Face Recognition for Location Detection of Occupants in a Building

By

Cheong Kok Siong

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR INFORMATION SYSTEMS (HONOURS)
BUSINESS INFORMATION SYSTEMS
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2023

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: Face Recognition for Location Detection of Occupants in a Building

Academic Session: JUNE 2023

I , CHEONG KOK SIONG declare that I allow this Final Year Project Report to be kept in Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

(Author's signature)

(Supervisor's signature)

Address:

5, Jalan Bahtera 1

Taman Bahtera, 86810

Mersing, Johor

Su Lee Seng

Supervisor's name

Date: 13 September 2023

Date: 13 September 2023

FACULTY OF INFORMATION AND COMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 13 September 2023

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that CHEONG KOK SIONG(ID No: *19ACB06441*) has completed this final year project entitled "Face Recognition System for Location Detection of Occupants in a Building" under the supervision of Mr. SU LEE SENG (Supervisor) from the Department of Digital Economy Technology, Faculty of Information and Communication Technology .

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

_____

CHEONG KOK SIONG

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Face Recognition System for Location Detection of Occupants in a Building**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : _Cheong Kok Siong_

Date : _____13_September 2023_____

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Mr Su Lee Seng who has given me this bright opportunity to engage in Face Recognition for Location Detection of Occupants in a Building. It is my first step to establish a career in Artificial Intelligent and Computer Vision. Mr Su also the person who provide simulation advice and motivation, as well as for helping me organise my project. A million thanks to you.

Also, I would like to express my gratitude to my parents and friends especially Tia-Kaztenie Giam, Tan Su Hua, Ng Suet Eng and Lim Wei for their ongoing encouragement, support and companionship. Without their help and encouragement, this project might not have been completed.

ABSTRACT

Face recognition technology has undergone remarkable advancements in recent times, largely owing to improvements in digital camera technology. This technology would benefit from being incorporated into current systems like building management systems to improve the quality of human life and strengthen security in varied contexts. The Face Recognition for Location Detection (FRLD) of Occupants in a Building system in the proposed project aims to fully use face recognition technology for pinpointing occupant location within a building. By utilising the strength of the current video network and an extensive database of registered residents, this modern technology will effectively identify people and locate their locations within the building. The system promises to revolutionise building management and improve security by seamlessly combining these components. The development of this project is based on existing infrastructure and equipment, and refers to the exploration and development of all similar systems in the market. The main purpose is to address the lack of effectiveness in tracking the location of all residents in the current process, inaccurate real-time data, and potential cybersecurity risks and privacy issues that may arise from the current process. In addition, the project will also address challenges related to user experience and facial recognition technology. Last but not least, the proposed system is expected to be implemented in various environments such as office buildings, hospitals, apartments, and universities to enhance existing building management and safety measures. Through this measure, we are expected to bring more efficient management and stronger.

In this report, we also provide a detailed introduction to methodology and software development lifecycle to clarify the main core of the software and explore into the process of subsequence to ensure smooth development and completion of the software. Although some challenges were encountered during the development process, leading to deviations in planning, but those have been replaced by other approaches and did not affect the final outcome.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CCTV	Close-circuit Television
FRLD	Face Recognition for Location Detection
IR4.0	Fourth Industrial Revolution
ICU	Intensive Care Units
RFID	Radio Frequency Identification
AI	Artificial Intelligent
LOS	Line of Sight
NLOS	Non-line of Sight
GPS	Global Positioning System
LFW	Labelled Face in Wild
CSV	Excel File Format
IDLE	integrated Development and Learning Environment
FDT	Face Detection Technology
CNN	Convolutional Neural Network
SSD	Single Shot Detectors
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
SSD	Single Shot Detector
FRT	Face Recognition Technology
ML	Machine Learning
DL	Deep Learning
DNA	Deoxyribonucleic Acid
RST	Rotation, Scaling, Translation
PCA	Principal Component Analysis
NHA	New Heuristic Algorithm
LBP	Local Binary Pattern
ADA	Adaptive Boost Algorithm
PIR	Passive Infrared
IT	Information Technology
IP	Internet Protocol

IoT	Internet of Things
SVM	Support Vector Machine
IoU	Intersection Over Union
DCNN	Deep convolutional Neural Network
MAC	Media Access Control
UI	User Interface
VMS	Video Management System
MMOD	Max-Margin
HOG	Histogram of Oriented Gradients

CHAPTER 1

Project Background

Technology innovation in the modern world has created a firm foundation for many sectors and has created a firm foundation for many sectors and has significantly improved human lives quality. For instance, the fierce rivalry in the marker for digital devices have prompted ongoing hardware and software updates that produce excellent user experiences. The majority of digital devices manufacturers have concentrated on developing camera hardware and software and the chips performance to improve the quality of vision capture and obliquely increase the camera's adaptability and the efficiency of system processing. Face recognition technology has been successfully developed and applied to several industries as a result.

The main goal of this project is to create a system that utilising existing algorithms and techniques based on Python programming language and OpenCV for Face Recognition for Location Detection of Occupants inside a Building. Face recognition is not only utilised for security considerations or identity verification based on biometric such as e-wallet login or banking system quick login. But it may also be used as safety measurement when used in closed-circuit television (CCTV) in a building. It can be used to keep an eye on people's action and spot any suspicious behaviour or potential safety issues. Another than that, it also serves as an efficient technological solution for enhancing building energy control systems. By integrating reinforcement learning technology into existing systems, the technology enables the automated reduction of unnecessary energy consumption within buildings through AI-based building management systems.

Introduction

The Fourth Industrial Revolution (IR4.0) has produces cutting-edge technology that can take place of regular tasks, make human daily life simpler, and offer convenience, allowing people to live better-quality lives. Face detection and recognition are one of the sub-technologies of object detection. There have been applied in a number of fields to improve the system performance including security system, access control, law enforcement, entertainment and also smart traffic light management. In year 2023, Malaysia had applied AI technology at traffic lights which including object detection, action detection and car plate detection to capture the illegal action of driver on the road as evidence to have better plan regulations and enforcement [1].

Humans perceive and interpret visual patterns, such as human characteristics and colour recognition, in distinct ways compared to digital computers. As humans utilize their eyes to capture visual data or any information that they saw, transform it into meaningful information, process it within their minds, and ultimately recognize and interpret it [2], but the result may been affected by external factor from the environment or human emotion. However, computers use algorithms to capture each pixel in an image or each frame from a video, detect human facial features, identify the person, and determine the coordinate system of the face in the image [3], which can formulate the processes and getting result accurately since computer is no emotion. Although humans can recognize faces easily compared to AI which requires complex algorithms and deep learning, AI able to distinguish and recognize them stably and accurately.

Face recognition relies on biometric technology that uses real-time photographs to discern individuals by cross-referencing them with data of a preexisting image captured earlier [4]. By comparing the captured photos with a database of previously recorded image, this advance technology allows for quick and accurate person identification in variety of applications, including security, access control and user authentication. It has been applied in various industries which include financing, healthcare, retail, entertainment, marketing and etc. Utilising face recognition technology to track attendance and improve security in the workplace and education institutions has signification traction in recent years. By enabling autonomous within the system, it replaces such manual tasks to improve the system performance and

efficiency. This is especially relevant when considering biometrics, which utilise unique bodily traits like fingerprints and facial features to verify a person's identity, ensuring reproducibility, ease of use, and accessibility [5]. For example, hospital staff often encounter difficulties in locating doctors, nurses and patients within the complex healthcare environment. This issue is further compounded in such restricted area like ICU rooms and operation theatres, where mobile devices are prohibited. However, through the implement face recognition, hospital can overcome this issue, as it provides a reliable and swift means to precisely identity and locate doctors, nurses and patient without relying on mobile or the devices. As a result, this technology has found application in diverse sector, encompassing functions such as monitoring, strengthening security precautions and streamlining management processes. In addition, its utilisation is particularly pronounced in environments like hospital and education institutes, where plays an important role in addressing such challenges that related to occupants' location, thus contributing to operational efficiency and overall effectiveness.

Face recognition technology has evolved into an important tool for ensuring security and enabling surveillance, notably excelling in tasks such as accurately locating the whereabouts of occupants within a building. By adeptly recognizing facial features from images or frames in videos captured by CCTV or cameras installed throughout a building, and this technology enables the identification of individuals' behaviours. Additionally, it makes system more easily to manage the occupants' movements carefully, which leads to accurate determination of their exact location within the building. Real-time information about occupants' locations is made available through this system, which enable to assist in building management and operations and improve overall security, safety and reduce the energy resources consuming. Last but not least, the implementation of FRLD in a hospital would be an effective solution to the problem of locating doctor, nurses and patients for improving the overall management of the hospital. However, it is important to balance the benefits of these systems with the need for individual privacy protection and to prevent misuse and abuse. Thus, Face Recognition for Location Detection (FRLD) of Occupants in a Building is importance and able to enhance human life quality such as safety, convincing level and ultimately improve patient outcomes.

1.1 Problem Statement and Motivation

Several industries have successfully applied the most recent face recognition technique to improve system performance. However, existing occupants' location detection system employing Bluetooth waves has given rise to an innovative idea for tracking building occupants by implement face recognition technique to overcome the shortcoming. The system will improve performance assurance, security, and overall management in various industries especially hospital by identifying and recording the personnel. However, the system has flaws, inconsistencies and ethical problems that may inhibit it from being universal and from producing useful applications.

I. Existing building management system lacks effectiveness in tracking the location of all occupants.

Most of the modern location-based services are relying on WiFi or Bluetooth signals. This shown those technique require each individual must always carry their mobile devices with them and have to connect to building's public WiFi network or turn on Bluetooth and maintain its active status in order to be tracked. However, some occupants may not consistently have their mobile devices on hand, which serve as the primary means of detecting their location for occupancy counting [6], which will lead to only for occupancy counting but hard to pinpoint the actual location of occupants and causing in terrible implications during emergencies and harming the safety of people. For instance, the system would not be able to locate an occupant if they intentional or unintentionally leave their phone behind. Additionally, mobile or digital gadgets are prohibited in several places including courtrooms, laboratories, operating theaters and intensive care units (ICU) in hospitals. This is particularly troublesome in emergency scenario where the location of individual is essential for rescue efforts, potentially endangering lives, and where finding medical personnel promptly results in needless delays in treating patients. For personal and privacy concerns, some residents might also protest to having the location detection application installed on their mobile devices. As a result, this may lead to new restrictions and undermine the system's efficiency. To ensure the safety and welfare of residents in emergency situations, location detecting

system must be implemented while keeping these restrictions in mind and account them.

II. Delayed and inaccuracy of real-time data

The current location detection system and building management system might not posses the required level of accuracy to provide precise positioning and location information regarding occupants within a building, particularly when navigating vast and complex structures or areas filled with numerous obstacles [7]. The current location detection system utilises WiFi access points and Bluetooth to track the location of mobile user time to time. However, these techniques could give rise to issues stemming from factors such as the technologies employed, and potentially causing delays in updating real-time data in cases of weak internet connections. These technologies also would be influenced by various factors, including the distance between sensors, the sensor's range and the environmental conditions such as obstacles or potential interference from electronic devices [8]. Another method for real time locating system which is radio frequency identification (RFID) which relies heavily on the environment and can be categorized into Line of Sight (LOS) and Non-Line of Sight (NLOS). In both situations, localization issues can be causes by various materials and LOS to NLOS transitions, so great accuracy is essential for real-time location tracking, especially for accident prevention [9]. Existing technique in building operation is very sensitive to environment such as the radio of electronic devices can easily affected or blocked sensor if there is high frequencies of digital devices or cluttered area which may leading to data inaccuracy. Another technology use for location detection is Global Positioning System (GPS), but it is unsuitable for indoor detecting occupants as it relies on network of satellites to calculate precise outdoor location. Thus, if numerous occupants are connected to the same network, it can only demonstrate that they are inside the building but cannot determine which floor they are on. Inaccurate real-time data can lead to reduce building efficiency, occupant misidentifications, false alarms causing wasteful evacuations and delay emergency response.

III. The existing organization procedures give rise to potential security risks and privacy concerns

Presently, the existing location detection system in the market are predominantly driven by RFID, GPS, satellite imaging and internet tracking, those are continuing to advance through ongoing technological innovations [10]. Any viruses may invade digital devices under the firewall or malware detection and steal personal privacy data through network connections, especially during the installation process of unauthenticated or unknow applications. The WiFi and Bluetooth signals used by the network connections for location detection have the potential to be intercepted by unauthorized devices, which could affect in security breaches that impact to all devices that connected to the same network [11]. Additionally, the risk of using public WiFi has potentially include man-in-the-middle attacks, malware distribution, WiFi snooping and sniffing and malicious hotspots due to unencrypted network [12]. The building's network may have holes that hacker can use to gain unauthorized access to company server, extracting sensitive information from their servers or devices, such as residents' detail, corporate activities and any privacy data that could be used for malicious purposes. Next, installing additional apps for WiFi or Bluetooth-based location tracking may encounter disagreement from occupants due to concerns about device privacy and how personal conflicting with work-related purpose, particularly when sensitive data is involved. This presents a challenge for organizations seeking to implement a location detection system of occupants in a building without compromising on privacy and security concerns of their occupants. Moreover, the physical registration at security station will also pose a risk of data leakage because everyone's information is written on the same registration form, making it possible for some people to record or capture the others personal information if there is a hidden camera. Some visitors may also provide fake personal information in the form and will unable to recognize the visitor's information and appearance due to lack of photos and other useful information. Next, those technologies and procedures for building management may not be reliable in some cases, so an alternative and safer solution should be developed overcome those shortcomings especially keep occupants' data in secure.

1.2 Project Objectives

The thesis intends to introduce modern face recognition algorithms and innovative soft biometric features for occupant location detection in buildings. Modern technology has increased living standards and reduced task-related human error. The suggested method utilizes machine learning and deep learning to improve location detection performance while integrating face detection and recognition into building and corporate management systems. The following are the planned result for fulfilling these goals.

- **Create a smart building environment using modern face recognition technology to reduce the interaction with occupants while streamlining data automation and integration for seamless management.**

  In pursuit of heightened building management efficiency, a sophisticated smart environment is being cultivated to minimize direct interaction to between building operations and occupants This is especially important in hospitals, where effective management is essential for delivering rapid and efficient patient care. The technique involves developing a system capable of discreetly recording the location of doctors, nurses and patients, thereby facilitating less intrusive data collecting through face recognition to identify the individual and sub-subsequent server-side uploads, which streamlines management procedures. Through the seamless integration of data within the hospital premises, the detection process is further enhanced by employing face recognition within the access control or CCTV systems. This comprehensive approach provides an extensive understanding of the occupant's movement withing the building, making it easier to quickly identify healthcare professionals and patients. In turn, this result in improved patient outcomes. Additionally, customized features can be created base of the precise requirements of building management such as search occupant function or filtering function. Those will increasing overall effectiveness and finally leading to the development of a sophisticated and streamlined building management system that optimizes resource allocation.

- **Utilizing modern face recognition technology to significantly enhance the speed, precision and comprehensiveness of real-time data capture.**

  Face recognition and detection have developed into crucial instruments for identifying people, and these techniques have spread to several industries. For example, by identifying the residents based on their distinctive face features, the integration of these technologies into building's location detection and occupants monitoring system will be improved for comprehensiveness of data. Those system related to building management able to gain a lot from this, including the ability to monitor and manage building condition in real-time. In addition, face recognition with specific function may significantly increase the accuracy of location detection and provide insightful data on occupancy pattern and trends. These perceptions can aid in improving building management, resulting in greater energy effectiveness and other advantages for businesses. The improved face recognition framework in system enhances interaction smoothness and recognition accuracy, including tracking capabilities for appearance changes. Face tracking is utilized in practical studies with robot to address concerns with body tracking mix-up and false detection [13]. Additionally, cloud computing or local databases enables the collection and processing of data in real-time, the storage of enormous volumes of data in a single location, and the ability to access that data from anywhere with an internet collection [14]. Throughout the proposed system, the location detection of occupants has been able to improve the precision and vastness of real-time data.

- **Utilise modern face recognition technology to efficiently reduce security concerns and streamline the data collection process**

  The vital value of cyber security is underlined by the need to protect the organization as a whole from potential harm, significant losses in terms of sensitive information, and financial consequences. Controlling the variety of data gathered is a reasonable strategy for reducing the potential hazards of data exposure. Basic occupant information like occupants' current location, clock-in / clock-out records are only couple of the types of data the system should concentrate on acquiring in order to achieve its intended purpose. Additionally,

it is essential to refrain from making occupants responsible for installing extra application to their devices or establishing connections to public WiFi networks. Occupants' resistance to tracking is lessened by this strategy, which also allays concerns about possible data breaches. Furthermore, the proposed system will also tackle the issues of counterfeit. Face recognition helps to create a more secure and controlled environment by decreasing the likelihood of unauthorized access [5], due to everyone face feature is unique. However, there are challenges with accuracy and mistake occurring. But there are many existing algorithms developed to improve face recognition performance which are Viola Jones, Local Binary Pattern, Neural Network-Based Face Detection, etc. [15]. Next, using Labelled Face in Wild (LFW) verification to predict the detected faces and LFW classification to determine who is who, the system will compare the existing record in server side once face recognition process is complete [3]. By seamlessly capturing face features and automating the population of personal detail, the system able to simplify the registration process for visitors, obviating the necessity for manual registration upon building entry, while simultaneously ensuring the confidentiality of visitor information. Consequently, face recognition will foster greater acceptance among occupants regarding the monitoring process, while simultaneously enhance their personal safety assurances.

1.3 Project Scope and Direction

The primary objective of this proposed system is developing an advanced Face Recognition for Location Detection (FRLD) system tailored for building occupancy management, thereby improving traditional building and corporation management practices. Face recognitions are seamlessly incorporated into this system to improve overall performance, which is based on concepts from surveillance, building management, and attendance management system. By assimilating the strengths and learning from the limitation of existing system, FRLD aim to engineer a novel management solution aligned with the specific demands of building management. FRLD is primarily focused on offering simple administration and precise real-time information to reduce potential dangers, creating a smart and safe environment such as a workplace or rest area around the globe. The principal actors engaged in the system comprise the security department, human resource division or individuals in higher positions within the corporation or building. However, access to this system is deliberately limited and not extended to all employees or residents to ensure the protection of privacy and sensitive data. The project outcome involves applying face recognition to location detection inside building management systems through verifying occupant authentication. The subsequent section outlines the project module and scopes.

Login Module:

i. Access to the system requires login credentials.
ii. Password Requirements for complexity.

View Building Record Module:

iii. System will list out all detected individual in table format.
iv. Filtering function is implemented to filter out the target date, status, and zone of entry.
v. Sorting is available for rearrange the sequence of displayed record based on selected features.
vi. Quick entry of occupants detected record is available once item in list is clicked.

View Occupants Module:

vii. Based on pre-defined registration in the system, the system is capable of identifying the inhabitants.

viii. The search tool enables the user to find information about an occupant by searching for the occupant's name or ID number.

ix. Blacklisted individual detection, highlight in the detected list.

x. System able to display all the detected record belong to targeted occupant.

Report Analysis Module:

xi. The system explores all the detected occupants record and visualize in graph.

xii. The system enables to export those records in excel file format (CSV) for further use.

xiii. The system will alert the building potential threats.

Face Recognition Module:

xiv. The system will be capable of identifying face traits from images and real-time video.

xv. The implemented algorithm is able to match recognised person in server side with previously recognised inhabitants from the prior procedure.

xvi. The process of face extraction and detection will be less impacted by facial expressions.

xvii. The system possesses the capability to simultaneously capture and recognize face features of multiple occupants.

1.4 Contributions

The proposed project proposal addresses the viability of biometric feature and efficient algorithm to improve the occupancy location detection in a building or corporation management system by detecting and recognising facial traits to perform identification.

Firstly, the application of Face Recognition for Location Detection (FRLD) holds relevance across industries, particularly in densely populated organizations like universities, corporate buildings, and hotels. Its implementation significantly elevates safety and security by effectively tracking the whereabouts of identified occupants.

Secondly, FRLD finds utility in diverse sectors such as automated door access or resources management, leveraging its capacity to capture facial features and authenticate identities for monitoring occupancy. Moreover, it brings enhancements to building and corporation management systems, making comprehensive utilization possible.

Thirdly, without human involvement or the use of any tangible objects like a key or smartphone, face recognition technology can reliably identify a person. These will boost convenience for residents since they no longer have to carry any form of identification and will lessen the likelihood that their identification card or smartphone will get lost or stolen.

Fourth, since it doesn't involve the transportation of any devices or personal information, the proposed system can assist solve privacy concerns that could arise from the usage of existing location detecting techniques like WiFi and Bluetooth signal.

Fifth, the proposed system has a strong feature that makes it possible to identify people who are on a blacklist. In order to draw the user's attention to them, the system will highlight any blacklisted individuals it finds in the building. This capability is very helpful in situations where the existence of someone on a blacklist could endanger security, safety, or other important issues.

1.5 Report Organization

This report will be organized in seven chapters which are introduction, literature review, system method or approach, system design, system implementation, system evaluation and discussion, and conclusion.

Chapter 1 is about the introduction, which include the project background, project introduction, problem statements, objectives, contribution are intended to give an overview of this project.

Chapter 2 – Literature review. It includes a discussion of certain relevant histories as well as information on similar systems and technologies that are already in use as well as face recognition algorithms.

Chapter 3 - System method or approach. It discusses about the system methodology for system developing process and strategy (Regular Prototyping model) and system design diagram.

Chapter 4 – System design. Describe the preliminary work of the proposed project. It will discuss about the system block diagram, system flow description, and system components interaction design.

Chapter 5 – System implementation. It includes the hardware and software setup, preliminary Integrated Development and Learning Environment (IDLE) setting and configuration, system operation with reference (Screenshot), implementation issues and challenges, and finally concluding remark.

Chapter 6 – System evaluation and discussion. Describe the system testing and performance evaluation which included survey about feedback or review after testing.

Chapter 7 – Conclusion which including the recommendation and future plan of proposed system.

CHAPTER 2
Literature Reviews

2.1 Review of technology

2.1.1 Face Detection Technology (FDT)

[2] a study report that was motivated by the use of AI-based computer technology to extract and recognise individuals within images or videos. By providing real-time people tracking and surveillance, the development of **face detection technology is enhancing numerous industries' achievement**, including security, entertainment, law enforcement, and personal safety.

Face detection has developed from fundamental computer vision approaches by utilising cutting-edge machine learning, artificial neural networks, and related algorithms. With the evolution of these technologies, face detection has gained popularity, benefiting from continuous performance improvements. It now serves as a **pivotal element** in a multitude of face tracking, analysis, and recognition applications, leading to enhanced precision on existing system.

