToken: googleAuth?.idToken,
    );

    final userCredential = await FirebaseAuth.instance.signInWithCredential(credential);
    final isNewUser = userCredential.additionalUserInfo?.isNewUser ?? false;
    final user = userCredential.user;

    if (user == null) return null;

    if (isNewUser) {
      // Prepare temp user model for role selection
      final tempUser = UserModel(
        email: user.email ?? '',
        fullName: user.displayName ?? '',
        phoneNo: user.phoneNumber ?? '',
        password: '', // Not used for Google sign-in
      );

      Get.to(() => RoleSelectionScreen(tempUser: tempUser));
    } else {
      // Go to dashboard based on stored role
      AuthenticationRepository.instance.setInitialScreen(user);
    }

    return userCredential;
  } on FirebaseAuthException catch (e) {
    final ex = TExceptions.fromCode(e.code);
    throw ex.message;
  } catch (e) {
    const ex = TExceptions();
    throw ex.message;
```

Figure 4.3.1.2 Sign In with Google

Log in with Email and Password

The function is designed to enable the user to log in with email and password and validate the input. When the user logs in with email and password, the system checks if the form inputs are valid. If the email and password are not valid, the process is stopped. Otherwise, call Firebase Authentication and send the email and password to it. If valid, Firebase logs in the user and redirects the user to the proper dashboard. If wrong, it throws an error.

```
Future<void> login() async {
  try {
    isLoading.value = true;
    if (!loginFormKey.currentState!.validate()) {
      isLoading.value = false;
      return;
    }
    final auth = AuthenticationRepository.instance;
    await auth.loginWithEmailAndPassword(email.text.trim(), password.text.trim());
    auth.setInitialScreen(auth.firebaseUser);
  } catch (e) {
    isLoading.value = false;
    Helper.errorSnackBar(title: tOhSnap, message: e.toString());
  }
}
```

Figure 4.3.1.3 Log In with Email and Password

4.3.2 Chat Module

Chat ID generator

This function is important because each pair of users should have only one unique chat room in a 1 to 1 chat system. The deterministic formula that always produces the same Chat ID is needed for the same two users, regardless of order. The function will remove any accidental spaces in user IDs to ensure consistency, and the compareTo() method will be used to compare user IDs alphabetically and then build the chat ID based on the results from compareTo(). This ensures the same chat ID no matter the order of user IDs and prevents duplicate chat rooms.

```dart
String chatIdFor(String uidA, String uidB) {
  final a = uidA.trim(), b = uidB.trim();
  return (a.compareTo(b) < 0) ? '${a}_$b' : '${b}_$a';
}
```

Figure 4.3.2.1 Chat ID Generator

Listening to Active Peer Selection

The _listenActivePeer() function monitors the logged-in user's Firestore document to manage real-time chat sessions. It begins by confirming the user authentication status and then establishes a listener on the user record. The function normalizes the user role, which is patients and caregivers, and checks the activePeerUid field to determine the current chat partner. If no partner is set, it clears the chat state and cancels any existing peer subscriptions. When a new partner is detected, it updates the local state, subscribes to the peer profile to display their name or email, and ensures a valid chat room exists between both users. In case of errors, it resets the state and shows an error message. Overall, the function guarantees dynamic role recognition, seamless partner switching, and consistent chat room management.

```dart
void _listenActivePeer() {
  final me = FirebaseAuth.instance.currentUser;
  if (me == null) return;

  _meSub?.cancel();
  _meSub = FirebaseFirestore.instance
      .collection('Users')
      .doc(me.uid)
      .snapshots()
      .listen((doc) async {
    final data = doc.data() ?? {};

    //normalize the role
    final rawRole = (data['role'] ?? data['Role'] ?? data['userRole'] ?? data['UserRole'])
        ?.toString()
        .trim()
        .toLowerCase();
    final detectedRole = (rawRole == 'patient' || rawRole == 'caregiver')
        ? rawRole
        : ((data['isCaregiver'] == true) ? 'caregiver' : (data['isPatient'] == true ? 'patient' : null));
    if (detectedRole != _myRole && mounted) {
      setState(() => _myRole = detectedRole);
    }

    final peerUid = (data['activePeerUid'] as String?)?.trim();

    // no active partner
    if (peerUid == null || peerUid.isEmpty) {
      setState(() {
        _otherUid = null;
        _otherName = 'No active partner';
        _items = [];
        _latestDocs = [];
        _indexByDocId.clear();
      });
      _peerSub?.cancel();
      return;
    }
```

Figure 4.3.2.2 Listening to Active Peer Selection

```
  if (peerUid != _otherUid) {
    setState(() {
      _otherUid = peerUid;
      _otherName = 'Chat';
      _items = [];
      _latestDocs = [];
      _indexByDocId.clear();
    });

    // resolve peer name live
    _peerSub?.cancel();
    _peerSub = FirebaseFirestore.instance
        .collection('Users')
        .doc(peerUid)
        .snapshots()
        .listen((peerDoc) {
      final d = peerDoc.data() ?? {};
      final full = (d['FullName'] ?? '').toString().trim();
      final display = full.isEmpty ? (d['Email'] ?? 'Unknown').toString() : full;
      if (mounted) setState(() => _otherName = display);
    });

    // ensure chat doc exists (only if connected)
    try {
      await ChatRoomService.instance.ensureChatExists(peerUid);
    } catch (e) {
      if (!mounted) return;
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Cannot start chat: $e')),
      );
      setState(() {
        _otherUid = null;
        _otherName = 'No active partner';
      });
    }
  }
});
}
```

Figure 4.3.2.3 Listening to Active Peer Selection

Voice Message Recording and Sending

The _toggleRecord() function handles both starting and stopping voice recording within the chat feature. When the user initiates recording, the function first checks for microphone permission and displays a warning if access is denied. If permitted, it creates a temporary file path by using the current timestamp and begins recording audio with specified configuration parameters. When recording, this function also subscribes to the amplitude update every 100 milliseconds to generate normalized peaks of the visual waveform display and keeps the size of the peak list by extracting old values. A timer updates the recording duration in seconds for UI display.

When the recording is stopped, the recorder halts, and all subscriptions and timers are canceled. The system will verify the existence of the recorded file, and its duration will be calculated. If an active chat partner is present, the audio file, its metadata such as filename, duration, waveform peaks, and content are sent to Firebase using the ChatRoomService.sendAudio() method, to ensure synchronization between users. After sending it, the function automatically scrolls the chat view to the most recent message for a seamless user experience.

```
Future<void> _toggleRecord() async {
  if (!_isRecording) {
    if (!await _recorder.hasPermission()) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text("Microphone permission is required.")),
      );
      return;
    }
    final dir = await getTemporaryDirectory();
    _recordPath = '${dir.path}/vm_${DateTime.now().millisecondsSinceEpoch}.m4a';

    await _recorder.start(
      RecordConfig(encoder: AudioEncoder.aacLc, bitRate: 128000, sampleRate: 44100),
      path: _recordPath!,
    );
    _peaks.clear();
    _ampSub?.cancel();
    _ampSub = _recorder
        .onAmplitudeChanged(const Duration(milliseconds: 100))
        .listen((amp) {
      final p = _normAmp(amp);
      if (_peaks.length < 120) { // cap bars
        _peaks.add(p);
      } else {
        // simple decimation: keep every second bar
        for (int i = 0; i < _peaks.length - 1; i += 2) {
          _peaks[i ~/ 2] = ((_peaks[i] + _peaks[i + 1]) ~/ 2);
        }
        _peaks.removeRange(_peaks.length ~/ 2, _peaks.length);
        _peaks.add(p);
      }
    });
```

Figure 4.3.2.4 Voice Message Recording and Sending

```
    _recStart = DateTime.now();
    setState(() {
      _isRecording = true;
      _recSeconds = 0;
    });
    _recTimer?.cancel();
    _recTimer = Timer.periodic(const Duration(seconds: 1), (_) {
      if (!mounted) return;
      setState(() => _recSeconds++);
    });  // Timer.periodic
  } else {
    final path = await _recorder.stop();
    _ampSub?.cancel();
    _recTimer?.cancel();
    _recTimer = null;

    setState(() => _isRecording = false);

    if (path == null || !File(path).existsSync()) return;

    final durationMs = DateTime.now().difference(_recStart ?? DateTime.now()).inMilliseconds;
```

Figure 4.3.2.5 Voice Message Recording and Sending

```
    // send to Firestore/Storage (syncs for both users)
    final other = _otherUid;
    if (other != null) {
      try {
        await ChatRoomService.instance.sendAudio(
          other,
          File(path),
          fileName: 'voice_${DateTime.now().millisecondsSinceEpoch}.m4a',
          durationMs: durationMs,
          peaks: List<int>.from(_peaks),
        );
      } catch (e) {
        if (!mounted) return;
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text('Voice send failed: $e')),
        );
      }
    }

    // scroll to bottom a moment later
    await Future.delayed(const Duration(milliseconds: 120));
    if (_items.isNotEmpty) {
      _itemScrollCtrl.scrollTo(
        index: _items.length - 1,
        duration: const Duration(milliseconds: 220),
        curve: Curves.easeOut,
      );
    }
  }

}
```

Figure 4.3.2.6 Voice Message Recording and Sending

Role-aware and Vision-aware System Prompt

The _composeSystemPrompt() function dynamically generates a system instruction string to guide how OpenAI should respond, tailored to both the user role and the presence of images. The base instruction is role-dependent: if the user is a patient, the OpenAI is instructed to use simple, warm, plain language while avoiding medical diagnoses; if the user is a caregiver, the assistant is directed to be concise, practical, and focused on actionable information and safety.

The function then appends additional guidance depending on whether an image is attached. When images are attached, OpenAI is instructed to prioritize describing what is visible in the image over prior chat content, provide a brief identification in 1 to 3 words, and follow it with a clear and detailed explanation. Without this instruction, the model could ignore or underweight the image context, and this ensures that visual analysis comes first. Considering that the patient has a cognitive problem, they may not even know what an object is or the object's function in the severe stage. Several rules

also enforce clarity and safety, such as avoiding speculation about brands, avoiding programming and code unless asked, providing up to two plausible options if uncertain, and being cautious when describing medication to avoid giving the wrong instructions to the patients and caregivers, because OpenAI may not be as professional as a doctor or medical specialist.

```
String _composeSystemPrompt({required bool hasImages}) {
  final base = (widget.role.toLowerCase() == "patient")
      ? "You are a warm, plain-language assistant for patients. Keep answers simple and short. Avoid jargon. Do not provide medical diagnoses."
      : "You are a concise, practical assistant for caregivers. Be clear and to the point. Prioritize actionable information and safety tips when relevant.";

  final vision = hasImages
      ? """
      When an image is attached, prioritize what you SEE in the image over any prior chat.
      If the user asks 'what is this' or similar, IDENTIFY the primary visible object or device.

      Formatting:
      - First line: the object in 1-3 words (e.g., "Laptop computer").
      - Second line: provide clearly and detailed explanation.

      Rules:
      - Do NOT mention programming/code unless explicitly asked.
      - Do NOT guess brand/model or unseen details. If unsure, say "Not sure—looks like …".
      - If multiple plausible options, give up to two (e.g., "Laptop or thin client").
      - If it appears to be medication: describe shape/color/imprint, avoid confirming identity; advise checking packaging or a pharmacist.
      - Keep it brief.
      """
            : """
      When no image is attached, answer concisely. Prefer simple, direct language appropriate for the user's role.
      """;

  return "$base\n$vision";
}
```

Figure 4.3.2.4 Role-aware and Vision-aware System Prompt

Maintain Chat History with Local Storage

This code manages the saving and restoring of chat history using GetStorage, a lightweight local storage solution in Flutter. A private storage key is defined dynamically based on the user role to ensure that each role, which includes patients and caregivers, has its own separate chat history. The _restoreConversation() function reads data from local storage under this key. If the stored data exists and is a list, it is converted back into a list of maps representing chat messages, assigned to _messages, and the UI is refreshed with setState(). Additionally, _scrollToBottom() is called to ensure the most recent messages are visible. On the other hand, _persistConversation() is responsible for saving the current list of _messages into local storage whenever changes occur.

```dart
final GetStorage _box = GetStorage();
String get _storageKey => 'openai_chat_history_${widget.role}';
void _restoreConversation() {
  final saved = _box.read(_storageKey);
  if (saved is List) {
    _messages =
        saved.map<Map<String, dynamic>>((e) => Map<String, dynamic>.from(e)).toList();
    setState(() {});
    _scrollToBottom(delayed: true);
  }
}

void _persistConversation() {
  _box.write(_storageKey, _messages);
}
```

Figure 4.3.2.5 Maintain Chat History with Local Storage

4.3.3 Diary Module

Firestore Query for Searching and Filtering

The code constructs a dynamic Firestore query to retrieve diary entries for a specific user to apply filters for search terms and optional date constraints. It begins by retrieving the user diary collection, sorted by creation time in descending order so the more recent entries appear first. If the user provides a search string, the query is refined based on the search mode. In hashtag search mode, the string is normalized into a hashtag, and entries containing that hashtag in their hashtags array field are returned. In person search mode, the string is normalized into a person identifier, and only entries referencing that person in the people array field are included. Furthermore, if a date filter is provided, the query is further restricted to include only diary entries created on that specific day. This can be achieved by calculating the start and end boundaries of the day and applying them as range filters on the date field. All of this ensures the users can filter diary entries by hashtags, people, or dates in combination to enhance the search functionality and effectiveness.

```
Query<Map<String, dynamic>> q = diaries.orderBy('createdAt', descending: true);

final s = search.trim();
if (s.isNotEmpty) {
  if (mode == _SearchMode.tag) {
    final tag = _normalizeTag(s);
    if (tag != null) q = q.where('tags', arrayContains: tag);
  } else if (mode == _SearchMode.person) {
    final p = _normalizePerson(s);
    if (p != null) q = q.where('people', arrayContains: p);
  }
}

if (date != null) {
  final start = DateTime(date!.year, date!.month, date!.day);
  final end = start.add(const Duration(days: 1));
  q = q
      .where('date', isGreaterThanOrEqualTo: Timestamp.fromDate(start))
      .where('date', isLessThan: Timestamp.fromDate(end));
}
```

Figure 4.3.3.1 Firestore Query for Searching and Filtering

Select Participants from live Firestore stream

This function creates a reactive list of people from Firestore and allows users to select or deselect them using checkboxes. It begins by copying the currently selected participants into a selected map. The StreamBuilder listens to the PeopleService.people(uid) collection, ordered by displayName to ensure that any changes in Firestore, such as adding or renaming a person, are reflected in real time in the UI. Inside the builder, the ListView.separated iterates through the retrieved documents and builds a CheckboxListTile for each person. Each tile displays the name of a person; by default, it is "Unknown" if missing and determines its checked state by checking whether the person ID exists in the selected map. When a user toggles a checkbox, the onChanged callback updates the local selected map: if checked, the person ID and name are added; if unchecked, they are removed.

```
Future<void> _openPeoplePicker() async {
  final uid = FirebaseAuth.instance.currentUser?.uid;
  if (uid == null) return;

  final result = await showDialog<Map<String, String>>(
    context: context,
    builder: (ctx) {
      final selected = {..._participants};
      return AlertDialog(
        title: const Text('Select participants'),
        content: SizedBox(
          width: 380,
          height: 420,
          child: StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
            stream: PeopleService.people(uid).orderBy('displayName').snapshots(),
            builder: (context, snap) {
              if (!snap.hasData) return const Center(child: CircularProgressIndicator());
              final docs = snap.data!.docs;
```

Figure 4.3.3.2 Select Participants from live Firestore stream

```
              return StatefulBuilder(
                builder: (context, setInner) => ListView.separated(
                  itemCount: docs.length,
                  separatorBuilder: (_, __) => const Divider(height: 1),
                  itemBuilder: (_, i) {
                    final d = docs[i];
                    final data = d.data();
                    final name = (data['displayName'] ?? data['name'] ?? 'Unknown').toString();
                    final id = d.id;
                    final checked = selected.containsKey(id);
                    return CheckboxListTile(
                      value: checked,
                      onChanged: (v) {
                        setInner(() {
                          if (v == true) {
                            selected[id] = name;
                          } else {
                            selected.remove(id);
                          }
                        });
                      },
                      title: Text(name),
                    );  // CheckboxListTile
                  },
                ),  // ListView.separated
              );  // StatefulBuilder
            },
          ),  // StreamBuilder
        ),  // SizedBox
        actions: [
          TextButton(onPressed: () => Navigator.pop(ctx), child: const Text('Cancel')),
          FilledButton(onPressed: () => Navigator.pop(ctx, selected), child: const Text('Done')),
        ],
      );  // AlertDialog
    },
  );
```

Figure 4.3.3.3 Select Participants from live Firestore Stream

```
  if (result != null) {
    setState(() {
      _participants
        ..clear()
        ..addAll(result);
    });
  }
}
```

Figure 4.3.3.4 Select Participants from live Firestore stream

Tag and Mention Pipeline

This code provides utility functions and validation rules to manage hashtags, mentions, and form requirements. The _normalizeTag and _normalizePerson functions standardize user input by trimming spaces, converting text to lowercase, and stripping leading special characters, which # for tags, @ for mentions. This ensures consistent storage and comparison regardless of how the user types them. The _extractTags function uses a regular expression to scan an input string for hashtags, supporting both standard and full-width #, and extracts alphanumeric or underscore-based tokens. These matches are normalized to lowercase and returned as a set to ensure uniqueness. Similarly, _extractMentions capture user mentions starting with both standard and full-width @, extracts valid usernames, and returns them as a set.

```
String? _normalizeTag(String? value) {
  if (value == null) return null;
  var v = value.trim().toLowerCase();
  if (v.startsWith('#')) v = v.substring(1);
  return v.isEmpty ? null : v;
}

String? _normalizePerson(String? value) {
  if (value == null) return null;
  var v = value.trim().toLowerCase();
  if (v.startsWith('@')) v = v.substring(1);
  return v.isEmpty ? null : v;
}

Set<String> _extractTags(String input) {
  final re = RegExp(r'(?:#|＃)([A-Za-z0-9_]+)');
  return re.allMatches(input).map((m) => m.group(1)!.toLowerCase()).toSet();
}

Set<String> _extractMentions(String input) {
  final re = RegExp(r'(?:@|＠)([A-Za-z0-9_]+)');
  return re.allMatches(input).map((m) => m.group(1)!.toLowerCase()).toSet();
}
```

Figure 4.3.3.5 Tag and Mention Pipeline

4.3.4 Face Recognition Module

Flow of Selecting an Image from Local Storage

The _pickImage() function manages the process of selecting an image, running face recognition, and displaying the result. When triggered, it first checks whether another recognition task is in progress by using the _busy flag to prevent simultaneous operations. Then, it opens the image picker with the given source, such as the camera or gallery. If the user cancels the selection, the process stops and _busy is reset. If an image is chosen, the file is passed to the FaceRecognizerService for preprocessing and embedding extraction. If no face is detected, the function records the result as “Not a face.” If a face embedding is found, the function compares it against stored embeddings in the Firebase using cosine similarity to find the best match of the similarity score, name, and person ID. If the highest similarity score is below the setting threshold of 0.70, the face is marked as “Unknown”. Otherwise, the match is confirmed, and the corresponding person ID is assigned if available. Finally, _busy is reset to allow new operations.

```dart
Future<void> _pickImage(ImageSource source) async {
  if (_busy) return;
  setState(() => _busy = true);

  try {
    final pickedFile = await _picker.pickImage(source: source);
    if (pickedFile == null) {
      setState(() => _busy = false);
      return;
    }

    final imageFile = File(pickedFile.path);
    final result = await _faceRecognizerService.preprocessAndEmbedFromFile(imageFile);
    if (result == null) {
      _setResult(imageFile, 'Not a face', 0.0, null);
      return;
    }

    double bestSim = 0.0;
    String bestName = 'Unknown';
    String? bestId;

    for (final m in _storedEmbeddings) {
      final dbEmb = List<double>.from(m['embedding']);
      final sim = _faceRecognizerService.cosineSimilarity(result.embedding, dbEmb);
      if (sim > bestSim) {
        bestSim = sim;
        bestName = (m['name'] ?? 'Unknown') as String;
        bestId = (m['personId'] ?? m['id'])?.toString();
      }
    }
```

Figure 4.3.4.1 Flow of Selecting an Image from Local Storage

```
    if (bestSim < 0.70) {
      _setResult(imageFile, 'Unknown', bestSim, null);
    } else {
      // Fallback to nam->id map if personId missing in embedding record
      bestId ??= _nameToPersonId[_fold(bestName)];
      _setResult(imageFile, bestName, bestSim, bestId);
    }
  } catch (e) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Recognition failed: $e')),
      );
    }
  } finally {
    if (mounted) setState(() => _busy = false);
  }
}
```

Figure 4.3.4.2 Flow of Selecting an Image from Local Storage

Preprocess and Embedding of the Image from Local Storage

The preprocessAndEmbedFromFile() function is responsible for preparing a face image and generating its numerical embedding for recognition. First and foremost, it decodes the input image file into a usable format; the process ends immediately if decoding fails. Then, it uses a face detector to identify all faces present in the image and selects the largest face to assume it to be the primary subject. This is determined by calculating the area of the bounding box of each face and selecting the one with the largest size.

Once the target face is identified, the function computes a square crop around it, centering on the bounding box and ensuring that the cropped region stays within the image boundaries. The cropped face is then resized to a fixed dimension of 112×112 pixels to standardize the input size for the recognition model. In the embedding step, the processed face image is passed into a feature extractor that produces a high-dimensional embedding vector. If the embedding is invalid, such as empty or containing NaN values, the function returns null; otherwise, it returns a FacePipelineResult that contains the cropped image, embedding, and the original bounding box. The pre-processing and embedding of the image from a live camera are most similar.

```dart
Future<FacePipelineResult?> preprocessAndEmbedFromFile(File file) async {
  final bytes = await file.readAsBytes();
  final decoded = img.decodeImage(bytes);
  if (decoded == null) return null;

  final faces = await _detector.processImage(InputImage.fromFile(file));
  if (faces.isEmpty) return null;

  Face target = faces.first;
  double bestArea = target.boundingBox.width * target.boundingBox.height;
  for (final f in faces) {
    final a = f.boundingBox.width * f.boundingBox.height;
    if (a > bestArea) { bestArea = a; target = f; }
  }

  final Rect b = target.boundingBox;
  final double side = math.max(b.width, b.height);
  int x = (b.center.dx - side / 2).round().clamp(0, decoded.width - 1);
  int y = (b.center.dy - side / 2).round().clamp(0, decoded.height - 1);
  int w = side.round().clamp(1, decoded.width - x);
  int h = side.round().clamp(1, decoded.height - y);

  final crop = img.copyCrop(decoded, x, y, w, h);
  final crop112 = img.copyResize(crop, width: 112, height: 112);

  final emb = await extractEmbedding(crop112);
  if (emb.isEmpty || emb.any((v) => v.isNaN)) return null;

  return FacePipelineResult(crop112: crop112, embedding: emb, faceBox: b);
}
```

Figure 4.3.4.3 Preprocess and Embedding the Image from Local Storage

```dart
Future<FacePipelineResult?> embedFromRgbAndBox(
    img.Image rgb,
    Rect faceBox, {
      bool mirror = false,
    }) async {
  final double side = math.max(faceBox.width, faceBox.height);

  int x = (faceBox.center.dx - side / 2).round().clamp(0, rgb.width  - 1);
  int y = (faceBox.center.dy - side / 2).round().clamp(0, rgb.height - 1);
  int w = side.round().clamp(1, rgb.width  - x);
  int h = side.round().clamp(1, rgb.height - y);

  img.Image crop = img.copyCrop(rgb, x, y, w, h);
  if (mirror) crop = img.flipHorizontal(crop);

  final img.Image face112 = img.copyResize(crop, width: 112, height: 112);
  final emb = await extractEmbeddingFrom112(face112);

  if (emb.isEmpty || emb.any((v) => v.isNaN)) return null;
  return FacePipelineResult(crop112: face112, embedding: emb, faceBox: faceBox);
}
```

Figure 4.3.4.4 Preprocess and Embedding the Image from Live Camera

Inference and L2 normalization of the Embedding

The extractEmbedding() function generates a numerical face embedding from an input image using a MonileFaceNet TensorFlow Lite model in this project. It begins by preprocessing the image through preprocessImage(), which ensures proper RGB format and resizing to the model's expected 112×112 resolution. The image is then converted into a Float32 tensor and reshaped into the [1, 3, 112, 112] format required for inference. An output buffer of size 1280 is prepared and reshaped into [1, 1280] to match the embedding dimension of the model. The TFLite interpreter runs inference to fill the output with the embedding vector. This raw embedding is then passed through l2Normalize(), which scales the vector, so its length (L2 norm) equals one to ensure consistency and comparability between embeddings. The function catches the error, logs it, and then returns an empty list if inference fails

```
Future<List<double>> extractEmbedding(img.Image image) async {

  final preprocessed = preprocessImage(image);
  final input = imageToFloat32(preprocessed).reshape([1, 3, 112, 112]);
  final output = List.filled(1280, 0.0).reshape([1, 1280]); // Change 192 to your model's output

  try {
    interpreter.run(input, output);
    // L2 normalize embedding
    return l2Normalize(List<double>.from(output[0]));
  } catch (e) {
    print('Error during inference: $e');
    return [];
  }
}

List<double> l2Normalize(List<double> v) {
  final norm = sqrt(v.fold(0.0, (a, b) => a + b * b));
  return norm == 0 ? v : v.map((e) => e / norm).toList();
}
```

Figure 4.3.4.5 Inference and L2 normalization of the Embedding

Matching Metric

The cosineSimilarity() function calculates the similarity between two numerical vectors, typically embeddings, by measuring the cosine of the angle between them. It begins with an assertion to ensure that both input lists are of equal length. Three variables are then initialized: dot for the dot product of the vectors, and na and nb for the squared magnitudes of each vector. A loop iterates through each index, simultaneously

accumulating the dot product (a[i] * b[i]) and the squared sums (a[i]^2 and b[i]^2). After the loop, the cosine similarity is computed as the dot product divided by the product of the square roots of the magnitudes (sqrt(na) * sqrt(nb)). The range of the result value should be from -1 to 1, where 1 indicates identical direction, meaning perfect similarity, 0 indicates orthogonality, meaning no similarity, and -1 indicates opposite directions.

```
double cosineSimilarity(List<double> a, List<double> b) {
  assert(a.length == b.length);
  double dot = 0.0;
  double normA = 0.0;
  double normB = 0.0;
  for (int i = 0; i < a.length; i++) {
    dot += a[i] * b[i];
    normA += a[i] * a[i];
    normB += b[i] * b[i];
  }
  return dot / (sqrt(normA) * sqrt(normB));
}
```

Figure 4.3.4.5 Matching Metric

Boot the camera and ML Kit

The _boot() function initializes the entire face recognition camera pipeline when the widget starts. It begins with a safeguard to ensure a valid subjectUid is provided. Next, it loads the face recognition model through the FaceRecognizerService so that embeddings can be generated. It then retrieves all stored embeddings associated with the subject user from Firebase, which will later be used for matching against live detections. It also preloads a people index in the background, caching names, avatars, and counts for quick access to improve performance. The function proceeds to configure the FaceDetector with accurate mode, tracking enabled, and a minimum face size threshold of 0.15 to ensure reliable detection. After that, it queries the available cameras, selects the front-facing one, and initializes a CameraController with high resolution, audio disabled, and a YUV420 image format suitable for ML processing.

Once the camera is successfully initialized, the state is updated to reflect readiness, and an image stream is started. Each incoming frame updates raw dimensions and is passed to the _process() function, which handles face detection and recognition in real time.

```
Future<void> _boot() async {
  if (widget.subjectUid.isEmpty) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('No active patient selected.')),
      );
      Navigator.pop(context);
    }
    return;
  }
  await _faceRecognizerService.loadModel();
  _storedEmbeddings = await _firebaseService.getAllEmbeddingsForRecognition(uid: widget.subjectUid);
  unawaited(_preloadPeopleIndex()); // this will also use subjectUid after we patch it below
  _faceDetector = FaceDetector(
    options: FaceDetectorOptions(
      performanceMode: FaceDetectorMode.accurate,
      enableTracking: true,
      enableLandmarks: false,
      enableContours: false,
      enableClassification: false,
      minFaceSize: 0.15,
    ),  // FaceDetectorOptions
  );  // FaceDetector
  final cameras = await availableCameras();
  final front = cameras.firstWhere((c) => c.lensDirection == CameraLensDirection.front);
  _controller = CameraController(
    front,
    ResolutionPreset.high,
    enableAudio: false,
    imageFormatGroup: ImageFormatGroup.yuv420,
  );
  await _controller.initialize();
  if (!mounted) return;
  setState(() => _isCameraInitialized = true);
  _controller.startImageStream((image) {
    _rawW = image.width;
    _rawH = image.height;
    _process(image);
  });
}
```

Figure 4.3.4.6 Boot the camera and ML Kit

4.3.5 People Enroll Module

Person Resolution and De-duplication

The _ensurePersonId() function ensures that a person entry exists in Firestore for a given display name and either retrieves or creates the appropriate identifier. It first checks whether the user is signed in. If a _personId has already been assigned earlier in

the session, it is returned immediately. Otherwise, the function normalizes the given display name into a folded form for case-insensitive searching. It then queries the People subcollection under the current user document to check if any entries already exist with the same folded name. If no matches are found, the user is prompted with a dialog to ask whether to create a new person entry. If the user agrees, a new Firestore document is created with the display name and timestamp metadata, and its ID is cached in _personId and returned. If matches are found, another dialog is shown to confirm whether the user wants to create a duplicate entry or cancel the operation. If the user cancels, a toast message is shown, and the function returns null. If the user chooses to create a new entry despite duplicates, a new document is added with the same structure, and its ID is returned. This can avoid duplicate names by prompting the user.

```
Future<String?> _ensurePersonId(String displayName) async {
  final uid = FirebaseAuth.instance.currentUser?.uid;
  if (uid == null) throw Exception('Must be signed in');
  if (_personId != null) return _personId!;

  final folded = _foldName(displayName);
  final peopleRef = FirebaseFirestore.instance
      .collection('Users')
      .doc(uid)
      .collection('People');
  final snap =
  await peopleRef.where('displayNameFolded', isEqualTo: folded).limit(10).get();

  if (snap.docs.isEmpty) {
    final create = await showDialog<bool>(
      context: context,
      barrierDismissible: false,
      builder: (c) => AlertDialog(
        title: const Text('Create new person?'),
        content: Text('No existing person called "$displayName". Create a new one now?'),
        actions: [
          TextButton(onPressed: () => Navigator.pop(c, false), child: const Text('Cancel')),
          ElevatedButton(onPressed: () => Navigator.pop(c, true), child: const Text('Create')),
        ],
      ),  // AlertDialog
    );
    if (create != true) {
      _toast('Canceled. Nothing created.');
      return null;
    }
    final newDoc = peopleRef.doc();
    await newDoc.set({
      'displayName': displayName,
      'displayNameFolded': folded,
      'createdAt': FieldValue.serverTimestamp(),
      'updatedAt': FieldValue.serverTimestamp(),
    });
    _personId = newDoc.id;
    return _personId!;
  }
```

Figure 4.3.5.1 Person Resolution and De-duplication

```
final action = await showDialog<_DupAction>(
  context: context,
  barrierDismissible: false,
  builder: (c) => AlertDialog(
    title: const Text('Same name found'),
    content: Text(
      'A person named "$displayName" already exists. '
          'Do you want to create a NEW entry, or cancel?',
    ),  // Text
    actions: [
      TextButton(onPressed: () => Navigator.pop(c, _DupAction.cancel), child: const Text('Cancel')),
      ElevatedButton(onPressed: () => Navigator.pop(c, _DupAction.createNew), child: const Text('Create NEW')),
    ],
  ),  // AlertDialog
);

if (action != _DupAction.createNew) {
  _toast('Canceled. Nothing created.');
  return null;
}

final newDoc = peopleRef.doc();
await newDoc.set({
  'displayName': displayName,
  'displayNameFolded': folded,
  'createdAt': FieldValue.serverTimestamp(),
  'updatedAt': FieldValue.serverTimestamp(),
});
_personId = newDoc.id;
return _personId!;
}
```

Figure 4.3.5.2 Person Resolution and De-duplication

4.3.6 GPS Tracking Module

Haversine Formula

The _calculateDistanceMeters() function computes the geographic distance between two latitude and longitude points using the Haversine formula, which accounts for the curvature of the Earth. It begins by defining the radius of the Earth (R) in meters. The latitude and longitude differences between the two points are calculated and converted from degrees to radians for trigonometric operations. The individual latitudes are also converted into radians. The function then applies the Haversine formula to compute an intermediate value h that represents the square of half the chord length between the points by using sine and cosine functions of the latitude and longitude differences. Lastly, 2 * R * atan2(sqrt(h), sqrt(1 - h)) is used to determine the great-circle distance, which gives the shortest distance over the Earth's surface between the two coordinates in meters.

```
double _calculateDistanceMeters(gmaps.LatLng a, gmaps.LatLng b) {
  const R = 6371000.0;
  final dLat = _degToRad(b.latitude - a.latitude);
  final dLon = _degToRad(b.longitude - a.longitude);
  final lat1 = _degToRad(a.latitude);
  final lat2 = _degToRad(b.latitude);
  final h = sin(dLat / 2) * sin(dLat / 2) +
      cos(lat1) * cos(lat2) * sin(dLon / 2) * sin(dLon / 2);
  return 2 * R * atan2(sqrt(h), sqrt(1 - h));
}
```

Figure 4.3.6.1 Haversine Formula

4.3.7 Return Route Module

Fetch Route from Directions API

The _fetchWalkingRoute() function retrieves and renders a walking route between two coordinates using the Google Maps Directions API, while also handling multiple fallback scenarios to ensure reliability. The function begins by checking if a valid API key is available. If not, the system will draw a straight polyline directly by default between the origin and destination. When an API key is present, the function marks a route fetch as in progress and constructs a Uri request with parameters for origin, destination, walking mode, and API key. It sends an HTTP GET request and, upon receiving a 200 response, parses the JSON body. If the API returns a status of OK with valid routes, the first route is selected. From its first leg, the estimated duration (_routeEtaText) and distance (_routeDistText) are extracted for display. The polyline path is then decoded from the overview polyline points, and a styled polyline is drawn on the map. If no valid route exists, or if the response status is not OK, the function falls back to rendering a straight line between the origin and destination. Additional fallbacks cover HTTP errors and exceptions such as network or parsing failures, with drawing different colors corresponding to different situations.

```dart
Future<void> _fetchWalkingRoute(gmaps.LatLng origin, gmaps.LatLng dest) async {
  if (_mapsKey.isEmpty) {
    // fall back to a straight line
    _routeLine = gmaps.Polyline(
      polylineId: const gmaps.PolylineId('fallback_line'),
      points: [origin, dest],
      width: 5,
      color: Colors.red,
    );  // gmaps.Polyline
    if (mounted) setState(() {});
    return;
  }

  _routeFetchInFlight = true;
  try {
    final uri = Uri.https(
      'maps.googleapis.com',
      '/maps/api/directions/json',
      {
        'origin': '${origin.latitude},${origin.longitude}',
        'destination': '${dest.latitude},${dest.longitude}',
        'mode': 'walking',
        'key': _mapsKey,
      },
    );

    final resp = await http.get(uri);
    if (resp.statusCode == 200) {
      final body = json.decode(resp.body) as Map<String, dynamic>;
      final status = (body['status'] ?? '').toString();

      if (status == 'OK') {
        final routes = (body['routes'] as List?) ?? const [];
        if (routes.isNotEmpty) {
          final route0 = routes.first as Map<String, dynamic>;

          final legs = (route0['legs'] as List?) ?? const [];
          if (legs.isNotEmpty) {
            final leg0 = legs.first as Map<String, dynamic>;
            _routeEtaText  = (leg0['duration']?['text'] ?? '').toString();
            _routeDistText = (leg0['distance']?['text'] ?? '').toString();
          } else {
            _routeEtaText = _routeDistText = null;
          }

          final overview = (route0['overview_polyline'] as Map?) ?? {};
          final points = (overview['points'] ?? '').toString();
```

Figure 4.3.7.1 Fetch Route from Directions API

```
            final path = points.isNotEmpty ? _decodePolyline(points) : <gmaps.LatLng>[origin, dest];

            _routeLine = gmaps.Polyline(
              polylineId: const gmaps.PolylineId('walking_route'),
              points: path,
              width: 6,
              color: Colors.blue,
            );  // gmaps.Polyline
            if (mounted) setState(() {});
            if (_follow) _fitToBounds(path);
            return;
          }
        }
        _routeEtaText = _routeDistText = null;  // <-- reset on fallbacks


        // Not OK or empty -> fallback straight line
        _routeLine = gmaps.Polyline(
          polylineId: const gmaps.PolylineId('fallback_line'),
          points: [origin, dest],
          width: 5,
          color: Colors.green,
        );  // gmaps.Polyline
        if (mounted) setState(() {});
      } else {
        // HTTP error -> fallback
        _routeLine = gmaps.Polyline(
          polylineId: const gmaps.PolylineId('fallback_line'),
          points: [origin, dest],
          width: 5,
          color: Colors.orange,
        );  // gmaps.Polyline
        if (mounted) setState(() {});
      }
    } catch (_) {
      // Network/parsing error -> fallback
      _routeLine = gmaps.Polyline(
        polylineId: const gmaps.PolylineId('fallback_line'),
        points: [origin, dest],
        width: 5,
        color: Colors.purple,
      );  // gmaps.Polyline
      if (mounted) setState(() {});
    } finally {
      _routeFetchInFlight = false;
    }
  }
```

Figure 4.3.7.2 Fetch Route from Directions API

4.3.8 Information Card Module

The _writeTagAndPublish() function integrates an NFC tag to write with Firestore publishing to create a shareable public information card for a patient by selecting some useful information only that is stored in the Information Card Module. The process begins by generating a new Firestore document in the PublicCards collection, obtaining its document ID (cid) to serve as both the database key and part of a unique URL (https://memoracare-27354.web.app/tag/$cid). The function then checks if NFC is

available on the device; if unavailable, it notifies the user and terminates. If NFC is supported, the device polls for a nearby NFC tag to ensure that it is both NDEF-compatible and writable. If the tag fails these conditions, the process is aborted gracefully with error feedback. When a writable tag is detected, the function writes the generated URL onto the tag as an NDEF record, effectively linking the physical tag to the patient's digital information. If the operation completes successfully, the required metadata and patient details will be stored in the Firebase, and the user is notified with a success message; otherwise, the function cleans up by deleting the partially created Firestore document, closing the NFC session with an error message, and alerting the user of the failure.

```
Future<void> _writeTagAndPublish(InfoCard c) async {
  final db  = FirebaseFirestore.instance;
  final doc = db.collection('PublicCards').doc(); // Firestore id
  final cid = doc.id;
  final url = Uri.parse('https://memoracare-27354.web.app/tag/$cid');

  try {
    final avail = await FlutterNfcKit.nfcAvailability;
    if (avail != NFCAvailability.available) {
      if (!mounted) return;
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('NFC not available on this device')),
      );
      return;
    }

    final tag = await FlutterNfcKit.poll(
      timeout: const Duration(seconds: 20),
      iosAlertMessage: 'Hold tag near phone',
      iosMultipleTagMessage: 'Multiple tags detected, try one',
    );

    if (tag.ndefAvailable != true || tag.ndefWritable != true) {
      await FlutterNfcKit.finish(iosErrorMessage: 'Tag not writable');
      if (!mounted) return;
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('This tag is not NDEF-writable')),
      );
      return;
    }

    await FlutterNfcKit.writeNDEFRecords([ndef.UriRecord.fromUri(url)]);
    await FlutterNfcKit.finish(iosAlertMessage: 'Tag written');
```

Figure 4.3.8.1 NFC Tag Writing and Publishing

```
    // Create the public doc
    await doc.set({
      'cid': cid,
      'subjectUid': c.subjectUid,
      'name': c.name,
      'phone': c.phone,
      'email': c.email,
      'addressLine1': c.addressLine1,
      'addressLine2': c.addressLine2,

      'city': c.city,
      'state': c.state,
      'postalCode': c.postalCode,
      'country': c.country,
      'emergencyName': c.emergencyContactName,
      'emergencyPhone': c.emergencyContactPhone,
      'updatedAt': FieldValue.serverTimestamp(),
      'published': true,
    });

    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Public tag created')),
    );
  } catch (e) {
    try { await doc.delete(); } catch (_) {}
    try { await FlutterNfcKit.finish(iosErrorMessage: 'Failed'); } catch (_) {}
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('NFC error: $e')),
    );
  }
}
```

Figure 4.3.8.2 NFC Tag Writing and Publishing

4.3.9 Workflow and Work Done for Developing the CNN Model

Dataset Used for for the CNN Model

Figure 4.3.9.1 Samples in the dataset

The dataset is collected from Kaggle, and there are some images of me that have been added to the dataset, which consists total of 2627 images labelled by 32 people. Before the dataset is used, it is separated to the training dataset and testing dataset manually with the ratio of 8:2. In the early stage of development and prototyping, this moderate dataset is used for easier observing, fasting the training time, building and validating the protype of the model quickly, and saving the memory and computational power. But the bigger dataset may be collected and found in the future to avoid problems like overfitting.

Data Loading and Preprocessing for the CNN Model

In this stage, TensorFlow's built-in utility function image_dataset_from_directory() is used to efficiently load and preprocess image data for separating the dataset into the train data and validation data for the Convolutional Neural Network (CNN) model. The training data and validation data are separated with a ratio of 8:2 from the training data set. A random seed that is 42 was set to ensure the split remains consistent across multiple runs. This method also enables automatic labelling based on subdirectory names and performs essential preprocessing steps such as resizing and batching. The images were resized to 128×128 pixels to standardize input dimensions for the CNN model, and a batch size of 32 was used for training. Both datasets were configured with label_mode="categorical", so means that the class labels are automatically converted into one-hot encoded vectors. This setup aligns with the use of the softmax activation function in the final layer of the CNN model, where the model predicts probabilities across multiple classes. These settings have also been used on the testing dataset with image size, batch size, and label_mode.

```
train_data_ori = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    validation_split=0.2,
    subset="training",
    seed=42,
    image_size=(128, 128),
    batch_size=32,
    label_mode="categorical"  # One-hot labels for softmax output
)

val_data_ori = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    validation_split=0.2,
    subset="validation",
    seed=42,
    image_size=(128 ,128),
    batch_size=32,
    label_mode="categorical"
)
```

Figure 4.3.9.2 Preprocessing training data and validation data

```
test_data_ori = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    image_size=(128, 128),
    batch_size=32,
    label_mode="categorical"
)
```

Figure 4.3.9.3 Preprocessing testing data

The data augmentation and normalization were conducted to improve the model's performance and generalization. The data augmentation is applied to the training data only in order to simulate variations in the dataset and prevent overfitting as large as possible. In this case, the augmentation is used, such as randomly flipping images horizontally, rotating the image by up to positive or negative 5%, randomly zooming into the image by up to 10%, and shifting he image horizontally and vertically by 10%.

```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.05),
    layers.RandomZoom(0.1),
    layers.RandomTranslation(0.1, 0.1),
])
```

Figure 4.3.9.4 Data Augmentation

After that, both the training, testing, and validation data are normalized to make the training process faster and stable. The normalization is implemented using TensorFlow's built-in layers.Rescaling(1./255) function. The purpose of this layer is to rescale the pixel values of the images from their original range of [0, 255], which is the standard RGB image values, to a [0, 1] range. This can be achieved by multiplying all pixel values by 1/255 via this function. The normalization and augmentation layers are applied using the .map() function. Caching and prefetching are applied to improve the efficiency of the input pipeline.

```
normalization_layer = layers.Rescaling(1./255)
```

```
#Apply augmentation only to training dataset
train_data = train_data_ori.map(lambda x, y: (normalization_layer(data_augmentation(x)), y))
val_data   = val_data_ori.map(lambda x, y: (normalization_layer(x), y))
test_data  = test_data_ori.map(lambda x, y: (normalization_layer(x), y))
```

```
 #Improve performance with caching and prefetching
AUTOTUNE = tf.data.AUTOTUNE
train_data = train_data.cache().prefetch(buffer_size=AUTOTUNE)
val_data = val_data.cache().prefetch(buffer_size=AUTOTUNE)
test_data = test_data.cache().prefetch(buffer_size=AUTOTUNE)
```

Figure 4.3.9.5 Normalization and augmentation layer is applied and then used Caching and Prefetching

Therefore, the training data occupies around 65%, the validation data occupies around 15%, and the testing data occupies 20% of the main dataset, with each batch size of 32.

Model Training for the CNN Model

Let's talk about the CNN model architecture. The CNN model is implemented using the Keras Sequential API with an input shape of (128, 128, 3), which corresponds to RGB images with size 128×128. The architecture consists of five convolutional blocks followed by fully connected layers. Each convolutional block includes a Conv2d layer with Relu activation and kernel size (5,5), BatchNormalization to improve stability and

convergence, and MaxPooling2D to reduce spatial dimensions. The number of filters of each block is ordered by 32, 32, 64, 128, and 256.

```
input_shape = (128, 128, 3)
num_classes = len(class_names)  # Should be 32
epochs = 30
```

Figure 4.3.9.6 Initial declarations

```
model = Sequential([
    Input(shape=input_shape),

    Conv2D(32, (5,5), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(),

    Conv2D(32, (5,5), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(),

    Conv2D(64,(5,5), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(),

    Conv2D(128, (5,5), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(),

    Conv2D(256, (5,5), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(),



    Flatten(),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dense(num_classes, activation='softmax')
])


# Show model summary
model.summary()
```

Figure 4.3.9.7 CNN Model Architecture

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	2,432
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 32)	25,632
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	51,264
batch_normalization_2 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	204,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	819,456
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 256)	1,048,832
batch_normalization_5 (BatchNormalization)	(None, 256)	1,024
dense_1 (Dense)	(None, 32)	8,224

Total params: 2,163,840 (8.25 MB)
Trainable params: 2,162,304 (8.25 MB)
Non-trainable params: 1,536 (6.00 KB)

Figure 4.3.9.8 CNN Model Architecture Summary

Then, the model is compiled by using the Adam optimizer with a learning rate of 0.001, and the categorical cross-entropy is used as the loss function. For the training configuration, the model is trained for 30 epochs by using train_data and val_data. The callbacks are also used for enhancing the processes, such as theReduceLROnPlateau, which helps to monitor the validation loss and reduces the learning rate by a factor of 0.5 if the loss does not improve after 3 epochs, while TensorBoard which used to log training metrics for visualization and debugging. Moreover, the model training is conducted using the fit.() function. The model will learn the features from the facial images provided and update its weights to minize the loss function during the training.

```
optimizer_adam = Adam(learning_rate=0.001)
model.compile(
    optimizer=optimizer_adam,
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Figure 4.3.9.9 Model Compilation

```
log_dir='logs'
```

```
callbacks = [
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, verbose=1),
    tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
]
```

Figure 4.3.9.10 Callbacks used

```
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=epochs,
    callbacks=callbacks
)
```

Figure 4.3.9.11 Model training

The model performance is monitored and visualized by TensorBoard. From the graph related to the loss function created, there are some key points that can be observed, such as the green line of the training loss decreases rapidly and reaches near-zero around epoch 10, summarizing that the model fits the data very well, while the orange line validation loss starts high and decreases until about epoch 10 and does not reach as low as training loss. Therefore, the key observation is that the model may be overfitting because the model learns the training data well, and the gap between training and validation loss is larger; this may cause the model cannot generalize well to unseen data.

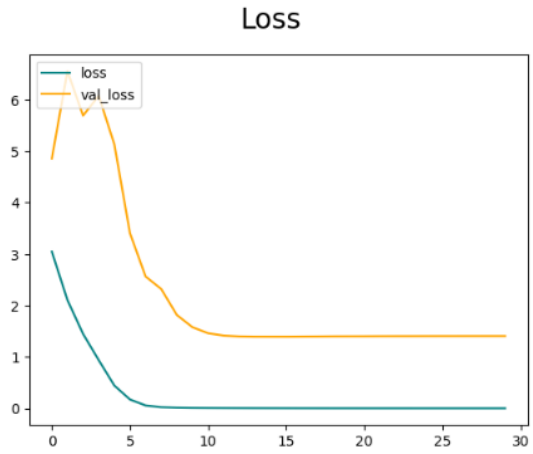


Figure 4.3.9.12 Graph of Loss Function of training and validation

By observing the graph related to the accuracy created, can also learn that the model may be overfitting because the gap between the training accuracy and validation accuracy is quite large, and the accuracy from the validation does not further improve with the increase in epochs.

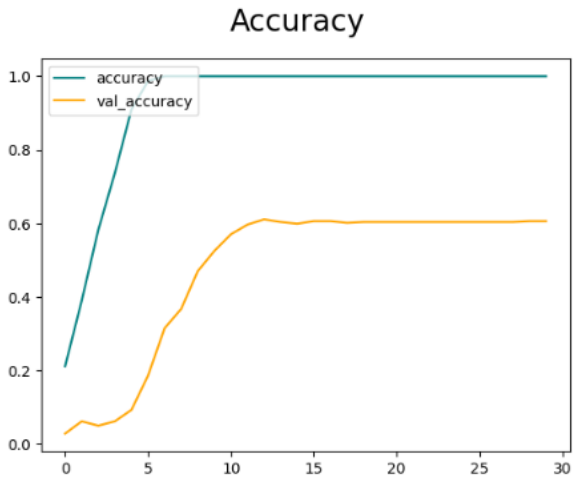


Figure 4.3.9.13 Graph of the accuracy of training and validation

Lastly, the precision, recall, accuracy, and f1 score from training both are achieved 100% also proven that the model is overfitting.

Classification Report:

	precision	recall	f1-score	support
Akshay Kumar	1.00	1.00	1.00	32
Alexandra Daddario	1.00	1.00	1.00	69
Alia Bhatt	1.00	1.00	1.00	48
Amitabh Bachchan	1.00	1.00	1.00	39
Andy Samberg	1.00	1.00	1.00	61
Anushka Sharma	1.00	1.00	1.00	44
Billie Eilish	1.00	1.00	1.00	70
Brad Pitt	1.00	1.00	1.00	83
Camila Cabello	1.00	1.00	1.00	60
Charlize Theron	1.00	1.00	1.00	43
Claire Holt	1.00	1.00	1.00	68
Courtney Cox	1.00	1.00	1.00	48
Dwayne Johnson	1.00	1.00	1.00	31
Elizabeth Olsen	1.00	1.00	1.00	44
Ellen Degeneres	1.00	1.00	1.00	51
Henry Cavill	1.00	1.00	1.00	66
Hrithik Roshan	1.00	1.00	1.00	66
Hugh Jackman	1.00	1.00	1.00	74
Jessica Alba	1.00	1.00	1.00	67
Kashyap	1.00	1.00	1.00	19
Lisa Kudrow	1.00	1.00	1.00	46
Margot Robbie	1.00	1.00	1.00	42
Marmik	1.00	1.00	1.00	18
Natalie Portman	1.00	1.00	1.00	66
Priyanka Chopra	1.00	1.00	1.00	72
Robert Downey Jr	1.00	1.00	1.00	73
Roger Federer	1.00	1.00	1.00	45
Tom Cruise	1.00	1.00	1.00	32
Vijay Deverakonda	1.00	1.00	1.00	68
Virat Kohli	1.00	1.00	1.00	36
Zac Efron	1.00	1.00	1.00	54
yiqian	1.00	1.00	1.00	45
accuracy			1.00	1680
macro avg	1.00	1.00	1.00	1680
weighted avg	1.00	1.00	1.00	1680

Figure 4.3.9.14 Classification Report of Training

Model Validation

After training the CNN, the model is evaluated by using the validation data to observe and visualize how well it generalizes to unseen data. This evaluation uses the predictions made by the model and compares them to the true labels in the validation set.

From the classification report, it can be observed that the accuracy of the model is 0.61, while the macro and weighted average of precision is 0.64, and recall is 0.61. This shows the model performs inconsistently across different classes. Furthermore, the class imbalance was observed, which is that class yiqian has perfect precision, recall, and F1-score. It is seen that this class has high-quality samples, which are images of

me. In the result, the new dataset may be needed to find in the future to improve the model performances.