In order to better reveal the boundaries of the face during the face detection process, it is necessary to use specific techniques that remove the background of the image. Using skin tone to identify faces in colour images can occasionally fail, especially when motion is involved, such as facial emotion and expression when capturing face features in real-time video.

The Viola-Jones framework has been developed and utilised in the process to improve the efficacy and efficiency of detecting faces from photos. This will improve the reliability of identifying faces in real-time video. It serves as a training model for identifying facial features by leveraging particular traits. Following that, it will be saved in the database for comparison with future images. In order to establish whether there is a face in the image being studied, Viola-Jones compares face traits at various phases. When a photo has completed all stages, it will be used as a face in further operations. Convolutional Neural Networks (CNN) and Single Shot Detectors (SSD), which are specifically made for processing pixel data, localising, and classifying objects in images, will also be used in deep learning to enhance the process.

Nevertheless, while face detection proves immensely valuable and yields numerous advantages in such technological contexts, it does carry certain **disadvantage**:

- Putative burden of handling extensive data volumes, given the reliance on robust data storage for machine learning.
- Potential susceptibility to detection error stemming from variations in appearance.
- Conceivable infringement upon privacy rights, as continuous tracking may raise concerns among individuals.

2.1.2 AI for Face Recognition Technology (FRT)

Face recognition technology may identify and trace a person's identity and personal information if the image of his or her face can be located and matched in the database. This is possible because humans today are very active on social media. According to Mc Frockman, artificial intelligence (AI) is intended to mimic human intellect by giving robots the capacity to learn and make judgements in a comparable way to humans. [16]. Artificial intelligence (AI) and machine learning (ML) serve as the key components of facial recognition technology (FRT). A computer system capable of performing human-like activities, such as visual perception and decision-making, is referred to as an artificial intelligence (AI) system. Its range includes organising and comprehending abstract ideas, successfully guiding circumstances with learned information. ML and Deep Learning (DL) stand out as the key AI techniques that provide FRT its impressive capabilities.

ML is a branch of AI that uses algorithms to spot patterns and analyse data with precise probabilities. It aids in understanding visual context but needs a lot of information to generate reliable results. However, the process of data training or learning patterns will be automated and completed without human involvement. Supervised ML, Unsupervised ML, and Semi-supervised ML are the three themes that machine learning is divided under [17].

- **Supervised ML** trains the algorithm by feeding it labelled data. The learning algorithm began by making predictions, and then it used back-propagation to find errors and correct them by comparing its output to the accurate results that were intended.

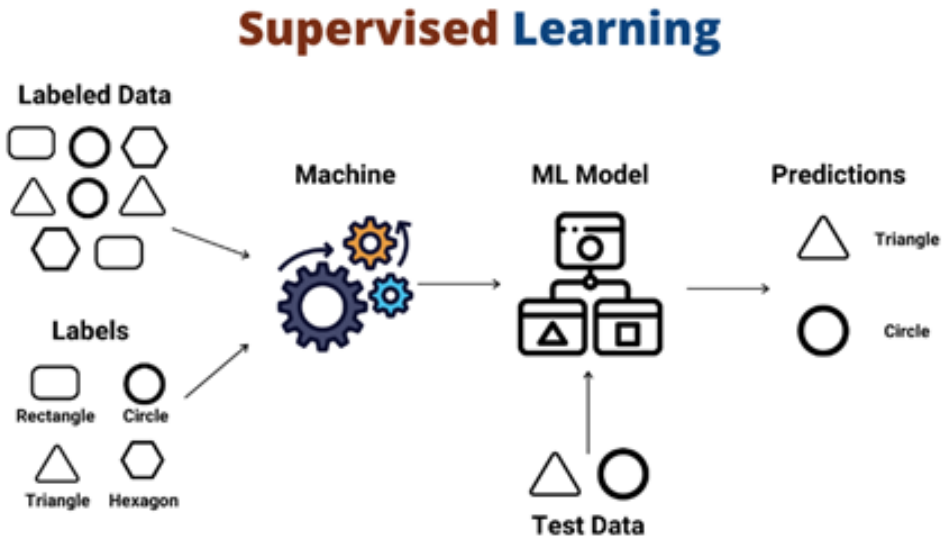


Figure 2.1.2.1 Supervised Learning [18]

- **Unsupervised ML** analyses and clusters unlabelled data to find hidden patterns or grouped data without involving a human. The three primary techniques used in unsupervised machine learning are K-means, dense-biased clustering, and hierarchical clustering.

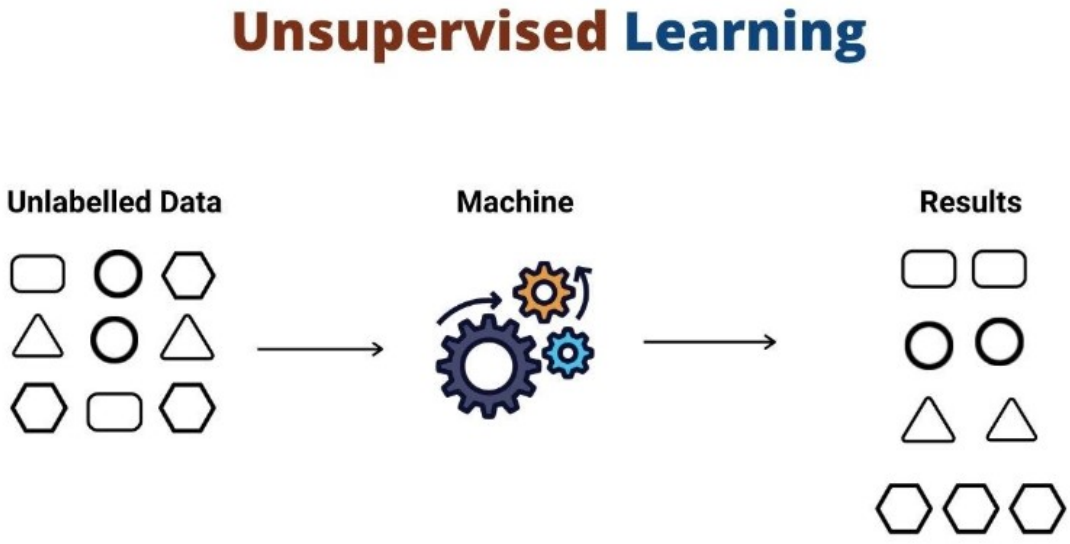


Figure 2.1.2.2 Unsupervised Learning [18]

- **Semi-supervised ML** also known as reinforcement ML which is the combination of supervised and unsupervised ML. It is a machine learning model that learns from feedback to help actions become better.

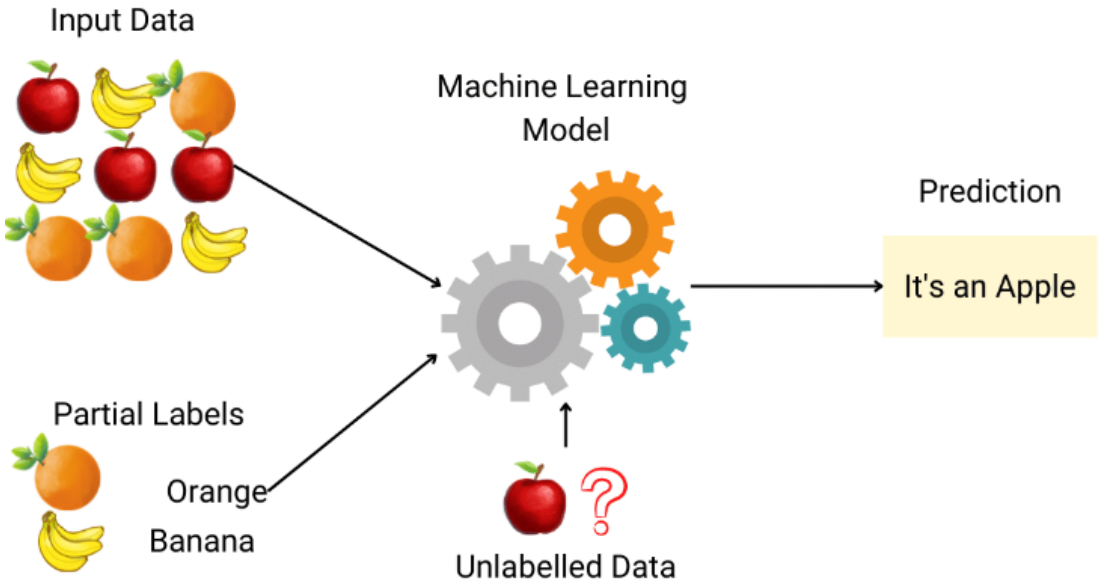


Figure 2.1.2.3 Semi-supervised Learning [18]

Regarding deep learning, it is a branch of machine learning (ML) that creates an architecture that resembles the human brain and is also referred to as multi neuron architecture. Without much human involvement, DL uses greater data to automatically extract the features. The term "deep" refers to the number of layers in the neural network, and it is scalable machine learning including Artificial Neural Network (ANN), Convolution Neural Network (CNN), Recurrent Neural Network (RNN):

- ANN is used for categorising images, such as training the input photo by extracting the facial traits.
- CNN uses a matrix-style "filter" to break up the image into little pixels, and the network then does a variety of calculations on those pixels to compare them to pixels that have a certain pattern. In order to attain accuracy, CNN needs a lot of training data and updates the filter value with each prediction using an error function.
- RNN are models that regulate the temporal pattern of videos and input the output of CNN into RNN for evaluation.

FRT is also very practical for many industries and has advanced in recent years thanks to AI backing and its ability to train up without human intervention. Face recognition is typically processed in three steps: face detection, picture conversion into data, which turns facial features into digital information, and match finding, which compares the digital information with already-existing database data. However, a separate form of FRT will have a different processing phase for more professional applications.

2.1.3 Biometric Face Detection and Face Recognition Algorithms

According to the International Journal for Modern Trends in Science and Technology, biometric systems recognise and authenticate a person's identity by employing specific physical qualities and particular characteristics. A biometric system offers a reliable way to confirm someone's identity. It frequently makes use of distinguishing traits like fingerprints, keystrokes, infrared thermograms, voice, signatures, DNA, and facial features.

The face characteristic is a very helpful biometric property for identifying an individual and is being integrated into more and more technologies and industries. To recognise and identify the face region or segment in an image or video, refer to the face detection technique. Cluttered backgrounds, lighting, occlusion, changes in rotation, variations in face pose and expression, scale, and translation are some of the aspects that can affect how well faces are detected and recognised. Therefore, when implementing face detection and identification technology, it is crucial to take these elements into account. The table below is showing each factors' detail:

Factor	Description
Clutter Background	External factors, including background and environmental conditions surrounding individuals in photos, exert a significant influence on the precision and effectiveness of face detection and recognition.
Occlusion	Occlusion can hinder the performance of face recognition algorithms by inadequately detecting face traits within the data.

Illumination	Face feature detection results may degrade due to inconsistencies in recalling face features, resulting from variation in lighting angles and shadow induced by differing intensities.
Face Pose	Postural variations arise from diverse angles and positions during image acquisition, leading to altered spatial relationships among face features. This distortion significantly impacts conventional appearance-based face recognition methods such as Fisherfaces and Eigenfaces.
Face Expression	Facial expression encompasses emotional shifts and convey feelings through changes in facial features, spatial relationships and alterations in facial contour.
Rotation, scaling and translation (RST)	RST poses challenges to face detection and recognition, demanding exhaustive searches across potential parameters during recognition.

Table 2.1.3.1 Factors Challenge Face Detection [19]

Even though face detection and identification technology face a number of barriers that will reduce the accuracy of the outcome, numerous algorithms have been created to enhance performance and efficiency. The descriptions of those algorithms are given below:

Algorithm	**Description**
Gabor Filter	A Gaussian function (2D Gabor filter) creates 40 images using edge detection at various angles and orientations. Resizing, pixel calibration, and border smoothing are all required for this. Each filtered image has maximum intensity points that are marked as fiducial points by the method. Based on determined distances, these points are then compared to known faces in the database.
Eigen Face	Eigenvectors, used in computer vision for face recognition, create a covariance matrix of face features from images. By comparing face representations using the basis set, recognition is achieved. This efficient technique prioritises

	encoding of illumination data over complex face features, placing speed over correctness.
Principal Component Analysis(PCA)	PCA is a robust statical technique that transforms a collection of potentially correlated variable into uncorrelated values. It is useful in both predictive modeling and exploratory data analysis and its sensitivity to relative scaling. PCA is notably the simplest eigenvector-based multivariate analysis. It clarifies data fluctuation by seeking for data's underlying structure. Additionally, PCA works well in situations where there are flattened images, projections or poor illumination
NHA	It serves as a standardization process, involving contrast starching or histogram stretching to widen the range of pixel intensity values. By guaranteeing a more consistent dynamic range across the dataset to reduce visual problems, this leads to higher image consistency and alignment with human perception. The inclusion of histogram equalization is critical, as it prevents the generation of unrealistic and undesired effects in photos with substantially higher color depth than the palette size.
Fisher Faces	The Representer's Hypothesis and Normal distribution are employed to tackle issues with invalid homoscedastic Normal classes, transforming them into piecewise mapping and addressing the challenge of determining the suitable number of mixtures per class. When there are enough training examples available to effectively learn nonlinear mappings, the piecewise method is advisable.
Viola Jones	The four steps it takes to process are choosing a Haar component, building a requisite picture, getting ready for ADA lifts, and cascading classifier. The process of recognition seeks for elements within rectangular regions of the image's pixels; to compute the value, subtract the total number of pixels inside white rectangles from the total

	number of pixels inside black rectangles. It swiftly provides results while continuously examining rectangle aspects.
Local Binary Pattern (LBP)	The neighbourhood structure of each pixel is represented by a decimal number that the initial LBP administrator assigns to each pixel. Its approach involves taking the central pixel out of the equation and comparing each pixel to its eight neighbours in a 3x3 neighbourhood. If the outcome is negative, the value will be replaced by 0, and if positive, by 1. In a clockwise motion, the acquired binary codes are concatenated.
Adaptive Boost Algorithm (AdaBoost)	By merging the outputs of various learners, Freund and Schapire's (2003) AdaBoost machine learning technique enhances performance. These 'weak learners' are weighted and modified iteratively, making it flexible and less prone to overfitting. Despite being noise-sensitive, it works even with weak learners who do only slightly better than at random. Particularly when using decision trees, AdaBoost is noted for its robust out-of-the-box performance.

Table 2.1.3.2 Algorithms for Face Detection and Recognition [19]

2.1.5 Summary of technology review

Face detection is employed to boost the efficiency and accuracy of the face recognition process. These technologies rely on AI, ML, and DL to detect, capture, recognise, and match facial traits in order to determine a person's identification. Finding faces in images, assessing facial features, comparing against recognised faces, and making predictions comprised the fundamental pipeline for face recognition. In addition, the security system, social media, and entertainment businesses have wisely employed them.

Face detection involves identifying facial features in an image or video using a variety of algorithms, including CNN, Viola Jones, Haar Cascade, and SSD. Once a face has been found, it will continue to use Eigenfaces, LBP, Fisher Face, and AdaBoost to process for recognition.

Overall, those algorithms have benefits and drawbacks that will influence the outcome of face recognition. The pros and drawbacks of such popular algorithms are shown in the table below:

Algorithm	Advantage	Disadvantage
Haar Cascade	Less complicated, real-time capable, quick to detect, and simple to apply.	Less accurate compared to DL-based technique, false-positive detection is higher
Viola Jones	The most admired technique for real-time face detection has great accuracy, a low false positive rate, and a high detection speed.	extended training period, few possible head postures, and unable to recognise faces in shadow.
LBP	High tolerance for recurrent fluctuations in illumination, efficient texturing feature, simple processing	Limited to binary and grey images, insufficient for non-monotonic illumination fluctuations, and lack of precision
AdaBoost	A simple and user-friendly programme with an adaptive	Overfitting and low margin are caused by

	algorithm doesn't require any prior knowledge.	weak classifiers, slow training progress.
PCA	Dimensional reduction, improved face recognition in noisy environments, and quick detection	A strong training set is necessary for sensors that are sensitive to occlusions, face expression, and lighting conditions.
Eigen Face	Able to perform in real-time, be cost-effective and efficient, and be able to capture facial features from various angles.	Faces with poses, lighting, and expressions that are sensitive to noise don't function well.
Fisher Face	High accuracy, compute quickly, low-dimensional feature	High precision, rapid computation, and low-dimensional feature

Table 2.1.4.1 Advantage and Disadvantage of Algorithms

2.2 Review of system

2.2.1 Occupancy detection systems for indoor environment

[20] proposed a study on an occupancy detection system, an automation system that offers smart services for indoor environments. One of the smart indoor environment services that has gained popularity recently and increased user experience by saving energy is occupancy detection. To enhance occupancy detection, optimise energy consumption, and increase user comfort, a variety of estimating techniques and communication technologies have been investigated and developed.

Occupancy detection systems can be applied in smart automation, which is a monitoring and control system that regulates energy consumption for any electronic equipment in the building, improve user comfort, and improve the efficiency of building management. This has important consequences for both residential and commercial environments. Smart buildings, security systems, energy-efficient infrastructure, healthcare monitoring, intelligent transportation systems, and building management and automation are just a few of its possible applications. Applications may need varying levels of occupancy data, including location, individual behaviours, occupant count, identity, and behaviour. Accurately measuring and detecting occupancy at a granular level presents considerable challenges due to factors like cost, deployability concerns related to sensors, and the complexity of required occupancy information.

Typically, monitoring, also referred to as sensing and prediction, and occupancy detection system, are two approaches that are classified. Prediction is based on occupant behaviour, form a prediction model by using statistical and ML techniques to control the equipment. Monitoring involves detecting the presence of people, relaying the collected information, and basing decisions and unit control on it. But there are also such restrictions and difficulties.

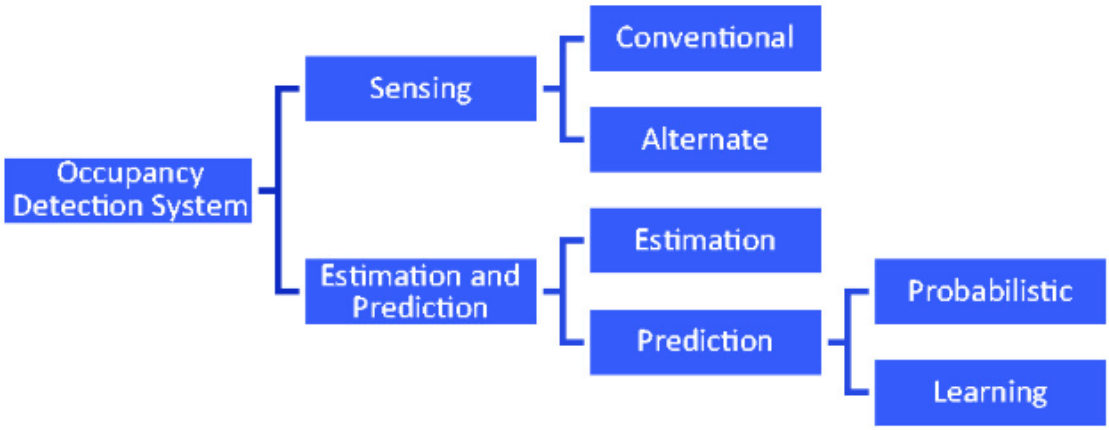


Figure 2.2.1.1 Occupancy Detection System [20]

Approach	Limitations / Challenges
Monitoring / Sensing	• The short lifespan of the sensor node • Requirement for high quantity of sensor devices lead to high cost • Complicated implementation of sensor units
Estimation and Prediction	• Necessity for a substantial pre-training dataset • Training the model is time-consuming

Table 2.2.1.1 Limitations and challenges of Occupancy Detection Approach

The crucial and fundamental standard for occupancy detection systems is monitoring or sensing. It is done by using a variety of occupant detection techniques, which are listed, described, and classified in the subsection below, along with their foundational methodologies:

- **Passive infrared (PIR) sensor** presents a method that can be used by all building management and lighting systems to detect indoor presence. Sensing the thermal radiation emitted by people in the monitored area allows for this.
- **Ultrasonic sensor** uses the transmission of ultrasonic signals and the analysis of the received signals to detect occupancy in a non-intrusive manner. It functions without requiring a direct line of sight and can also provide information about people's whereabouts.

- **Camara sensor** is employed to oversee building occupancy. It can offer diverse tiers of data like occupant whereabouts, identification, monitoring, headcount, and conduct. A wireless network of cameras is capable of collecting data on zone occupancy.
- **Wearable sensor** is extra devices require hanging all the time for the continuous monitoring of both activity levels and vital signs.
- **Tag-based sensor** able to effectively detect occupants and pinpoint their actual location and count the number of available occupants within a defined range by monitoring electromagnetic field fluctuations. These sensors will only be activated when more than two individuals within a defined range.
- **Smart device-based sensing** is implemented to track occupants by interacting with or identifying the installed application on their devices, providing insights into their presence, and detecting the presence. Given the widespread use of smartphones in the modern era, this strategy is especially valuable.
- **Network activity** refers to the collection of connections and online transactions conducted by building occupants using the current Information Technology (IT) facilities. In order to estimate occupancy's location, this involves tracking internet protocol (IP) traffic within the network.

There are several existing methods for performing occupancy detection for indoor environments, each of which has advantages and disadvantages that affect the outcome.

2.2.2 Building Occupancy Detection and Localization Using CCTV camera and Deep Learning

[21] is an academic publication that examines the improvements in management of buildings and user comfort brought about by the development of Internet of Things (IoT) technology. Although Closed-Circuit Television (CCTV) systems are widely used in monitoring occupants inside the building. However, it's important to emphasize that existing CCTV systems remain underutilized, presenting an opportunity to incorporate additional functionalities into these devices. For instance, by integrating technologies like OpenCV and face recognition, it becomes feasible to swiftly and

accurately process videos containing multiple individuals, enabling precise identification of occupant locations and identities.

First off, building management benefits from gathering occupancy data by using it to analyse it and gain a thorough understanding of a building's energy performance as well as pinpoint any potential risks or problems within the building. There are many different IoT sensors that can detect occupancy in buildings, but those sensors have their limitations. For instance, PIR sensors perform poorly in complex environments and have a low accuracy when detecting multiple objects, while RFID and WiFi sensors require extra equipment and are impacted by noise. To overcome the limitations, CCTV been suggested to replace those sensors since is widely available within buildings for safety and security purposes and to fully utilise the devices. Although they are still being developed, CCTV cameras for occupancy detection still face difficulties such as background noise and object obfuscation.

In the quest to augment the capabilities of CCTV for occupants' detection and localisation, the system has made use of machine learning. The performance has been improved by using both shallow learning algorithms like support vector machine (SVM) and deep learning (DL) methods.

- Machine learning is proficient in performing both classification and regression tasks.
- Support vector machine have limited capacity when it comes to accomplishing occupant localization.
- Deep learning amplifies learning capabilities through the utilization of neural networks structures that are both deeper and more intricate.

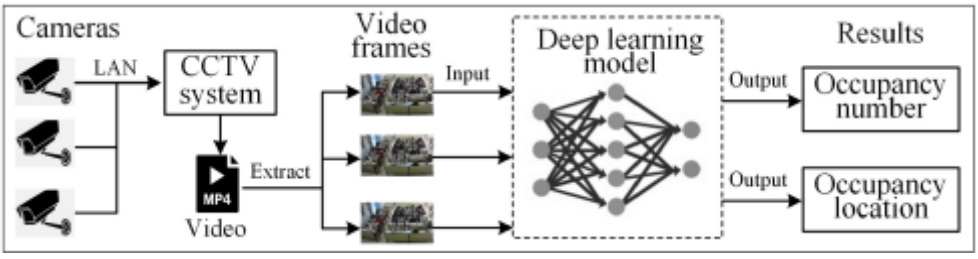


Figure 2.2.2.1 Layout of Occupancy Detection System Using CCTV [21]

The figure depicted above provides a concise overview of the occupancy detection system by using CCTV. It involves the collection of information from the CCTV

system, the extraction of video frames, their subsequent analysis and estimation within a deep learning model to produce the desired outcomes such as the number of occupants and where they are located.

Within the deep learning model, the system initiates the process by extracting image features through face detection algorithms before proceeding to three-stage decision methodology. In order to gradually improve Intersection Over Union (IoU) thresholds and produce high quality of hypotheses, this three-stage decision approach uses a serious of uniform detectors that each go through a sub-training operation.

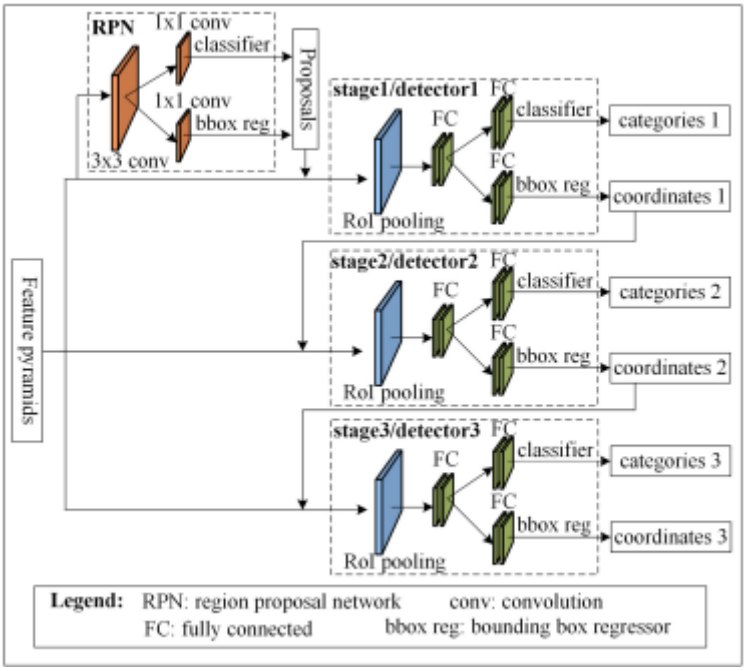


Figure 2.2.2.2 Three-stage detection [21]

A thorough evaluation of the various existing detection methods, including head detection, occupancy localization, occupancy presence, overhead detection and the suggested model was also done. The deep convolutional neural network (DCNN) and three-stage detection method were combined to create the model which displayed impressive performance results with precision rate up to 89.7% and a true positive rate of 89.2%. In addition, this model successfully expanded the occupant detection range through the whole building, make it flexible and superior to other approaches in terms of effectiveness.

2.2.3 FaceMe

FaceMe is an application developed by CyberLink which is a global frontrunner in the realm of face recognition and face attribute technologies. In the NIST Face Recognition Vendor Test, FaceMe is their incredibly accurate AI engines, holds a prominent position, securing recognition as one of the top performers, particularly in the VISA and WILD test categories.

FaceMe Security serves as a ready-to-use security and access control solution. It facilitates identity verification, attendance tracking and access control utilizing face recognition, along with live monitoring and instant notification. Furthermore, it has capability to trace an individual's action using facial image or physical characteristics. When integrated with prominent video management system (VMS) platforms, FaceMe Security streamlines and enhance intelligent security management.

FaceMe is introducing a comprehensive access control and AI smart monitoring solution that combine robust personnel access control, efficient record management and advanced face recognition. It seamlessly integrates with VMS, support various access of control protocols and it is compatible with both IP cameras and face recognition terminals. FaceMe is offering real-time face recognition, blocklist detection with prompt which includes blacklist and greylist, and precise video searches. Instant alerts via various channels are also available in its management interface, which sync with VMS databases and security cameras. In addition, FaceMe support multi-person identification with high accuracy of face recognition result even masks are used. People Tracker add-on which will speed up the video surveillance investigation for tracking people based on the physical features across multiple locations.

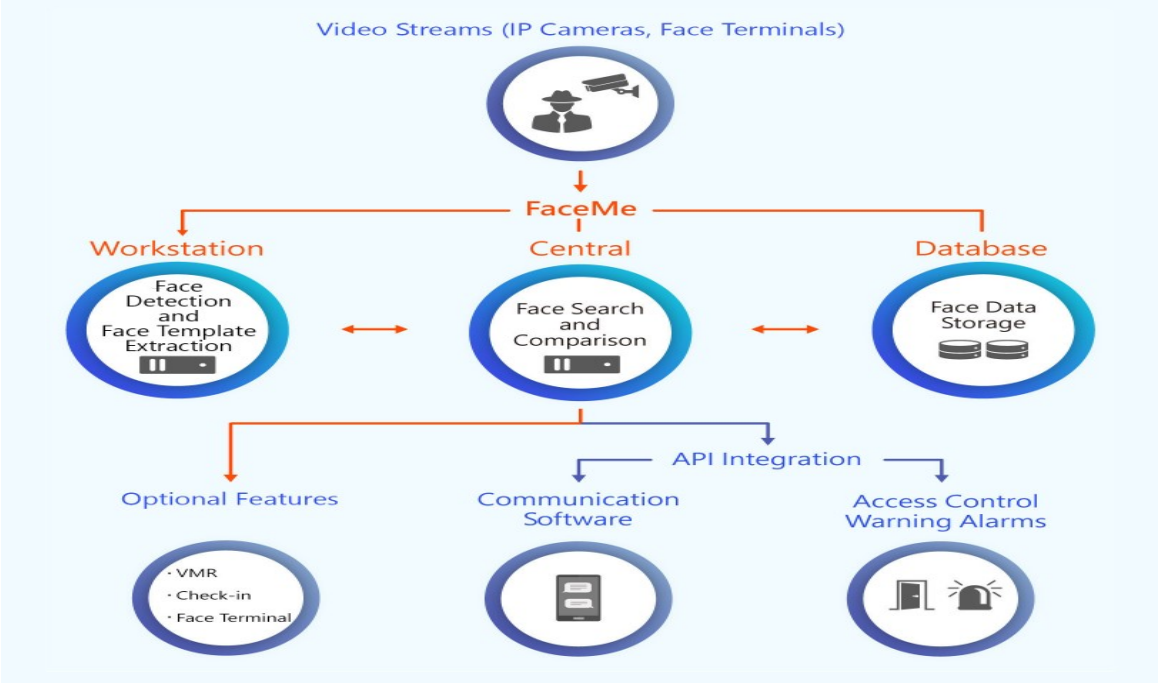


Figure 2.2.3 FaceMe system structure [22]

Based on figure above, FaceMe Security's decentralized architecture ensure flexibility and high availability. It evaluates the characteristics value, compares them at the second tiers which is main database, and control access. When specific conditions, such as blocklists occur, Central notifies those responsible or VMS. In second tier, which encompasses workstations, the Central and the database, each station necessitates dedicated server support to ensure seamless operation. For example, the Central demands a minimum configuration of 16GB of RAM, i5 CPU, Microsoft SQL Server, a 2GB HDD, and the Window 11 OS to effectively facilitate the workflow. Similarly, workstations require 16GM of RAM but with an i7 CPU to robustly support the workflow process.

FaceMe offers an additional API which is People Tracker designed for AI Vision-based tracking and recognition using physical characteristics. It includes face recognition, multi-camera tracking, person attribute recognition, Person Re-ID, and person identification. Once a person has been identified and tracked, it able to reduce the search area by selecting the time and relevant camera. It able to narrow the search by entering specific face features, after which you can watch the video from all the available cameras. Finally, examine the videos chronologically to trace the individual's movements.

2.2.4 Summary of system review

There are several technologies and methods support and improve the performance of location detection systems, especially for indoor environment according to review of those journal articles. The process of detecting occupancy is completed by location detection system since it is using the data to carry out additional operations. It is highly advised to use CCTV to perform location detection as it able to provide multiple level of information which including the identity, location tracking, headcount, and behavior of the occupants. Even smart device-based, tag-based, and network activity are also capable of providing that information but are also carrying extra devices or installing additional applications are burdens that the occupants must bear. However, CCTV is widely used in many building for safety and security purposes, it is one of the best options for face recognition for occupant location detection within a building structure.

	PIR	Ultrasonic	Wearable	Tag-based	Smart device-base	Network Activity	Proposed System
1. Require additional application on personal devices (intrusiveness)	X	X	X	X	✓	✓	X
2. Existing infrastructure	X	X	X	X	✓	✓	✓
3. Capable of accurately counting the number of occupant present	X	✓	✓	✓	✓	✓	✓

4. Able to track occupant's identity	X	X	✓	✓	✓	✓	✓
5. Suitable for huge environment	X	X	X	X	✓	✓	✓
6. Cost	Low	Low	Medium	Low	High	Low	Low
7. Accuracy	Low	Low	High	High	High	Low	High

Table 2.2.3.1 System's comparison result

Furthermore, by comparing the proposed system with "Building Occupancy Detection and Localization using CCTV camera and Deep Learning", the concept of those systems is similar, but the proposed system places a distinct emphasis on ease of management and flexibility. The proposed system's goal is to offer a straightforward and adaptable computer application platform, elevating both building management and security levels. [21] An improved understanding of the solution is made possible by the clearly defined sequence of the process of the location detection system using CCTV.

FaceMe AI engines have achieved remarkable accuracy in the NIST Face Recognition Vendor Test, earning recognition as a top performer. Given this, FaceMe is frequently serves as a reference system during the development process to ensure the comprehensive and practicality of the proposed system.

CHAPTER 3 System Methodology / Approach

3.1 System Planning

3.1.1 Gantt Chart

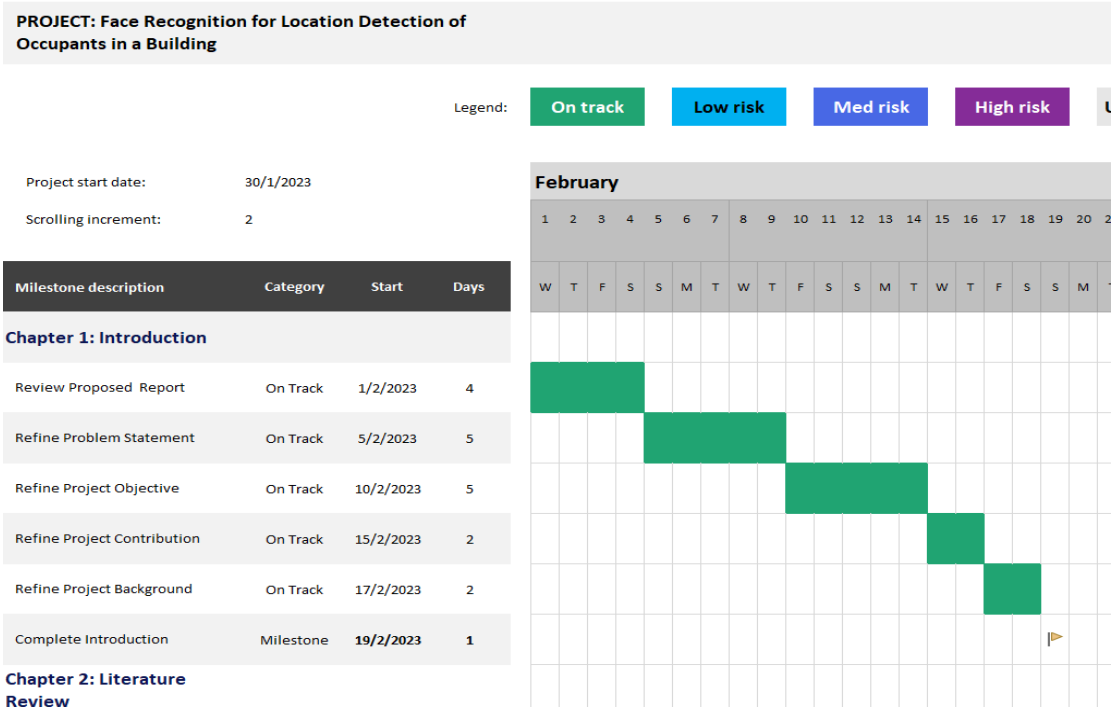


Figure 3.1.1.1 GanttChart (1/5)

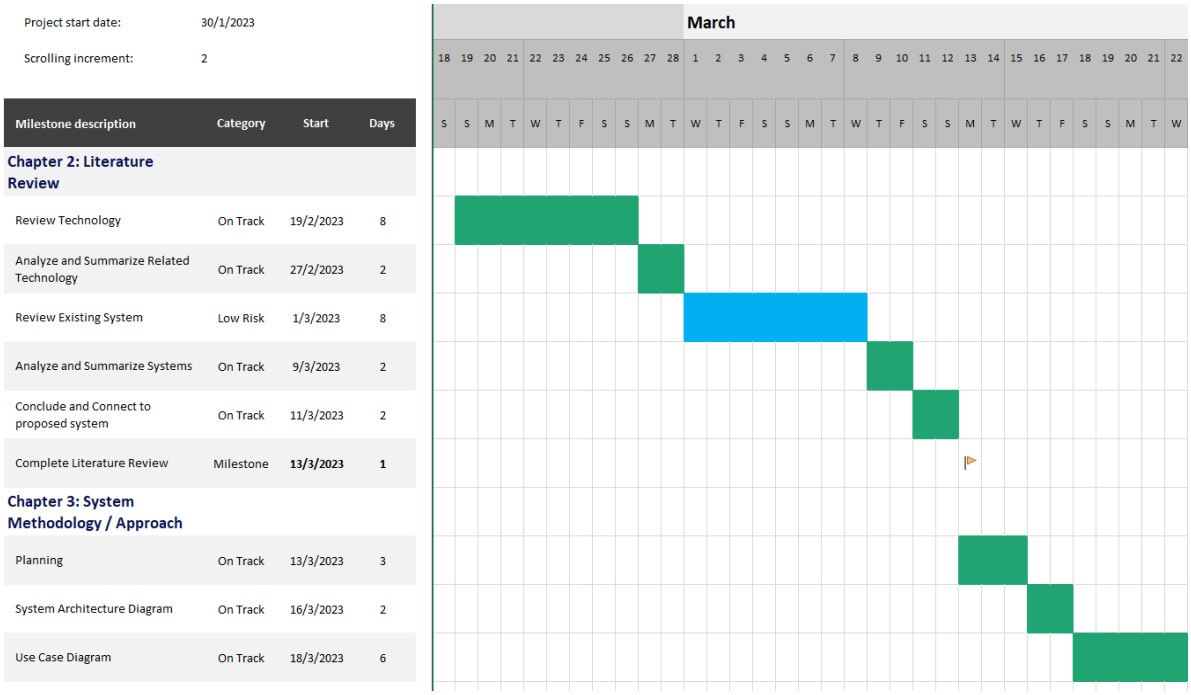


Figure 3.1.1.2 GanttChart (2/5)

Project start date: 30/1/2023

Scrolling increment: 2

Milestone description	Category	Start	Days
Use Case Diagram	On Track	18/3/2023	6
Activity Diagram	On Track	24/3/2023	6
Complete System Methodology	Milestone	**30/3/2023**	**1**
Chapter 4: System Design			
System Flow Diagram	Low Risk	30/3/2023	3
Database Class Diagram	Low Risk	2/4/2023	3
System User Interface Design	Low Risk	5/4/2023	7
System Components Iteraction Operation	On Track	12/4/2023	10
Complete System Design	Milestone	**22/4/2023**	**1**
Chapter 5: System			

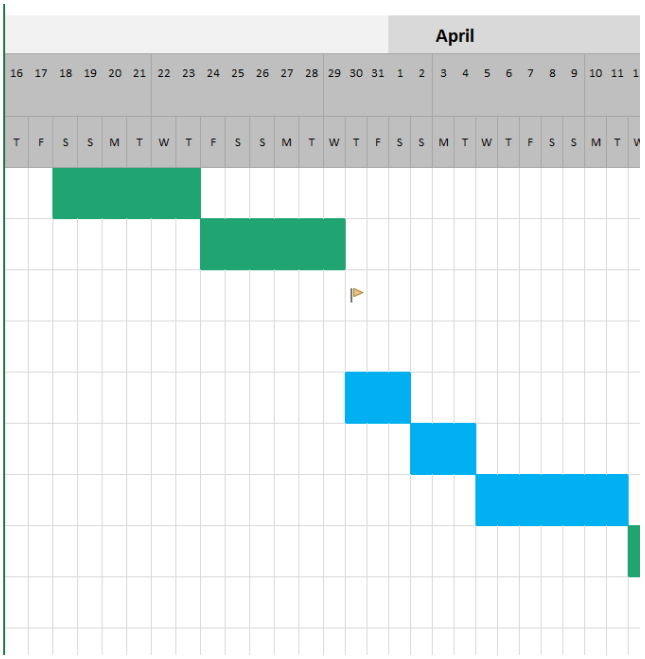


Figure 3.1.1.3 GanttChart (3/5)

Project start date: 30/1/2023

Scrolling increment: 2

Milestone description	Category	Start	Days
System Components Iteraction Operation	On Track	12/4/2023	10
Complete System Design	Milestone	**22/4/2023**	**1**
Chapter 5: System Implementation			
Hardware Setup	On Track	22/4/2023	1
Software Setup	On Track	23/4/2023	2
Setting Configuration	Low Risk	25/4/2023	5
Software Development Life Cycle	Med Risk	30/4/2023	110
Complete Systen Implentation	Milestone	**18/8/2023**	**1**
Chapter 6: System Evaluation and Discussion			
System Testing and Performace	Low Risk	18/8/2023	5

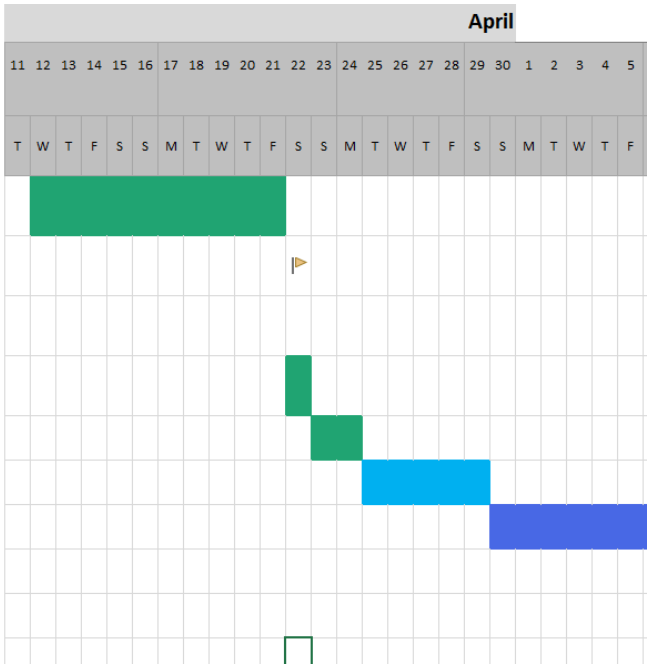
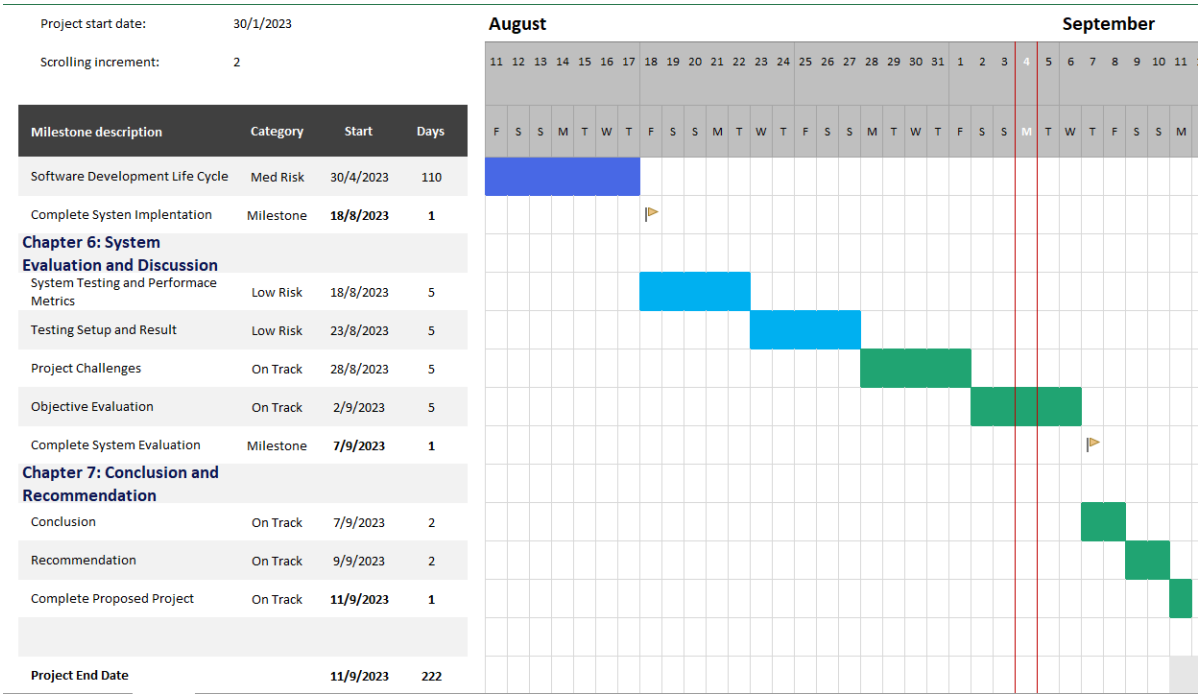


Figure 3.1.1.4 GanttChart (4/5)



Project start date: 30/1/2023

Scrolling increment: 2

Milestone description	Category	Start	Days
Software Development Life Cycle	Med Risk	30/4/2023	110
Complete Systen Implentation	Milestone	**18/8/2023**	**1**
Chapter 6: System Evaluation and Discussion			
System Testing and Performace Metrics	Low Risk	18/8/2023	5
Testing Setup and Result	Low Risk	23/8/2023	5
Project Challenges	On Track	28/8/2023	5
Objective Evaluation	On Track	2/9/2023	5
Complete System Evaluation	Milestone	**7/9/2023**	**1**
Chapter 7: Conclusion and Recommendation			
Conclusion	On Track	7/9/2023	2
Recommendation	On Track	9/9/2023	2
Complete Proposed Project	On Track	**11/9/2023**	**1**
Project End Date		**11/9/2023**	**222**

Figure 3.1.1.5 GanttChart (5/5)

The Gantt Chart shown above was meticulously drafted to provide an in-depth visual representation of our project's schedule. Its goal is to make it easier to track progress, spot potential delays early, and make the necessary adjustments. This Gantt Chart is our tool for keeping a close eye on project activities and ensuring that everything is in line with the overall goals and schedule of the project. Its value lies in facilitating efficient project management, quick detection of schedule deviations, and prompt application of corrective actions to guarantee the project's successful completion.