Classification Report:

	precision	recall	f1-score	support
Akshay Kumar	0.67	0.50	0.57	8
Alexandra Daddario	0.36	0.57	0.44	7
Alia Bhatt	0.71	0.42	0.53	12
Amitabh Bachchan	0.94	1.00	0.97	15
Andy Samberg	0.35	0.46	0.40	13
Anushka Sharma	0.12	0.25	0.17	4
Billie Eilish	0.90	0.82	0.86	11
Brad Pitt	0.54	0.70	0.61	20
Camila Cabello	0.80	0.80	0.80	10
Charlize Theron	0.82	0.53	0.64	17
Claire Holt	0.50	0.64	0.56	11
Courtney Cox	0.58	0.47	0.52	15
Dwayne Johnson	0.67	0.67	0.67	3
Elizabeth Olsen	0.36	0.36	0.36	11
Ellen Degeneres	0.67	0.80	0.73	5
Henry Cavill	0.43	0.50	0.47	20
Hrithik Roshan	0.25	0.47	0.33	15
Hugh Jackman	0.53	0.48	0.50	21
Jessica Alba	0.57	0.62	0.59	21
Kashyap	0.25	0.50	0.33	2
Lisa Kudrow	0.75	0.67	0.71	9
Margot Robbie	1.00	0.71	0.83	14
Marmik	0.75	0.75	0.75	4
Natalie Portman	0.50	0.43	0.46	21
Priyanka Chopra	0.64	0.78	0.70	9
Robert Downey Jr	0.61	0.48	0.54	23
Roger Federer	0.82	0.88	0.85	16
Tom Cruise	0.80	0.57	0.67	14
Vijay Deverakonda	0.82	0.79	0.81	29
Virat Kohli	0.50	0.75	0.60	4
Zac Efron	0.56	0.25	0.34	20
yiqian	1.00	1.00	1.00	15
accuracy			0.61	419
macro avg	0.62	0.61	0.60	419
weighted avg	0.64	0.61	0.61	419

Figure 4.3.9.15 Classification Report for Validation

Model Testing for the CNN Model

Lastly, the model is tested by using the testing data to observe and visualize how well it generalizes to completely unseen data. The test set is untouched during training and validation, giving a true reflection of the real-world performance of the model. From the classification report, it can be observed that the accuracy of the model is around 0.61, while the macro and weighted average of precision is 0.61, and recall is 0.61. This shows the model performs inconsistently across different classes. Furthermore, the class imbalance was observed, which is that class yiqian has perfect precision, recall,

and F1-score. It is seen that this class has high-quality samples, which are images of me.

Classification Report (Test Set):

	precision	recall	f1-score	support
Akshay Kumar	0.33	0.30	0.32	10
Alexandra Daddario	0.59	0.81	0.68	16
Alia Bhatt	0.38	0.26	0.31	19
Amitabh Bachchan	0.87	1.00	0.93	20
Andy Samberg	0.65	0.72	0.68	18
Anushka Sharma	0.33	0.30	0.32	20
Billie Eilish	0.62	0.59	0.61	17
Brad Pitt	0.68	0.76	0.72	17
Camila Cabello	0.65	0.88	0.75	17
Charlize Theron	0.83	0.56	0.67	18
Claire Holt	0.52	0.71	0.60	17
Courtney Cox	0.78	0.41	0.54	17
Dwayne Johnson	0.67	0.73	0.70	11
Elizabeth Olsen	0.58	0.44	0.50	16
Ellen Degeneres	0.95	0.95	0.95	19
Henry Cavill	0.43	0.45	0.44	20
Hrithik Roshan	0.43	0.45	0.44	20
Hugh Jackman	0.67	0.59	0.62	17
Jessica Alba	0.34	0.55	0.42	20
Kashyap	0.25	0.11	0.15	9
Lisa Kudrow	0.79	0.73	0.76	15
Margot Robbie	0.64	0.44	0.52	16
Marmik	0.56	0.50	0.53	10
Natalie Portman	0.38	0.33	0.35	18
Priyanka Chopra	0.57	0.76	0.65	21
Robert Downey Jr	0.63	0.71	0.67	17
Roger Federer	0.72	0.81	0.76	16
Tom Cruise	0.43	0.25	0.32	12
Vijay Deverakonda	0.62	0.72	0.67	18
Virat Kohli	0.75	0.67	0.71	9
Zac Efron	0.69	0.53	0.60	17
yiqian	1.00	1.00	1.00	21
accuracy			0.61	528
macro avg	0.60	0.59	0.59	528
weighted avg	0.61	0.61	0.60	528

Figure 4.3.9.16 Classification Report of Testing

4.3.10 Workflow and Work Done for Developing the Siamese Network

Dataset Used for the Siamese Network

The dataset used is the same.

Data Preprocessing for the Siamese Network

In the development of the Siamese Network, a crucial preprocessing step involves generating paired data for training. This pairing process is essential because the Siamese Network learns by comparing two input images to determine whether they belong to the same class (positive pair) or different classes (negative pair).

To create these pairs, a custom function get_images_grouped_by_person(folder_path) was implemented. This function scans a specified directory (folder_path), where each subfolder represents a different individual. It builds a dictionary mapping each person's name to a list of their corresponding image file paths. Only individuals with at least two images are included, ensuring that positive pairs can be created.

```
def get_images_grouped_by_person(folder_path):
    """Returns a dictionary mapping person_name -> list of image paths"""
    person_to_images = {}
    for person_name in os.listdir(folder_path):
        person_dir = os.path.join(folder_path, person_name)
        if os.path.isdir(person_dir):
            image_paths = glob.glob(os.path.join(person_dir, '*.jpg'))
            if len(image_paths) >= 2:  # At least 2 images needed for positive pairs
                person_to_images[person_name] = image_paths
    return person_to_images
```

Figure 4.3.10.1 Function get_images_grouped_by_person(folder_path)

Positive pairs are generated by randomly selecting two different images belonging to the same individual. Negative pairs are generated by randomly selecting one image from one individual and pairing it with an image from a different individual. The positive pairs will be labelled with 1 while the negative pairs will be labelled with 0. The positive image path, the comparison image path, and the corresponding label (positive or negative) are stored separately in anchor_paths, compare_paths, and labels lists, respectively.

```
# Load positive images only (32 people)
positive_images = get_images_grouped_by_person("data/positive")

all_people = list(positive_images.keys())

positive_paths = []
compare_paths = []
labels = []

for person in all_people:
    imgs = positive_images[person]

    for a_img in imgs:
        # --- Positive pair (1) ---
        possible_pos_imgs = [img for img in imgs if img != a_img]
        if possible_pos_imgs:
            p_img = random.choice(possible_pos_imgs)
            positive_paths.append(a_img)
            compare_paths.append(p_img)
            labels.append(1)

        # --- Negative pair (0) ---
        other_people = [p for p in all_people if p != person]
        if other_people:
            neg_person = random.choice(other_people)
            neg_imgs = positive_images[neg_person]
            n_img = random.choice(neg_imgs)
            positive_paths.append(a_img)
            compare_paths.append(n_img)
            labels.append(0)

print(f"Generated {len(positive_paths)} total pairs ({labels.count(1)} positive, {labels.count(0)} negative).")
```

Figure 4.3.10.2 Create the images pairs and labels

To prepare the input data for training the Siamese Network, three TensorFlow datasets were created which is positive_ds, compare_ds, and label_ds. Then, these three datasets will be merged into a single dataset for training.

```
positive_ds = tf.data.Dataset.from_tensor_slices(positive_paths)
compare_ds = tf.data.Dataset.from_tensor_slices(compare_paths)
label_ds = tf.data.Dataset.from_tensor_slices(tf.cast(labels, tf.float32))
# Combined dataset
data = tf.data.Dataset.zip((positive_ds, compare_ds, label_ds))
```

Figure 4.3.10.3 Create the dataset for training

Almost somethings will be implemented to create the testing data.

```
test_positive_ds = tf.data.Dataset.from_tensor_slices(test_positive_paths)
test_compare_ds = tf.data.Dataset.from_tensor_slices(test_compare_paths)
test_label_ds = tf.data.Dataset.from_tensor_slices(tf.cast(test_labels, tf.float32))

testing_data = tf.data.Dataset.zip((test_positive_ds, test_compare_ds, test_label_ds))
```

Figure 4.3.10.4 Create the dataset for testing

The preprocessing will be applied to all image inputs via the two custom functions called preprocess(file_path) and preprocess_twin(input_img, validation_img, label). The function preprocess decodes the image from JPEG format into a tensor using tf.io.decode_jpeg() and resized to 100×100 pixels with 3 color channels (RGB) to maintain uniform input dimensions required by the Siamese Network. Then, pixel values are normalized to a [0, 1] range by dividing by 255.0. The function preprocess_twin applies the preprocess() operation to both the images in the image pairs.

```
def preprocess(file_path):

    # Read in image from file path
    byte_img = tf.io.read_file(file_path)
    # Load in the image
    img = tf.io.decode_jpeg(byte_img)

    # Preprocessing steps - resizing the image to be 100x100x3
    img = tf.image.resize(img, (100,100))
    # Scale image to be between 0 and 1
    img = img / 255.0

    # Return image
    return img
```

Figure 4.3.10.5 Function preprocess(file_path)

```
def preprocess_twin(input_img, validation_img, label):
    return(preprocess(input_img), preprocess(validation_img), label)
```

Figure 4.3.10.6 Function preprocess_twin(input_img, validation_img, label)

The additional processing steps were applied to optimize the training pipeline. The map(preprocess_twin) operation applies the previously defined preprocess_twin function to each dataset element. This ensures that both anchor and comparison images are decoded, resized to 100×100×3, normalized to a [0,1] range, and prepared in the correct input format for the Siamese model. The cache() operation was used to store the preprocessed dataset in memory after the first epoch. This greatly improves efficiency by preventing redundant preprocessing during each training epoch aims to faster data retrieval and reduced training time. The shuffle(buffer_size=1024) operation randomly shuffles the dataset with a buffer size of 1024 elements. Shuffling introduces randomness in the order of training samples to prevent the model from learning any

order-based patterns and promoting better generalization. Also, do the same things on testing data.

```
# Build dataloader pipeline
data = data.map(preprocess_twin)
data = data.cache()
data = data.shuffle(buffer_size=1024)
```

Figure 4.3.10.7 Build the pipeline for training

```
testing_data = testing_data.map(preprocess_twin)
testing_data = testing_data.cache()
testing_data = testing_data.shuffle(buffer_size=1024)
```

Figure 4.3.10.8 Build the pipeline for testing

After building and preprocessing the paired dataset for the Siamese Network, the data was partitioned into training and validation sets to evaluate model performance. 80% of the data was assigned for training using take(), and 20% for validation using skip(). Both sets were batched with a size of 16 to optimize training efficiency. Prefetching with a buffer size of 8 was applied to both partitions to speed up data loading during model training and evaluation.

```
# Training partition
train_data = data.take(round(len(data)*.8))
train_data = train_data.batch(16)
train_data = train_data.prefetch(8)
```

```
# validation partition
val_data = data.skip(round(len(data)*.8))
val_data = val_data.take(round(len(data)*.2))
val_data = val_data.batch(16)
val_data = val_data.prefetch(8)
```

Figure 4.3.10.9 Separate to training and validation set

Model Training for the Siamese Network

The embedding model design serves as the feature extractor for the Siamese Network to transform input face images into fixed-length feature vectors (embeddings) that capture important characteristics for similarity comparison. The model accepts images of size 100×100×3. For the first cconvolutional block, a Conv2D layer with 64 filters

and a 10×10 kernel size, followed by a MaxPooling2D layer with a 2×2 pooling window with the ReLU function is implemented. For the second convolutional block, a Conv2D layer with 128 filters and a **7×7** kernel size, followed by the same MaxPooling2D layer with the ReLU function is implemented. For the third convolutional block, a Conv2D layer with 128 filters and a 4×4 kernel size, followed by the same MaxPooling2D layer with the ReLU function is implemented. Lastly, a Conv2D layer with 256 filters and a 4×4 kernel size, followed by a Flatten layer that converts the 2D feature maps into a 1D vector. A Dense layer with 4096 units and a sigmoid activation function generates the final embedding vector. The design is referred to in the article [12].

```
def make_embedding():
    inp = Input(shape=(100,100,3), name='input_image')

    # First block
    c1 = Conv2D(64, (10,10), activation='relu')(inp)
    m1 = MaxPooling2D(64, (2,2), padding='same')(c1)

    # Second block
    c2 = Conv2D(128, (7,7), activation='relu')(m1)
    m2 = MaxPooling2D(64, (2,2), padding='same')(c2)

    # Third block
    c3 = Conv2D(128, (4,4), activation='relu')(m2)
    m3 = MaxPooling2D(64, (2,2), padding='same')(c3)

    # Final embedding block
    c4 = Conv2D(256, (4,4), activation='relu')(m3)
    f1 = Flatten()(c4)
    d1 = Dense(4096, activation='sigmoid')(f1)


    return Model(inputs=inp, outputs=d1, name='embedding')
```

Figure 4.2.10 Embedding model design

```
Model: "embedding"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_image (InputLayer)    [(None, 100, 100, 3)]     0

 conv2d (Conv2D)             (None, 91, 91, 64)        19264

 max_pooling2d (MaxPooling2D  (None, 46, 46, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 40, 40, 128)       401536

 max_pooling2d_1 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_2 (Conv2D)           (None, 17, 17, 128)       262272

 max_pooling2d_2 (MaxPooling  (None, 9, 9, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 6, 6, 256)         524544

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 4096)              37752832

=================================================================
Total params: 38,960,448
Trainable params: 38,960,448
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.3.10.11 Embedding model design summary

To enable the Siamese Network to compute the similarity between two input images, a custom layer called L1Dist was implemented. This layer inherits from TensorFlow's Layer class and defines the L1 distance operation, which calculates the absolute difference between the two embedding vectors. The Initialization (__init__) is the constructor method simply calls the parent Layer class initializer. The Forward Pass (call) receives the parameter which is a tuple of two input embeddings, one is positive embedding and the comparison embedding. It computes the element-wise absolute difference between the two embeddings using tf.math.abs(). The measurements of this L1 distance are a smaller distance indicates higher similarity while A larger distance indicates lower similarity.

```
# Siamese L1 Distance class
class L1Dist(Layer):

    # Init method - inheritance
    def __init__(self, **kwargs):
        super().__init__()

    #  similarity measurements
    def call(self, inputs):
        input_embedding, validation_embedding = inputs
        return tf.math.abs(input_embedding - validation_embedding)
```

Figure 4.3.10.12 Layer to calculate distances

The Siamese Network architecture was built to compare two input images and predict whether they belong to the same class or not. For the input layers, both inputs are resized to 100×100×3 dimensions to match the preprocessing pipeline. Each input image is passed through the same embedding model to ensure that both images are transformed into comparable feature vectors. The two embeddings are compared using a custom L1 Distance Layer (L1Dist), which calculates the distances or similarity between two images. The classification layer computed distance vector is passed into a dense layer with one neuron and a sigmoid activation function. The architecture is referred to in the article [12].

```python
def make_siamese_model():

    input_image = Input(name='input_img', shape=(100,100,3))

    validation_image = Input(name='validation_img', shape=(100,100,3))

    siamese_layer = L1Dist()
    siamese_layer._name = 'distance'
    distances = siamese_layer([embedding(input_image), embedding(validation_image)])

    # Classification Layer
    classifier = Dense(1, activation='sigmoid')(distances)

    return Model(inputs=[input_image, validation_image], outputs=classifier, name='SiameseNetwork')
```

```python
siamese_model = make_siamese_model()
```

Figure 4.3.10.13 Siamese Network architecture

The Siamese Network was trained using the Binary Cross-Entropy loss, appropriate for binary classification tasks where the model predicts whether two images belong to the same class. The Adam optimizer with a learning rate of 0.0001 was selected to ensure stable and efficient convergence. To safeguard training progress, TensorFlow's checkpoint mechanism was implemented to save both the optimizer state and model weights at defined intervals.

```python
binary_cross_loss = tf.losses.BinaryCrossentropy()
```

```python
optimizer = tf.keras.optimizers.Adam(1e-4) # 0.0001
```

```python
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, 'ckpt')
checkpoint = tf.train.Checkpoint(opt=optimizer, siamese_model=siamese_model)
```

Figure 4.3.10.14 Definition of loss, optimizer, and training checkpoint for model training

A custom training step function train_step(batch) was implemented to manually control the training process of the Siamese Network. This function was decorated with @tf.function to optimize execution by compiling it into a TensorFlow graph for faster performance. It takes a batch of image pairs and labels, runs them through the model to get predictions, calculates the binary cross-entropy loss, and then uses GradientTape to compute and apply gradients for learning.

```python
@tf.function
def train_step(batch):
    X, y_true = batch[:2], batch[2]
    with tf.GradientTape() as tape:
        y_pred = siamese_model(X, training=True)
        loss = binary_cross_loss(y_true, y_pred)
    gradients = tape.gradient(loss, siamese_model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, siamese_model.trainable_variables))
    return loss, y_true, y_pred
```

Figure 4.3.10.15 Function train_step(batch)

A custom train(data, EPOCHS) function was implemented to manage the Siamese Network training. For each epoch, metrics like recall, precision, and accuracy were initialized and updated batch-by-batch using the train_step function. The average loss was calculated at the end of each epoch, and performance metrics were printed for monitoring. Additionally, model checkpoints were saved every 10 epochs to ensure training progress could be restored if needed.

```python
def train(data, EPOCHS):
    for epoch in range(1, EPOCHS + 1):
        print('\nEpoch {}/{}'.format(epoch, EPOCHS))
        progbar = tf.keras.utils.Progbar(len(data))

        recall = Recall()
        precision = Precision()
        accuracy = BinaryAccuracy()

        epoch_losses = []

        for idx, batch in enumerate(data):

            loss, y_true, y_pred = train_step(batch)

            # Update metrics
            recall.update_state(y_true, y_pred)
            precision.update_state(y_true, y_pred)
            accuracy.update_state(y_true, y_pred)

            epoch_losses.append(loss.numpy())

            progbar.update(idx + 1, [("loss", loss.numpy())])

        avg_loss = sum(epoch_losses) / len(epoch_losses)

        print("Epoch {} Summary: Loss: {:.4f} | Recall: {:.4f} | Precision: {:.4f} | Accuracy: {:.4f}".format(
            epoch,
            avg_loss,
            recall.result().numpy(),
            precision.result().numpy(),
            accuracy.result().numpy()
        ))

        # Save model every 10 epochs
        if epoch % 10 == 0:
            checkpoint.save(file_prefix=checkpoint_prefix)
```

Figure 4.3.10.16 Function train(data, EPOCHS)

After 20 epochs of training, the Siamese Network achieved a loss of 0.62, recall of 0.56, precision of 0.64, and accuracy of 0.62.

```
Epoch 20/20
210/210 [==============================] - 43s 204ms/step - loss: 0.6266
Epoch 20 Summary: Loss: 0.6266 | Recall: 0.5628 | Precision: 0.6410 | Accuracy: 0.6254
```

Figure 4.3.10.17 Training results

Model Validation for the Siamese Network

On the validation set, the Siamese Network achieved a recall of 0.67, precision of 0.65, and accuracy of 0.66.

```
Recall: 0.6725979
Precision: 0.65968585
Accuracy: 0.6614056
```

Figure 4.3.10.18 Validation results

Model Testing for the Siamese Network

On the testing set, the Siamese Network achieved a recall of 0.65, precision of 0.65, and accuracy of 0.65.

```
Recall: 0.65266484
Precision: 0.6497302
Accuracy: 0.6492977
```

Figure 4.3.10.19 Testing results

4.3.11 Workflow and Work Done for Developing the MobileFaceNet

Dataset Used for the MobileFaceNet

The dataset used is the same.

Data Preprocessing for the MobileFaceNet

First and foremost, load the data from the correct paths and separate into training data and testing data. Then, defines a custom dataset class called FaceDataset that inherits

from PyTorch's Dataset class. Its purpose is to organize face image data into a structured format that can be directly used for model training and evaluation. It stores image paths and their labels, provides the dataset size through __len__, and retrieves individual images with preprocessing via __getitem__. This setup ensures efficient data handling and smooth integration with PyTorch's training pipeline.

```
class FaceDataset(Dataset):
    def __init__(self, path, transform=None, selected_classes=None):
        self.path = path
        self.transform = transform
        self.classes = []            # filtered class folder names
        self.class_to_idx = {}       # mapping: class name -> label index
        self.images_path = []
        self.targets = []

        for i, cls in enumerate(sorted(os.listdir(path))):
            if selected_classes is not None and cls not in selected_classes:
                continue
            self.classes.append(cls)
            self.class_to_idx[cls] = len(self.class_to_idx)  # re-index
            cls_path = os.path.join(path, cls)
            for filename in os.listdir(cls_path):
                file_path = os.path.join(cls_path, filename)
                self.images_path.append(file_path)
                self.targets.append(self.class_to_idx[cls])

    def __len__(self):
        return len(self.targets)

    def __getitem__(self, idx):
        image_path = self.images_path[idx]
        image = Image.open(image_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        label = self.targets[idx]
        return image, label
```

Figure 4.3.11.1 FaceDataset Class

The code sets up preprocessing for training and testing datasets. Training uses data augmentation such as resize, flips, rotation, and color jitter with normalization to improve generalization, while testing only applies resizing and normalization for consistent evaluation. Both pipelines are applied through the FaceDataset class to prepare images before model training and testing.

```
transform_train = v2.Compose([
    v2.ToImage(), # Convert input to PIL Image if needed
    v2.Resize((224,224)), # Resize to 224x224
    v2.RandomHorizontalFlip(), # Randomly flip horizontally
    v2.RandomVerticalFlip(p=0.5),  # Added vertical flip
    v2.RandomRotation(degrees=10), # Randomly rotate the image by up to 10 degrees
    v2.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),  # Soften color jitter
    v2.ToDtype(torch.float32, scale=True), # Scale to [0,1] and make float32
    v2.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225]),# Normalization
])

transform_test = v2.Compose([
    v2.ToImage(),
    v2.Resize((224,224)),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

```
train_dataset = FaceDataset(train_path, transform_train)
test_dataset = FaceDataset(test_path, transform_test)
```

Figure 4.3.11.2 Data Augmentation

The code applies StratifiedShuffleSplit to divide the training dataset into 80% training and 20% validation while keeping class distributions balanced. It ensures reproducibility with a fixed random seed and prints the number of samples and their indices in each split.

```
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
train_idx, val_idx = next(splitter.split(train_dataset.images_path, train_dataset.targets))

print('Number of training samples:', len(train_idx), "| indices:", train_idx)
print('Number of val samples     :', len(val_idx), "| indices:", val_idx)
```

Figure 4.3.11.3 Stratified Split

After that, creates PyTorch data loaders for training, validation, and testing. It uses SubsetRandomSampler to select the correct samples for training and validation and sets a batch size of 32. The test loader does not shuffle to keep order consistent.

```
train_sampler = SubsetRandomSampler(train_idx)
val_sampler = SubsetRandomSampler(val_idx)
```

```
train_loader = DataLoader(train_dataset, batch_size=32, sampler=train_sampler)
val_loader = DataLoader(train_dataset, batch_size=32, sampler=val_sampler)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
x_batch, y_batch = next(iter(train_loader))
print(f'{x_batch.shape = }')
print(f'{y_batch.shape = }')
```

Figure 4.3.11.4 Data Loaders for Training, Validation, and Testing

Model Training (Fine Tune) for the MobileFaceNet

Then, the MobileNetV2 is initialized as a backbone model for feature extraction by loading pretrained ImageNet weights. The default classifier layer of MobileNetV2, which is normally used for classification tasks, is removed and replaced with nn.Identity(), meaning the network will output raw feature embeddings instead of class predictions. The embedding dimension is then set using backbone.last_channel, which for MobileNetV2 equals 1280, representing the size of the output feature vector.

```
backbone = models.mobilenet_v2(weights='IMAGENET1K_V1')  # or weights=None
backbone.classifier = nn.Identity()  # Remove default classifier
embedding_dim = backbone.last_channel  # 1280 for MobileNetV2
```

Figure 4.3.11.5 Initializes MobileNetV2

The ArcFace class implements the ArcFace loss in PyTorch for face recognition. A learnable weight matrix is created and initialized using Xavier uniform distribution. In the forward method, both embeddings and weights are L2-normalized to ensure features lie on a unit hypersphere. Cosine similarity between embeddings and weights is then computed, followed by clamping values for numerical stability. An angular margin is added to the cosine similarity of the target class, which forces embeddings of the same class to be closer together while pushing apart embeddings of different classes. The labels are one-hot encoded, and the margin is applied only to the true class logits.

```python
class ArcFace(nn.Module):
    def __init__(self, in_features, out_features, s=64.0, m=0.5):
        """
        in_features: embedding dimension
        out_features: number of classes
        s: scaling factor
        m: angular margin
        """
        super(ArcFace, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.s = s
        self.m = m

        # Learnable weights
        self.weight = nn.Parameter(torch.FloatTensor(in_features, out_features))
        nn.init.xavier_uniform_(self.weight)  # glorot_uniform equivalent

    def forward(self, embeddings, labels):
        # L2 normalize embeddings and weights
        embeddings = F.normalize(embeddings, p=2, dim=1)
        weight = F.normalize(self.weight, p=2, dim=0)

        # Cosine similarity
        cos_theta = torch.matmul(embeddings, weight)
        cos_theta = cos_theta.clamp(-1.0 + 1e-7, 1.0 - 1e-7)  # numerical stability

        # Arc margin
        theta = torch.acos(cos_theta)
        target_logits = torch.cos(theta + self.m)

        # One-hot encode labels
        one_hot = torch.zeros_like(cos_theta)
        one_hot.scatter_(1, labels.view(-1, 1), 1.0)

        # Combine: only apply margin to target class
        logits = cos_theta * (1 - one_hot) + target_logits * one_hot
        logits *= self.s

        return logits
```

Figure 4.3.11.6 Custom ArcFace Class

The MobileNetV2ArcFace class combines MobileNetV2 as the backbone for feature extraction with an ArcFace head for classification. The input is first converted into embeddings by the backbone, then passed to ArcFace with labels to produce logits, tto enable efficient feature learning with discriminative margins for face recognition.

```python
class MobileNetV2ArcFace(nn.Module):
    def __init__(self, backbone, arcface_head):
        super(MobileNetV2ArcFace, self).__init__()
        self.backbone = backbone
        self.arcface = arcface_head

    def forward(self, x, labels):
        embeddings = self.backbone(x)
        logits = self.arcface(embeddings, labels)
        return logits
```

Figure 4.3.11.7 MobileNetV2ArcFace Class

The final face recognition model is built by combining MobileNetV2 as the backbone with an ArcFace head. It sets the number of classes from the dataset, configures ArcFace with the embedding size and class count, and moves the complete model to the chosen GPU for training or inference

```python
num_classes = len(class_to_idx)

arcface_head = ArcFace(in_features=embedding_dim, out_features=num_classes)
model = MobileNetV2ArcFace(backbone, arcface_head).to(device)
```

Figure 4.3.11.8 Builds Final Model

The AdamW optimizer is set up with separate learning rates: a smaller one for the pretrained MobileNetV2 backbone and a larger one for the newly added ArcFace head. It also applies weight decay as default 1e-3 to improve generalization and reduce overfitting.

```python
optimizer = torch.optim.AdamW([
    {"params": model.backbone.parameters(), "lr": config['lr_features']},
    {"params": model.arcface.parameters(), "lr": config['lr_classifier']}
], weight_decay=config.get('weight_decay', 1e-3))
```

Figure 4.3.11.9 Set Up AdamW optimizer

Then, runs the training and validation loop for fine-tuning the face recognition model. It trains with CrossEntropyLoss, updates weights using backpropagation, and calculates training loss and accuracy. In validation mode, it evaluates performance without updating weights, checking embedding stability and computing validation loss and accuracy.

```
criterion = nn.CrossEntropyLoss()

for epoch in range(config['num_epochs']):
    model.train()
    train_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        logits = model(inputs, labels)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * inputs.size(0)
        _, preds = torch.max(logits, 1)
        correct += preds.eq(labels).sum().item()
        total += labels.size(0)

    train_loss /= total
    train_acc = 100 * correct / total

    # Validation
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        embeddings = model.backbone(inputs)
        print("Batch embeddings mean:", embeddings.mean().item(), "std:", embeddings.std().item())
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            logits = model(inputs, labels)
            loss = criterion(logits, labels)
            val_loss += loss.item() * inputs.size(0)
            _, preds = torch.max(logits, 1)
            val_correct += preds.eq(labels).sum().item()
            val_total += labels.size(0)

    val_loss /= val_total
    val_acc = 100 * val_correct / val_total

    wandb.log({
        'epoch': epoch + 1,
        'train_loss': train_loss,
        'train_acc': train_acc,
        'val_loss': val_loss,
        'val_acc': val_acc
    })

    print(f"[Epoch {epoch+1}] Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")
```

Figure 4.3.11.10 Fine Tuning Process

From the classification report, it can be observed that the training accuracy of the model is 0.986, while the macro and weighted average of precision is 0.987, and recall is 0.986.

	precision	recall	f1-score	support
Akshay Kumar	1.000	1.000	1.000	32
Alexandra Daddario	1.000	1.000	1.000	61
Alia Bhatt	1.000	0.938	0.968	48
Amitabh Bachchan	1.000	0.977	0.988	43
Andy Samberg	0.983	1.000	0.992	59
Anushka Sharma	0.884	1.000	0.938	38
Billie Eilish	0.985	1.000	0.992	65
Brad Pitt	0.988	1.000	0.994	82
Camila Cabello	1.000	0.982	0.991	56
Charlize Theron	0.979	0.979	0.979	48
Claire Holt	0.984	1.000	0.992	63
Courtney Cox	0.980	0.960	0.970	50
Dwayne Johnson	0.964	1.000	0.982	27
Elizabeth Olsen	1.000	0.909	0.952	44
Ellen Degeneres	1.000	0.978	0.989	45
Henry Cavill	1.000	0.971	0.985	69
Hrithik Roshan	1.000	1.000	1.000	65
Hugh Jackman	0.962	1.000	0.981	76
Jessica Alba	0.986	1.000	0.993	70
Kashyap	0.850	1.000	0.919	17
Lisa Kudrow	0.936	1.000	0.967	44
Margot Robbie	1.000	0.956	0.977	45
Marmik	1.000	0.944	0.971	18
Natalie Portman	1.000	1.000	1.000	70
Priyanka Chopra	1.000	0.985	0.992	65
Robert Downey Jr	1.000	1.000	1.000	77
Roger Federer	1.000	0.939	0.968	49
Tom Cruise	0.973	0.973	0.973	37
Vijay Deverakonda	1.000	1.000	1.000	77
Virat Kohli	1.000	1.000	1.000	32
Zac Efron	1.000	1.000	1.000	59
yiqian	1.000	1.000	1.000	48
accuracy			0.986	1679
macro avg	0.983	0.984	0.983	1679
weighted avg	0.987	0.986	0.986	1679

Figure 4.3.11.11 Classification Report of train_loader

Lastly, saves the trained model weights (state_dict) into a file named mobilefacenet_xyq_arcface.pth.

```
5]: model_name = "mobilefacenet_xyq"
    torch.save(model.state_dict(), f"{model_name}_arcface.pth")
```

Figure 4.3.11.12 Save Model

Model Validation for the MobileFaceNet

Then, we conduct the model validation by using the val_loader. From the classification report, it can be observed that the validation accuracy of the model is 0.817, while the macro and weighted average of precision is 0.839, and recall is 0.817.

```
report = get_classification_report(
    model,
    val_loader,
    class_to_idx=class_to_idx,
    device=device,
    use_arcface=True
)
print(report)
```

```
                  precision    recall  f1-score   support

      Akshay Kumar      0.625     0.625     0.625         8
Alexandra Daddario      1.000     0.933     0.966        15
       Alia Bhatt       0.643     0.750     0.692        12
  Amitabh Bachchan      1.000     0.818     0.900        11
      Andy Samberg      0.765     0.867     0.812        15
    Anushka Sharma      0.500     0.600     0.545        10
     Billie Eilish      0.714     0.938     0.811        16
         Brad Pitt      0.900     0.857     0.878        21
    Camila Cabello      0.800     0.857     0.828        14
   Charlize Theron      0.875     0.583     0.700        12
       Claire Holt      0.824     0.875     0.848        16
     Courtney Cox       0.714     0.769     0.741        13
    Dwayne Johnson      0.778     1.000     0.875         7
   Elizabeth Olsen      0.833     0.455     0.588        11
   Ellen Degeneres      1.000     0.909     0.952        11
      Henry Cavill      0.824     0.824     0.824        17
    Hrithik Roshan      0.733     0.688     0.710        16
      Hugh Jackman      0.810     0.895     0.850        19
      Jessica Alba      0.783     1.000     0.878        18
           Kashyap      0.300     0.750     0.429         4
       Lisa Kudrow      0.733     1.000     0.846        11
     Margot Robbie      0.636     0.636     0.636        11
            Marmik      0.667     0.500     0.571         4
   Natalie Portman      0.938     0.882     0.909        17
   Priyanka Chopra      1.000     0.562     0.720        16
  Robert Downey Jr      0.938     0.789     0.857        19
     Roger Federer      1.000     0.917     0.957        12
       Tom Cruise       0.889     0.889     0.889         9
Vijay Deverakonda      0.900     0.900     0.900        20
      Virat Kohli      1.000     0.750     0.857         8
        Zac Efron      1.000     0.800     0.889        15
           yiqian      1.000     1.000     1.000        12

         accuracy                          0.817       420
        macro avg      0.816     0.801     0.796       420
     weighted avg      0.839     0.817     0.818       420
```

Figure 4.3.11.13 Classification Report of val_loader

Model Testing for the MobileFaceNet

Finally, we conduct the model validation by using the test_loader. From the classification report, it can be observed that the testing accuracy of the model is 0.805, while the macro and weighted average of precision is 0.837, and recall is 0.805.

```
report = get_classification_report(
    model,
    test_loader,
    class_to_idx=class_to_idx,
    device=device,
    use_arcface=True
)
print(report)
```

```
                   precision    recall  f1-score   support

      Akshay Kumar      0.417     0.500     0.455        10
Alexandra Daddario      0.938     0.938     0.938        16
        Alia Bhatt      0.812     0.684     0.743        19
  Amitabh Bachchan      1.000     1.000     1.000        20
      Andy Samberg      0.895     0.944     0.919        18
    Anushka Sharma      0.441     0.750     0.556        20
     Billie Eilish      0.875     0.824     0.848        17
         Brad Pitt      0.789     0.882     0.833        17
    Camila Cabello      0.650     0.765     0.703        17
   Charlize Theron      0.938     0.833     0.882        18
       Claire Holt      1.000     0.706     0.828        17
      Courtney Cox      0.909     0.588     0.714        17
    Dwayne Johnson      0.900     0.818     0.857        11
   Elizabeth Olsen      0.867     0.812     0.839        16
   Ellen Degeneres      0.895     0.895     0.895        19
      Henry Cavill      1.000     0.750     0.857        20
    Hrithik Roshan      0.773     0.850     0.810        20
      Hugh Jackman      0.739     1.000     0.850        17
      Jessica Alba      0.621     0.900     0.735        20
           Kashyap      0.400     0.444     0.421         9
       Lisa Kudrow      0.750     1.000     0.857        15
     Margot Robbie      0.750     0.938     0.833        16
            Marmik      1.000     0.600     0.750        10
   Natalie Portman      0.889     0.444     0.593        18
   Priyanka Chopra      1.000     0.476     0.645        21
  Robert Downey Jr      0.842     0.941     0.889        17
     Roger Federer      0.938     0.938     0.938        16
        Tom Cruise      0.583     0.583     0.583        12
Vijay Deverakonda      0.938     0.833     0.882        18
      Virat Kohli      0.818     1.000     0.900         9
        Zac Efron      1.000     0.824     0.903        17
           yiqian      1.000     1.000     1.000        21

         accuracy                          0.805       528
        macro avg      0.824     0.796     0.795       528
     weighted avg      0.837     0.805     0.806       528
```

Figure 4.3.11.14 Classification Report of test_loader

4.4 System Operation

4.4.1 Onboarding Activity

The onboarding screen is designed as a series of introductory pages shown to users when they open the app for the first time. Its purpose is to introduce the application's purpose, highlight key features and guide first-time users to attract and motivate users before they reach the main functionality, like login and signup functionalities. The onboarding screen enables patients who may have cognitive difficulties and caregivers who need clear, efficient tools to understand the features of the application quickly and determine whether the application is what they need. This can reduce their confusion and increase user retention. Only shown once is the technical behaviour inside the onboarding activity, by controlling the local storage flag, so the onboarding screen does not appear again after the first launch. User can click the button at the last onboarding page or skip to jump to the welcome screen that consists of login and signup options.

Figure 4.4.1.1 Onboarding Screen Page 1

Figure 4.4.1.2 Onboarding Screen Page 2

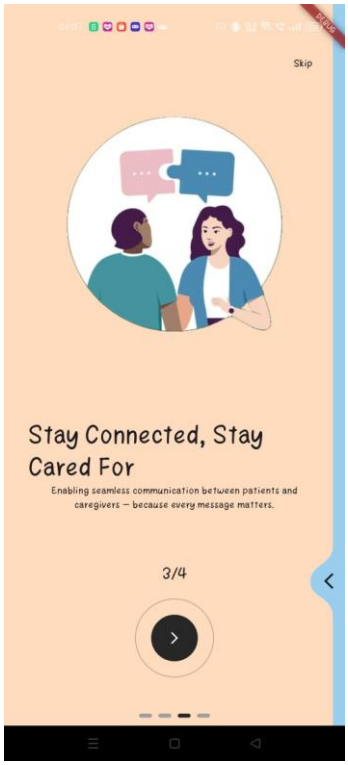


Figure 4.4.1.3 Onboarding Screen Page 3

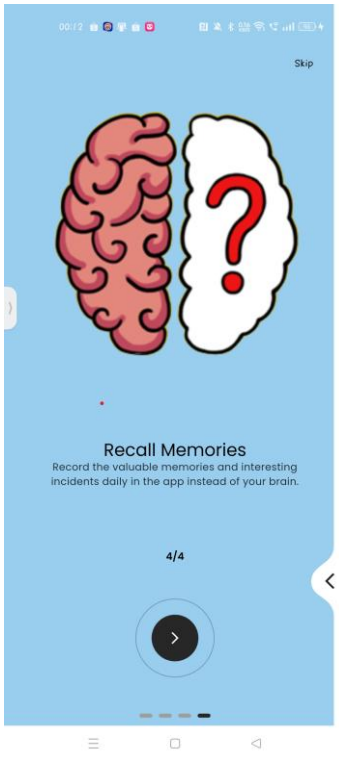


Figure 4.4.1.4 Onboarding Screen Page 4

4.4.2 Signup activity

The Sign-Up Activity is a crucial entry point of the mobile application, allowing new users, whether patients or caregivers, to create an account and gain access to the application's functionalities. This activity plays a fundamental role in establishing secure, personalized access to ensure that users can be uniquely identified and their data securely managed.

The signup form typically collects essential details such as full name, email address, phone number, and password. The real-time input validation is applied to prevent empty or incorrectly formatted fields. For example, the input fields cannot be empty if they want to submit the form. There are other validations to the input fields, such as the email input field should have the symbol @ or dot, while the password input field only accepts 8 characters with uppercase, lowercase, numbers, and symbols. Once the user successfully submits the form, their credentials are processed through Firebase Authentication, which securely stores account information and handles account

creation, meanwhile storing the data in the Firestore database for future use. After that, the user will come to the email verification screen, and the email will be sent to the registered email for verification with the link. The user needs to click the link to complete the verification. If the sign-up and verification are successful, the user is expected to be redirected to the different dashboard based on the role. For this case, the core method for signing up is using the email.

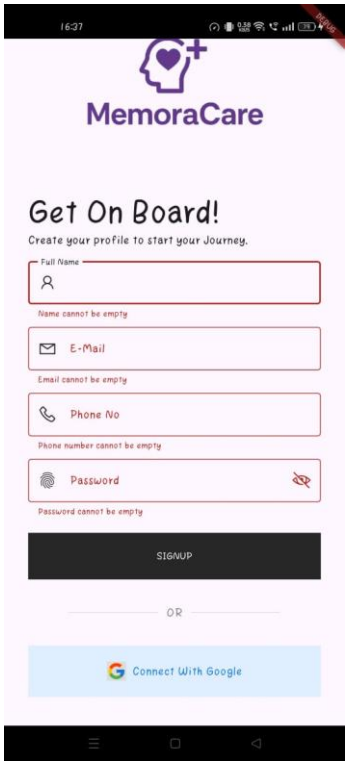


Figure 4.4.2.1 Input fields cannot be empty

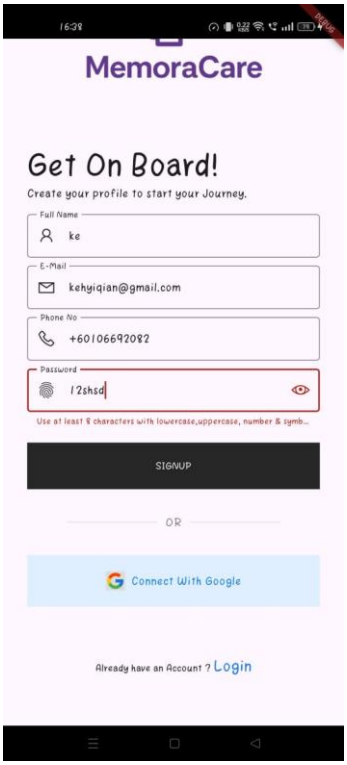


Figure 4.4.2.2 Password Format validation

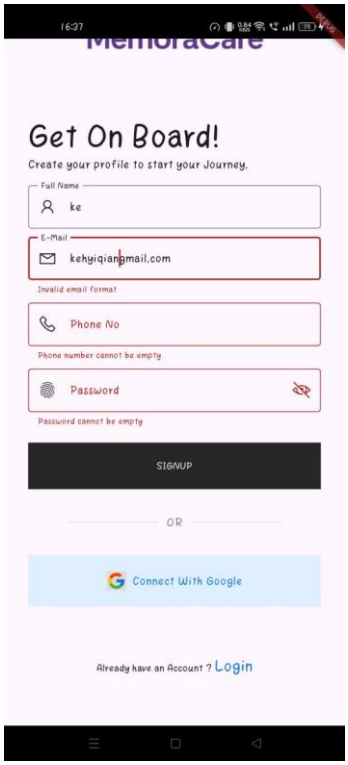


Figure 4.4.2.3 Email format validation (lack @)

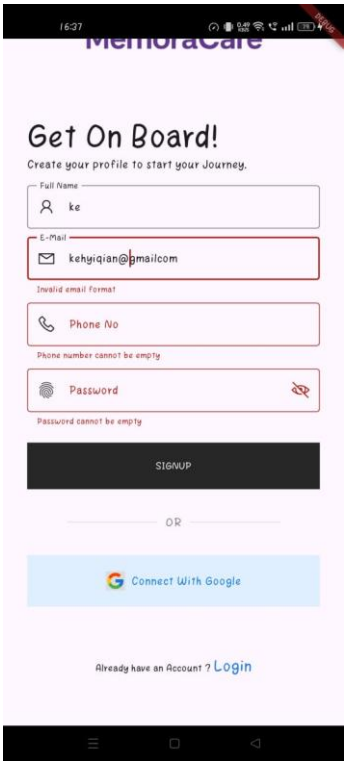


Figure 4.4.2.4 Email format validation (lack dot)

Figure 4.4.2.5 Email verification page

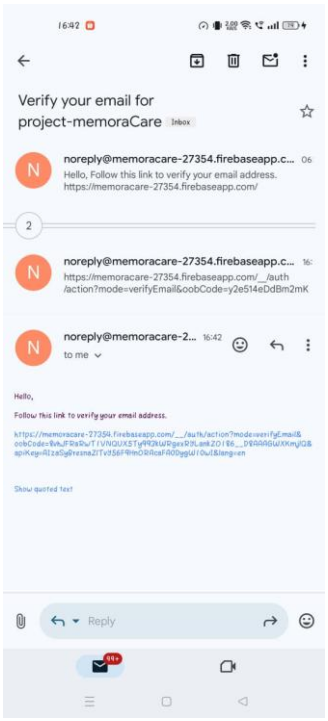


Figure 4.4.2.6 Receive verification email

Figure 4.4.2.7 Done verification

Furthermore, users can also sign up for the application by connecting with Google, which is a quite simple and convenient way for the user to sign up compared to signing up with an email. The credentials and data will also be stored in the Firebase authentication and Firestore database. Whatever method to use to sign up, the user needs to select their role after signing up.

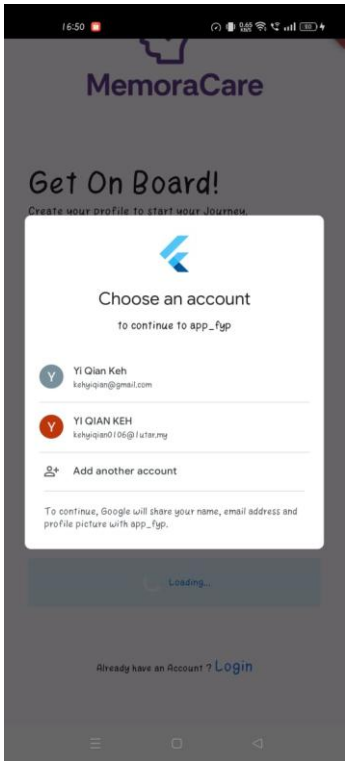


Figure 4.4.2.8 Choose an account for Google Sign in

Figure 4.4.2.9 Choose the Role

By implementing this activity, the application establishes the foundation for secure user access, session management, and future data personalization, such as linking patient profiles to GPS tracking.

4.4.3 Login activity

The Login Activity is another core component of the mobile application that enables users, whether patients or caregivers, to access their personalized content securely for accessing the application. This screen allows users to log in using their email and password credentials that have been previously registered during the signup process. There is a form used to prompt the users to enter their email and password in the input textfield, and simple validation has been applied on the fields, such as the input cannot be empty and the validation of email format.

Once the form is validated, the user clicks the login button, which will trigger the loginWithEmailAndPassword() method in the login controller. The Firebase

Authentication will automatically verify the user's credentials. The login activity contributes significantly to the application's security and usability to prevent unauthorized access. Users can also choose to connect with Google to access the app directly.

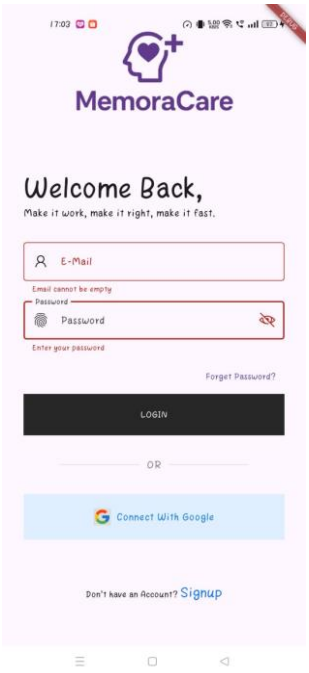


Figure 4.4.3.1 Login screen and input validation

4.4.4 Dashboard

The dashboard is different based on the role; there are some limitations to accessing the modules for different roles. There are 7 modules open to the patients, which are Information Card, People Enroll, Face Recognition, Gallery, Chat, Return Route, and Diary modules, as shown in Figure 4.4.4.1 and Figure 4.4.4.2, while only 4 modules are open to the caregiver: Information Card, Face Recognition, Chat, and GPS Tracking modules, as shown in Figure 4.4.4.3. The Information Card, Face Recognition, and Chat modules are allowed for both patients and caregivers to access. For the reason why we block some modules for patients in order to protect the privacy and personal information about the patient, because of some modules, like People Enroll, Gallery, and Diary modules, will record the patient's personal memories. Then, the information card is opened to both patients and caregivers because the card is public, which will record the fundamental personal information about the patient. It is useful if the patient

wanders. For the face recognition module, it is also open to both because caregivers can use this module to recognize visitors to avoid the patient from not recognizing the visitor due to cognitive and memory loss problems. The Chat module opens to both so that the patients and caregivers can communicate with each other. The GPS tracking is the specific module designed for the caregiver to monitor the location of the patient, while the Return Route module is designated for the patient to find the way back if wandering.

The arrangement of the modules looks spacious to avoid all the modules from squeezing together, so the user can slide down or click the down arrow button to access the other modules. We can also observe that there are some descriptions below the modules to let the user have a simple understanding of each module. Furthermore, the welcome header is placed at the upside of both modules, the blue word represents the connected user's name, and it is clickable to access the connected user management quickly, and the phone button allows the user to call the set phone number faster.

Figure 4.4.4.1 Patient's Dasboard1

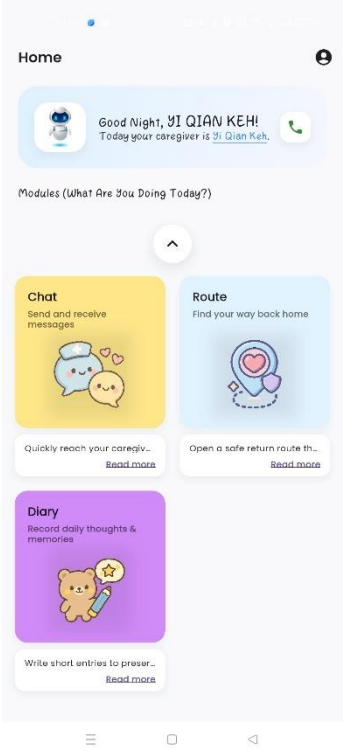


Figure 4.4.4.2 Patient's Dasboard2

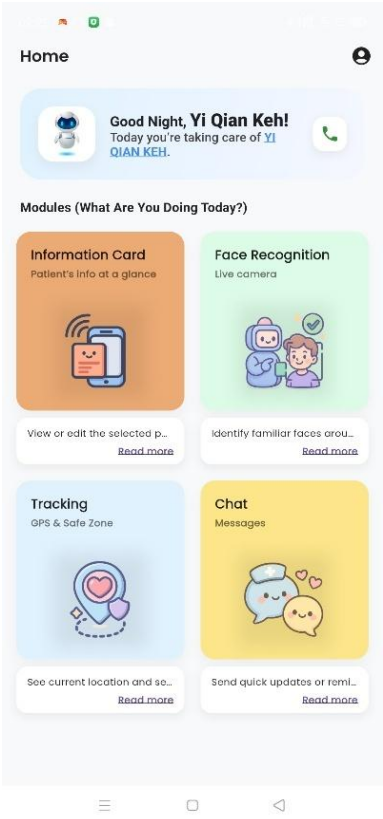


Figure 4.4.4.3 Caregiver's Dashboard

4.4.5 User Profile

The user profile provides identical functions to both patients and caregivers and has been integrated with other components like Settings, Notifications, Invitations, and User Management, as shown in Figure 4.4.5.1. At the top, the head portrait, name, and email are displayed and retrieved from Firebase. Then, the edit button allows the user to update their name, email, and phone number, which are stored in the Firebase as shown in Figure 4.4.5.2.

Furthermore, the settings as shown in Figure 4.4.5.3 allow the user to manually upload their location to Firebase, as shown in Figure 4.4.5.4; the stored latitude and longitude in the Firebase are not renewed or updated if this setting is not open. Below is the contact list that manages the phone number at the welcome header located in the

dashboard, which allows the CRUD actions to the phone number, and the data will be stored in the Firebase as shown in Figure 4.4.5.5.

Moreover, the invitations and notifications are related; the user can send the connection link to other users by using their unique and specific invitation ID to send to the desired connected user with a message to clarify the reason for the connection, as shown in Figure 4.4.5.6, so that the receiver's side knows who the sender is or what happens. Once the invitation is sent, the Firebase will be updated with the contents of the invitation as shown in Figure 4.4.5.7. Below is the user's own invitation ID that is hidden to prevent the ID from leaking easily, and can be copied directly to easily pass the ID to another user. When the invitation is sent, the notification page on the receiver's side can receive the invitation and decide whether to accept it, as shown in Figure 4.4.5.8. Only if the receiver side agrees to the invitation, as shown in Figure 4.4.5.9, is the connection between the two sides formally established to avoid unknowingly linking, and the Firebase will be updated in the patient's database with the caregiver ID or vice versa, as shown in Figure 4.4.5.10. Besides that, the sender can cancel the invitation via the notification page, as shown in Figure 4.4.5.11.

For the user management, it allows the user to manage the connected user, either select to activate the connection, such as when the caregiver selects a patient, then can access the database of that patient with certain limitations, and can only activate one patient at the same time or unlink the connection with the opposite side, as shown in Figure 4.4.5.12. The Firebase will be updated with the existing active user, which can only be one, as shown in Figure 4.4.5.13. Lastly, the user can log out through this page.