3.1.2 Methodology Model

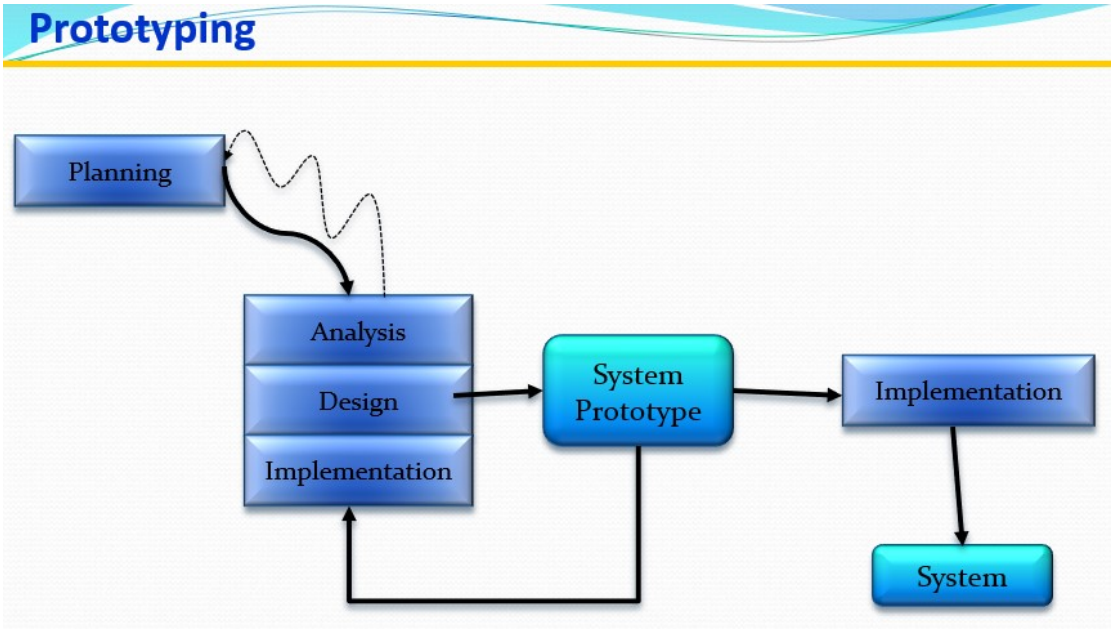


Figure 3.1.2.1 Regular Prototyping Model [23]

The regular prototyping model is an iterative approach to system development which involving cycles of design. Testing, and performance enhancement. This methodology holds promise for enhancing building management systems. Specifically, it has the potential to significantly improve the identification of building occupants and their precise locations within the structure.

Planning and requirement gathering come first. The ability to accurately recognise faces, the ability to allocate people inside the building, and the ability to detect and track people based on their face features are all prerequisites for face recognition.

The second phrase is “analysis, design, implementation” (also known as “prototype development”), which identifies the needs for the system. System’s architecture must be designed which including system prototype, diagram, technology and algorithms, based on those needs. This phrase states that the process will be refined until it satisfies user requirement.

After creating the prototype, the proposed system will be tested those features and functionality to cut down on errors and bugs. The coding process will be significantly more efficient and streamlined because the system design and

requirements are carefully and precisely obtained through repeated evaluation and refinement that match the requirements.

Overall, prototyping model, which is successful strategy for software development, can be used to create FRLD system. By implementing the proposed proposal recommendations, it is able to create an accurate and reliable face recognition system that can be used to improved building management and security.

3.2 System Design Diagram

3.2.1 System Architecture Diagram

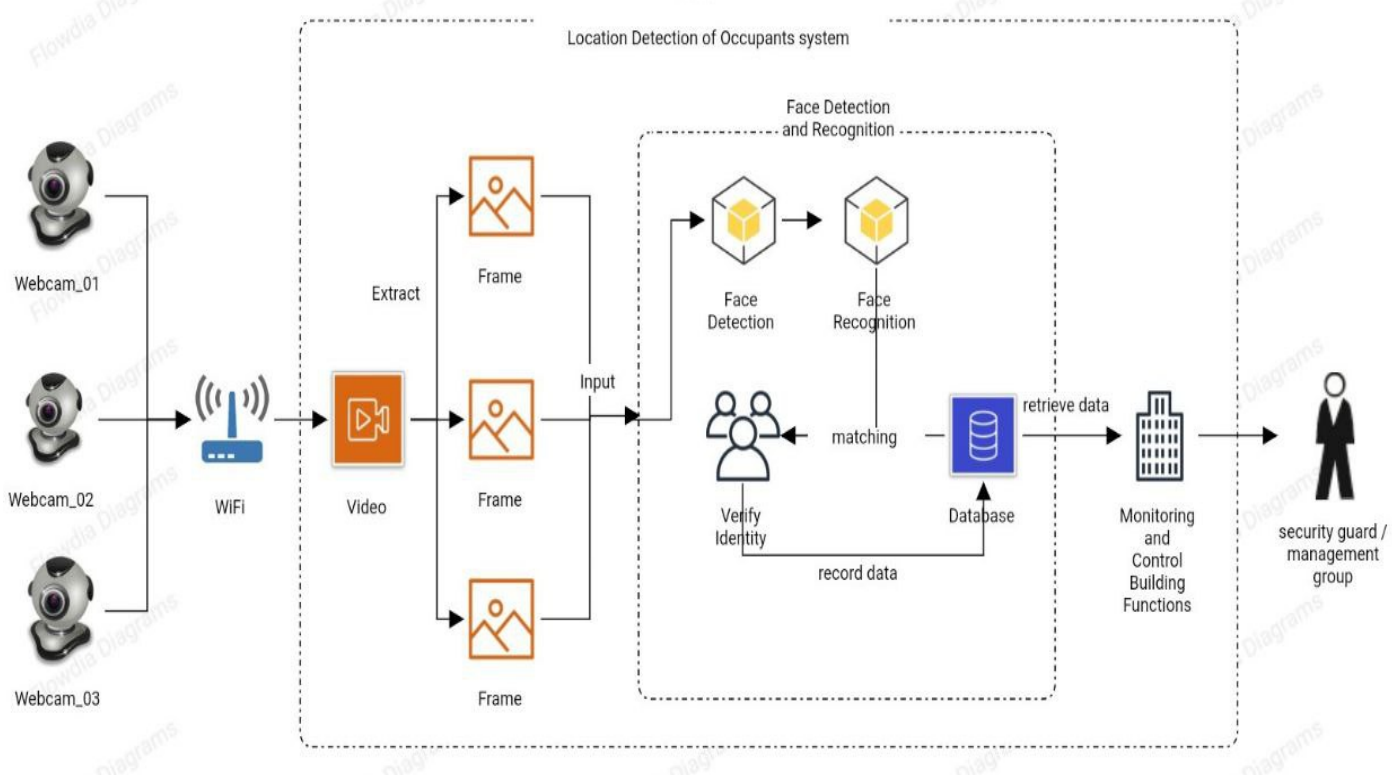


Figure 3.2.1.1 FRLD system architecture

Figure 3.2.1 above is presenting the proposed system architecture which give an in-depth description. Firstly, the system strategically places cameras throughout the building to capture video footage of individual and connecting to the system via WiFi. Next, it starts the analyse the face feature from the captured video frame by frame for proceeding on face detection and recognition process. Thirdly, to determine a person's identity, the system will compare the detected faces with the database of well-known people. Fourthly, it includes a location detection and tracking module that tracks the whereabouts of the identified occupants inside the building by using their location, save and update into database. Then, for efficient management, it analyses, groups and filters this data. In summary, the Face Recognition for Location Detection(FRLD) system's goal is to provide accurate and reliable location data, enhancing the capabilities of the current building management system and enhancing occupant safety as a whole.

3.2.2 System Use Cases Diagram

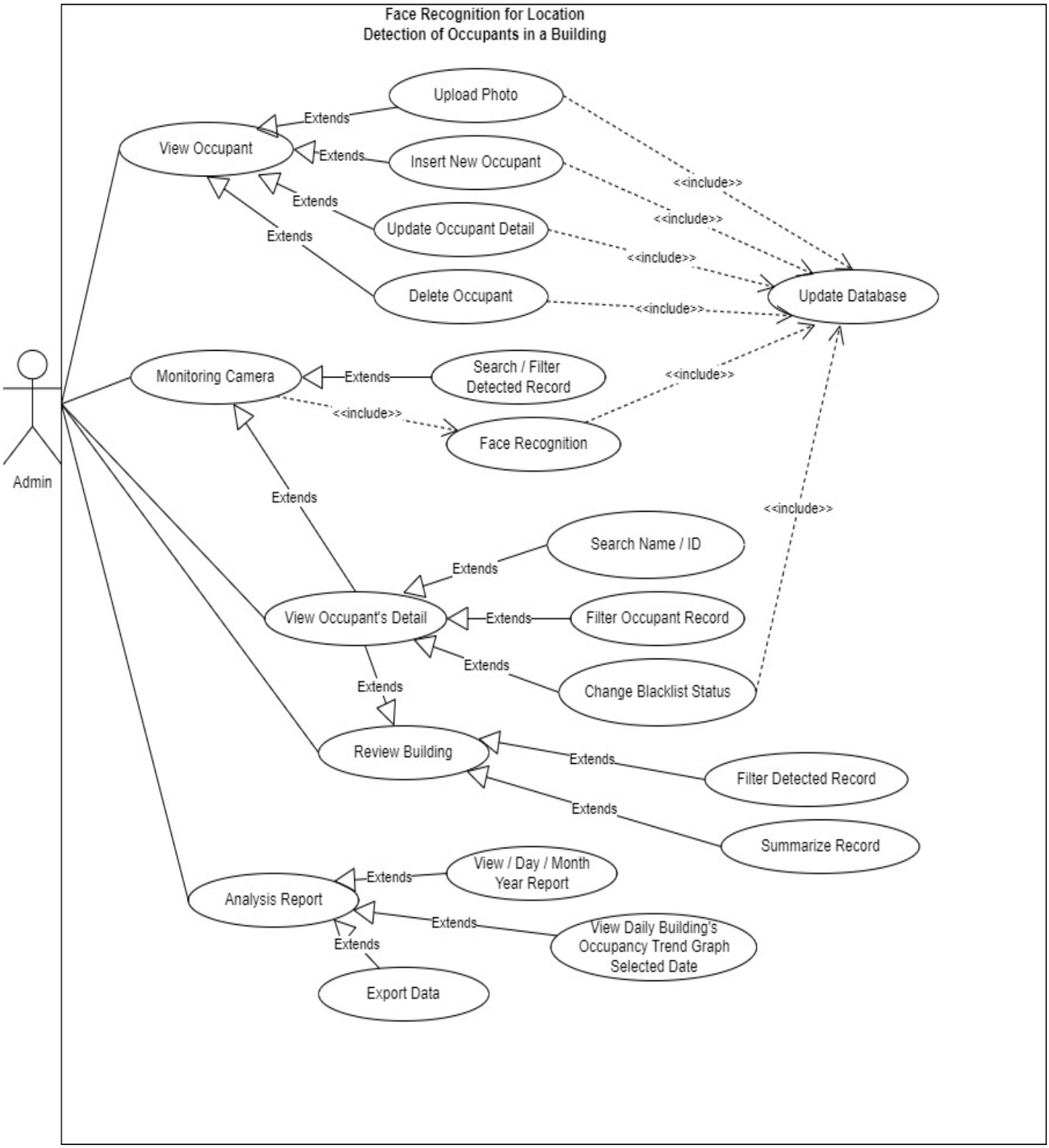


Figure 3.2.2.1 FRLD System Use-Case Diagram

Use Case Name: View Occupants	**ID:** 1	**Importance Level:** High
Primary Actor: Admin	**Use Case Type:** Detail, Essential	
Stakeholder and Interest: Admin – manage the occupants' personal detail within the building		
Brief Description: This use case describes the features of manage occupants' registration		
Trigger: Admin view, manage and insert of building's occupants		
Relationships **Association:** No Application **Include : U**pdate database **Extend :** Upload photo, insert new occupants, update occupants detail, delete occupants		
Normal Flow of Events: 1. Admin view the available occupants which recorded in the building. 2. Admin able to upload occupant's photo. 3. Admin is allowed to insert new occupant inside the system. 4. Admin have the right update the occupant's detail or remove the occupants from the system.		
Sub Flows: Not Applicable		
Alternate/Exceptional Flows: 1a. Fail to connect the system's database. 2a. Image format not allowed. 3a. All details are not filled.		

Table 3.2.2.1 Use Case Description (1/5)

Use Case Name: Monitoring Camera	**ID:** 2	**Importance Level:** High
Primary Actor: Admin	**Use Case Type:** Detail, Essential	
Stakeholder and Interest: Admin – monitoring the camera and detected record		
Brief Description: This use case shown the monitoring on the captured video and detected occupants		
Trigger: Admin monitor on the detect record and capture video.		

Relationships **Association:** **Include :** Update database, face recognition **Extend :** Search or filter detected record
Normal Flow of Events: 1. Admin able to view and monitor on captured video. 2. Admin is allowed to view detected occupancy's detail. 3. Admin narrow down the detected record by using keyword like occupant's name or id to filter the record.
Sub Flows: Not Applicable
Alternate/Exceptional Flows: 1a. More than one occupant face detected. 1b. System fail to record the occupant's detail since more than one face recognize at same time. 3a. Keyword is not fined in the detected list.

Table 3.2.2.2 Use Case Description (2/5)

Use Case Name: View Occupants' Detail	**ID:** 3	**Importance Level:** High
Primary Actor: Admin	**Use Case Type:** Detail, Essential	
Stakeholder and Interest: Admin – monitor the occupants' detail with detected record		
Brief Description: This use case describes how to review the occupants' detail with detected record		
Trigger: Admin able to view comprehensive details about an occupant within the building management system		
Relationships **Association:** **Include :** Update database **Extend :** search name / ID , filter occupant record, change blacklist status		

Normal Flow of Events: 1. Navigate to view occupant's detail with detected record. 2. Narrow down the data range with defined features. 3. Blacklist the occupants or remove from blacklist. 4. Update the blacklist status change into database.
Sub Flows:
Alternate/Exceptional Flows:

Table 3.2.2.3 Use Case Description (3/5)

Use Case Name: Review Building	**ID:** 4	**Importance Level:** High
Primary Actor: Admin	**Use Case Type:** Detail, Essential	
Stakeholder and Interest: Admin – able to review all the detected record within in building.		
Brief Description: This use case shown the steps involved in reviewing a building's detected record within system.		
Trigger: Admin able to comprehensively assess and evaluate the status, and occupancy of a particular building.		
<u>Relationships</u> **Association:** **Include :** retrieve database **Extend :** filter detected record, summarize detected record, sorting the data.		
Normal Flow of Events: 1. Admin able to view all the recognized occupant's record. 2. Admin narrow down the range for view by selecting the features. 3. Admin able to sort the table to have more streamlined view of detailed records.		
Sub Flows: Not Applicable		
Alternate/Exceptional Flows: 1a. Record missing or corrupted. 1b. Some records may not be available or accurate 3a. Display error message if sorting cannot be completed 3b. Choose to retry the sorting operation or report issue for resolution		

Table 3.2.2.4 Use Case Description (4/5)

Use Case Name: Analysis Report	**ID:** 5	**Importance Level:** High
Primary Actor: Admin	**Use Case Type:** Detail, Essential	
Stakeholder and Interest: Admin – interesting in generating and analyzing occupants reports.		
Brief Description: This use case describes the steps for admin to request, generate and analyze occupancy reports.		
Trigger: Admin able to generate an occupancy analysis report.		
Relationships **Association:** No Application **Include :** Retrieve Database **Extend :** View day /month / year Report, View Daily Building's Occupancy Trend Graph		
Normal Flow of Events: 1. Admin select the desired date for which they want to generate report. 2. Admin determines the report's date range, its format (day, month or year). 3. The system retrieve the relevant data from database 4. The system process the data, generate and display the report in dashboard		
Sub Flows: Not Applicable		
Alternate/Exceptional Flows: 3a. insufficient data available to generate requested report 4a. system will display error message if error occurs while processing the data		

Table 3.2.2.5 Use Case Description (5/5)

3.3 Activity Diagram

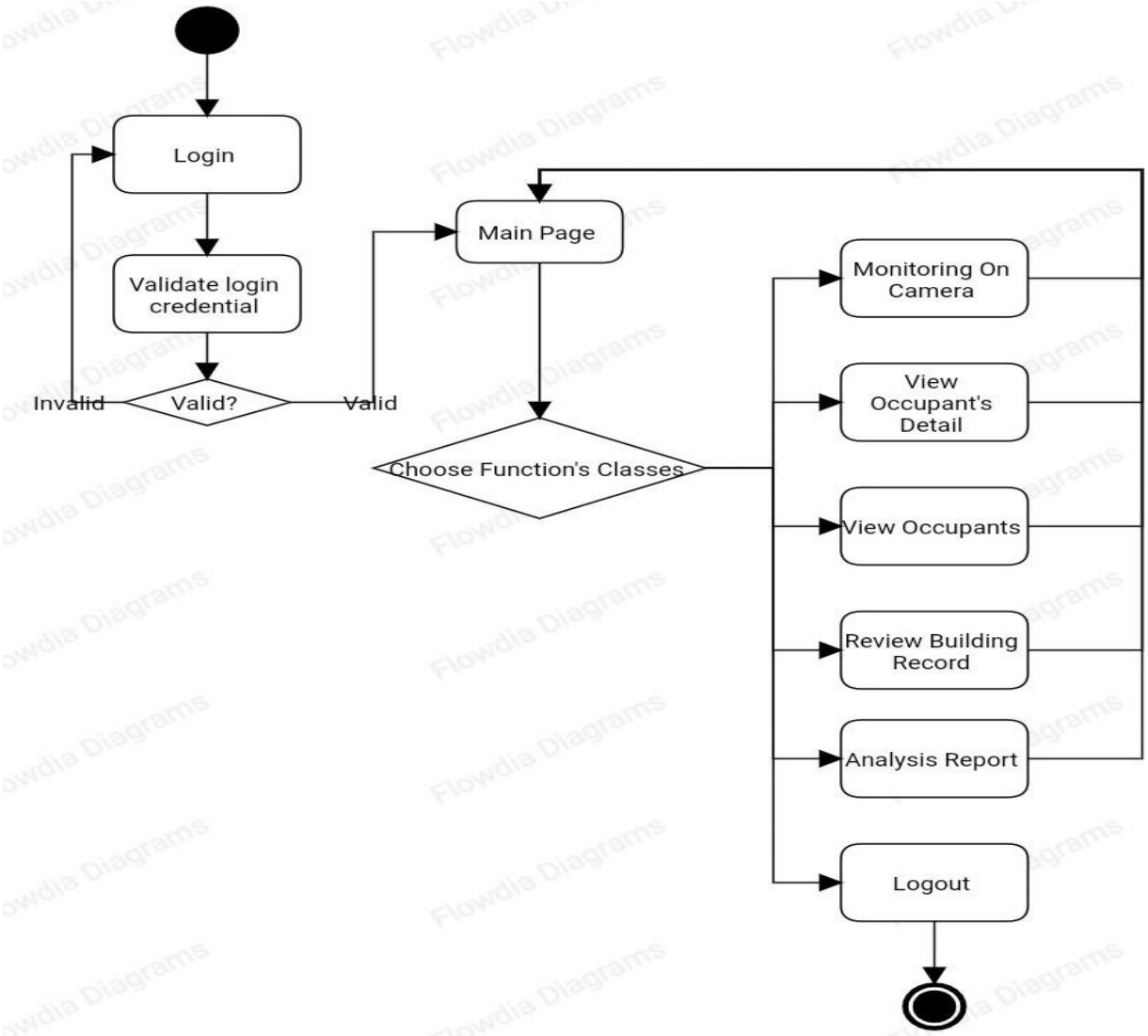


Figure 3.3.1 FRLD Activity Diagram (1/6)

To ensure secure access and proper functionality within the proposed system, users are required to log in before utilising its features. Only authorised users, typically administrators and other authorised staff are allowed access. After the login credential have been successfully validated, the system brings users to main interface, where they can choose from a number of modules made to accommodate various functionalities. These modules include “Monitoring on Camera”, “View Occupant’s Detail”, “View Occupants”, and “Review Building Record”. User must log out in order to leave the system once they have finished their tasks.