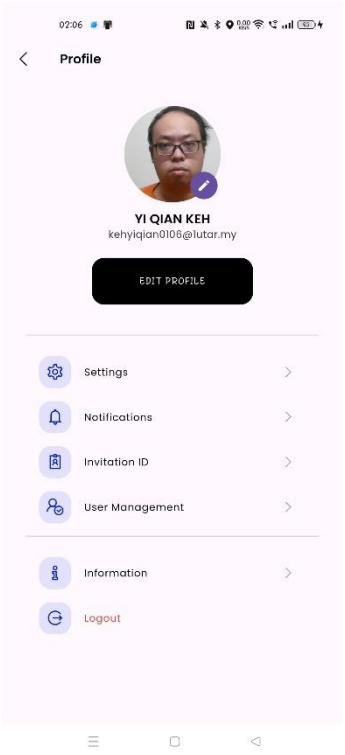


Figure 4.4.5.1 User Profile Screen

Figure 4.4.5.2 Edit User Profile Screen

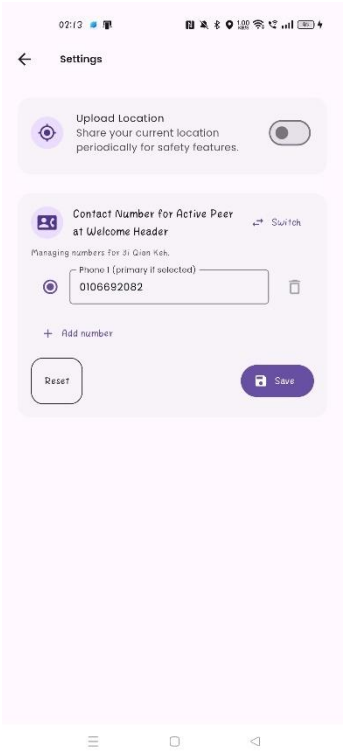


Figure 4.4.5.3 Settings Page

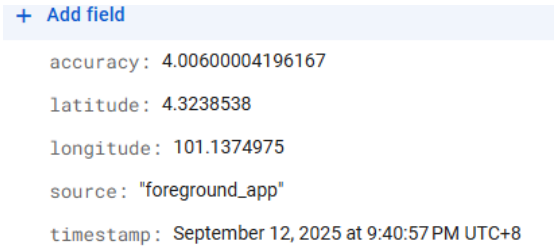


Figure 4.4.5.4 Stored Locations

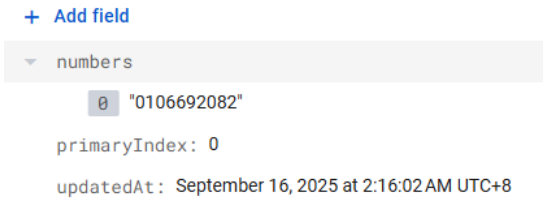


Figure 4.4.5.5 Stored Contact Number

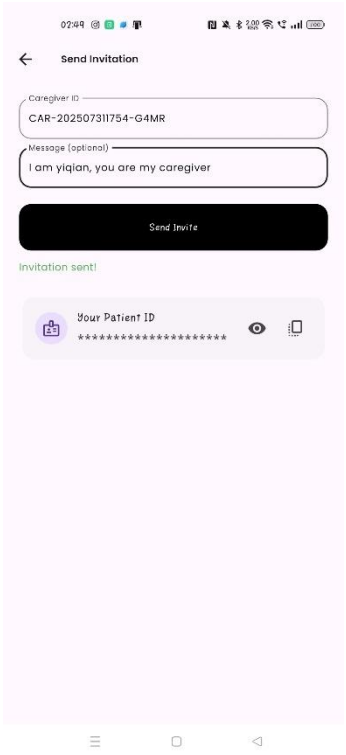


Figure 4.4.5.6 Send Invitation

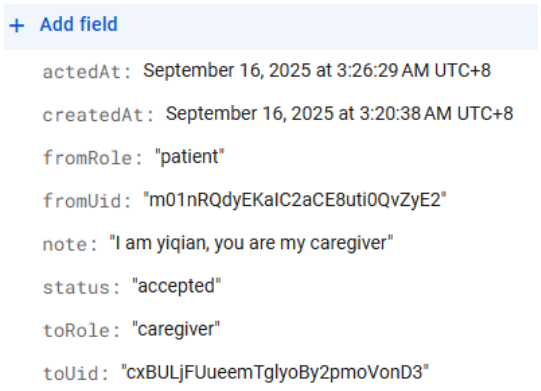


Figure 4.4.5.7 Stored Invitations Data

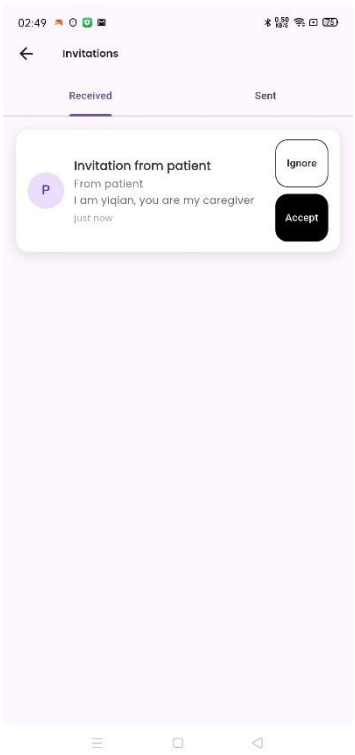


Figure 4.4.5.8 Receiver Receives the Invitation

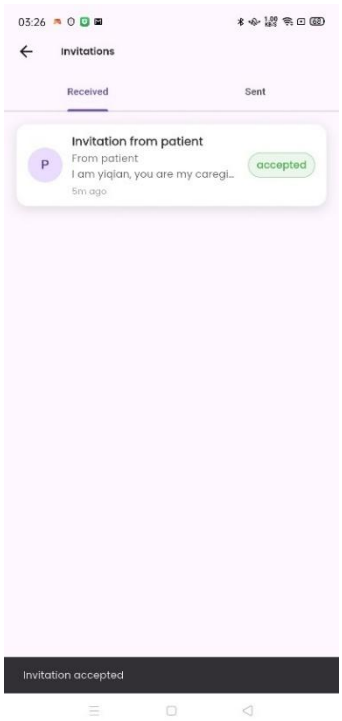


Figure 4.4.5.9 Receiver Agrees to the Invitation

Figure 4.4.5.10 The Caregiver is saved

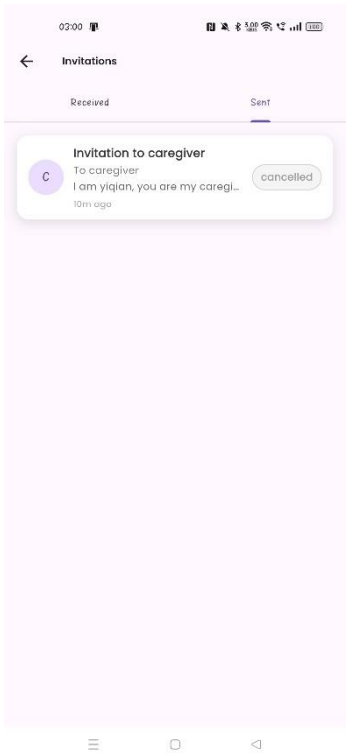


Figure 4.4.5.11 Sender Cancels the Invitation

Figure 4.4.5.12 User Management Screen

Figure 4.4.5.13 ID for Existing Active User

4.4.6 Face Recognition Module

First, the Face Recognition module has its own dashboard, as shown in Figure 4.4.6.1. It can be separated into two main sections: one is instruction that guides the user on how to use the module effectively to obtain accurate results, and the other is face recognition, which allows the user to recognize people stored in Firebase. The face recognition section is composed of three components: the first is an image stage that displays the uploaded image. Second is the result, and third is the two buttons that allow the user to choose the methods for face recognition. The user can choose to conduct face recognition in real time with a live camera or select a photo from local storage.

There is a fixed square box assigned at the center of the screen. The system will guide the user to place the face in the box at the top of the camera, and a bounding box will be displayed if a face is detected. Only the face is placed properly at the center, then the green bounding box and the guidance statement appears “Perfect! Hold still” as shown in Figure 4.4.6.2 will be displayed together with the name and confidence so the results of the recognition is more accurate and can observe that beside the name there has a show more button which connect to the stored memory page of that person, the user can click it to see and recall their stored memories otherwise it will displayed red box as shown in Figure 4.4.6.3. The threshold is set at 0.70 in this project, and the face is detected by using ML Kit. The face recognition goes through the computation and comparison of the embeddings that were generated from a real frame and stored embeddings of the Firebase as shown in the Figure, to realize the function of recognizing the person.

Moreover, the image can be selected from local storage and do the recognition, same as real-time recognition, it will use ML Kit to detect face, and the threshold is also set as 0.70, the name, confidence, and the show more buttons will be presented if the face is recognized as shown in Figure 4.4.6.4. More testing examples of face recognition will be conducted and shown in the testing part of Chapter 5.

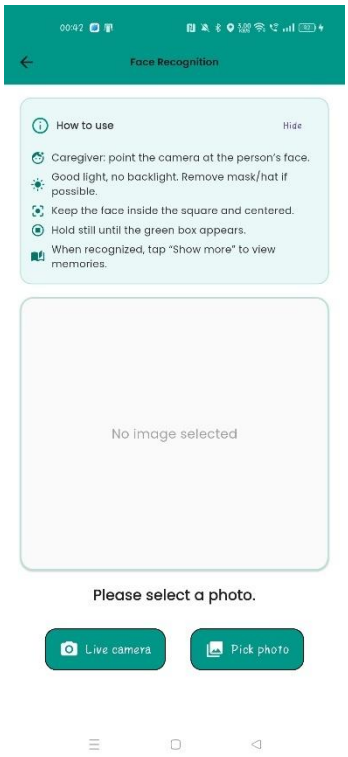


Figure 4.4.6.1 Face Recognition's Dashboard

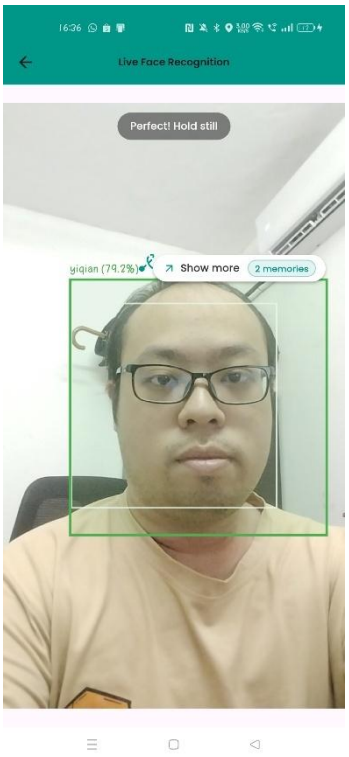


Figure 4.4.6.2 Green Box is Displayed

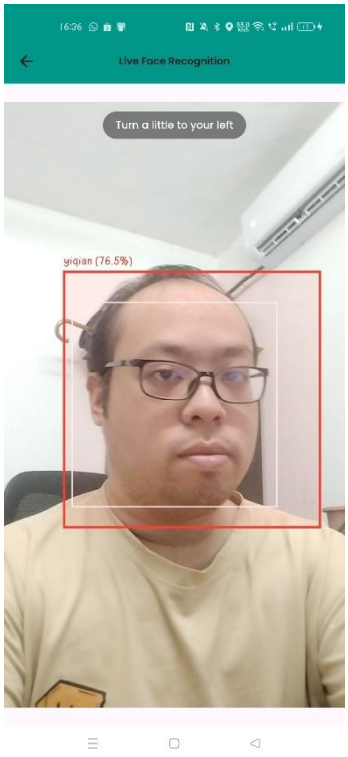


Figure 4.4.6.3 Red Box is Displayed

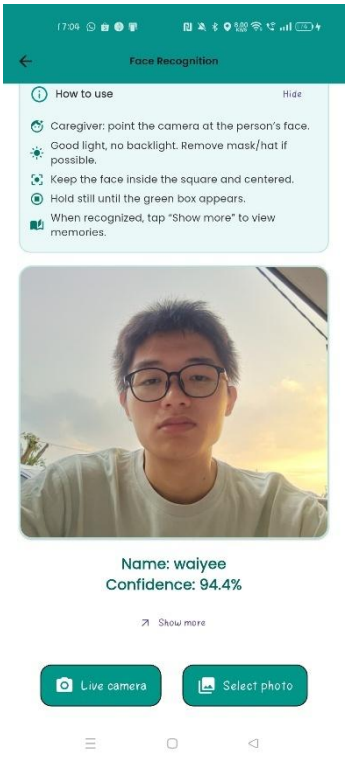


Figure 4.4.6.4 Recognized by Selecting Image from Local Storage

4.4.7 People Enroll Module

From the visualization of the dashboard of the People Enroll module, as shown in Figure 4.3.7.1, one can observe that there are five main regions for the user to complete the enrolment process step by step. First and foremost, the user needs to read the instructions and then label the name of the person that they want to save. After that, the user needs to upload exactly 10 images to generate the embeddings; they can either select the images from local storage or capture by the system's customized camera. The Add Memory button will be lit if the precondition is met, which is that both the name and image are placed completely and are valid, as shown in Figure 4.4.7.2.

Through the Add Memory button, the user can enter the page that is used to record valuable memories or personal information about the person, such as relationships, notes, and certain media, as shown in Figure 4.4.7.3. Then, after all name, images, and memories are added, the upload button is lit as shown in Figure 4.4.7.4, and the user can click the button to upload all metadata to the Firebase with prompt the confirmation

to the user as shown in Figure 4.4.7.5, if the user is confirmed the data will be stored as shown in Figure 4.4.7.6 and Figure 4.4.7.7.

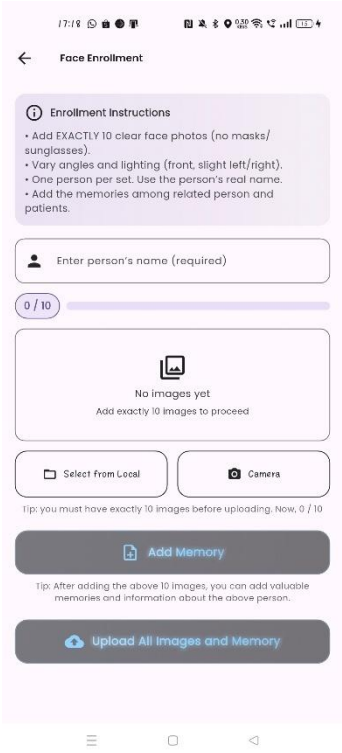


Figure 4.4.7.1 People Enroll Dashboard

Figure 4.4.7.2 Add Memory button is Lit if Name and Images are placed

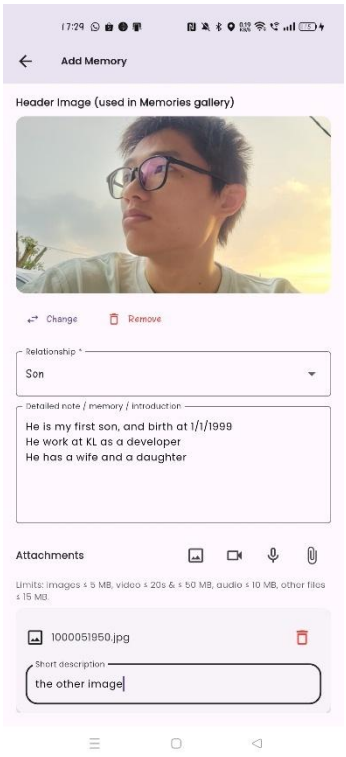


Figure 4.4.7.3 Add Memory Page

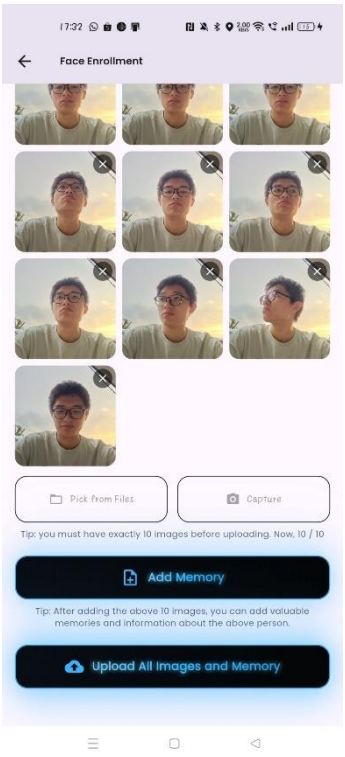


Figure 4.4.7.2 Upload button is Lit if Name, Images, and Memories are placed

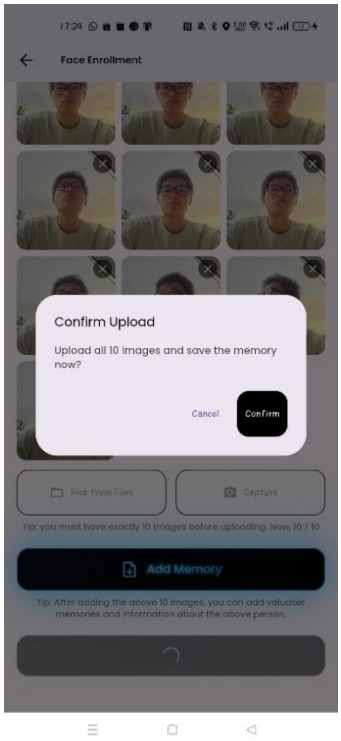


Figure 4.4.7.5 Upload Confirmation

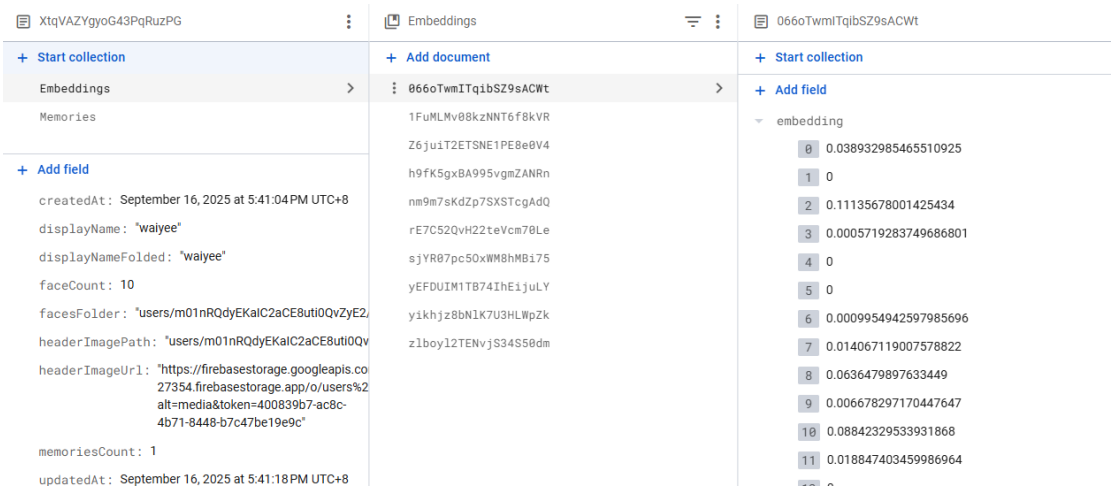


Figure 4.4.7.6 Stored Embeddings

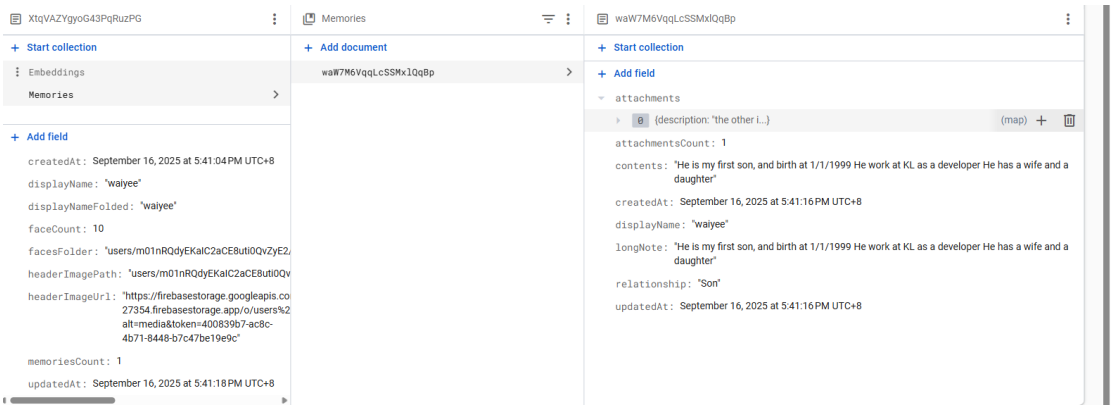


Figure 4.4.7.7 Stored Memories

4.4.8 Gallery Module

The Gallery module is the module designed to manage the stored data of the enrolled person, such as name, faces, and memories. The dashboard of the Gallery includes four main components, as shown in Figure 4.4.8.1, which are the search bar for searching by name as shown in Figure 4.4.8.2, the filter for categorizing the relationships as shown in Figure 4.4.8.3, sorting based on enrollment time either ascending or descending as shown in Figure 4.4.8.4, and lastly, the results or displayed stored people.

The user can click the header portrait of the person to manage the information of that person. First, the user will enter the page that presents the overall information of the person, such as the header image, name, number of stored faces, and number of stored

memory items, as shown in Figure 4.4.8.4. The user can do the normal actions like view, edit, and delete to the header image and name. Then, the delete button at the bottom will delete the entire information and data from Firebase about the person if clicked and the system will prompt confirmation for the deletion, as shown in Figure 4.4.8.5.

The user can click the Faces card to manage the faces that are stored in the Firebase, as shown in Figure 4.4.8.6. This page allows the user to read, add, and delete the faces stored in the Firebase, and it will obey the conditions about the number of faces that ensure the images are more than or equal to 10.

The user can click the Memories card to manage the memories that are stored in the Firebase, as shown in Figure 4.4.8.7. This page allows the user to read, add, and delete the memories stored in the Firebase, and also prompts for confirmation for the deletion of whatever items they want to delete, as shown in Figure 4.4.8.8.

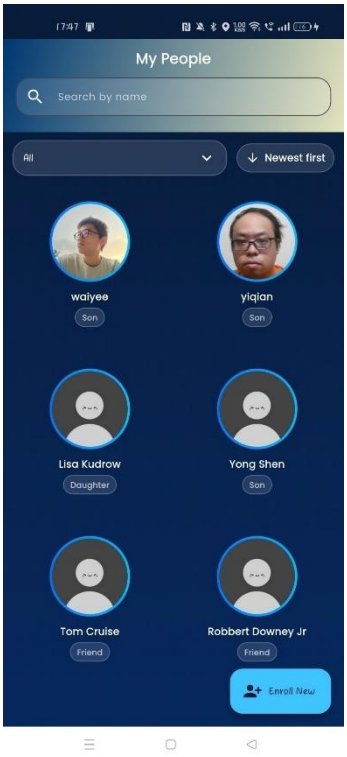


Figure 4.4.8.1 Gallery Dashboard

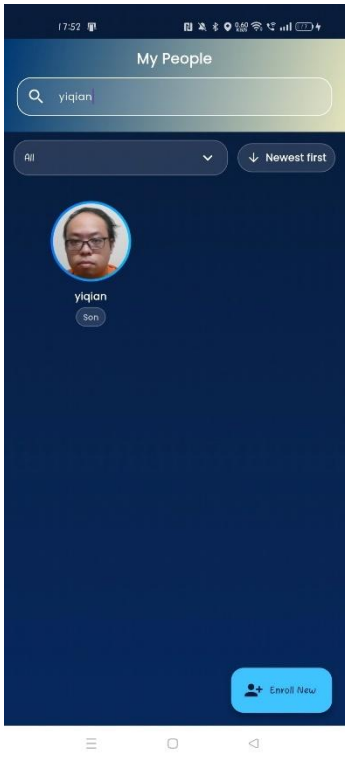


Figure 4.4.8.2 Search Name

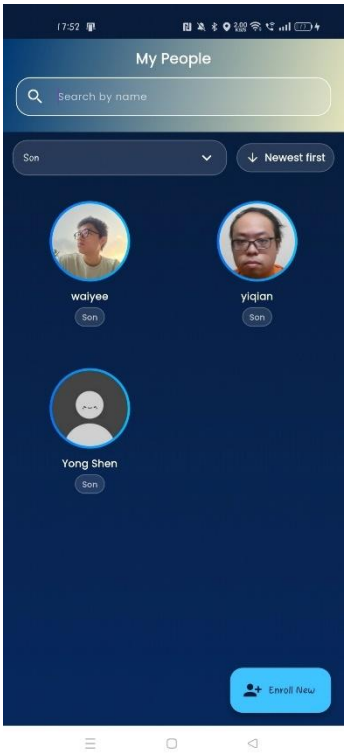


Figure 4.4.8.3 Filter Relationship

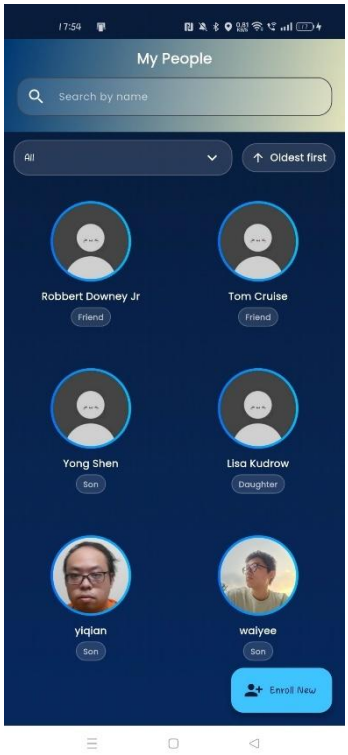


Figure 4.4.8.4 Sort with Ascending Order

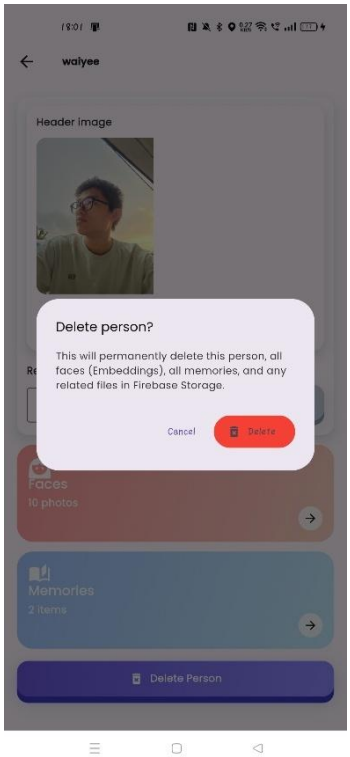


Figure 4.4.8.5 Delete Entire Person

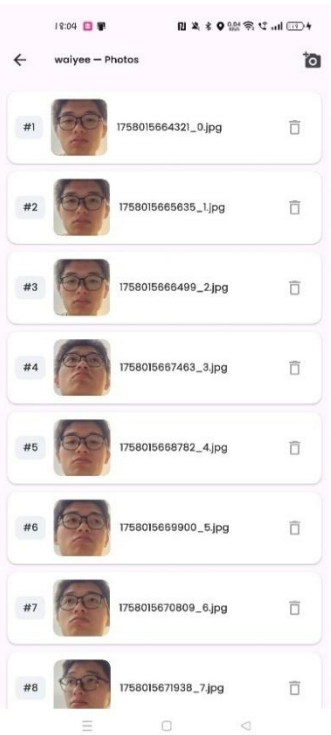


Figure 4.4.8.6 Manage Faces

Figure 4.4.8.7 Manage Memories

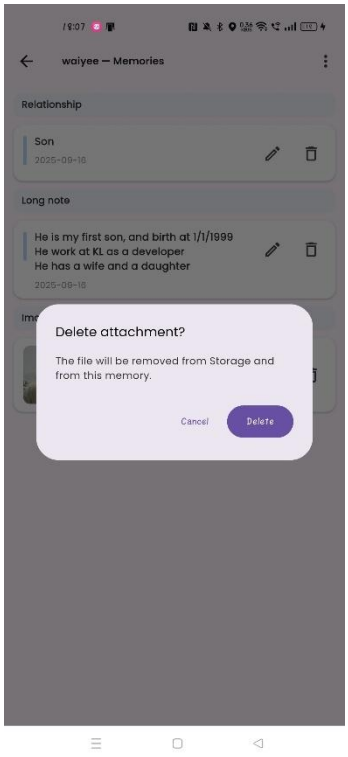


Figure 4.4.8.8 Delete Memories Item

4.4.9 Information Card

The Information Card Module, as shown in Figure 4.3.9.1, allows the user to create a card for a patient. It is designed to record personal information of the patient, emergency contact, medical conditions, and so on. This card is a role-like identification card. It is useful and helpful when the patient is wandering by showing this card to other people, and it can also be shown to the caregiver to add to the initial understanding and impress them about the patient.

The system allows the CRUD actions on the card, which are create action as shown in Figure 4.4.9.2, read action as shown in Figure 4.4.9.3, update action as shown in Figure 4.4.9.4, and delete action. Before the deletion of the card, there is also prompt confirmation for the deletion, as shown in Figure 4.4.9.5. Both the data will be stored and updated in the Firebase, as shown in Figure 4.4.9.9.

Through the card, the user can directly call the phone number of the patient on the card as shown in Figure 4.4.9.6, and also write the NDEF URL about the information on the

card into the NFC tag. In this project, the type of NFC tag used is the Ntag 213. It will write the URL to the NFC tag if the system detects the tag in the surroundings by clicking the button "Write nfc tag". The message "Public tag created" is displayed if the system successfully wrote the NDEF URL to the NFC tag, as shown in Figure 4.4.9.7 and the public information card will be stored in the Firebase, as shown in Figure 4.4.9.10, so that the information will update and renew if the user edits the information card in the module. The other user can detect the NFC tag to access the webpage that presents some information about the patient, as shown in Figure 4.4.9.8.

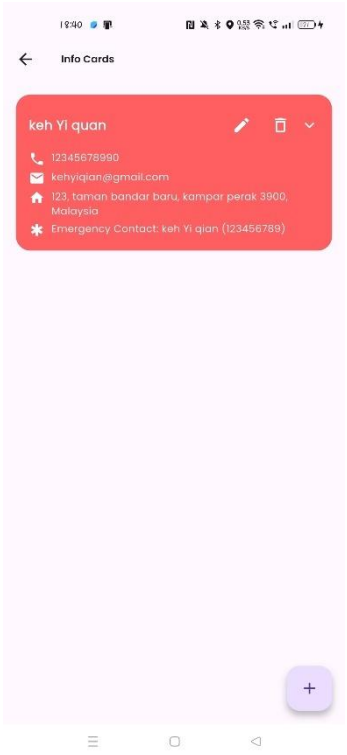


Figure 4.4.9.1 Information Card Screen

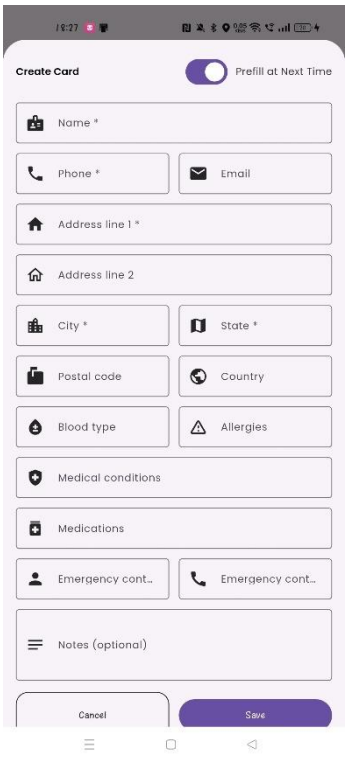


Figure 4.4.9.2 Create Card

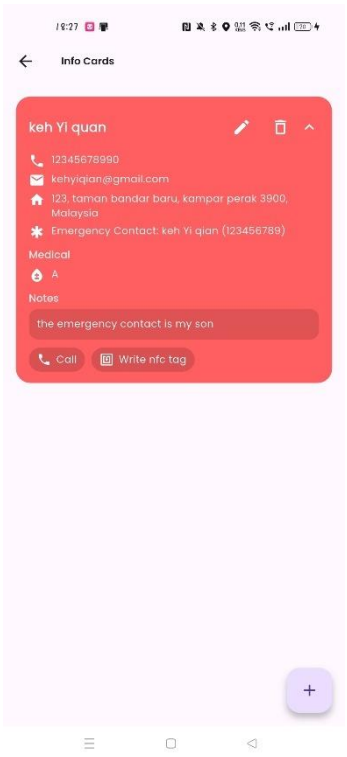


Figure 4.4.9.3 Read Card

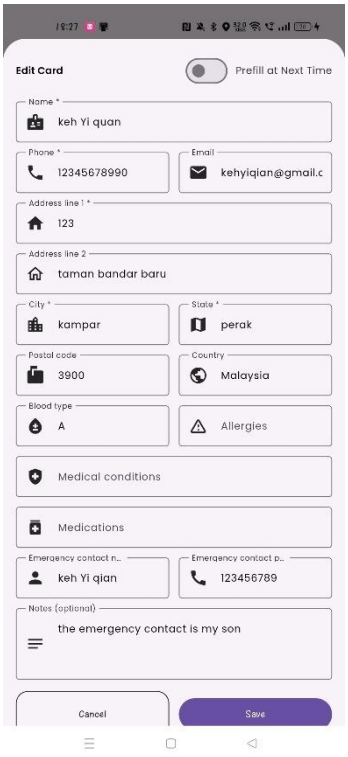


Figure 4.4.9.4 Update Card

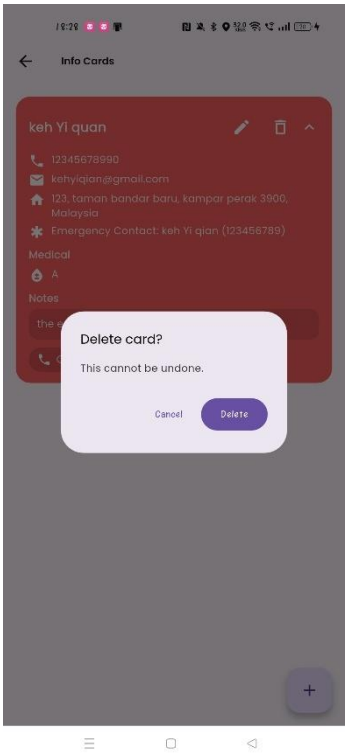


Figure 4.4.9.5 Delete Card

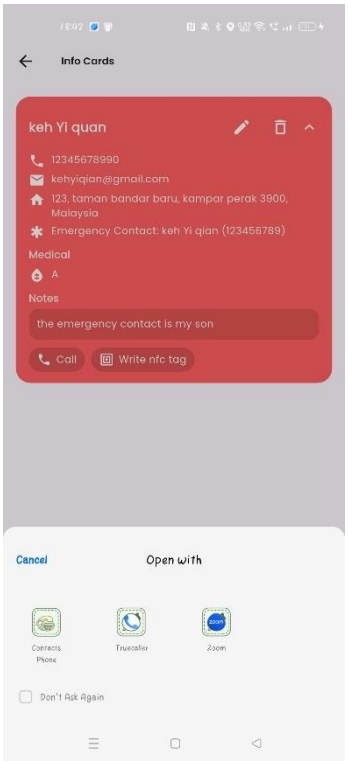


Figure 4.4.9.6 Call Patients

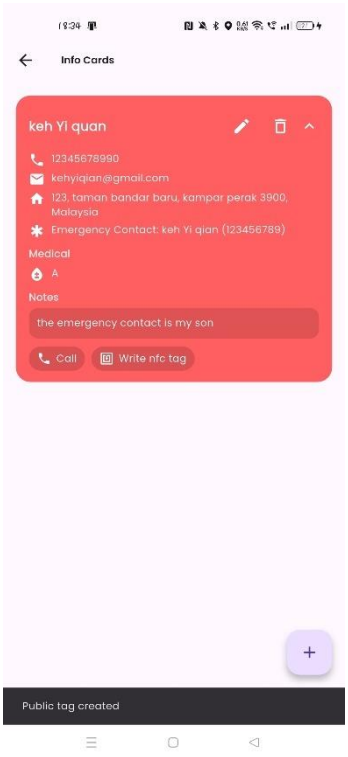
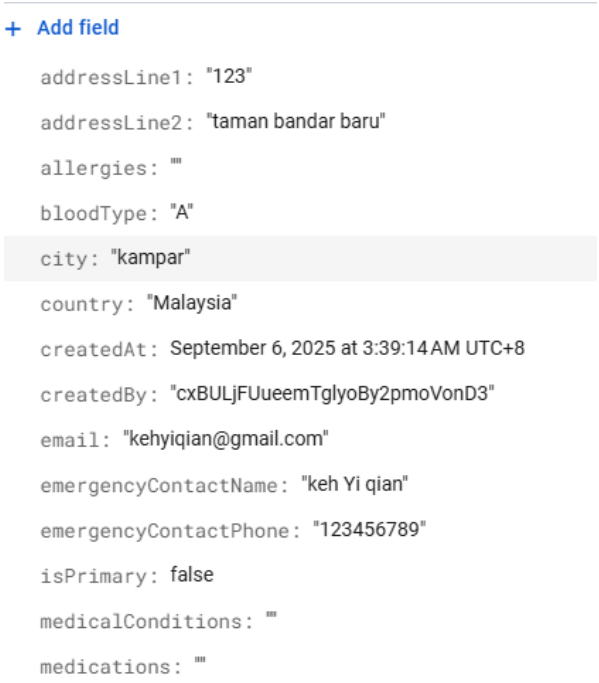


Figure 4.4.9.7 Display Message About Successfully Writing to the NFC Tag

Figure 4.4.9.8 Access the URL via NFC Tag



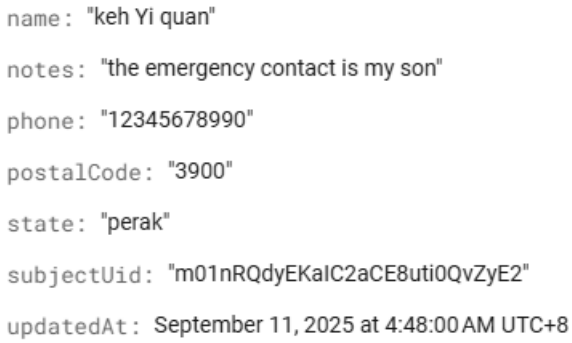


Figure 4.4.9.9 Stored Information Card

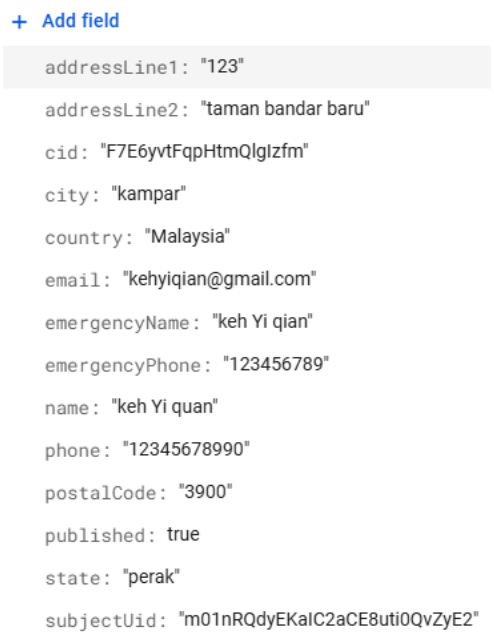


Figure 4.4.9.10 Stored Public Information Card that was Published on Web

4.4.10 Chat Module

The Chat module, as shown in Figure 4.4.10.1, has two chat rooms provided, one allows patients to chat with caregivers or vice versa, and the other allows both patients and caregivers to chat with OpenAI. When entering the Chat module, it will enter the chat room where users can chat with other users by default. The user can click the "Ask ChatGPT" to switch to the OpenAI chat room.

The Chat module supports text, voice, and media messages such as images, videos, audio, and files for conversation, as shown in Figure 4.4.10.2 and Figure 4.4.10.3. All of the messages will be stored in the Firebase as shown in Figure 4.4.10.4 and will retrieve the previous conversation every time you enter the chatroom again. At the top, there are three components, which are the search bar to search for the text that appeared in the conversation as shown in Figure 4.4.10.5, the three-dot vertical icon allows the user to access both sent media, links, and documents as shown in Figure 4.4.10.6, Figure 4.4.10.7 and Figure 4.4.10.8, or allows the sender clean the chat history on their owns side means that the receiver side still can see the previous conversation but not

sender side, the system prompts the confirmation before the deletion as shown in Figure 4.4.10.9. The icon beside that, which looks like a link, is just an icon that can let the user go to the active user management page to select the user that wants to activate, as shown in Figure 4.4.10.10. The text message is allowed to copy directly, and each message whatever is text or media message sent by the sender can be deleted with two methods as shown in Figure 4.4.10.11; one is "Delete for me" means that only delete the message in sender side's chat room while "Delete for everyone" means that delete the sent message by the sender in both sides. Therefore, the sender can only perform "Delete for me" on the message sent by the receiver, as shown in Figure 4.4.10.12.

The OpenAI chat room, as shown in Figure 4.4.10.13, allows the user to chat with OpenAI using text messages and can also attach an image as shown in Figure 4.4.10.14. The system will send the message to OpenAI via HTTP and wait for the reply from it. The conversation between the user and OpenAI will be stored in the local storage and allows the deletion of all conversations at a time by clicking the trash icon. The system will prompt confirmation before the deletion as shown in Figure 4.4.10.15.

Figure 4.4.10.1 Chat Room for Chat with Users

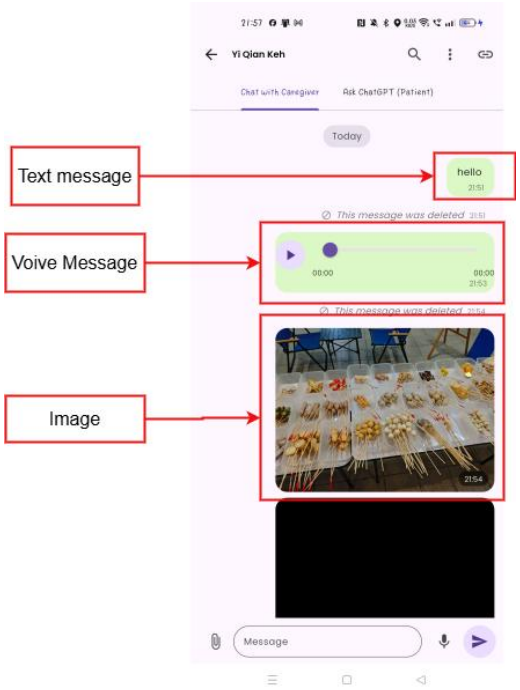


Figure 4.4.10.2 Type of Messages that are Allowed to be Sent

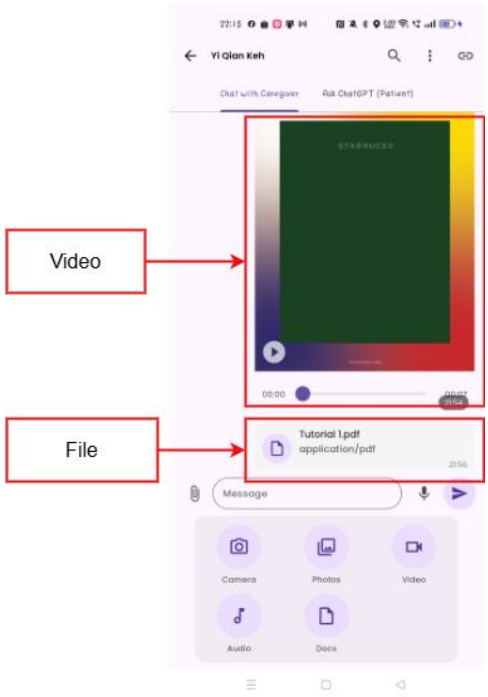


Figure 4.4.10.3 Type of Messages that are Allowed to be Sent

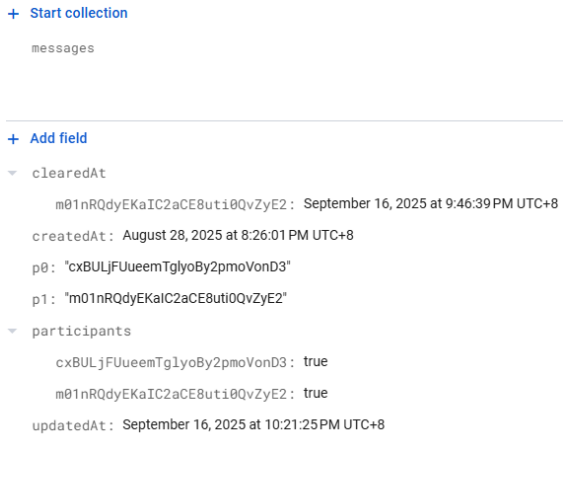


Figure 4.4.10.4 Stored Messages

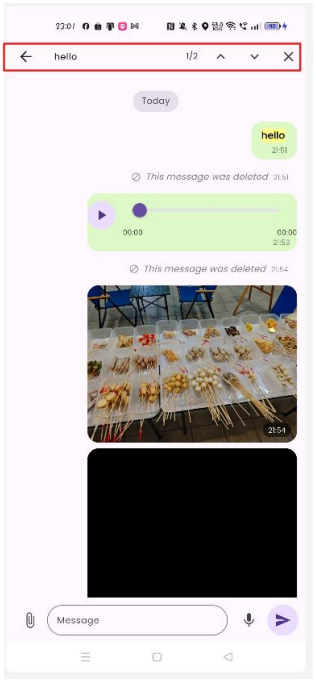


Figure 4.4.10.5 Search Bar

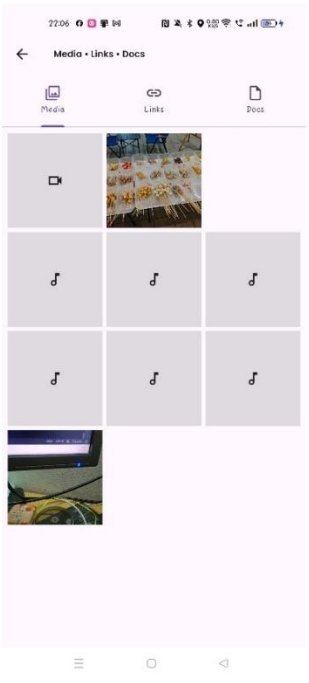


Figure 4.4.10.6 Quick Access for All Sent Media

Figure 4.4.10.7 Quick Access for All Sent Links

Figure 4.4.10.7 Quick Access for All Sent Documents

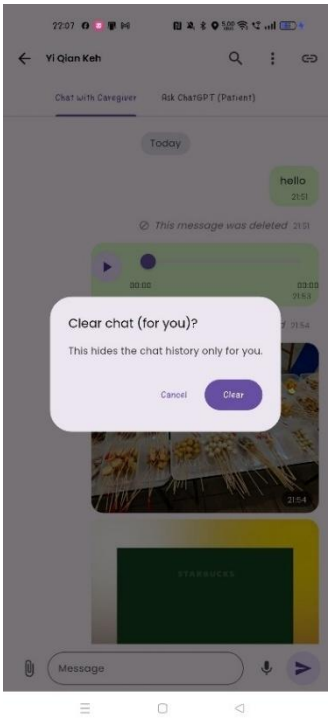


Figure 4.4.10.9 Clear for Me

Figure 4.4.10.10 Go to this Page

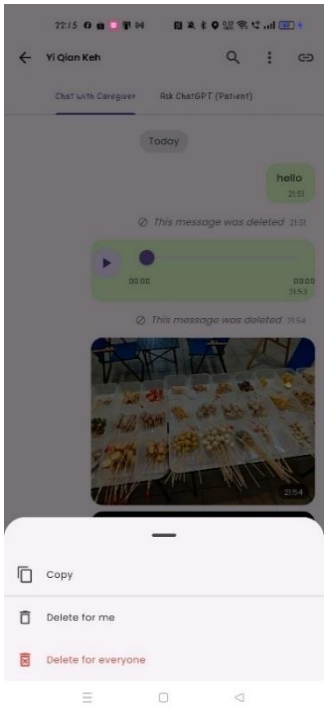


Figure 4.4.10.11 Delete Options to the Sender: Delete its Own Sent Message

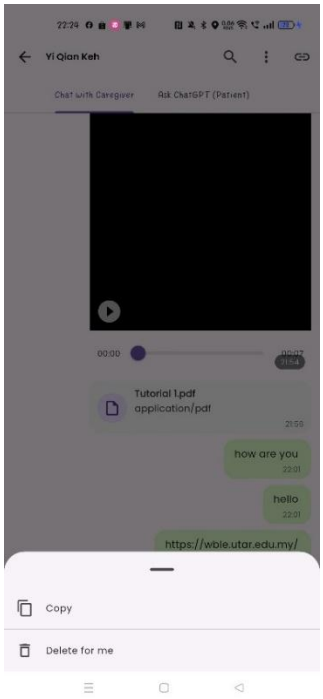


Figure 4.4.10.12 Delete Options to the Sender: Delete its Own Sent Message

Figure 4.4.10.13 Chat Room for OpenAI

Figure 4.4.10.14 Sent Message to OpenAI and Get Response

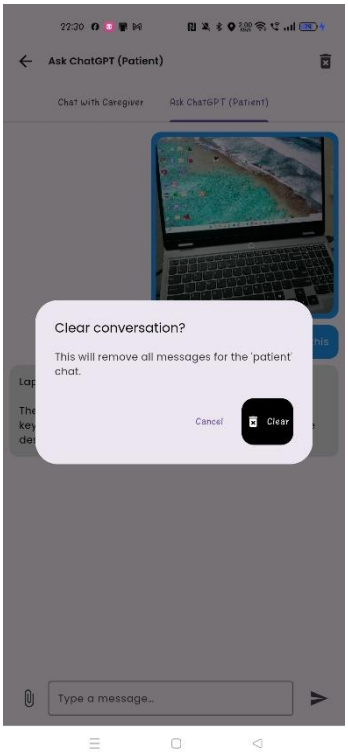


Figure 4.4.10.15 Delete Confirmation

4.4.11 Diary Module

The Diary module as shown in Figure 4.4.11.1, is designed to enable the patient to record their daily memories related to certain familiar people, interesting incidents, and valuable memories. First and foremost, the search bar is placed at the top, and the user can search for the diary entry by using the hashtags with the prefix # or participants with the prefix @ that are assigned when creating the diary, as shown in Figure 4.4.11.2. Below is the date searching tool, which the user can use to filter and find diary entries quickly by utilizing the date of the diary created, as shown in Figure 4.4.11.3. The diary's list is ordered in descending order, which means it will display the most recently created diary at the top.

This module allows the user to implement CRUD for the diary entry. The user can click the New Entry button to create a new diary; the user is required to fill in the header image, date, title, contents, and at least one hashtag, while filling in the mood, photos, and participants is optional. The participants are chosen from the enrolled persons in the Firebase to encourage the patient to record more valuable memories about the

enrolled persons in the app and guide the patient to link the diary to the person when writing the diary, as shown in Figure 4.4.11.4. After the diary form is filled in as shown in Figure 4.4.11.5 and Figure 4.4.11.6, the user can click the save button at the top right of the page to upload the metadata and information about the diary to the Firebase, as shown in Figure 4.4.11.7. Each diary entry is displayed like a card, and the read, update, and delete actions can be implemented by using the corresponding icons that are placed at the bottom right of the card. The read mode only allows the user to review the diary and cannot be edited, as shown in Figure 4.4.11.8. The user can edit the diary in edit mode, and the contents of the diary will be retrieved from Firebase. The user can then review the previously filled fields in the diary and edit them if needed, as shown in Figure 4.4.11.9. The system will prompt delete confirmation for the user if they implement the delete action, as shown in Figure 4.4.11.10.