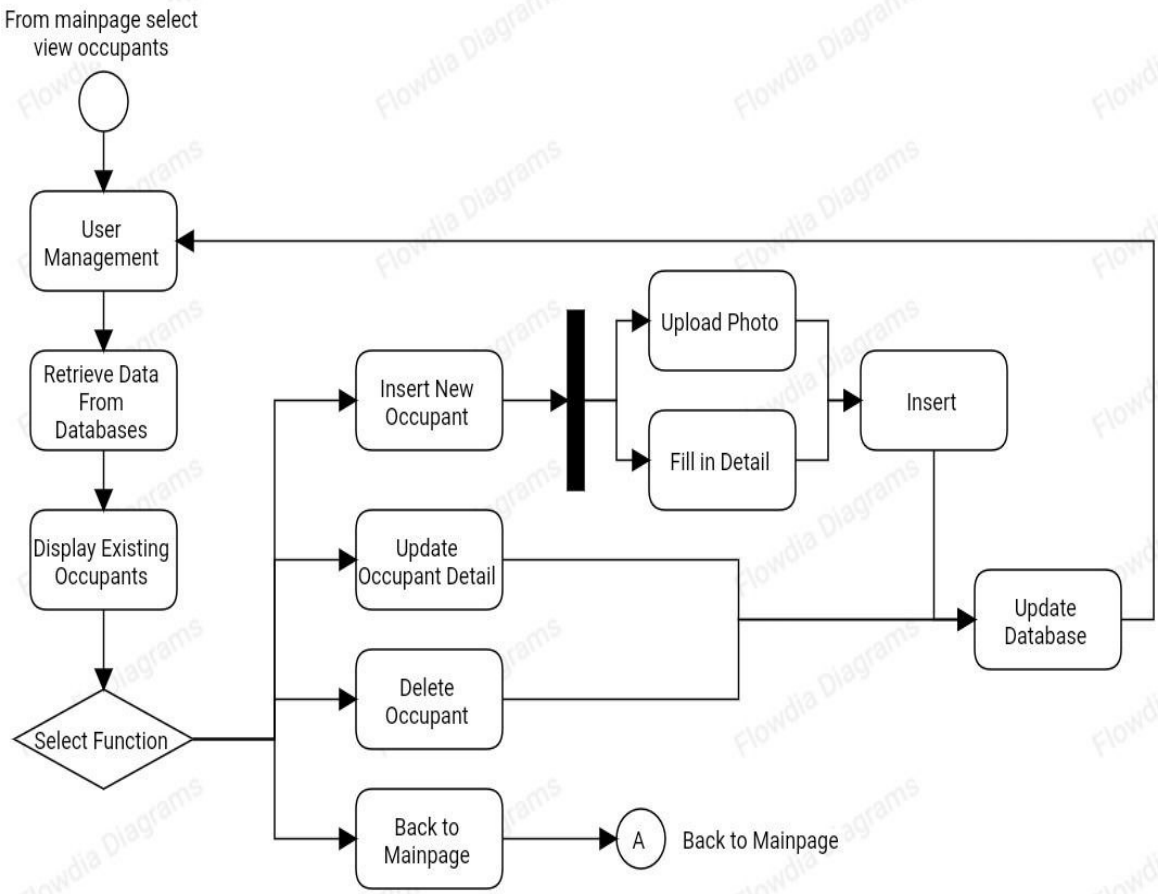


Figure 3.3.2 FRLD Activity Diagram (2/6)

Figure 3.3.2 describe the activity process of the View Occupant module. Once user selected View Occupants module. The technology smoothly gets comprehensive data from the database and displays a thorough list of every occupant who is currently registered. In this system, administrators are given important powers for effective occupant management. These features include the option to add new people, amend current occupant information, and remove residents as necessary. Administrators are prompted to upload the resident's photo and enter any important personal data when adding a new occupant. Administrators can then submit the information for the new occupier, and all transactions are immediately and automatically updated in the database, ensuring real-time information synchronization.

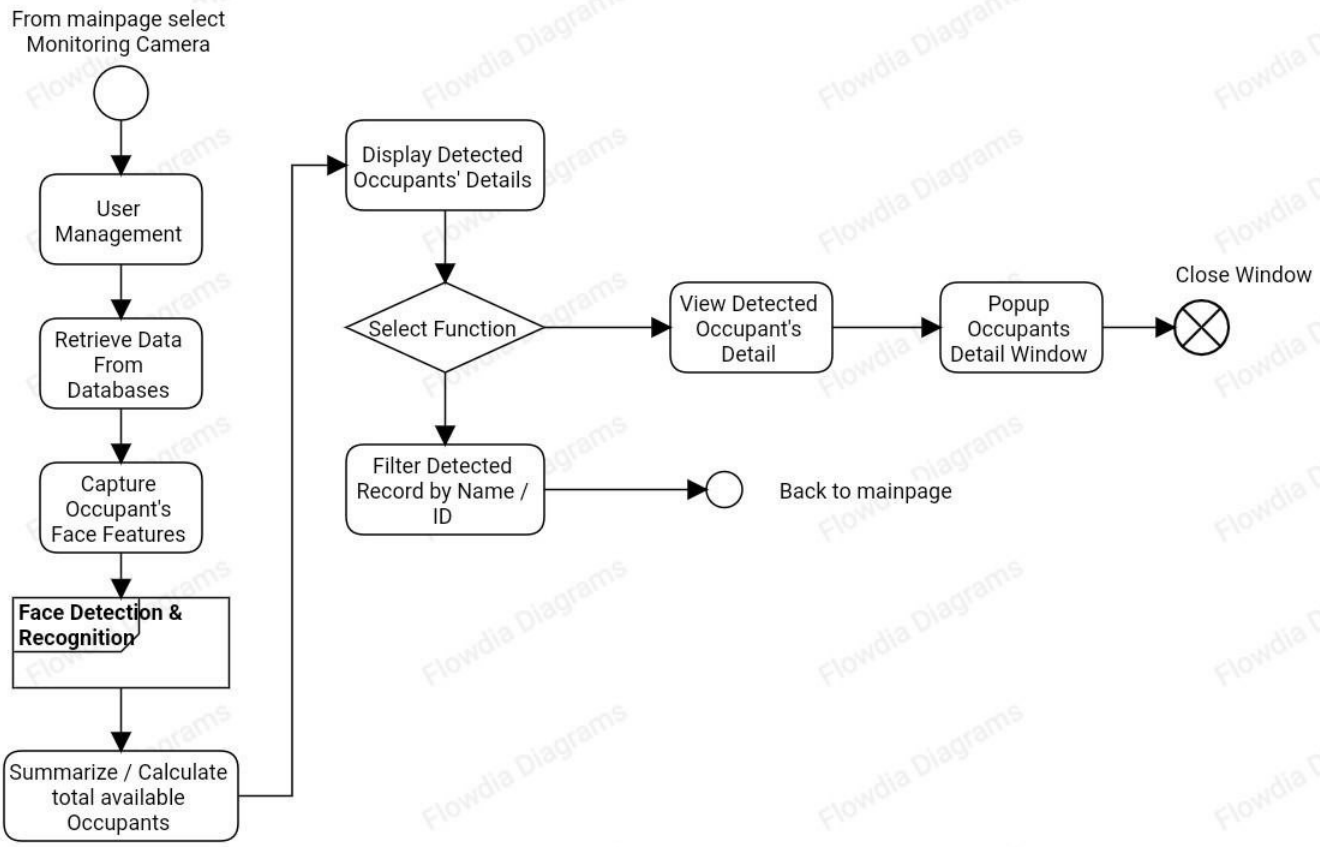


Figure 3.3.3 FRLD Activity Diagram (3/6)

Figure 3.3.3 describe the activity process of the Monitoring Camera. It is outlining the systematic process of monitoring and managing occupants within the system. It is started from obtaining information from database, capturing occupants' face features through camera, employing face detection and recognition techniques to identify individuals. Users are having option to access information about specific occupants or filter data by name of ID, and the system will calculate the total number of occupants shows discovered occupants' detail in real time. This comprehensive approach ensures effective real-time monitoring and control, improving building security and management.

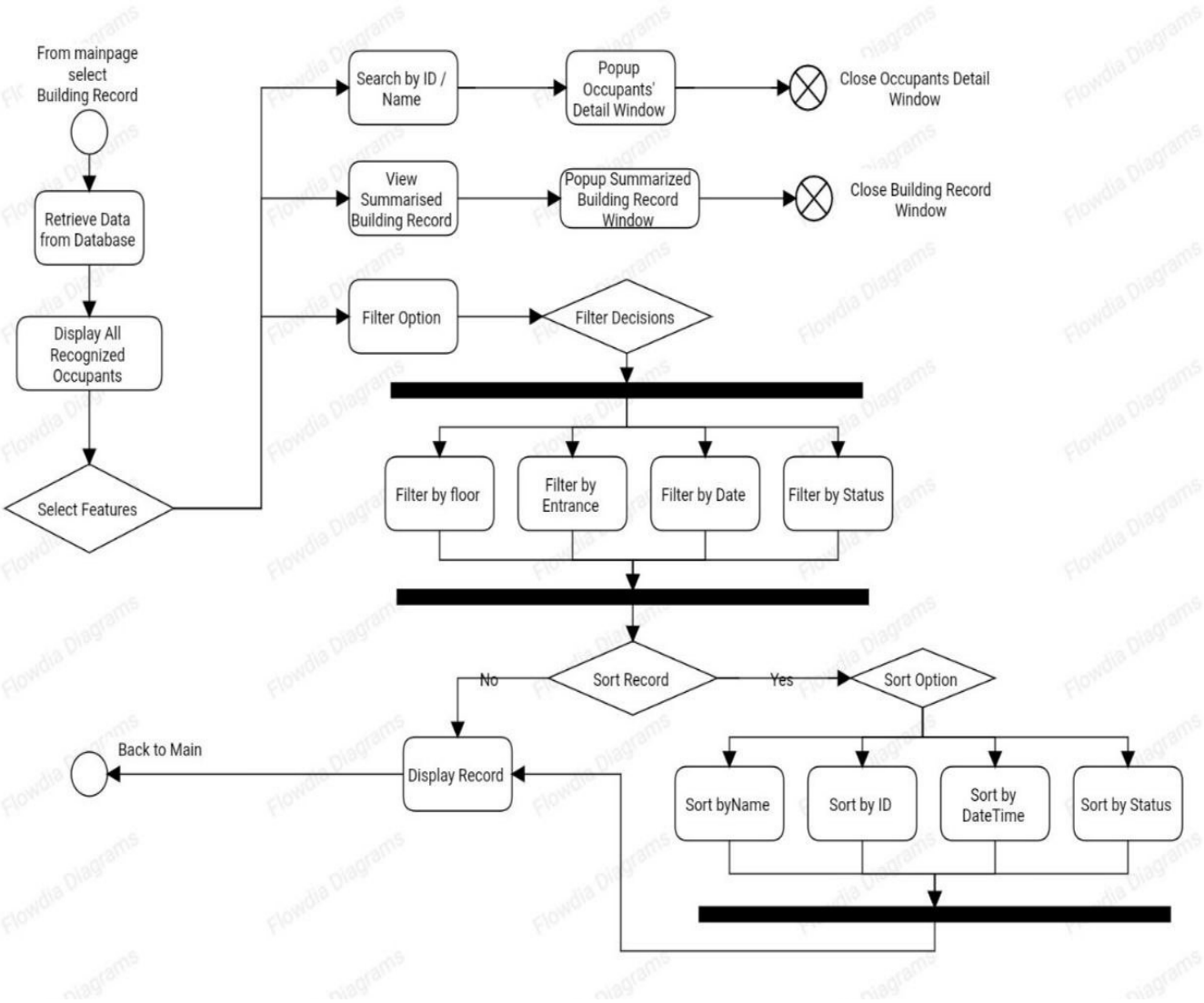


Figure 3.3.4 FRLD Activity Diagram (4/6)

Figure 3.3.4 illustrates the activity of "View Building Record" module, that enabling user or admin to monitor occupant's clock-in and clock-out records, movement and status within the system. Initially, the system retrieves and displays all recognized record from the database, which have been proceeds through AI, ML and DL algorithms for accurate recognition. Then, users can interact with the retrieved data using three essential elements which are search by ID or name, view summarized building record, filter option. Users of the first features can look up occupancy by name or ID, which opens a detailed occupant window for specialized viewing. The second feature offers a streamlined building record view that concentrates on status changes like clock-ins and clock-outs while removing redundant entries of data. The third features provide filtering

options that let user select which record are presented based on floor, entrance, data, status specification. Through streamlined data display, the data is updated in real-time to fit the selected filters and may be sorted by name, ID, datetime or status that ensuring an effective and efficient approach to building management.

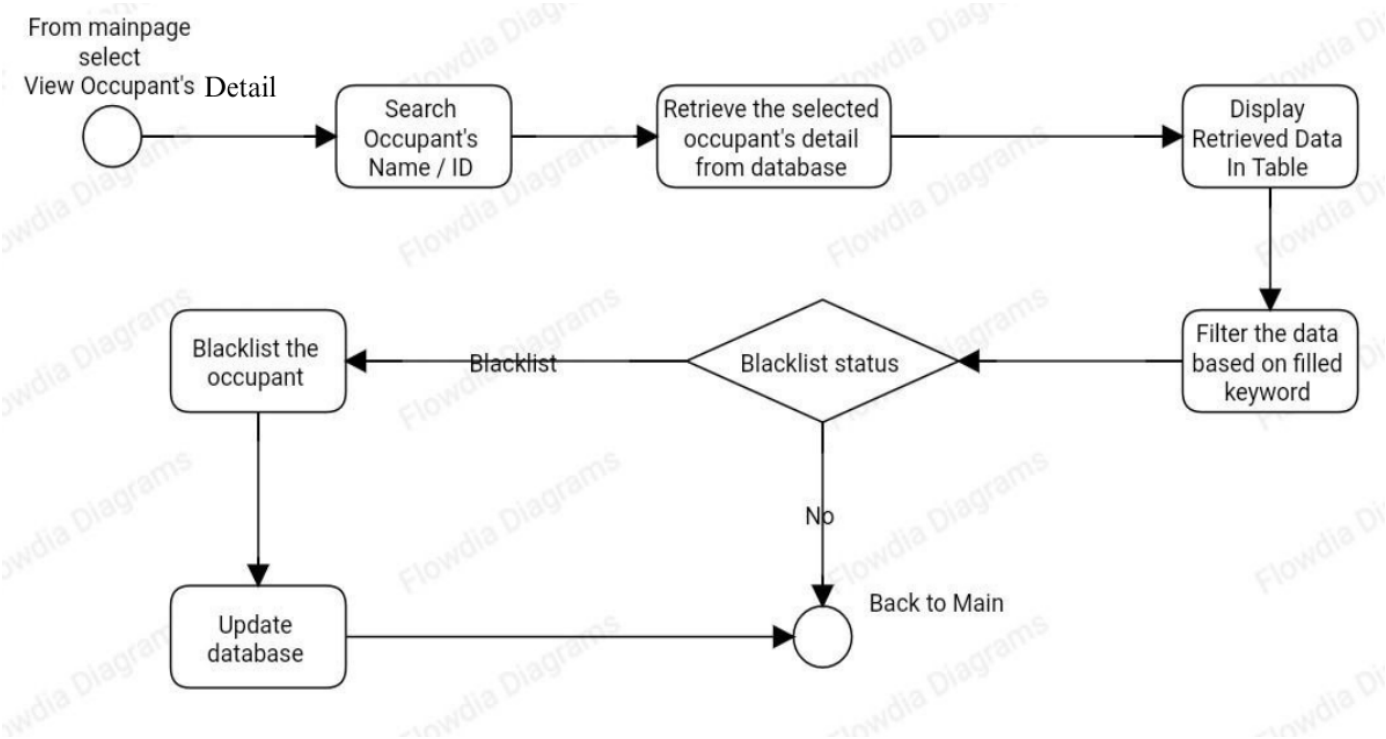


Figure 3.3.5 FRLD Activity Diagram (5/6)

Figure 3.3.5 is describing the module of "View Occupant's Detail" as a well-structured and user-friendly module for retrieving specific information about target occupants within the building. It starts by allowing users to search for particular occupant using their name or ID, which triggers the retrieval of occupant's detail from the database and display them in table view for easy access. To streamline the search process, the filter function is available, enabling users to narrow down the dataset for display by utilizing keywords such as datatime, entrance number, floor level or status. There is an interesting feature which is the ability to manage the blacklist status of occupants within this module, offering the option to blacklist or remove individual from blacklist. The database is rapidly updated is any status changed. Overall, this module gives building management skill a simple and efficient method for managing and keeping an eye on certain occupancy.

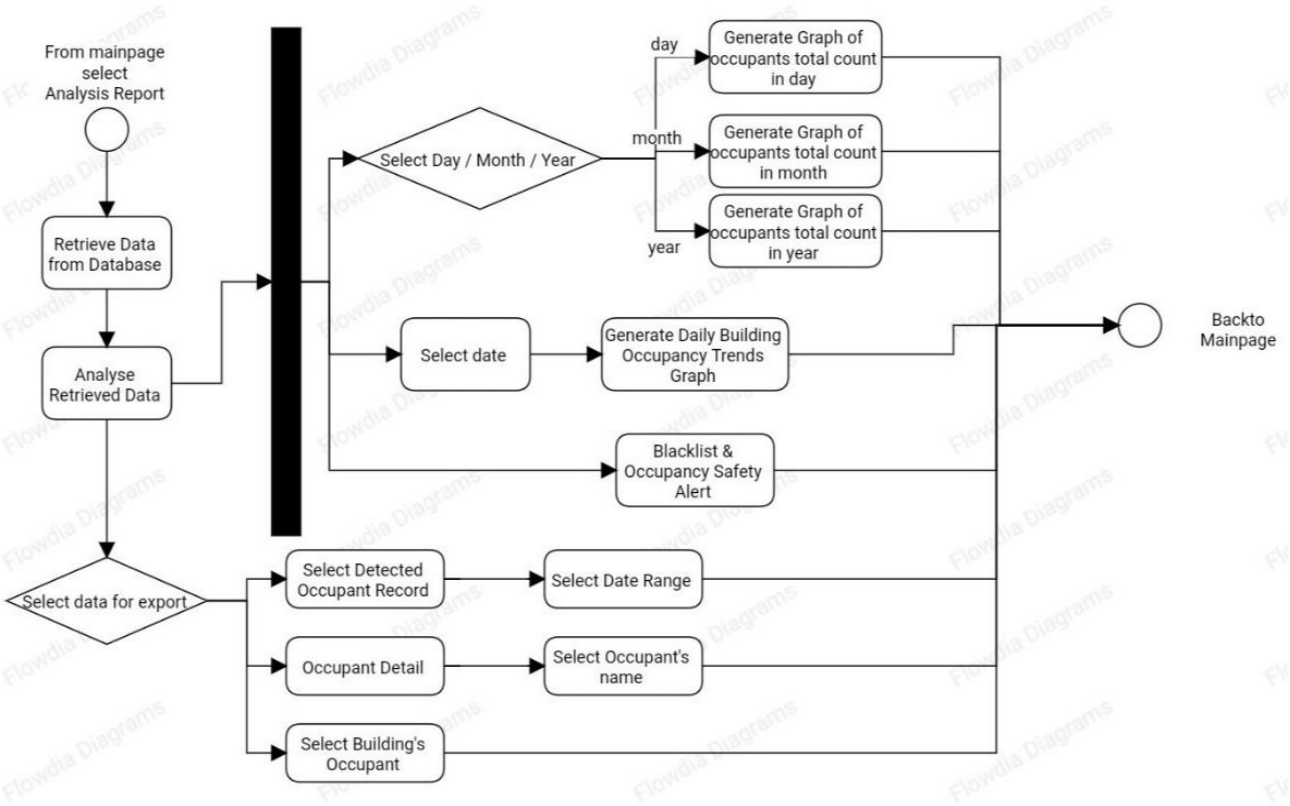


Figure 3.3.6 FRLD Activity Diagram (6/6)

Figure 3.3.6, the "Analysis Report" activity diagram outline a comprehensive module designed to retrieve, analyze, and present data which related to building occupancy. It starts by retrieving data from database, after which the system proceeds to analyze this data based on predefined rules. The analysed data is then displayed in graph by using default options for presentation. User will have the flexibility to choose which graph they would like to view with options such as total count of occupants in day, month or year. Additionally, the system will also generate a daily building occupancy trend graph based on the selected date. Simultaneously, the system incorporates a blacklist and occupancy safety alert system to notify admins or users of any building-related alerts. Another than that, an essential feature of this module is data export which enables users to export data in Excel file format based on chosen applications, such as detected occupant record during a given period range, particular occupant details, or all registered occupants' data in the building. In conclusion, this module serves as a dashboard for administrators, providing quick access to crucial building occupancy information.

CHAPTER 4

System Design

4.1 System Block Diagram

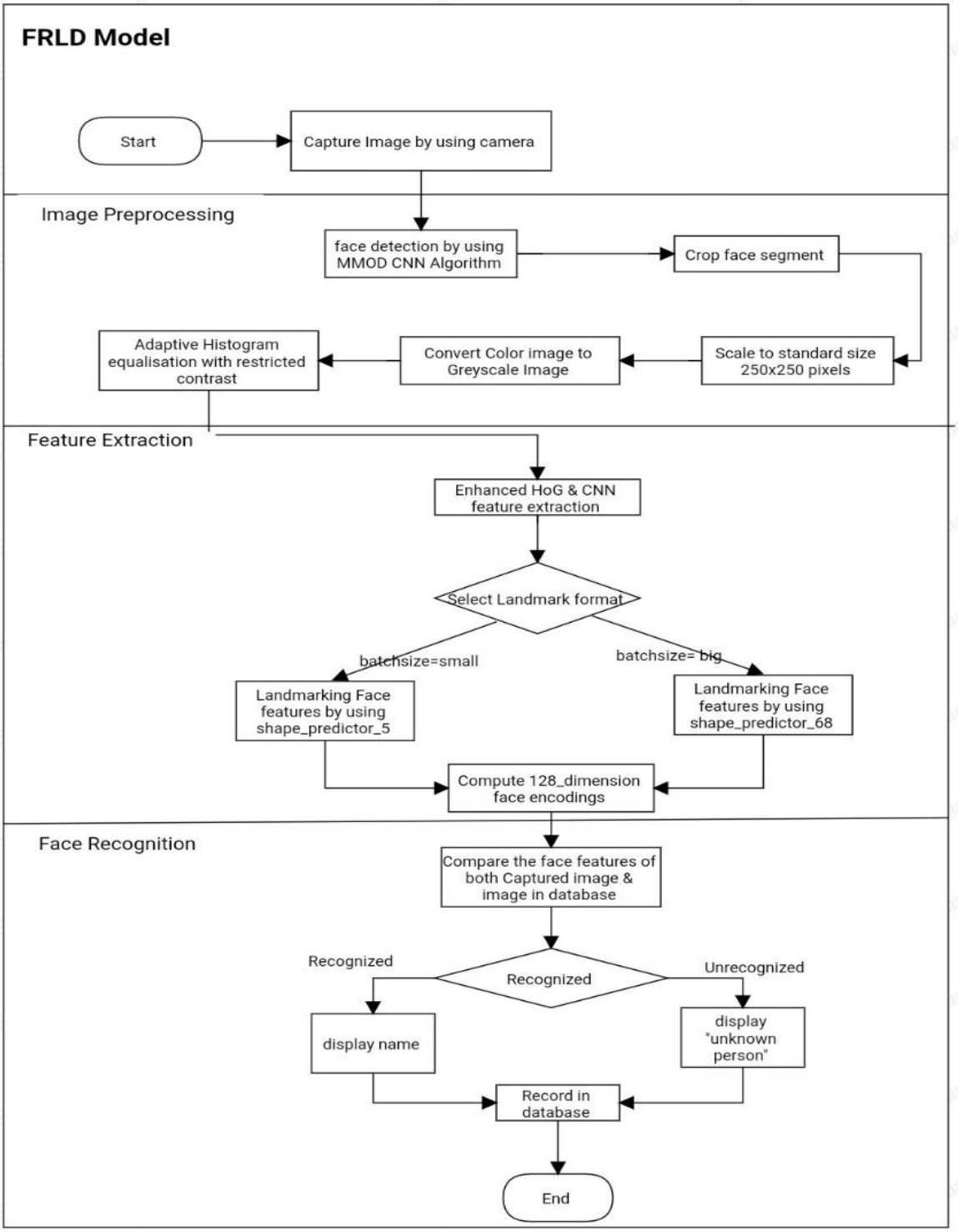


Figure 4.1.1 FRLD Block Diagram

FRLD block diagram (Figure 4.1.1) which visualize the overview of the system process and concept. The initial step represents the process of capturing image from the record real-time video frame by frame. The system process can be clustering into three main stages which is image preprocessing, features extraction and recognition.