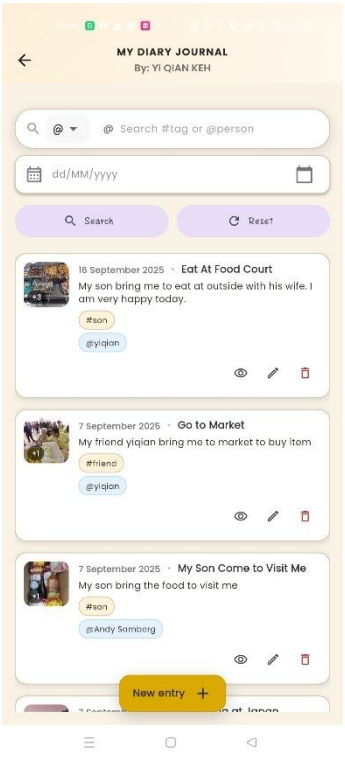


Figure 4.4.11.1 Diary Dashboard

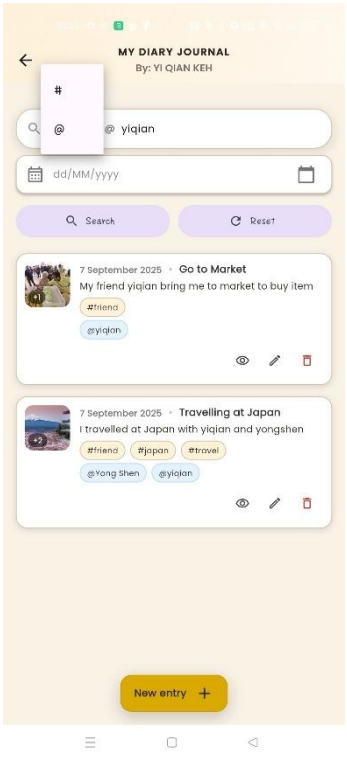


Figure 4.4.11.2 Search Diary

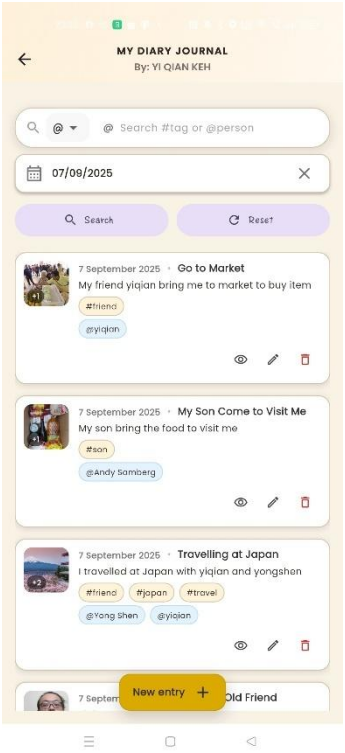


Figure 4.4.11.3 Search Date

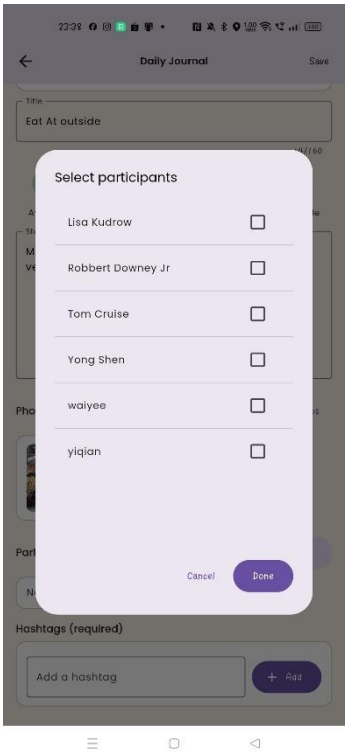


Figure 4.4.11.4 Select Enrolled Person

Figure 4.4.11.5 Create Diary

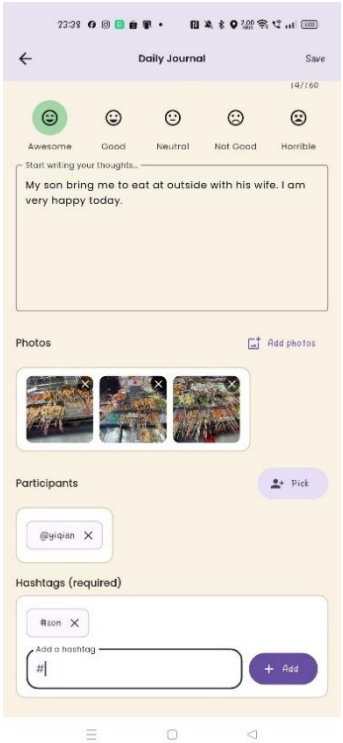


Figure 4.4.11.6 Create Diary

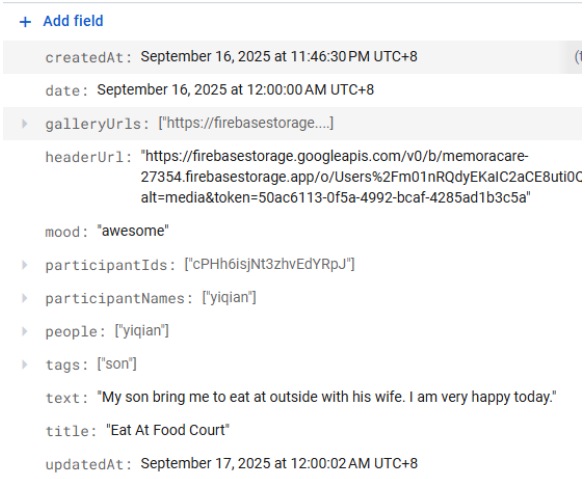


Figure 4.4.11.7 Stored Diary

Figure 4.4.11.8 Read Diary

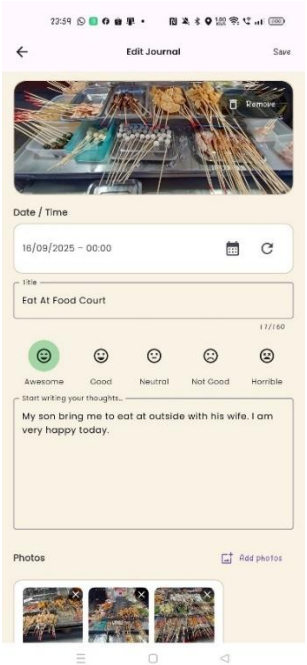


Figure 4.4.11.9 Update Diary

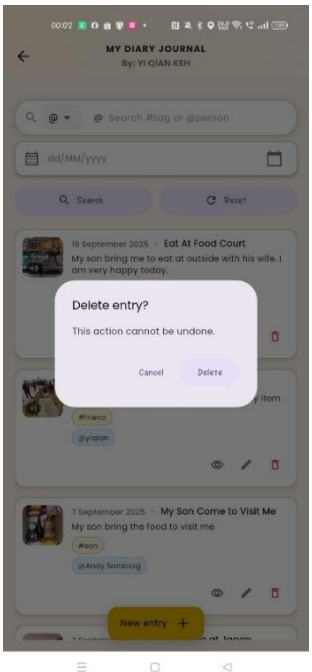


Figure 4.4.11.10 Delete Diary

4.4.12 Return Route Module

The Return Route module, as shown in Figure 4.4.12.1, displays the route between the patient and caregiver so that the patient can return via the route or ask the helping from other people when they wander. The route is suggested by the Directions API. The designated destination is based on the caregiver's side setting. If the caregiver's side GPS Tracking module is set to monitor the distance between the caregiver and patient, then the destination of the route is the caregiver's location, and if the caregiver is set to monitor the location between the patient and a fixed location, then the destination is the fixed location. At the bottom, it presents some information about the location, the distance from the patient's location to the destination, and the walking ETA. If the patient is out of the range of the safe zone set by the caregiver, the route will be displayed as shown in Figure 4.4.12.2 and send the alert notifications to the patient's side with the alarm sound as shown in Figure 4.4.12.3.

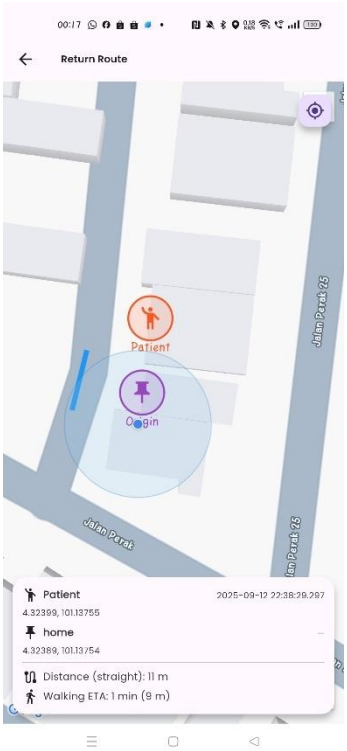


Figure 4.4.12.1 Return Route Screen

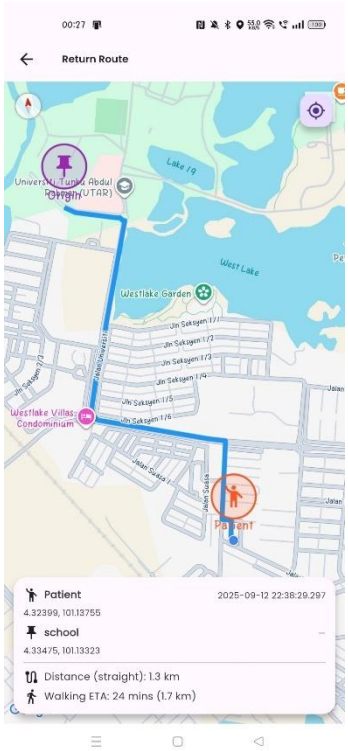


Figure 4.4.12.2 Route Suggestion

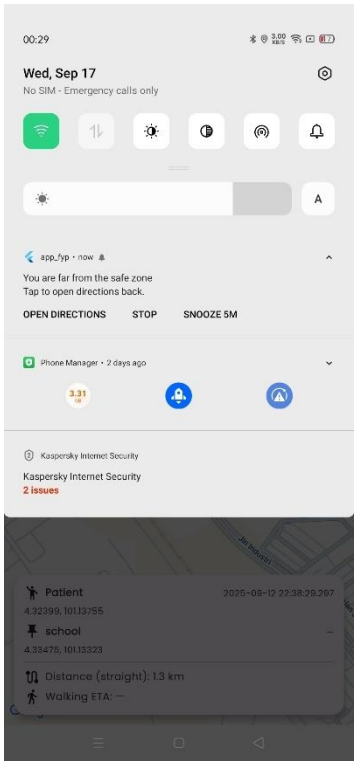


Figure 4.4.12.3 Alert Notifications

4.4.13 GPS Tracking Module

The GPS Tracking module enables the caregiver to monitor the patient's location and set a customized safe zone to prevent the patient from leaving too far away from the caregivers, as shown in Figure 4.4.13.1. There are two tracking ways for the caregiver: the first is to track the distance between the patient and the caregiver's own location, both of them locations is obtained by phone’s geolocator and then upload to the Firebase, and the second is to track the distance between the patient and a fixed location that is predefined by the caregiver, with a maximum of 3 locations that can be set, as shown in Figure 4.4.13.2. The fixed location can be added via searching places through the Places API or using current locations, as shown in Figure 4.4.13.3, and the data of the location will be stored in the Firebase, as shown in Figure 4.4.13.4.

It can also set the emergency contact to call the phone number directly. It will be triggered if the patient is out of the safe zone. The number of contact numbers can be set to a maximum of 3 phone numbers, as shown in Figure 4.4.13.5, and the default number is 123 if no phone number is set. These emergency contact numbers will be

stored in the Firebase as shown in Figure 4.4.13.6. When the patient is out of the safe zone, the countdown of calling the emergency contact with the alarm sound will be triggered, as shown in Figure 4.4.13.7, and call directly if the caregiver does not cancel within 10 seconds to avoid the caregiver from ignoring the alarm sound.

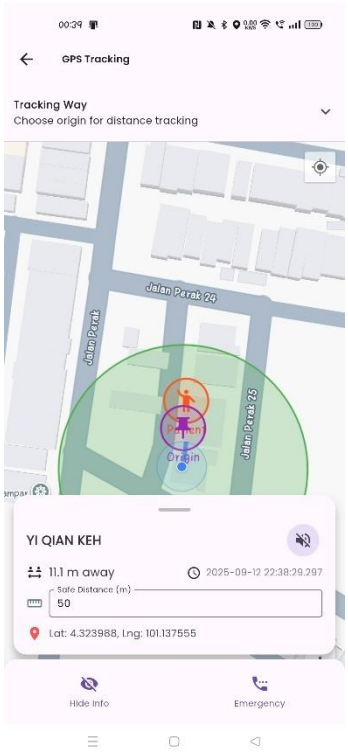


Figure 4.4.13.1 GPS Tracking Screen

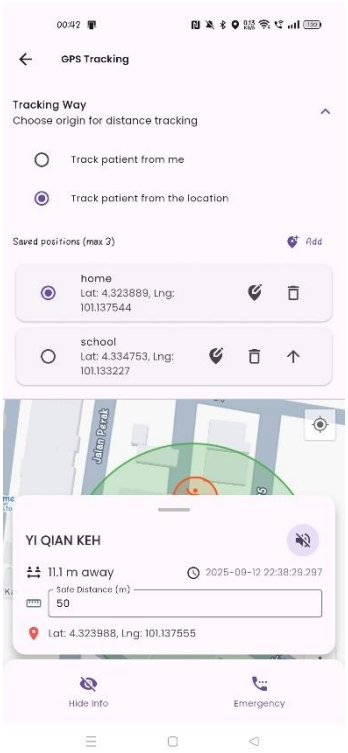


Figure 4.4.13.2 Tracking Ways

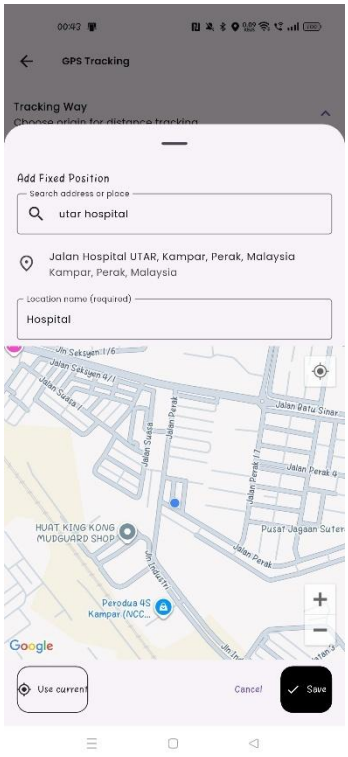


Figure 4.4.13.3 Set the Fixed Location

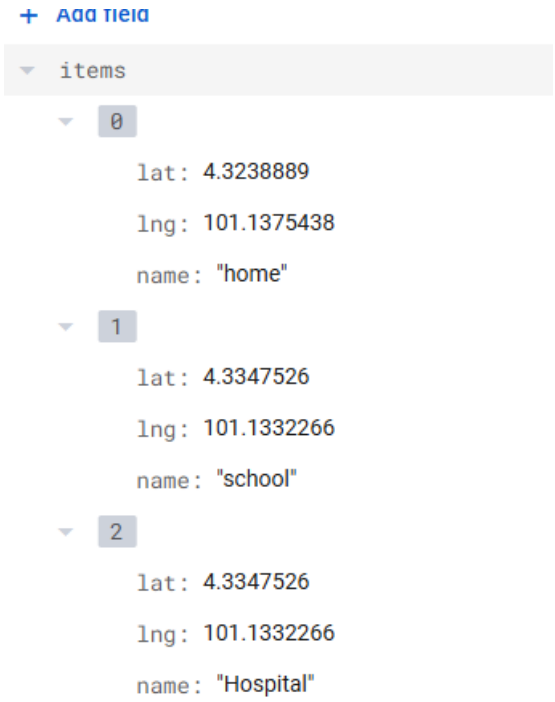


Figure 4.4.13.4 Stored Fixed Locations

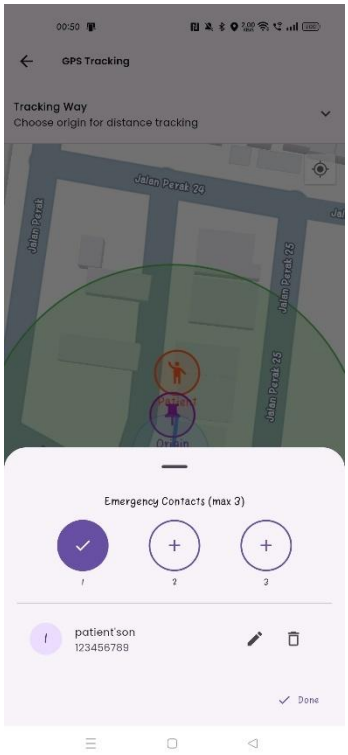


Figure 4.4.13.5 Set the Emergency Contact

Figure 4.4.13.6 Stored Emergency Contact

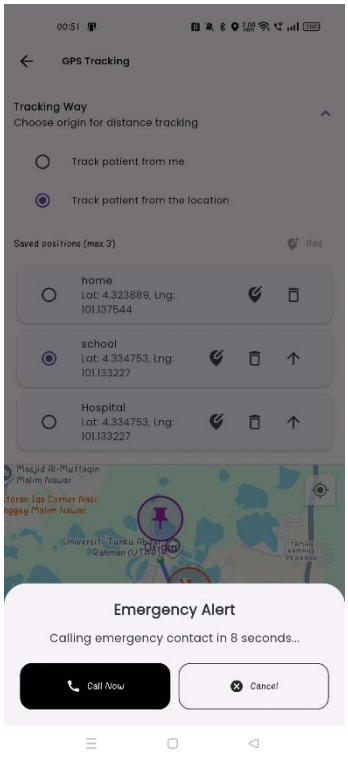


Figure 4.4.13.7 Triggering Emergency Contact's Countdown with Alarm

CHAPTER 5 System Evaluation and Discussion

5.1 Comparison between Three Models and Discussion

Table 5.1 Comparison between Three Models

Models	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy (%)
Custom CNN Model	100	61	61
Siamese Network	62.5	66.1	64.9
MobiFaceNet	98.6	81.7	80.5

Among the three evaluated models, MobileFaceNet is the most appropriate choice due to its strong balance between training performance and generalization capability. The custom CNN model, while achieving 100% training accuracy, suffers from severe overfitting with validation and testing accuracy dropping to 61%, indicating poor adaptability to unseen data. The Siamese network demonstrates consistent but relatively low performance across 62.5% of training accuracy, 66.1% of validation accuracy, and 64.9% of testing accuracy, meaning that it may suffer from underfitting and limited discriminative power. In contrast, MobileFaceNet achieves 98.6% of high training accuracy while maintaining significantly better 81.7% validation accuracy and 80.5% of testing accuracy, the performance is better than other models to prove its effectiveness and robustness. Furthermore, as a lightweight architecture specifically optimized for face recognition tasks, MobileFaceNet is well-suited for real-world deployment on resource-constrained devices. Thus, MobileFaceNet is the most reliable and practical model to adopt for this project.

5.2 System Testing and Performance Metrics

Authentication Module (Both can access)

Test Case Description	Input	Expected Result	Actual Result
Sign up with an invalid email format	Name: yiqian Email: test123 (Invalid) Phone number: 010123456789 Password: 123qweA@ (valid)	The system displays an invalid email format message	Achieved the expected result (Figure 5.2.1)
Sign up with an invalid password format	Name: yiqian Email: yiqiankeh@gmail.com Phone number: 010123456789 Password: 123 (invalid)	The system displays an invalid password format message	Achieved the expected result (Figure 5.2.2)
Sign up with a valid password and email format	Name: yiqian Email: yiqiankeh@gmail.com (valid) Phone number: 010123456789 Password: 123qweA (valid)	The system redirects to the role selection page and sends an email verification	Achieved the expected result (Figure 5.2.3)

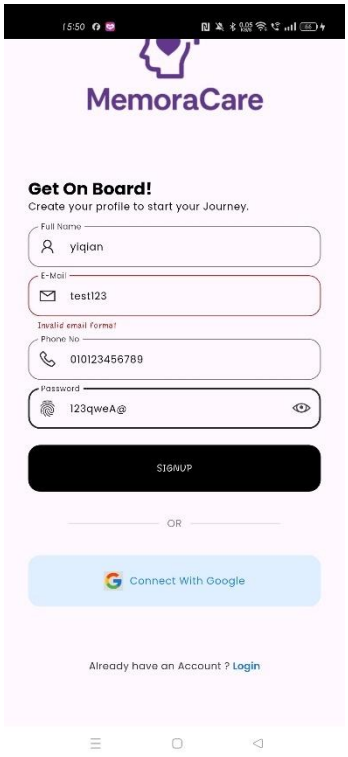


Figure 5.2.1 Displays an Invalid Email Format Message

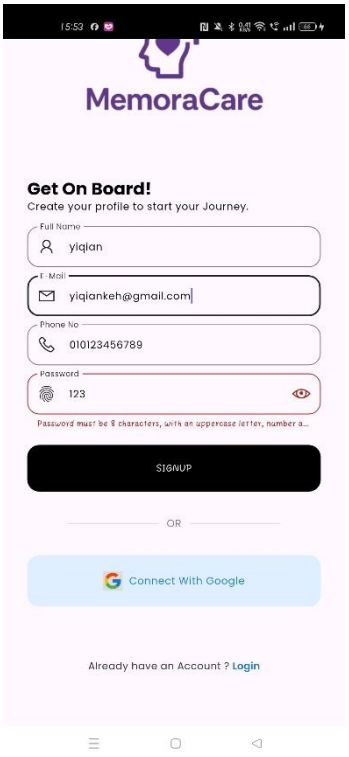


Figure 5.2.2 Displays an Invalid Password Format Message

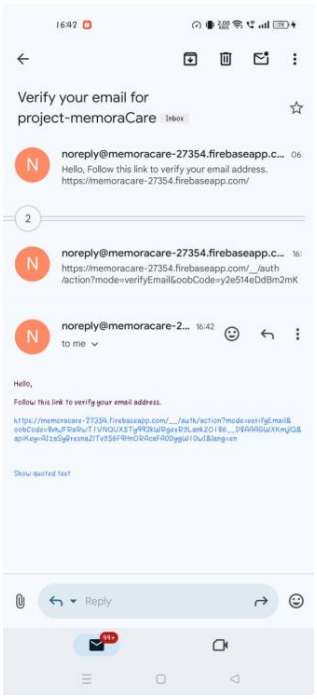


Figure 5.2.3 Received Verification Email

People Enrollment Module (Only Patients can access)

Test Case Description	Input	Expected Result	Actual Result
Add a new person	The patient fills in the name, captures 10 images, and adds memories.	The system should store the person's data in the Firebase	Same as expected result
Duplicate enrollment	The patient tries to enroll the same person again with the stored name	The system detects duplicates and prompts the user	Same as expected result (Figure 5.2.4)
Insufficient number of images	The patient captured 7 images only	The system is unable to upload and does not allow the user to click the upload button	Same as expected result (Figure 5.2.5)
Add the Images without a face	The patient captured 9 images with a face and 1 object image	The system displays an invalid message and prompts the user	Same as expected result (Figure 5.2.6)
Add the memory without a relationship	The patient lacks a choice of the relationship in the Add Memory Page	The system should display an alert message	Same as expected result (Figure 5.2.7)

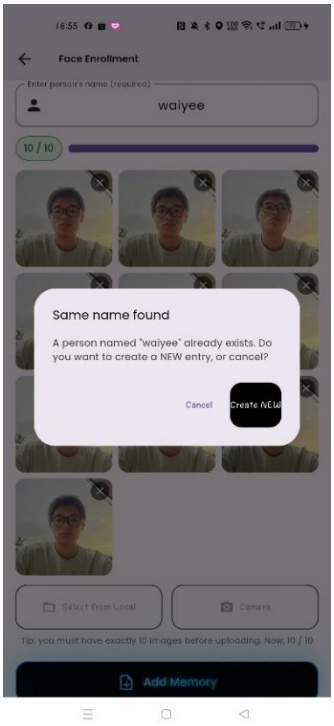


Figure 5.2.4 Duplicate Enrollment

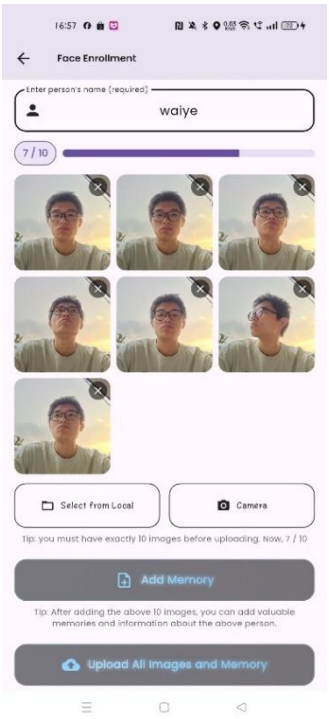


Figure 5.2.5 Insufficient number of images

Figure 5.2.6 Add the Images without a face

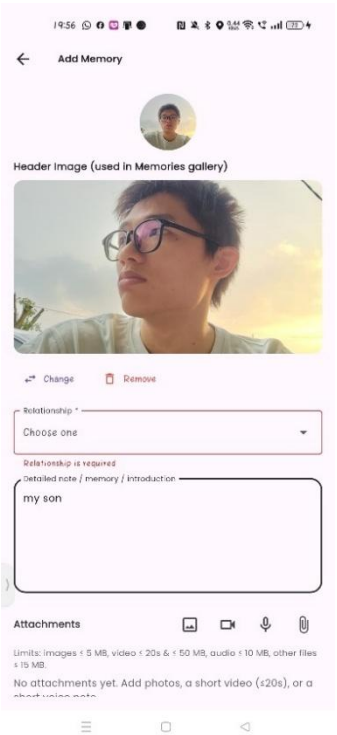


Figure 5.2.7 Add the memory without a relationship

Face Recognition Module (Both can access)

Test Case Description	Input	Expected Result	Actual Result
Real-time Recognition			
Attempts to recognize the enrolled person	Faces of the enrolled person	The system should label the name correctly and show confidence	Same as expected result (Figure 5.2.8)
Attempts to recognize the non-enrolled person	Faces of the non-enrolled person	The system should label the person as “Unknown”	Same as expected result (Figure 5.2.9)
Guide the user to place the face in the centre of the white bounding box	Faces of the person who is not in the white box	The system should display the arrow for guidance and prompt the user with the message	Same as expected result (Figure 5.2.10)
Guide the user to place the positive face in the centre of the white bounding box	Faces of the person who is in the white box, but the head is raised up	The system should prompt the user with the message to adjust their head	Same as expected result (Figure 5.2.11)
Attempts to recognize the enrolled person with a mask	Faces of the person who is in the white box, but wearing the mask	The system should label the name correctly and show confidence	The system mismatched the labelled name with the actual person (Figure 5.2.12)

Attempts to recognize the enrolled person with a hat	Faces of the person who is in the white box, but wearing the hat	The system should label the name correctly and show confidence	The system mismatched the labelled name with the actual person (Figure 5.2.13)
Fetch Images from local storage for recognition			
Attempts to recognize the enrolled person	Faces of the enrolled person	The system should label the name correctly and show confidence	Same as expected result (Figure 5.2.14)
Attempts to recognize the non-enrolled person	Faces of the non-enrolled person	The system should display an unknown message	Same as expected result (Figure 5.2.15)
Attempts to recognize the image without a face	An object image	The system should display the message “This is not a face.”	Same as expected result (Figure 5.2.16)

Test Case Description	Input	Expected Result	Actual Result
Patients			
The patients can click the show button and go to the page that can manage the memories, and	Faces of the recognized person and clicks the show more button	The system should redirect to the page that manages the memories and allows editing.	Same as expected result (Figure 5.2.17)

allows them to edit			
Caregivers			
The patients can click the show button and go to the page that can manage the memories, and not allow them to edit	Faces of the recognized person and clicks the show more button	The system should redirect to the page that manages the memories and not allow for editing to improve the data of the patients.	Same as expected result (Figure 5.2.18)