During **image preprocessing** stage, Max-Margin (MMOD) CNN algorithm is utilized for face detection. The DLIB-based face detection method known as MMOD CNN accepts only one parameter, model Path, which specifies the location of the pre-trained mmod_human_face_detector.dat file stored on discs and bounding boxes and draw on output. This method prioritises accuracy above performance speed [24]. Once a face is detected, the system extracts the face by utilizing the bounding box provided by MMOD. This step is crucial for face recognition, especially when there are multiple faces in the same frame. Following this, the system resizes the cropped image from frame to a standardized 250x250-pixel format, ensuring consistency in preprocessing. This standardization simplifies further analysis by enabling models and algorithms to operate on faces with a known and fixed size. Additionally, the conversion to grayscale and adaptive contrast enhancement enhances facial data for analysis. This preprocessing streamlines the processing, reduces computational complexity, and sharpens the image, improving the visibility of facial features and details.

In **feature extraction** stage, the system will proceed with enhanced Histogram of Oriented Gradients (HOG) and Convolutional Neural Network (CNN) features extraction methods. These techniques are used to capture the complex details and distinctive features of human photos. HOG is applying hard-negative mining which record the feature vector that associated with false-positive patch together with the probability of the classification if there incorrectly labels a particular window as an object and it will undoubtedly be false-positives. However, CNN rely on three key components for feature extraction from facial image which are CNN connect hidden neurons to localized regions of input images and allow them focus on specific image details through learned weights and biases (Local Receptive Fields); Neurons with same stride in local receptive fields share the same weights and biases, which reduce the network complexity and encourages the detection of common features in various input locations (Shared Weights and Biases); The pooling layer reduce spatial dimensions while keeping important information by choosing the maximum activation

within small input regions to condense the convolutional layer's output features map [25]. Following feature extraction, the system proceeds to determine the landmark format based on the batch size of image. If the batch size is small, the system will opt for the shape_predictor_5 landmark format. However, in the case of larger batch size, shape_predictor_68 format will be chosen which provide more comprehensive landmarking face features. Face landmark which is localizing and labelling specific face regions within (x,y)-coordinates. These regions encompass main facial components which are mouth, right eyebrow, left eyebrow, left eye, right eye, nose and jaw. This landmarking procedure is important for precisely identifying and distinguishing different face features for subsequent analysis and recognition tasks. Once face features completely landmarked, system will compute 128-dimensional face encoding which convert image into informative numerical representation of face for efficient and accurate face recognition during subsequent stages of processing.

After encoding the photo, system will the process of comparing the face features extracted from the captured image in the recorded video with those stored in the database. Once a match detected, system will promptly display the occupant's name else will display unknown and log the information into the database for further reference.

4.2 System Flow Description

A system flow description is an important document that illustrate how several modules interact and communicate within a software environment to achieve functionality or task. It acts a comprehensive guide that explains the relationship between several modules, the information exchange and how they dependence on one another, all with the aim successfully implementing face recognition. This description offers a clear road map for understanding the coordinated cooperation between these modules within the software system.

4.2.1 main_faceRecognition.py

Main.py | main_faceRecogntion.py

```
import face_recognition
import cv2
import os
import glob
import numpy as np

Cheong Kok Siong
class Facerec:
    Cheong Kok Siong
    def __init__(self):
        self.known_face_encodings = []
        self.known_face_names = []

        # Resize frame for a faster speed
        self.frame_resizing = 0.25

    Cheong Kok Siong
    def load_encoding_images(self, images_path):
        """
        Load encoded images from images path
        :param images_path:
        :return:
        """
        images_path = glob.glob(os.path.join(images_path, "*.*"))
        print("{} encoding images found.".format(len(images_path)))

        # Store image encoding and names
        for img_path in images_path:
            img = cv2.imread(img_path)
```

Figure 4.2.1.1 main_faceRecogntion.py (1/3)

```
        img = cv2.imread(img_path)
        rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Get the filename from the initial file path.
        basename = os.path.basename(img_path)
        (filename, ext) = os.path.splitext(basename)

        # Get encoding image
        img_encoding = face_recognition.face_encodings(rgb_img)[0]
        #print(filename)
        #print(img_encoding)

        # Store file name and file encoding
        self.known_face_encodings.append(img_encoding)
        self.known_face_names.append(filename)
        print(img_encoding)
        print(filename)
    print("Encoding images loaded")

Cheong Kok Siong
def detect_known_faces(self, frame):
    small_frame = cv2.resize(frame, (0, 0), fx=self.frame_resizing, fy=self.frame_resizing)
    # Detect faces and face encodings in the current frame of video
    # Convert the image from BGR color (which OpenCV uses) to RGB color (which face_recognition uses)
    rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)
    face_locations = face_recognition.face_locations(rgb_small_frame)
    face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)
```

Figure 4.2.1.2 main_faceRecogntion.py (2/3)

```
    face_locations = face_recognition.face_locations(rgb_small_frame)
    face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

    face_names = []
    for face_encoding in face_encodings:
        # match for the known face(s)
        matches = face_recognition.compare_faces(self.known_face_encodings, face_encoding)
        name = "Unknown"

        face_distances = face_recognition.face_distance(self.known_face_encodings, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = self.known_face_names[best_match_index]
        face_names.append(name)

    # Convert to numpy array to adjust coordinates with frame resize quickly
    face_locations = np.array(face_locations)
    face_locations = face_locations / self.frame_resizing
    return face_locations.astype(int), face_names
```

Figure 4.2.1.3 main_faceRecogntion.py (3/3)

This module (main_faceRecogntion.py) is define a class called “Facerec” for face recognition which based on those python libraries included “face_recogniton”, “cv2”, “”os”, “glob”, and “numpy” to perform face recognition on photo. This class start by

creating empty lists to store known face encodings and names, and also have to set a frame resizing factor for speed optimization. The import module pre-defined function "load_encoding_images" loads images from targeted path, encodes the faces in those photos and store the encoding along with the filenames. The function "detect_known_faces" will takes the input frame which captured from live recorded video, resize it and detect the face in each frame, then computes face encodings and match them against the known face encodings. The names and locations of recognized faces in each frame are returned.

4.2.2 Main.py

Main.py

```
import sys
from PySide2.QtWidgets import QApplication, QMainWindow, QTabWidget, QWidget, QLabel, QVBoxLayout, QHBoxLayout, \
    QMessageBox
from main_cam1 import FVWindow
from main_cam2 import FVWindow2
from main_occupants import MainOccupantsWindow
from main_building_record import BuildingRecord_Window
from main_occupant_detail import MainOccupants_DetailWindow
from UI_login import Ui_MainWindow
from main_report import MainReport_Window



Cheong Kok Siong
class LoginWindow(QMainWindow):
    Cheong Kok Siong
    def __init__(self):
        super().__init__()

        # Create an instance of the Ui_MainWindow class
        self.ui = Ui_MainWindow()

        # Set up the user interface
        self.ui.setupUi(self)

        # Connect the login button to the login function
        self.ui.pushButton.clicked.connect(self.login)


```

Figure 4.2.2.1 Main.py (1/6)

```
29          # Set the fixed size of the main window
30          self.setFixedSize(self.size())
31
            Cheong Kok Siong
32      def login(self):
33          username = self.ui.textEdit.toPlainText()
34          password = self.ui.textEdit_4.text()
35          print(username)
36          print(password)
37
38          # Check if the username and password are valid
39          if username == "aaa" and password == "aaa":
40              QMessageBox.information(self, "Login Successful", "Welcome, admin!")
41              self.open_fv_window()
42          else:
43              QMessageBox.warning(self, "Login Failed", "Invalid username or password. Please try again.")
44
            Cheong Kok Siong
45      def open_fv_window(self):
46          # Create the FVWindow instance
47          self.myapp = MyApplication()
48
49          # Hide the LoginWindow
50          self.hide()
51
52          # Show the FVWindow
53          self.myapp.show()
54
```

Figure 4.2.2.2 Main.py (2/6)

```
55      def closeEvent(self, event):
56          # Close the FVWindow if it is open
57          if hasattr(self, 'MyApplication'):
58              self.myapp.close()
59
60          event.accept()
61
62
            Cheong Kok Siong *
63  class MyApplication(QMainWindow):
            Cheong Kok Siong
64      def __init__(self):
65          super().__init__()
66
67          self.init_ui()
68
            Cheong Kok Siong *
69      def init_ui(self):
70          self.setWindowTitle("Tab Widget Example")
71          self.setGeometry(100, 100, 800, 600)
72          self.resize(1476, 871)
73          self.setFixedSize(self.size())
74
75          # Create a tab widget
76          tab_widget = QTabWidget()
77
78          # Create tabs and add them to the tab widget
79          #tab1 = QWidget()
80          tab2 = QWidget()
```

Figure 4.2.2.3 Main.py (3/6)

```
79          #tab1 = QWidget()
80          tab2 = QWidget()
81          tab3 = QWidget()
82          tab4 = QWidget()
83          tab5 = QWidget()
84          tab6 = QWidget()
85          tab7 = QWidget()
86
87          # tab_widget.addTab(tab1,"MultiCam")
88          tab_widget.addTab(tab2, "MainCam")
89          tab_widget.addTab(tab3, "ExitCam")
90          tab_widget.addTab(tab4, "Occupant_Record")
91          tab_widget.addTab(tab5, "Building_Record")
92          tab_widget.addTab(tab6,"Occupant_Detail")
93          tab_widget.addTab(tab7, "Analysis Report")
94
95
96          layout2 = QVBoxLayout()
97          fv_window = FVWindow()  # Create an instance of FVWindow
98          layout2.addWidget(fv_window)
99          tab2.setLayout(layout2)
100
101         layout3 = QVBoxLayout()
102         fv_window2 = FVWindow2()
103         layout3.addWidget(fv_window2)
104         tab3.setLayout(layout3)
105
106         layout4 = QVBoxLayout()
107         occ_window = MainOccupantsWindow()
```

Figure 4.2.2.4 Main.py (4/6)

```
106         layout4 = QVBoxLayout()
107         occ_window = MainOccupantsWindow()
108         layout4.addWidget(occ_window)
109         tab4.setLayout(layout4)
110
111         layout5 = QVBoxLayout()
112         building_window = BuildingRecord_Window()
113         layout5.addWidget(building_window)
114         tab5.setLayout(layout5)
115
116         layout6 = QHBoxLayout()
117         occ_detail_window = MainOccupants_DetailWindow()
118         layout6.addStretch()  # Add stretchable space before the content
119         layout6.addWidget(occ_detail_window)  # Add the content (occupant detail) to the layout
120         layout6.addStretch()  # Add stretchable space after the content
121         tab6.setLayout(layout6)
122
123         layout7 = QVBoxLayout()
124         report_window = MainReport_Window()
125         layout7.addWidget(report_window)
126         tab7.setLayout(layout7)
127
128         # Set the central widget to the tab widget
129         self.setCentralWidget(tab_widget)
130
            Cheong Kok Siong
131     def main():
132         # Create the QApplication instance
133         app = QApplication(sys.argv)
```

Figure 4.2.2.5 Main.py (5/6)

```
127
128             # Set the central widget to the tab widget
129             self.setCentralWidget(tab_widget)
130
        Cheong Kok Siong
131     def main():
132         # Create the QApplication instance
133         app = QApplication(sys.argv)
134
135         # Create the LoginWindow instance
136         loginWindow = LoginWindow()
137
138         # Show the login window
139         loginWindow.show()
140
141         # Start the application event loop
142         sys.exit(app.exec_())
143
144 if __name__ == '__main__':
145     main()
146
```

Figure 4.2.2.6 Main.py (6/6)

Main.py which is the main module that as a central module to connect all the developed modules. The user interfaces for main.py is developed based on PySide2 library like UI_login.py. It starts with login window where user enter his/her login credentials, if login success, a main application window is displayed with tabs for different functions, such as managing camera feeds, handling occupant and building records, viewing occupant details, and generating analysis reports. The code organizes these functions using layout and widgets. Overall, it is a user-friendly interface for managing building occupancy data and related tasks.

4.2.3 main_cam1.py & main_cam2.py

```python
import sys
import re
import cv2
from UI_cam1 import Ui_FVWindow
from PySide2.QtGui import QPixmap, QImage, QFontMetrics, QFont, QColor
from PySide2.QtWidgets import QApplication, QMainWindow, QLabel, QFrame, QMessageBox, QAction, QListWidgetItem, \
    QListWidget
from PySide2.QtCore import Qt, QTimer, QRect, QCoreApplication
from PySide2.QtSql import QSqlDatabase, QSqlQuery
from datetime import datetime
import sqlite3

from test_main_occupants import MainOccupantsWindow
from main_occupant_detail import MainOccupants_DetailWindow

from main_faceRecogntion import Facerec

sfr = Facerec()
sfr.load_encoding_images("images/")


...
```

Figure 4.2.3.1 main_cam1.py & main_cam2.py (1/20)

```python
Cheong Kok Siong
class FVWindow(QMainWindow):
    Cheong Kok Siong
    def __init__(self):
        super().__init__()

        # Create an instance of the Ui_FVWindow class
        self.ui = Ui_FVWindow()

        # Set up the user interface
        # custom_color = QColor(176, 224, 230)
        # self.setStyleSheet(f"background-color: {custom_color.name()};")

        self.ui.setupUi(self)

        self.detail_window = None

        # Set the fixed size of the FVWindow
        self.setFixedSize(self.size())

        # Add any additional setup for the FVWindow
        self.create_detected_occupants_table()

        # Open the camera
        self.cap = cv2.VideoCapture(1)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1201)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 550)

```

Figure 4.2.3.2 main_cam1.py & main_cam2.py (2/20)

```
97
98          # self.cap2 = cv2.VideoCapture(1)
99          # self.cap2.set(cv2.CAP_PROP_FRAME_WIDTH, 1201)
100         # self.cap2.set(cv2.CAP_PROP_FRAME_HEIGHT, 550)
101
102         # Create a QLabel widget to display the image
103         self.image_label = QLabel(self.ui.frame)
104         self.image_label.setGeometry(QRect(0, 0, 1201, 550))
105         self.image_label.setScaledContents(True)
106
107         self.photo_label = QLabel(self.ui.frame_2)
108         self.photo_label.setScaledContents(True)
109         self.photo_label.setGeometry(QRect(5, 0, 230, 220))
110
111         self.listWidget = QListWidget(self.ui.listWidget)
112         self.listWidget.setGeometry(0,0,275,1500)
113         self.listWidget.setAutoScroll(True)
114
115         self.listWidget.itemClicked.connect(self.click_list_item)
116
117
118         # Start the timer to update the frame
119         # self.timer = QTimer()
120         # self.timer.timeout.connect(self.update_frame)
121         # self.timer.timeout.connect(self.update_list_background)
122         # self.timer.start(30)  # Update every 30 milliseconds
123
124         text_browser_height = 35
```

Figure 4.2.3.3 main_cam1.py & main_cam2.py (3/20)

```
124         text_browser_height = 35
125         self.ui.textBrowser_13.setFixedHeight(text_browser_height)
126         self.ui.textBrowser_12.setFixedHeight(text_browser_height)
127         self.ui.textBrowser_14.setFixedHeight(text_browser_height)
128
129         self.ui.plainTextEdit.setFixedHeight(text_browser_height)
130
131         # Set the font size of the textBrowser widgets
132         font_size = 12
133         font = QFont()
134         font.setPointSize(font_size)
135         self.ui.textBrowser_12.setFont(font)
136         self.ui.textBrowser_13.setFont(font)
137         self.ui.textBrowser_14.setFont(font)
138
139
140         # Start the timer to update the current time
141         self.time_timer = QTimer()
142         self.time_timer.timeout.connect(self.update_frame)
143         #self.time_timer.timeout.connect(self.update_list_background)
144         self.time_timer.timeout.connect(self.update_datetime)
145         self.time_timer.timeout.connect(self.count_occ_in_building)
146         self.time_timer.start(1000)  # Update every 1 second
147
148         self.ui.pushButton.clicked.connect(self.filter_list_widget)
149
150         # Uncomment the following lines to connect the "Occupants" menu item
151
EVWindow  >  init ()
```

Figure 4.2.3.4 main_cam1.py & main_cam2.py (4/20)

```
148         self.ui.pushButton.clicked.connect(self.filter_list_widget)
149
150         ...
158
159     #         self.actionOccupants_Details.triggered.connect(self.openOccupantsWindow)
160
            Cheong Kok Siong
161     def create_detected_occupants_table(self):
162         # Connect to the database
163         db = QSqlDatabase.addDatabase("QSQLITE")
164         db.setDatabaseName("Occupants.db")
165         if not db.open():
166             print("Failed to connect to the database.")
167             return
168
169         # Execute a query to create the "Detected_Occ" table
170         query = QSqlQuery()
171         query.prepare(
172             '''CREATE TABLE IF NOT EXISTS Detected_Occ(
173                     RecordID INTEGER PRIMARY KEY AUTOINCREMENT,
174                     OID INTEGER,
175                     Name VARCHAR(255),
176                     Contact CHAR(11),
177                     DetectedDT DATETIME,
178                     Photo BLOB,
179                     Zone CHAR(10),
180                     Status CHAR(5)
181                     )
```

Figure 4.2.3.5 main_cam1.py & main_cam2.py (5/20)

```
180                     Status CHAR(5)
181                     )
182                     ''')
183
184         if not query.exec_():
185             print("Failed to create the Detected_Occ table.")
186             db.close()
187             return
188
189         # Close the database connection
190         db.close()
191
192     ...
219
            Cheong Kok Siong
220     def update_status_in_occ_status(self, detected_name, new_status, zone):
221         try:
222             # Connect to the database
223             db = QSqlDatabase.addDatabase("QSQLITE")
224             db.setDatabaseName("Occupants.db")
225             if not db.open():
226                 print("Failed to connect to the database.")
227                 return
228
229             # Create a QSqlQuery object
230             query = QSqlQuery()
231
232             # Retrieve the current status before updating
233             query.prepare("SELECT Status FROM occ_status WHERE Name = ?")
```

Figure 4.2.3.6 main_cam1.py & main_cam2.py (6/20)

```
231
232             # Retrieve the current status before updating
233             query.prepare("SELECT Status FROM occ_status WHERE Name = ?")
234             query.addBindValue(detected_name)
235             if query.exec_() and query.next():
236                 current_status = query.value(0)
237             else:
238                 print(f"Failed to retrieve current status for '{detected_name}' from 'occ_status' table.")
239                 db.close()
240                 return
241
242             # Only update if the new status is different
243             if current_status != new_status:
244                 current_datetime = datetime.now().strftime("%m/%d/%Y %H:%M:%S")
245
246                 # Update 'occ_status' table
247                 query.prepare("UPDATE occ_status SET Status = ?, LastDeteced = ?, Zone = ? WHERE Name = ?")
248                 query.addBindValue(new_status)
249                 query.addBindValue(current_datetime)
250                 query.addBindValue(zone)
251                 query.addBindValue(detected_name)
252
253                 if query.exec_():
254                     print(f"Updated 'Status' for '{detected_name}' to '{new_status}' in 'occ_status' table.")
255                 else:
256                     print(f"Failed to update 'Status' for '{detected_name}' in 'occ_status' table.")
257
258                 # Insert into 'Detected_Trans' table
259                 query.prepare("INSERT INTO Detected_Trans (OID, Name, LastDetectedDT, Zone, Status) "
```

Figure 4.2.3.7 main_cam1.py & main_cam2.py (7/20)

```
259                 query.prepare("INSERT INTO Detected_Trans (OID, Name, LastDetectedDT, Zone, Status) "
260                               "SELECT OID, Name, LastDeteced, Zone, Status FROM occ_status WHERE Name = ?")
261                 query.addBindValue(detected_name)
262
263                 if query.exec_():
264                     print(f"Inserted into 'Detected_Trans' table: {detected_name}, {current_datetime}, {new_status}")
265                 else:
266                     print(f"Failed to insert into 'Detected_Trans' table for '{detected_name}'.")
267
268             else:
269                 print(f"No change in status for '{detected_name}'.")
270
271             # Close the database connection
272             db.close()
273
274         except Exception as e:
275             print("Error updating status:", e)
276
        Cheong Kok Siong
277     def count_occ_in_building(self):
278         try:
279             db = QSqlDatabase.addDatabase("QSQLITE")
280             db.setDatabaseName("Occupants.db")
281             if not db.open():
282                 print("Failed to connect to the database.")
283                 return
284
285             query = QSqlQuery()
```

Figure 4.2.3.8 main_cam1.py & main_cam2.py (8/20)

```
            query = QSqlQuery()
            if query.exec_("SELECT COUNT(*) FROM occ_status WHERE Status = 'IN'"):
                if query.next():
                    in_count = query.value(0)
                    text = f"Head Count: {in_count}"
                    self.ui.textBrowser_14.setPlainText(text)
                else:
                    print("Failed to retrieve count from 'occ_status' table.")
            else:
                print("Failed to execute query.")

            db.close()

        except Exception as e:
            print("Error:", e)

    Cheong Kok Siong
    def update_frame(self):
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
        ret, frame = self.cap.read()
        if ret:
            gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

            # Detect faces
            faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=4)

            face_locations, face_names = sfr.detect_known_faces(frame)
```

Figure 4.2.3.9 main_cam1.py & main_cam2.py (9/20)

```
            ...
            for (x, y, w, h), face_name in zip(face_locations, face_names):

                is_blacklisted = self.check_blacklist(face_name)

                for face_loc, name in zip(face_locations, face_names):
                    # top=y1 left=x1 bottom=y2 right=x2
                    y1, x1, y2, x2 = face_loc[0], face_loc[1], face_loc[2], face_loc[3]

                    cv2.putText(frame, name, (x1, y1 - 10), cv2.FONT_HERSHEY_DUPLEX, 1, (0, 0, 0), 2)
                    if is_blacklisted:
                        color = (0, 0, 200)

                    elif "vip" in name:
                        color = (200, 0, 0)
                    else:
                        color = (0, 200, 0)

                    cv2.rectangle(frame, (x1, y1), (x2, y2), color, 4)

                # Retrieve data from the database based on the recognized face name
                data = retrieve_data_from_database(face_name)

                # Display the retrieved data
                self.ui.textBrowser_7.setText(str(data.get("ID", "")))
                self.ui.textBrowser_4.setText(data.get("NAME"))
                #self.ui.textBrowser_7.setText("Contact: " + data.get("CONTACT", ""))
```

Figure 4.2.3.10 main_cam1.py & main_cam2.py (10/20)

```
344             self.ui.textBrowser_4.setText(data.get("NAME"))
345             #self.ui.textBrowser_7.setText("Contact: " + data.get("CONTACT", ""))
346             self.ui.textBrowser_6.setText(data.get("COMPANY"))
347
348             current_datetime = datetime.now().strftime("%m/%d/%Y %H:%M:%S")
349             self.ui.textBrowser_10.setText(current_datetime)
350
351             photo_data = data.get("PHOTO", None)
352
353             #add new record in Detected_Occ
354             db = QSqlDatabase.addDatabase("QSQLITE")
355             db.setDatabaseName("Occupants.db")
356             if not db.open():
357                 print("Failed to connect to database.")
358                 return
359
360             query = QSqlQuery()
361             query.prepare("INSERT INTO Detected_Occ (OID, Name, Contact, DetectedDT, Photo, Zone, Status) VALUES(?, ?, ?, ?, ?, ?, ?)")
362             query.addBindValue(data.get("ID", ""))
363             query.addBindValue(data.get("NAME"))
364             query.addBindValue(data.get("CONTACT", ""))
365             query.addBindValue(current_datetime)
366             query.addBindValue(data.get("PHOTO", None))
367             query.addBindValue(data.get("ZONE", "LV01_Z01"))
368             query.addBindValue(data.get("STATUS", "IN"))
369             new_zone = data.get("ZONE", "LV01_Z01")
370
371             oid = data.get("ID", "")
372             detected_dt = current_datetime
```

Figure 4.2.3.11 main_cam1.py & main_cam2.py (11/20)

```
371             oid = data.get("ID", "")
372             detected_dt = current_datetime
373             contact = data.get("CONTACT", "")
374             status = data.get("STATUS", "IN")
375
376             if not query.exec_():
377                 print("Failed to insert record into Detected_Occ:", query.lastError().text())
378
379             self.update_status_in_occ_status(name, "IN",new_zone)
380
381             if photo_data is not None:
382                 # Create a QImage from the photo data
383                 image = QImage.fromData(photo_data)
384
385                 if not image.isNull():
386                     # Scale the image to fit the label size
387                     scaled_image = image.scaled(self.photo_label.size(), Qt.AspectRatioMode.KeepAspectRatio,
388                                                 Qt.SmoothTransformation)
389                     # Create a QPixmap from the scaled image
390                     pixmap = QPixmap.fromImage(scaled_image)
391                     self.photo_label.setPixmap(pixmap)
392
393             # Draw a rectangle around the face
394             cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
395
396             datetime_obj = datetime.strptime(detected_dt, "%m/%d/%Y %H:%M:%S")
397             detected_dt_formatted = datetime_obj.strftime("%d-%m-%Y %H:%M:%S")
398
399             item_text = f"OID: {oid}\nName: {name}\nContact: {contact}\nDetected Time: {detected_dt_formatted}\nStatus: {status}\n"
```

Figure 4.2.3.12 main_cam1.py & main_cam2.py (12/20)

```
398
399             item_text = f"OID: {oid}\nName: {name}\nContact: {contact}\nDetected Time: {detected_dt_formatted}\nStatus: {status}\n"
400
401             item = QListWidgetItem(item_text)
402
403             is_blacklisted = self.check_blacklist(name)
404
405             if is_blacklisted:
406                 item.setBackground(QColor(255, 200, 200))
407             else:
408                 item.setBackground(QColor(255, 255, 255))
409
410             self.listWidget.insertItem(0, item)
411
412             db.close()
413
414             ...
425
426         # Convert the frame from BGR to RGB and display it
427         frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
428         image = QImage(frame_rgb.data, frame_rgb.shape[1], frame_rgb.shape[0], QImage.Format_RGB888)
429         pixmap = QPixmap.fromImage(image)
430         self.image_label.setPixmap(pixmap)
431
432         #self.retrieve_available_occ()
433         #self.retrieve_detected_occupants()
434         self.filter_list_widget()
435
436     # Close the database connection
```

Figure 4.2.3.13 main_cam1.py & main_cam2.py (13/20)

```
515
    Cheong Kok Siong
516 def update_datetime(self):
517     # Update current date in textBrowser_13
518     current_date = datetime.now().strftime("%d-%m-%Y")  # Changed format here
519     self.ui.textBrowser_13.setText(current_date)
520
521     # Update current time in textBrowser_12
522     current_time = datetime.now().strftime("%H:%M:%S")
523     self.ui.textBrowser_12.setText(current_time)
524
525     # Update current date and time in textBrowser_10
526     #current_datetime = datetime.now().strftime("%m/%d/%Y %H:%M:%S")
527
    Cheong Kok Siong
528 def openOccupantsWindow(self):
529     # Create the MainOccupantsWindow instance
530     self.occupantsWindow = MainOccupantsWindow()
531
532     # Hide the FVWindow
533     self.hide()
534
535     # Show the MainOccupantsWindow
536     self.occupantsWindow.show()
537
```

Figure 4.2.3.14 main_cam1.py & main_cam2.py (14/20)

```
537
                Cheong Kok Siong
538         def retrieve_detected_occupants(self):
539             # Connect to the database
540             db = QSqlDatabase.addDatabase("QSQLITE")
541             db.setDatabaseName("Occupants.db")
542             if not db.open():
543                 print("Failed to connect to the database.")
544                 return
545
546             # Execute a query to fetch records from Detected_Occ table
547             query = QSqlQuery()
548             query.prepare("SELECT * FROM Detected_Occ")
549             if not query.exec_():
550                 print("Failed to fetch records from Detected_Occ table.")
551                 db.close()
552                 return
553
554             # Clear the list widget
555             self.listWidget.clear()
556
557             # Populate the list widget with the retrieved data
558             while query.next():
559                 oid = query.value(1)
560                 name = query.value(2)
561                 contact = query.value(3)
562                 detected_dt = query.value(4)
563                 status = query.value(7)
```

Figure 4.2.3.15 main_cam1.py & main_cam2.py (15/20)

```
563             status = query.value(7)
564             item_OID = oid
565
566
567             datetime_obj = datetime.strptime(detected_dt, "%m/%d/%Y %H:%M:%S")
568             detected_dt_formatted = datetime_obj.strftime("%d-%m-%Y %H:%M:%S")
569
570             item_text = f"OID: {oid}\nName: {name}\nContact: {contact}\nDetected Time: {detected_dt_formatted}\nStatus: {status}\n"
571
572             item = QListWidgetItem(item_text)
573
574             self.listWidget.insertItem(0, item)
575
576         # Close the database connection
577         db.close()
578
        Cheong Kok Siong
579     def update_list_background(self):
580         for row in range(self.listWidget.count()):
581             item = self.listWidget.item(row)
582             item_text = item.text()
583
584             lines = item_text.split("\n")
585             name_line = next(line for line in lines if "Name:" in line)
586             name = name_line.split(":")[1].strip()
587
588             is_blacklisted = self.check_blacklist(name)
```

Figure 4.2.3.16 main_cam1.py & main_cam2.py (16/20)

```
590             if is_blacklisted:
591                 item.setBackground(QColor(255, 200, 200))
592             else:
593                 item.setBackground(QColor(255, 255, 255))
594
        Cheong Kok Siong
595     def click_list_item(self, item):
596         # Access the clicked item
597         clickedItem = item.text()
598
599         print(clickedItem)
600         match = re.search(r"\b\d+\b", clickedItem)
601         if match:
602             clicked_OID = match.group()
603             print(clicked_OID)
604
605         # If the detail window doesn't exist or has been closed, create a new instance
606         if not self.detail_window or not self.detail_window.isVisible():
607             self.detail_window = MainOccupants_DetailWindow()
608
609         # Set the clicked item name in the detail window
610         self.detail_window.set_clicked_item(clickedItem, clicked_OID)
611
612         # Show the detail window
613         self.detail_window.show()
614
        Cheong Kok Siong
```

Figure 4.2.3.17 main_cam1.py & main_cam2.py (17/20)

```
        Cheong Kok Siong
615     def filter_list_widget(self):
616         filter_text = self.ui.plainTextEdit.toPlainText().strip().lower()
617
618         for index in range(self.listWidget.count()):
619             item = self.listWidget.item(index)
620             item_text = item.text().lower()
621             if filter_text in item_text:
622                 item.setHidden(False)
623             else:
624                 item.setHidden(True)
625
        Cheong Kok Siong
626     def check_blacklist(self, facename):
627         db = QSqlDatabase.addDatabase("QSQLITE")
628         db.setDatabaseName("Occupants.db")
629
630         if not db.open():
631             print("Failed to connect to the database.")
632             return False
633
634         query = QSqlQuery()
635         query.prepare("SELECT COUNT(*) FROM Blacklist WHERE NAME = :name")
636         query.bindValue(":name", facename)
637
638         if not query.exec_():
639             print("Failed to check if occupant is in Blacklist.")
640             db.close()
```

Figure 4.2.3.18 main_cam1.py & main_cam2.py (18/20)

```
637
638         if not query.exec_():
639             print("Failed to check if occupant is in Blacklist.")
640             db.close()
641             return False
642
643         query.next()
644         count = query.value(0)
645
646         db.close()
647         return count > 0
648
        Cheong Kok Siong
649 def retrieve_data_from_database(face_name):
650     try:
651         # Connect to the database
652         conn = sqlite3.connect("Occupants.db")
653         c = conn.cursor()
654
655         # Execute a SELECT query to retrieve the data based on the face name
656         c.execute("SELECT * FROM Occupants WHERE Name = ?", (face_name,))
657         data = c.fetchone()
658
659         # Close the database connection
660         conn.close()
661
662         # Return the retrieved data as a dictionary
663         if data:
```

Figure 4.2.3.19 main_cam1.py & main_cam2.py (19/20)

```
663         if data:
664             return {
665                 "ID": data[0],
666                 "NAME": data[1],
667                 "CONTACT": data[2],
668                 "COMPANY": data[3],
669                 "DATE/TIME": data[4],
670                 "PHOTO":data[8]
671             }
672
673     except sqlite3.Error as e:
674         print("Error retrieving data from database:", e)
675
676     return {}
677
678 if __name__ == "__main__":
679     # Create the QApplication instance
680     app = QApplication(sys.argv)
681     # Create the FVWindow instance
682     fvWindow = FVWindow()
683     # Show the main window directly
684     fvWindow.show()
685     # Start the application event loop
686     sys.exit(app.exec_())
687
688
```

Figure 4.2.3.20 main_cam1.py & main_cam2.py (20/20)

The "main_cam1.py" and "main_cam2.py" files define the "FVWindow" class, crucial for system functionality. They import various modules such as Pyside2, sqlite3, cv2, re, sys, and predefined classes like "Facerec" from "main_faceRecognition" for loading encoding images.

The process begins with the "retrieve_data_from_database" function, fetching data using SQL commands. The "create_detected_occupants_table" function structures data storage for detected occupants, ensuring a database file is present. The heart of the system lies in the "update_frame" function, employing OpenCV's Haar Cascade Classifier for face detection and processing video frames. It detects faces, recognizes known occupants via the "detect_known_faces" function, updates the GUI with occupant information, inserts new records into the "Detected_Occ" table, and highlights blacklisted individuals in red.

"Update_status_in_occ_status" is responsible for updating the "occ_status" table with new status, last detection timestamp, and zone, also inserting corresponding records into "Detected_Trans." "Filter_list_widget" allows users to filter detected occupants based on keywords. "Click_list_item" manages interactions when users click on list items, displaying detailed occupant information. The "check_blacklist" function determines if an occupant is blacklisted and is used in "update_list_background," where blacklisted individuals are displayed in red. Both "main_cam1.py" and "main_cam2.py" classes and functions share similarities but serve different purposes. "main_cam1.py" acts as an entry camera, recording occupants' status as "IN," while "main_cam2.py" serves as an exit detector with recorded statuses as "OUT."

4.2.4 main_occupants.py

```python
import os
import numpy as np
from UI_occupant_record import Ui_MainWindow
from PySide2.QtWidgets import QMainWindow, QApplication, QTextEdit, QTableWidgetItem, QSizePolicy, QHeaderView, QLabel, QMessageBox, QFileDialog, QLineEdit
from PySide2.QtGui import QFont, QPixmap, QImage
from PySide2.QtCore import Qt, QTimer, QDate, QByteArray, QBuffer, QIODevice, QRect
from PySide2.QtSql import QSqlDatabase, QSqlQuery
from datetime import datetime

from main_faceRecogntion import Facerec
 Cheong Kok Siong *
class MainOccupantsWindow(QMainWindow, Ui_MainWindow):
    Cheong Kok Siong *
    def __init__(self):
        super(MainOccupantsWindow, self).__init__()

        self.setupUi(self)

        self.setFixedSize(self.size())  # Set fixed window size
        self.setupReadOnlyFields()  # Set OID and ip_dateacc as read-only
        self.setupFieldStyle()  # Set style for the dateEdit field
        self.setFieldFont()  # Set font size for QTextEdit fields

        self.photo_label = QLabel(self.frame)
        self.photo_label.setScaledContents(True)
        self.photo_label.setGeometry(QRect(160, 0, 330, 320))

        self.create_occupants_table()
```

Figure 4.2.4.1 main_occupants (1/19)

```python
        self.create_occupants_table()

        ...
        self.timer = QTimer()
        self.timer.timeout.connect(self.populateTable)
        self.timer.start(10000)  # Refresh every 10 seconds

        # Populate the table initially
        self.populateTable()
        self.tableWidget.cellClicked.connect(self.populateTextEdit)  # Connect cellClicked signal to populateTextEdit slot
        self.adjustTableSize()  # Adjust table widget size and column widths

        self.btn_refresh.clicked.connect(self.refreshWindow)
        self.btn_add.clicked.connect(self.addOccupant)
        self.btn_update.clicked.connect(self.updateOccupant)
        self.btn_dlt.clicked.connect(self.deleteOccupant)
        self.btn_refresh.clicked.connect(self.refreshWindow)
        self.upload_photo.clicked.connect(self.uploadButtonClicked)

        #self.btn_bck.clicked.connect(self.back_to_main())

        self.ip_photo = QLineEdit()  # Create the ip_photo attribute as a QLineEdit object

    Cheong Kok Siong
    def create_occupants_table(self):
        # Connect to the database
        db = QSqlDatabase.addDatabase("QSQLITE")
        db.setDatabaseName("Occupants.db")
        if not db.open():
            print("Failed to connect to the database.")
            return
```

Figure 4.2.4.2 main_occupants (2/19)

```
63          print("Failed to connect to the database.")
64          return
65
66      # Execute a query to insert the new occupant into the database
67      query = QSqlQuery()
68      query.prepare(
69          '''CREATE TABLE IF NOT EXISTS Occupants(
70                  OccupantsID INTEGER PRIMARY KEY AUTOINCREMENT,
71                  Name VARCHAR(255),
72                  Contact CHAR(11),
73                  DOB DATE,
74                  Gender VARCHAR(255),
75                  Company VARCHAR(255),
76                  Position VARCHAR(255),
77                  DateAccessed DATETIME,
78                  Photo BLOB
79                  )
80                  ''')
81
    Cheong Kok Siong
82  def populateTable(self):
83
84      # Connect to the database
85      db = QSqlDatabase.addDatabase("QSQLITE")
86      db.setDatabaseName("Occupants.db")
87      if not db.open():
88          print("Failed to connect to the database.")
89          return
```

Figure 4.2.4.3 main_occupants(3/19)

```
88              print("Failed to connect to the database.")
89              return
90
91          # Execute a query to fetch records
92          query = QSqlQuery()
93          query.prepare("SELECT * FROM Occupants")
94          if not query.exec_():
95              print("Failed to fetch records.")
96              db.close()
97              return
98
99          # Clear the table widget
100         ##self.tableWidget.clearContents()
101         self.tableWidget.setRowCount(0)
102
103         # Populate the table widget with the retrieved data
104         row = 0
105         # while query.next():
106         #     self.tableWidget.insertRow(row)
107         #     for column in range(query.record().count()):
108         #         item = QTableWidgetItem(str(query.value(column)))
109         #         item.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled)  # Set item flags
110         #         item.setTextAlignment(Qt.AlignCenter)
111         #         self.tableWidget.setItem(row, column, item)
112         #     row += 1
113
114         while query.next():
115             self.tableWidget.insertRow(row)
116             for column in range(query.record().count()):
117                 item = QTableWidgetItem(str(query.value(column)))
```

Figure 4.2.4.4 main_occupants(4/19)

```
117                 item = QTableWidgetItem(str(query.value(column)))
118                 if column == 0:  # Check if it's the OID column
119                     item.setFlags(item.flags() | Qt.ItemIsSelectable | Qt.ItemIsEnabled)
120                     item.setFlags(item.flags() & ~Qt.ItemIsEditable)
121                     item.setTextAlignment(Qt.AlignCenter)  # Set alignment to center
122                     item.setData(Qt.UserRole, query.value(column))  # Store the OID value in UserRol
123                 else:
124                     item.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled)  # Set item flags
125                     item.setTextAlignment(Qt.AlignCenter)  # Set alignment to center
126
127                 self.tableWidget.setItem(row, column, item)
128
129                 table_item = self.tableWidget.item(row, column)
130                 if table_item is not None:
131                     table_item.setFont(QFont("Arial", 12))  # Set font size to 12
132
133             row += 1
134
135         # Close the database connection
136         db.close()
137
    Cheong Kok Siong
138     def populateTextEdit(self, row, column):
139         # Get the value of the selected row's OID column
140         oid_value = self.tableWidget.item(row, 0).data(Qt.UserRole)
141
142         # Connect to the database
143         db = QSqlDatabase.addDatabase("QSQLITE")
144         db.setDatabaseName("Occupants.db")
145         if not db.open():
146             print("Failed to connect to the database.")
```

Figure 4.2.4.5 main_occupants(5/19)

```
146             print("Failed to connect to the database.")
147             return
148
149         # Execute a query to fetch the record based on the OID value
150         query = QSqlQuery()
151         query.prepare("SELECT * FROM Occupants WHERE OccupantsID = :oid")
152         query.bindValue(":oid", oid_value)
153         if not query.exec_():
154             print("Failed to fetch the record.")
155             db.close()
156             return
157
158         # Fetch the record
159         if query.next():
160             # Populate the input fields with the retrieved data
161             self.OID.setText(str(query.value(0)))
162             self.ip_name.setText(query.value(1))
163             self.ip_contact.setText(query.value(2))
164             #self.dateEdit.setDate(query.value(3))
165             self.ip_gender.setText(query.value(4))
166             self.ip_comp.setText(query.value(5))
167             self.ip_pos.setText(query.value(6))
168             self.ip_dateacc.setText(query.value(7))
169
170             # Retrieve the dob from the database
171             dob = query.value(3)
172
173             # Convert dob to QDate object
174             dob_date = QDate.fromString(dob, "yyyy-MM-dd")
175
176             # Format dob as "dd/mm/yyyy"
```

Figure 4.2.4.6 main_occupants(6/19)

```
174             dob_date = QDate.fromString(dob, "yyyy-MM-dd")
175
176             # Format dob as "dd/mm/yyyy"
177             dob_formatted = dob_date.toString("dd/MM/yyyy")
178
179             # Set dob in the dateEdit widget
180             self.dateEdit.setDate(dob_date)
181             self.dateEdit.setDisplayFormat("dd/MM/yyyy")
182
183             ...
200             photo_data = query.value(8)
201             #print(photo_data)
202             image = QImage.fromData(photo_data)
203
204             if not image.isNull():
205                 # Scale the image to fit the label size
206                 scaled_image = image.scaled(self.photo_label.size(), Qt.AspectRatioMode.KeepAspectRatio,
207                                             Qt.SmoothTransformation)
208                 # Create a QPixmap from the scaled image
209                 pixmap = QPixmap.fromImage(scaled_image)
210                 self.photo_label.setPixmap(pixmap)
211
212             ...
217         # Close the database connection
218         db.close()
219
            Cheong Kok Siong
220     def setupFieldStyle(self):
221         style = """
222                 QDateEdit {
```

Figure 4.2.4.7 main_occupants(7/19)

```
220     def setupFieldStyle(self):
221         style = """
222                 QDateEdit {
223                     background-color: #FFFFFF;  /* Set background color */
224                     border: 1px solid #CCCCCC;  /* Set border */
225                     padding: 8.5px;  /* Set padding */
226                     color: #000000;  /* Set text color */
227                 }
228                 """
229         self.dateEdit.setStyleSheet(style)  # Apply the style to dateEdit
230
        Cheong Kok Siong
231     def setupReadOnlyFields(self):
232         self.OID.setEnabled(False)  # Set OID field as read-only
233         self.ip_dateacc.setEnabled(False)  # Set ip_dateacc field as read-only
234         #self.dateEdit.setEnabled(False)  # Set dateEdit field as read-only
235
        Cheong Kok Siong
236     def setFieldFont(self):
237         font = QFont()
238         font.setPointSize(10)  # Set font size to 10
239
240         textEdits = self.findChildren(QTextEdit)  # Find all QTextEdit widgets
241
242         for textEdit in textEdits:
243             textEdit.setFont(font)  # Apply the font to each QTextEdit
244
        Cheong Kok Siong
245     def adjustTableSize(self):
246         # Set the table widget size to match the scroll area
```