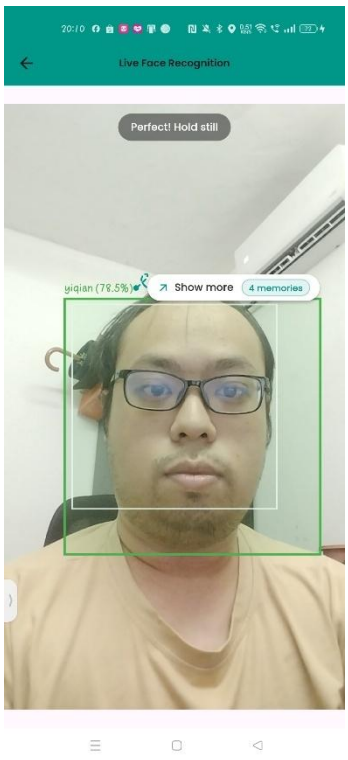


Figure 5.2.8 Recognize the enrolled person

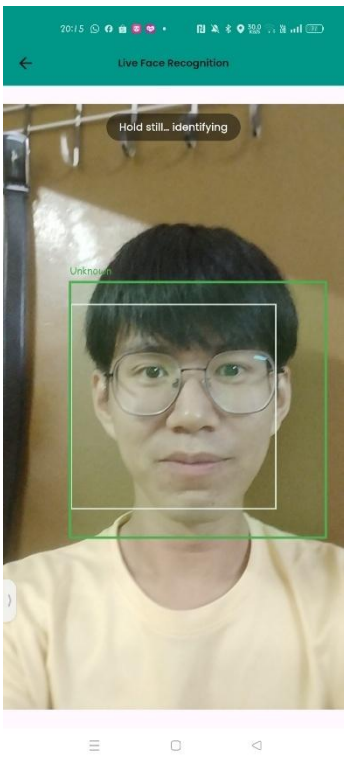


Figure 5.2.9 Unrecognized enrolled person

Figure 5.2.10 Guide User for Placing the Face in the White Box

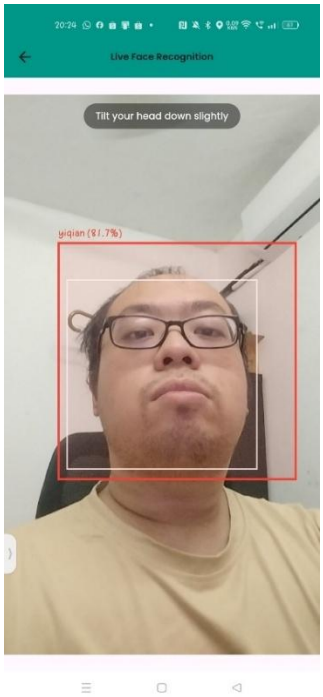


Figure 5.2.11 Guide User for Adjusting the Face in the White Box

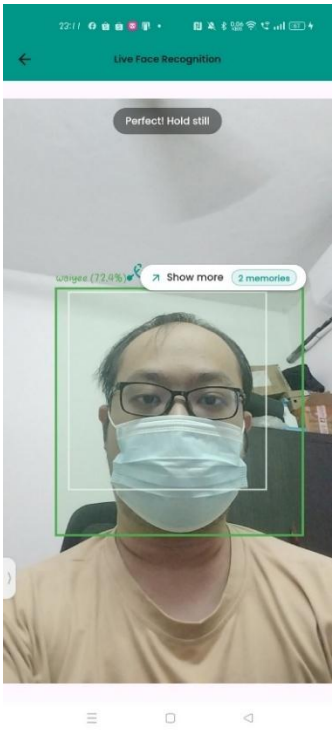


Figure 5.2.12 The Labelled Name and Actual Person Mismatched

Figure 5.2.13 The Labelled Name and Actual Person Mismatched

Figure 5.2.14 Recognize the Enrolled Face that was Fetched from Local Storage

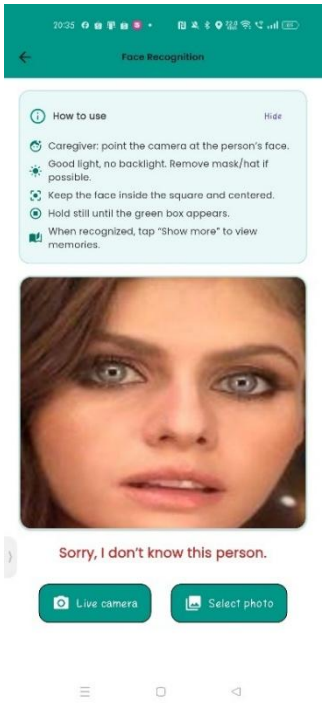


Figure 5.2.15 Unrecognize the Enrolled Face that was Fetched from Local Storage

Figure 5.2.16 Display "This is not a face." message

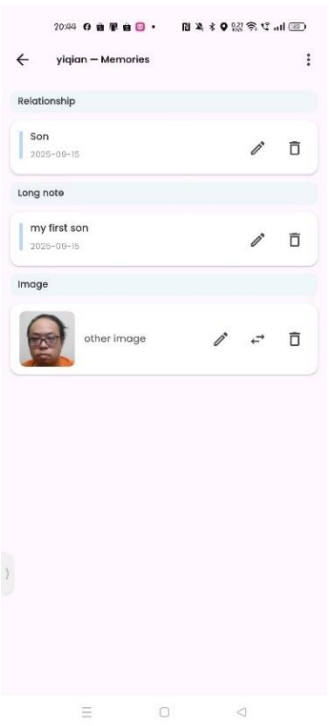


Figure 5.2.17 Editable Page on the Patient side

Figure 5.2.18 Not Editable Page on the Caregiver side

Gallery Module (Only Patients can access)

Test Case Description	Input	Expected Result	Actual Result
Opens the Gallery Module, and can view all the enrolled persons	The patient opens the Gallery module	The system displays all enrolled people with thumbnails	Same as expected result (Figure 5.2.19)
Delete the enrolled person	The patient clicks the delete button	The system displays all enrolled people with thumbnails, except for the deleted person	Same as expected result (Figure 5.2.20)

Edit the person's details by adding an image to the enrolled person for the memories	The patient attaches one more image	The system should add the image to the Firebase and display the newly attached image on the page	Same as expected result (Figure 5.2.21)

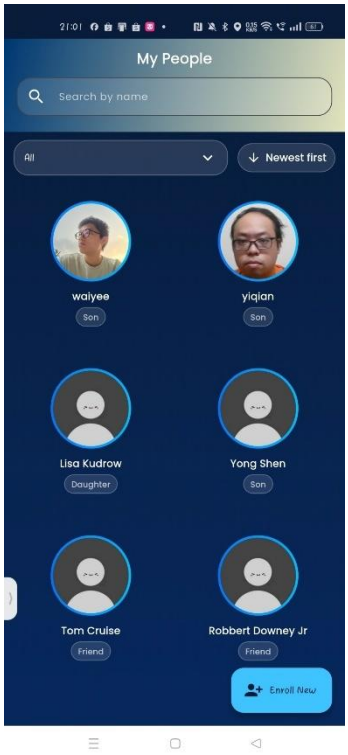


Figure 5.2.19 Display the enrolled person's list

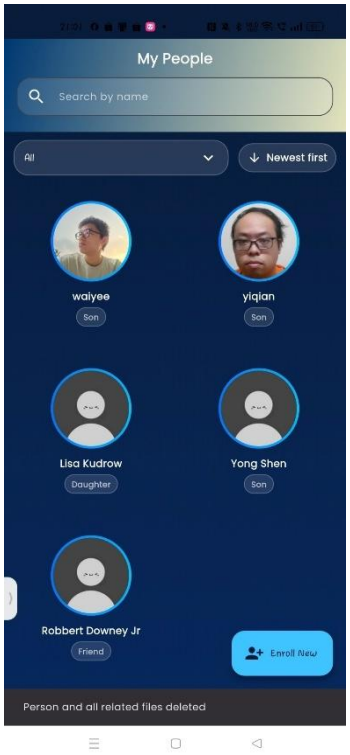


Figure 5.2.20 Page after Deletion

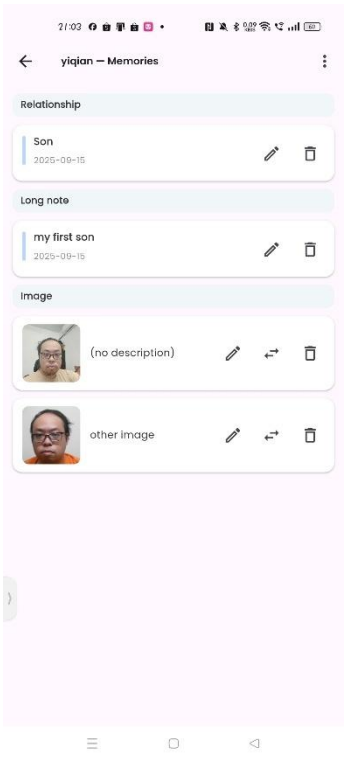


Figure 5.2.21 New Image is added

Chat Module (Both can access)

Test Case Description	Input	Expected Result	Actual Result
The patient sends a message to the caregiver	Patient types and sends a text message	Message delivered, and appears in the caregiver chat room instantly	Same as expected result (Figure 5.2.22)
The caregiver sends a reply to the patient	Caregiver types and sends a text message	Message delivered, and appears in the patient chat room instantly	Same as expected result (Figure 5.2.23)

The user can send the media files	User uploads an image	File uploaded to Firebase and image shown in chat	Same as expected result (Figure 5.2.24)
Test the AI assistant query	User types and sends a question	OpenAI is expected to answer it properly	Same as expected result (Figure 5.2.25)
Share the Images to the other social media	User clicks the share button to share the media	Present the shareable platform	Same as expected result (Figure 5.2.26)

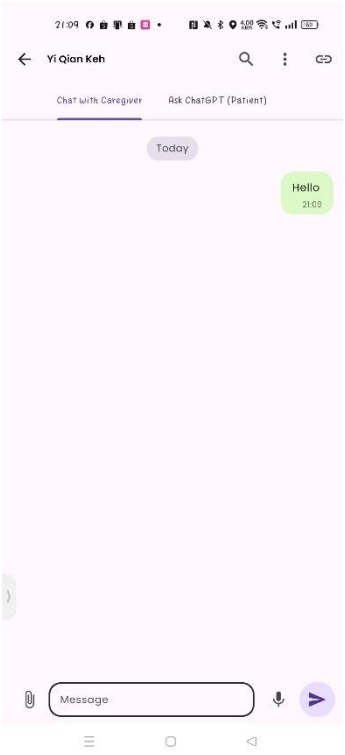


Figure 5.2.22 Patient Send the Message

Figure 5.2.23 Caregiver Replies to the Message

Figure 5.2.24 Patient Sends an Image

Figure 5.2.25 OpenAI Responses to Query

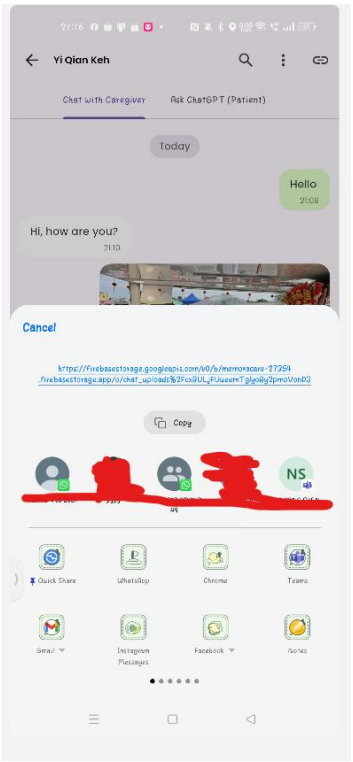


Figure 5.2.26 Patient Share the Image

Diary Module (Only Patients can access)

Test Case Description	Input	Expected Result	Actual Result
Create a diary	The patient inserts the valid inputs	The system should store the diary and display it on the diary dashboard	Same as expected result
Submit an empty diary	The patient clicks the save button directly	The system should prompt the user to fill in the required fields	Same as expected result (Figure 5.2.27)
Edit diary entries	The patient updates the data in the fields	The system should update the latest data in the diary	Same as expected result
Delete diary entries	The patient deletes the diary	The system should remove the related data in the Firebase and update the diary dashboard	Same as expected result
View diary entries	The patient views the diary dashboard	The system should display the stored diaries	Same as expected result

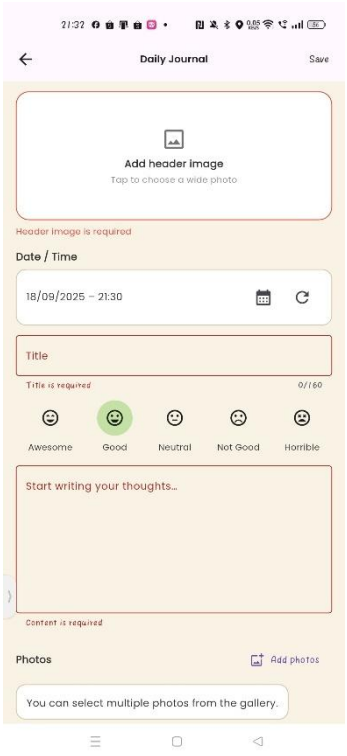


Figure 5.2.27 Submit Empty diary

Return Route Diary (Only Patients can access)

Test Case Description	Input	Expected Results	Actual Results
Generate return route	The patient maintains some distance from the caregiver	The system should generate the route and display it on the map	Same as expected result
Send notifications if out of the range of the safe zone	The patient is far away from the safe zone	The system should release the notification with an alert sound to the patient	Same as expected result

The updated of the return route	The caregiver changes the designated destination of the patient	The system should update the route and display it on the map	Same as expected result

Information Card Module (Both can access)

Test Case Description	Input	Expected Results	Actual Results
Create an information card	The user fills in the required valid input to the form	The system should upload the data to the Firebase and display the created card in the dashboard of the module	Same as expected result
Update an information card	The user updates the data in the form	The system should update the data to the Firebase and display the created card in the dashboard of the module with the latest data	Same as expected result
View an information card	The user views the data on the information card	The system should display all the stored data on the information card	Same as expected result
Delete an information card	The user deletes the entire information card	The system should remove the deleted information card from the dashboard and also	Same as expected result

		delete the data in the Firebase.	
View card via NFC	The bystander can access the publicly published card via the NFC	The web should allow access	Same as expected result
Write the NDEF URL into the NFC tag	The user clicks the write nfc tag button to scan a new null NFC tag to write the tag	The system should write the NDEF URL to the NFC Tag and display successful messages	Same as expected result
Write the NDEF URL without detecting the NFC tag	The user clicks the write nfc tag button and the system cannot detect the tag within 20 seconds	The system should display a failure message.	Same as expected result (Figure 5.2.28)

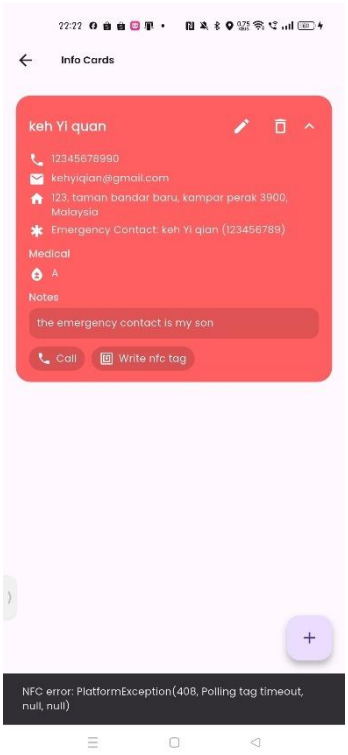


Figure 5.2.28 Failure message

GPS Tracking Module (Caregiver only can access)

Test Case Description	Input	Expected Result	Actual Result
Try the location tracking	The patient brings the phone and moves outdoors	The system should update the visual maker on the map	Same as expected result
Safe zone alarm	The patient is out of the range of the safe zone	The system should release the alarm and start the countdown for the emergency contact	Same as expected result
Trigger an emergency call	After the emergency contact's countdown ends, the phone will call the contact automatically.	The system should redirect to the call on the phone and call out automatically	Same as expected result

5.3 Project Challenge

One of the major challenges encountered during the development of this project was the limited timeframe available, within 28 weeks, for completing the system. The project scope included multiple advanced features such as real-time face recognition, people enrolment, gallery management, GPS tracking, return routes, diary management, chat communication, and information card integration. Each of these modules required not only design and implementation but also extensive testing, integration, and optimization to ensure they worked seamlessly together. The restricted time forced the project to be carefully conducted to prioritize tasks and adopt an iterative development approach, to focus first on core functionalities such as authentication, people enrollment, gallery, face recognition, and chat before integrating

supportive modules such as GPS tracking, return route, diary, and information card. Despite these constraints, the project successfully delivered a functional prototype that met the key objectives. However, the time limitation highlighted the importance of agile planning, strict milestone tracking, and scope management in balancing innovation with deadlines.

Furthermore, the significant challenge faced during this project was the need to self-learn Flutter because it was new to me and not in the syllabus at UTAR. Flutter is a powerful framework for cross-platform mobile development, but it comes with a steep learning curve, especially when integrating advanced features such as Firebase services, real-time camera processing, face recognition with TensorFlow Lite, GPS tracking, and chat modules. Since no prior experience existed, I had to spend additional time exploring documentation, online tutorials from Google and YouTube, community forums, and trial-and-error experiments to understand concepts such as state management, widget tree optimization, asynchronous programming, and platform channel integration. However, this challenge also became a learning opportunity because I gained hands-on expertise in Flutter, Firebase, and machine learning integration.

Another key limitation of this project is that the application currently does not support iOS devices, which focus only on Android. This challenge arose due to several reasons, such as plugin restrictions, hardware access limitations, time constraints, and a lack of iOS testing devices. While this reduces the capability of the cross-platform, the focus on Android was practical since it is the dominant mobile platform among the intended users.

Moreover, the challenge faced was the limited opportunity to conduct extensive user testing with real Alzheimer's patients and caregivers. Due to time, resource, and ethical constraints, the system was primarily tested by the developer and in small controlled scenarios rather than with a broad group of actual end users. This limitation meant that while functional and system testing validated technical correctness, the project could not fully capture usability concerns, accessibility feedback, or real-world interactions between patients and caregivers. Despite this, the prototype demonstrates the feasibility

of the system. Future iterations should include focusing on structured usability studies, practicing the caregiver and patient trials, and collecting the iterative feedback loops to ensure the app aligns closely with the needs of Alzheimer's patients and their caregivers.

Finally, the major technical challenge faced was that the face recognition system struggled to identify individuals wearing masks, hats, or other face coverings. Since the recognition model relies on visible facial landmarks such as eyes, nose, and mouth, occlusions will reduce accuracy significantly. During testing, the system performed well under normal conditions but often failed or misclassified when faces were partially covered. This limitation affects the reliability of recognition in uncontrolled environments. Although the guidance and instructions are listed in the app, we cannot ensure that both users will obey them.

5.4 Objectives Evaluation

Objectives	Work Done	Evaluation
To Develop a Mobile Application Designed to Assist Alzheimer's Patients and Their Caregivers	The mobile application has been successfully developed with multiple integrated features, including real-time face recognition, people enrollment, and a diary function to help patients recall their memories. Additional safety features such as GPS tracking, return route assistance, and digital information cards ensure the well-being of patients in case they wander or become lost. To support social interaction, a chat room feature is provided to strengthen communication between patients and caregivers. Through these functionalities, the application assists patients by enabling memory recall via face recognition, accessing stored memories in Firebase, and reviewing personal diary entries, while also guiding	Objective Done

	them back safely when needed. At the same time, caregivers benefit from tools to monitor patients' real-time locations, publish patient information cards online for emergency contact, and maintain ongoing communication, thereby enhancing both safety and emotional support for patients.	
To Implement Real-Time Facial Detection and Recognition to Recognize Familiar People of Alzheimer's Patients	The application achieves real-time face detection and recognition by integrating MobileFaceNet with the ML Kit Flutter package. Through the live camera feed, the system can accurately detect faces, extract embeddings, and recognize individuals by matching them against stored profiles in Firebase. Once a match is identified, the application retrieves and displays the corresponding person's stored memories from Firebase to create a seamless connection between recognition and memory recall. This real-time linkage not only personalizes the patient's experience but also reinforces memory stimulation and supports interaction between caregiver and patient in a meaningful way.	Objective Done
To Establish Communication between Alzheimer's Patients and Caregivers and Integrate with	The application includes a dedicated one-to-one chat room that serves as a private communication channel between patients and caregivers. Within this chat module, users can exchange text messages, share media files, and send voice messages to ensure flexible and accessible communication. To further enhance interaction, an OpenAI assistant is integrated into the chat system. This assistant can accompany patients in	Objective Done

AI-driven Assistance	conversation, provide answers to simple queries such as identifying objects, this particularly useful for those patients experiencing cognitive challenges, and offer caregivers practical guidance on how to better engage and interact with patients. By combining personal communication with intelligent assistance, the chat room not only strengthens relationships between patients and caregivers but also provides continuous support tailored to their needs.	

Chapter 6 Conclusion and Recommendations

6.1 Conclusion

In conclusion, this project successfully delivered a fully functional mobile application that integrates several critical features to support both Alzheimer's patients and caregivers. Alzheimer's often leads to memory loss, disorientation, and communication difficulties, placing significant strain on both patients and caregivers. The system seeks to address these challenges by incorporating face enrollment, real-time face detection, face recognition, communication modules, GPS tracking, route suggestion assistance, diary management, and information card generation. A key achievement of the project is the face recognition module, which was fine-tuned using MobileFaceNet and evaluated using datasets found on Kaggle. The model achieved 98.6% training accuracy, 81.7% validation accuracy, and 80.5% testing accuracy to demonstrate its capability to recognize familiar faces reliably under normal conditions. Real-time facial recognition, powered by MobileFaceNet and ML Kit, helps patients identify familiar individuals and recall associated memories, while the enrolment and diary modules enable them to capture and revisit meaningful experiences by storing those memories in the cloud instead of their brains, so that they can review the stored valuable memories even if they no longer remember.

Not only integrate the recognition feature, but it also proves effective in integrating real-time GPS tracking and safety alerts to allow caregivers to monitor patients accurately by using the geolocator and Google Map API. Additionally, the return route feature that displays the suggested route from the Directions API has been developed so that the patient can refer to the return way via the route. Moreover, the chat enables responsive 1 to 1 communication channels among the patients and caregivers to foster their communication and relationship while the AI chat assists by utilizing the OpenAI to realize the communication between AI and user. Besides that, the information card modules that are integrated with the function of writing the NDEF URL that consists of certain info from the information card into the NFC tag offer patients and caregivers a streamlined way to preserve and share important personal information.

By combining face recognition, memory support, safety features, and communication tools, the application aspires to simplify caregiving, strengthen patient autonomy, and mitigate the cognitive and emotional difficulties associated with Alzheimer's disease. Another main point of this project lies in its dual-target design, prioritizing both patients and caregivers equally, because this is the gap that existing applications often overlook. Moreover, the idea of integrating real-time face detection and recognition specifically to trigger memory recall among Alzheimer's patients introduces a new dimension to mobile health technology for AD care. This mobile application has been developed using Flutter, which implements the Dart programming language and Android as the operating system, also utilizing the native device sensors such as camera, geolocator, and NFC to capture images, obtain locations, and read and write NFC tags. Firebase was adopted as the backend service to provide seamless support for authentication, database management, and web hosting. It enables efficient control of user access, enforces role-based constraints, and ensures secure interaction between patients, caregivers, and other system components.

Overall, the project demonstrated a thoughtful, user-centred approach to combining emerging technologies into a practical application that supports and empowers Alzheimer's patients and caregivers in meaningful ways. The challenges and implementation issues will be addressed by putting more effort into the future and continuing to develop the new functionality that has already been designed.

6.2 Recommendations

Currently, the application is limited to Android devices, but to keep future iOS support feasible, it is recommended to extend compatibility to iOS devices to broaden accessibility for caregivers and patients who use Apple devices. This will require replacing or adapting plugins that have limited iOS support, refactoring the codebase, and setting up proper testing environments. For example, ensure that TensorFlow Lite models, GPS tracking, and NFC integration are built with packages that provide official iOS support, or set up an iOS testing environment, including macOS, Xcode, and physical iOS devices, to validate features like background GPS tracking, push notifications, and NFC reading.

Moreover, one of the key limitations was the inability to conduct extensive user testing with Alzheimer's patients and caregivers due to time and resource restrictions. In future iterations, it is highly recommended to establish structured usability studies with actual Alzheimer's patients and caregivers during early prototypes. For example, can collaborate with healthcare institutions, NGOs, or caregiver communities to obtain practical feedback. Such studies would allow the system to be evaluated under real-world conditions to reveal and discover critical insights into usability, accessibility, and reliability. Testing should not only validate technical performance but also focus on ease of use for elderly patients, who may struggle with cognitive challenges. Gathering feedback through iterative trials will enable continuous refinement of the interface to ensure that the app is not just functional but also practical, user-friendly, and trustworthy in everyday caregiving scenarios.

The face recognition module achieved strong accuracy under normal conditions (98.6% training, 81.7% validation, and 80.5% testing) but struggled when users wore masks, hats, or other coverings. To overcome this, future work should consider enhancing the dataset with occluded faces during training and fine-tuning the model to improve robustness. Advanced recognition techniques such as mask-invariant face recognition or multi-modal biometrics could further improve reliability.

REFERENCES

[1] "Alzheimer's Disease," in The Heart of Alzheimer's Care & Cure, n.d., [Online]. Available:https://www.alzsd.org/wp-content/uploads/2016/05/ALZHEIMERS-Understanding-the-disease.pdf

[2] J.C. Meyer, P. Harirari, and N. Schellack, Eds., "Overview of Alzheimer's disease and its management," in ResearchGate. Dec. 2015. [Online]. Available: https://www.researchgate.net/publication/316214529_Overview_of_Alzheimer's_disease_and_its_management

[3] "Dementia statistics." Alzheimer's Disease International. Accessed: Sep. 5, 2024. [Online]. Available: https://www.alzint.org/about/dementia-facts-figures/dementia-statistics/

[4] C. A. Des Roches, I. Balachandran, E. M. Ascenso, Y. Tripodis, and S. Kiran, Eds., "Effectiveness of an impairment-based individualized rehabilitation program using an iPad-based software platform," in Frontiers in Human Neuroscience. Jan. 2015. Available:https://doi.org/10.3389/fnhum.2014.01015.

[5] A. Saperstei. "Lumosity: A Professional Review." *One Mind PsyberGuide.* Accessed: Sep. 5, 2024. [Online]. Available: https://onemindpsyberguide.org/guide/expert-review/lumosity-expert-review/

[6] Lumosity. Accessed: Sep. 5, 2024. Available: https://www.lumosity.com/app/v4/dashboard

[7] K. Bainbridge and R. E. Mayer, Eds., "Shining the Light of Research on Lumosity," in Journal of Cognitive Enhancement. Oct. 2017. Available: https://doi.org/10.1007/s41465-017-0040-5.

[8] C. McGoldrick, S. Crawford, and J. J. Evans, Eds., "MindMate: A single case experimental design study of a reminder system for people with dementia," in Neuropsychological Rehabilitation. Aug. 2019. Available:https://doi.org/10.1080/09602011.2019.1653936.

[9] "12 Apps Designed for People Living with Alzheimer's Disease and Their Caregivers." Alzheimer's Caregivers Network. Accessed: Sep. 5, 2024. [Online]. Available: https://alzheimerscaregivers.org/2023/12/05/12-apps-designed-for-people-living-with-alzheimers-disease-and-their-caregivers/

[10] "Medisafe Platform, App Increase Medication Adherence." Medisafe. Accessed: Sep. 5, 2024. [Online]. Available: https://www.medisafe.com/education-resources/case-study-medisafe-platform-app-increase-medication-adherence/

[11] A. P. Leone. "Why Do People With Dementia Wander?." *www.hebrewseniorlife.org*. Accessed: Sep. 5, 2024. [Online]. Available: https://www.hebrewseniorlife.org/blog/why-do-people-dementia-wander#:~:text=About%2060%20percent%20of%20people

[12] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition". n.d. Available: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

Appendices

A.1 Poster