Figure 4.2.4.8 main_occupants(8/19)

```
246         # Set the table widget size to match the scroll area
247         self.tableWidget.setSizeAdjustPolicy(self.tableWidget.AdjustToContents)
248         self.tableWidget.setSizePolicy(self.scrollAreaWidgetContents.sizePolicy())
249
250         # Expand the size of columns to fit the widget
251         header = self.tableWidget.horizontalHeader()
252         header.setSectionResizeMode(QHeaderView.Stretch)
253
254         header.setSectionResizeMode(0, QHeaderView.Fixed)
255         header.resizeSection(0, 110)
256
257         header.setSectionResizeMode(2, QHeaderView.Fixed)
258         header.resizeSection(2, 140)
259
260         header.setSectionResizeMode(3, QHeaderView.Fixed)
261         header.resizeSection(3, 130)
262
263         header.setSectionResizeMode(4, QHeaderView.Fixed)
264         header.resizeSection(4, 120)
265
266         header.setSectionResizeMode(5, QHeaderView.Fixed)
267         header.resizeSection(5, 180)
268
269         header.setSectionResizeMode(6, QHeaderView.Fixed)
270         header.resizeSection(6, 160)
271
272         header.setSectionResizeMode(7, QHeaderView.Fixed)
273         header.resizeSection(7, 220)
274
        Cheong Kok Siong
275     def refreshWindow(self):
```

Figure 4.2.4.9 main_occupants(9/19)

```
    Cheong Kok Siong
275 def refreshWindow(self):
276     # Clear the input fields
277     self.OID.clear()
278     self.ip_name.clear()
279     self.ip_contact.clear()
280     self.dateEdit.clear()
281     self.ip_gender.clear()
282     self.ip_comp.clear()
283     self.ip_pos.clear()
284     self.ip_dateacc.clear()
285
286     # Clear the photo frame
287     self.clearPhotoFrame()
288
289     # Refresh the table
290     self.populateTable()
291
292     # Adjust the table size
293     self.adjustTableSize()
294
    Cheong Kok Siong
295 def clearPhotoFrame(self):
296     # Clear the photo label from the frame
297     self.photo_label.clear()
298
    Cheong Kok Siong *
299 def addOccupant(self):
300     sfr = Facerec()
301     name = self.ip_name.toPlainText()
302     contact = self.ip_contact.toPlainText()
```

Figure 4.2.4.10 main_occupants(10/19)

```
299     def addOccupant(self):
300         sfr = Facerec()
301         name = self.ip_name.toPlainText()
302         contact = self.ip_contact.toPlainText()
303         gender = self.ip_gender.toPlainText()
304         company = self.ip_comp.toPlainText()
305         position = self.ip_pos.toPlainText()
306 
307         # Get the current date and time
308         current_datetime = datetime.now()
309 
310         # Convert the current date and time to the desired format
311         date_acc = current_datetime.strftime("%Y-%m-%d %H:%M:%S")
312 
313         # Convert the date of birth to the desired format
314         dob = self.dateEdit.date().toString("yyyy-MM-dd")
315 
316         # Get the selected photo path
317         photo_path = self.ip_photo.text()
318 
319         # Load the photo from the selected file
320         image = QImage(photo_path)
321         photo_data = QByteArray()
322         # buffer = QBuffer(photo_data)
323         # buffer.open(QIODevice.WriteOnly)
324         # image.save(buffer, "PNG")
325 
326         image_directory = "images"
327         if not os.path.exists(image_directory):
328             os.makedirs(image_directory)
329 
```

Figure 4.2.4.11 main_occupants(11/19)

```
329
330
331         image_name = name
332         image_filename = f"{image_name}.jpg"
333
334         image_path = os.path.join(image_directory, image_filename)
335         image.save(image_path)
336
337
338         buffer = QBuffer(photo_data)
339         buffer.open(QIODevice.WriteOnly)
340         image.save(buffer, "PNG")
341
342         ...
348         sfr.load_encoding_images("images/")
349
350         # Connect to the database
351         db = QSqlDatabase.addDatabase("QSQLITE")
352         db.setDatabaseName("Occupants.db")
353         if not db.open():
354             print("Failed to connect to the database.")
355             return
356
357         # Execute a query to insert the new occupant into the database
358         query = QSqlQuery()
359         query.prepare(
360             "INSERT INTO Occupants (Name, Contact, DOB, Gender, Company, Position, DateAccessed, Photo) VALUES (?, ?, ?, ?, ?, ?, ?, ?)")
361         query.addBindValue(name)
362         query.addBindValue(contact)
363         query.addBindValue(dob)
```

Figure 4.2.4.12 main_occupants(12/19)

```
362         query.addBindValue(contact)
363         query.addBindValue(dob)
364         query.addBindValue(gender)
365         query.addBindValue(company)
366         query.addBindValue(position)
367         query.addBindValue(date_acc)
368         query.addBindValue(photo_data)
369
370         if not query.exec_():
371             error_message = "Failed to add occupant:\n" + query.lastError().text()
372             QMessageBox.critical(self, "Error", error_message)
373             db.close()
374             return
375
376         occupant_id = query.lastInsertId()
377
378         status_query = QSqlQuery()
379         status_query.prepare(
380             "INSERT INTO occ_status (OccupantsID, Name, LastDeteced, Status, Zone) VALUES (?, ?, ?, ?, ?)")
381         status_query.addBindValue(occupant_id)
382         status_query.addBindValue(name)
383         status_query.addBindValue("")
384         status_query.addBindValue("")
385         status_query.addBindValue("")
386
387         if not status_query.exec_():
388             error_message = "Failed to insert status into occ_status table:\n" + status_query.lastError().text()
389             QMessageBox.critical(self, "Error", error_message)
390             db.close()
391             return
```

Figure 4.2.4.13 main_occupants(13/19)

```
391             return
392
393         # Close the database connection
394         db.close()
395
396         # Clear the input fields after adding the occupant
397         self.ip_name.clear()
398         self.ip_contact.clear()
399         self.ip_gender.clear()
400         self.ip_comp.clear()
401         self.ip_pos.clear()
402
403         # Refresh the window to update the table and other components
404         self.refreshWindow()
405
    Cheong Kok Siong
406     def qimage_to_ndarray(self,qimage):
407         width = qimage.width()
408         height = qimage.height()
409         buffer = qimage.bits().ascontiguousbytes()
410         image = np.frombuffer(buffer, dtype=np.uint8).reshape((height, width, 4))
411         return image
412
    Cheong Kok Siong
413     def uploadButtonClicked(self):
414         # Open a file dialog to select the photo file
415         file_dialog = QFileDialog()
416         file_dialog.setFileMode(QFileDialog.ExistingFile)
417         file_dialog.setNameFilter("Images (*.png *.jpg *.jpeg *.bmp)")
418
419         if file_dialog.exec_():
```

Figure 4.2.4.14 main_occupants(14/19)

```
419         if file_dialog.exec_():
420             selected_files = file_dialog.selectedFiles()
421             if selected_files:
422                 photo_path = selected_files[0]
423
424                 # Load the photo from the selected file
425                 image = QImage(photo_path)
426
427                 photo_data = QByteArray()
428                 buffer = QBuffer(photo_data)
429                 buffer.open(QIODevice.WriteOnly)
430                 image.save(buffer, "PNG")
431
432                 if not image.isNull():
433                     # Scale the image to fit the label size
434                     scaled_image = image.scaled(self.photo_label.size(), Qt.AspectRatioMode.KeepAspectRatio,
435                                                 Qt.SmoothTransformation)
436                     # Create a QPixmap from the scaled image
437                     pixmap = QPixmap.fromImage(scaled_image)
438                     self.photo_label.setPixmap(pixmap)
439
440                 # Update the photo path in the text field
441                 self.ip_photo.setText(photo_path)
442
    Cheong Kok Siong
443     def updateOccupant(self):
444         # Get the values from the input fields
445         oid = self.OID.toPlainText()
446         name = self.ip_name.toPlainText()
447         contact = self.ip_contact.toPlainText()
448         gender = self.ip_gender.toPlainText()
```

Figure 4.2.4.15 main_occupants(15/19)

```
445         oid = self.OID.toPlainText()
446         name = self.ip_name.toPlainText()
447         contact = self.ip_contact.toPlainText()
448         gender = self.ip_gender.toPlainText()
449         company = self.ip_comp.toPlainText()
450         position = self.ip_pos.toPlainText()
451 
452         # Convert the date of birth to the desired format
453         dob = self.dateEdit.date().toString("yyyy-MM-dd")
454 
455 
456         # Get the selected photo path
457         photo_path = self.ip_photo.text()
458 
459         # Load the photo from the selected file
460         image = QImage(photo_path)
461         photo_data = QByteArray()
462         buffer = QBuffer(photo_data)
463         buffer.open(QIODevice.WriteOnly)
464         image.save(buffer, "PNG")
465 
466         # Connect to the database
467         db = QSqlDatabase.addDatabase("QSQLITE")
468         db.setDatabaseName("Occupants.db")
469         if not db.open():
470             print("Failed to connect to the database.")
471             return
472 
473         # Execute a query to update the existing occupant in the database
474         query = QSqlQuery()
475         query.prepare(
```

Figure 4.2.4.16 main_occupants(16/19)

```
472 
473     # Execute a query to update the existing occupant in the database
474     query = QSqlQuery()
475     query.prepare(
476         "UPDATE Occupants SET Name = ?, Contact = ?, DOB = ?, Gender = ?, Company = ?, Position = ?, Photo = ? WHERE OccupantsID =
477     query.addBindValue(name)
478     query.addBindValue(contact)
479     query.addBindValue(dob)
480     query.addBindValue(gender)
481     query.addBindValue(company)
482     query.addBindValue(position)
483     query.addBindValue(photo_data)
484     query.addBindValue(oid)
485 
486     if not query.exec_():
487         error_message = "Failed to update occupant:\n" + query.lastError().text()
488         QMessageBox.critical(self, "Error", error_message)
489         db.close()
490         return
491 
492     status_query = QSqlQuery()
493     status_query.prepare(
494         "UPDATE occ_status SET Name = ? WHERE OccupantsID = ?")
495     status_query.addBindValue(name)
496     status_query.addBindValue(oid)
497 
498     if not status_query.exec_():
499         error_message = "Failed to update name in occ_status table:\n" + status_query.lastError().text()
500         QMessageBox.critical(self, "Error", error_message)
501         db.close()
502         return
```

Figure 4.2.4.17 main_occupants(17/19)

```
502             return
503         # Close the database connection
504         db.close()
505
506         # Refresh the window to update the table and other components
507         self.refreshWindow()
508
    Cheong Kok Siong
509     def deleteOccupant(self):
510         # Get the value of the selected row's OID column
511         current_row = self.tableWidget.currentRow()
512         if current_row < 0:
513             QMessageBox.warning(self, "Delete Error", "Please select a record to delete.")
514             return
515
516         oid_value = self.tableWidget.item(current_row, 0).data(Qt.UserRole)
517
518         # Connect to the database
519         db = QSqlDatabase.addDatabase("QSQLITE")
520         db.setDatabaseName("Occupants.db")
521         if not db.open():
522             print("Failed to connect to the database.")
523             return
524
525         # Execute a query to delete the record based on the OID value
526         query = QSqlQuery()
527         query.prepare("DELETE FROM Occupants WHERE OccupantsID = :oid")
528         query.bindValue(":oid", oid_value)
529
530         if not query.exec_():
531             error_message = "Failed to delete the occupant:\n" + query.lastError().text()
```

Figure 4.2.4.18 main_occupants(18/19)

```
530         if not query.exec_():
531             error_message = "Failed to delete the occupant:\n" + query.lastError().text()
532             QMessageBox.critical(self, "Error", error_message)
533             db.close()
534             return
535
536         # Close the database connection
537         db.close()
538
539         # Refresh the window to update the table and other components
540         self.refreshWindow()
541
542         # Clear the input fields after a successful deletion
543         self.refreshWindow()
544
545 if __name__ == "__main__":
546     app = QApplication([])
547     window = MainOccupantsWindow()
548     window.show()
549     app.exec_()
```

Figure 4.2.4.19 main_occupants(19/19)

This python code defines a GUI application using the PySide2 framework for managing occupant's records. The "MainOccupantWindow" class extends QMainWindow and interfaces with a user-friendly GUI created with PySide2 elements. It is importing various python predefined modules which are PySide2 widgets, Pyside2 GUI elements, PyQt5 database and SQLite database interactions. Within the class, there are many functions defined for tasks such as populating tables with occupant data, adding, updating, and deleting record, refreshing the GUI and handling image uploads. Those functions are "populateTable", "populateTextEdit", "refreshWindow", "clearPhotoFrame", "addOccupant", "qimage_to_ndarray", "uploadButtonClicked", "updateOccupant", "deleteOccupant". Most of the functions are required to connect to database for retrieving data from targeted table by using SQL query.

The "addOccupant" function in the GUI is responsible for capturing various occupant details such as name, contact information, gender, company, and more. Once these details are collected, they are inserted into the database, and the GUI is updated to reflect the new occupant's information. Additionally, the occupant's photo is saved in a predefined path with the occupant's name as the filename. The photo is also converted into a byte format and stored in the database for future reference.

The "populateTextEdit" function is a crucial component of the GUI for managing occupant records. When a cell in the table is clicked, the system retrieves and displays detailed occupant information. It begins by obtaining the Occupant ID (OID) from the selected row, which serves as a unique identifier, and uses it to fetch the occupant's details. Additionally, the function converts the Date of Birth (DOB) to the "dd/MM/yyyy" format and presents it within a date-edit widget. Notably, the occupant's photo, stored as binary data, is retrieved and converted to an image using QImage. It is then displayed, with the image automatically scaled to fit a label while preserving its original aspect ratio. This function enhances the user experience when viewing occupant records.

4.2.5 main_building_record.py

```python
import sys
from PySide2.QtGui import QFont
from PySide2.QtCore import Qt, QDateTime, QTimer, QDate
from PySide2.QtSql import QSqlDatabase, QSqlQuery
from PySide2.QtWidgets import QApplication, QMainWindow, QFormLayout, QWidget, QHBoxLayout, QTableWidgetItem, \
    QHeaderView, QTableWidget, QStyledItemDelegate
from UI_building_record import Ui_BuildingrecordWindow
from main_occupant_detail import MainOccupants_DetailWindow as MOD
from main_building_summarized import UI_BuildingSum_Window as SUM

xml_ui = '''...'''

Cheong Kok Siong *
class BuildingRecord_Window(QMainWindow, Ui_BuildingrecordWindow):
    Cheong Kok Siong *
    def __init__(self):
        super(BuildingRecord_Window, self).__init__()

        self.setupUi(self)

        self.setFixedSize(self.size())
        self.adjustTableSize()
        self.tableWidget.horizontalHeader().setStretchLastSection(True)
        self.tableWidget.setEditTriggers(QTableWidget.NoEditTriggers)

        default_date = QDate()
        self.date_edit.setDate(default_date)
```

Figure 4.2.5.1 main_building_record.py (1/13)

```python
        self.formLayout.setRowWrapPolicy(QFormLayout.WrapLongRows)
        item_height = 35
        for i in range(self.formLayout.rowCount()):
            layout_item = self.formLayout.itemAt(i, QFormLayout.FieldRole)
            if layout_item and layout_item.widget():
                layout_item.widget().setFixedHeight(item_height)

        self.load_data_from_database()

        ...

        self.btn_filter.clicked.connect(self.apply_filters)
        self.btn_searchID.clicked.connect(self.open_occupant_detail_window)
        self.btn_summarize.clicked.connect(self.open_building_summarized_window)

        ...

    Cheong Kok Siong
    def load_data_from_database(self):
        # Establish the connection to the database
        db = QSqlDatabase.addDatabase("QSQLITE")
        db.setDatabaseName("Occupants.db")
        if not db.open():
            print("Error: Failed to connect to the database.")
            return

        # Retrieve data from Detected_Occ and Occupants tables
        query = QSqlQuery()
        query.prepare('''SELECT Detected_Occ.OID, Detected_Occ.DetectedDT, Detected_Occ.Zone, Detected_Occ.Status, Occupants.Name, Occupants.Company
                         FROM Detected_Occ
```

Figure 4.2.5.2 main_building_record.py (2/13)

```
                    FROM Detected_Occ
                    LEFT JOIN Occupants ON Detected_Occ.OID = Occupants.OID''')

        if not query.exec_():
            print("Error: Failed to execute the query.")
            db.close()
            return

        # Clear the existing table contents
        self.tableWidget.setRowCount(0)
        self.tableWidget.setColumnCount(6)

        # Populate the table with data
        row_index = 0
        while query.next():
            oid = query.value(0)
            detected_dt = query.value(1)
            name = query.value(4)
            company = query.value(5)
            zone = query.value(2)
            status = query.value(3)

            detected_dt = QDateTime.fromString(detected_dt, "MM/dd/yyyy HH:mm:ss")
            detected_dt_formatted = detected_dt.toString("dd-MM-yyyy HH:mm:ss")

            ...
            item = QTableWidgetItem(str(oid))
            self.tableWidget.insertRow(row_index)
```

Figure 4.2.5.3 main_building_record.py (3/13)

```
            item = QTableWidgetItem(str(oid))
            self.tableWidget.insertRow(row_index)
            self.tableWidget.setItem(row_index, 0, item)

            self.tableWidget.setItem(row_index, 1, QTableWidgetItem(str(name)))
            self.tableWidget.setItem(row_index, 2, QTableWidgetItem(str(company)))
            self.tableWidget.setItem(row_index, 3, QTableWidgetItem(str(detected_dt_formatted)))
            self.tableWidget.setItem(row_index, 4, QTableWidgetItem(str(zone)))
            self.tableWidget.setItem(row_index, 5, QTableWidgetItem(str(status)))
            row_index += 1

        # Close the database connection
        db.close()

    ...

    Cheong Kok Siong
    def adjustTableSize(self):
        # Set the column width to expand to fit the contents
        self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)

        # Set fixed sizes for specific columns
        header = self.tableWidget.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.Fixed)
        header.resizeSection(0, 150)

        header.setSectionResizeMode(1, QHeaderView.Fixed)
        header.resizeSection(1, 300)
```

Figure 4.2.5.4 main_building_record.py (4/13)

```
        header.setSectionResizeMode(2, QHeaderView.Fixed)
        header.resizeSection(2, 170)

        header.setSectionResizeMode(3, QHeaderView.Fixed)
        header.resizeSection(3, 280)

        header.setSectionResizeMode(4, QHeaderView.Fixed)
        header.resizeSection(4, 190)

        header.setSectionResizeMode(5, QHeaderView.Fixed)
        header.resizeSection(5, 190)

        # font = QFont("Arial", 15)
        # Set the alignment of all cells to center
        for row in range(self.tableWidget.rowCount()):
            for col in range(self.tableWidget.columnCount()):
                item = self.tableWidget.item(row, col)
                if item is not None:
                    item.setTextAlignment(Qt.AlignCenter)

        delegate = FontSizeDelegate(self.tableWidget)
        self.tableWidget.setItemDelegate(delegate)

        font = QFont("Arial", 12)  # Adjust the font size as needed
        self.tableWidget.horizontalHeader().setFont(font)
        self.tableWidget.horizontalHeader().setDefaultAlignment(Qt.AlignCenter)

```

Figure 4.2.5.5 main_building_record.py (5/13)

```
def sort_table(self,sorting_option):

    # Sort the data in the table based on the selected option
    if sorting_option == "Occupant ID":
        self.sort_table_by_occupant_id()
        #self.tableWidget.sortItems(0, Qt.AscendingOrder)
    elif sorting_option == "Name":
        self.tableWidget.sortItems(1, Qt.AscendingOrder)
    elif sorting_option == "Company":
        self.tableWidget.sortItems(2, Qt.AscendingOrder)
    elif sorting_option == "Date Time":
        self.tableWidget.sortItems(3, Qt.AscendingOrder)
    elif sorting_option == "Entrance":
        self.tableWidget.sortItems(4, Qt.AscendingOrder)
    elif sorting_option == "Status":
        self.tableWidget.sortItems(5, Qt.AscendingOrder)

 Cheong Kok Siong
def sort_table_by_occupant_id(self):
    rows = []
    for row in range(self.tableWidget.rowCount()):
        item = self.tableWidget.item(row, 0)
        oid_text = item.text()
        numeric_oid = int(oid_text) if oid_text.isdigit() else 0
        rows.append((numeric_oid, [self.tableWidget.item(row, col).text() for col in range(self.tableWidget.columnCount())]))

    rows.sort(key=lambda x: x[0])
    for row, (value, row_data) in enumerate(rows):
        for col, data in enumerate(row_data):
```

Figure 4.2.5.6 main_building_record.py (6/13)

```
227             for col, data in enumerate(row_data):
228                 self.tableWidget.setItem(row, col, QTableWidgetItem(data))
229
230         ...
235
            Cheong Kok Siong *
236     def filter_table_by_floor(self):
237         selected_floor = self.combo_floor.currentText().strip()
238
239         # Check if the selected entrance is empty
240         if not selected_floor:
241             # If empty, show all rows in the table and return
242             for row in range(self.tableWidget.rowCount()):
243                 self.tableWidget.showRow(row)
244             return
245
246         # Create a mapping from entrance names to their corresponding codes
247         entrance_mapping = {
248             "Floor 1": "LV01",
249             "Floor 2": "LV02",
250             "Floor 3": "LV03",
251             "Floor 4": "LV04",
252         }
253         # Reload the data from the database
254         self.load_data_from_database()
255
256         # Get the corresponding code from the mapping
257         floor_code = entrance_mapping.get(selected_floor)
```

Figure 4.2.5.7 main_building_record.py (7/13)

```
259         # Apply the filter and populate the table with filtered data
260         for row in range(self.tableWidget.rowCount()):
261             entrance_item = self.tableWidget.item(row, 4)
262
263             if floor_code and floor_code.lower() not in entrance_item.text().lower():
264                 self.tableWidget.hideRow(row)
265             else:
266                 self.tableWidget.showRow(row)
267
            Cheong Kok Siong *
268     def filter_table_by_entrance(self):
269         selected_entrance = self.combo_entrance.currentText().strip()
270
271         # Check if the selected entrance is empty
272         if not selected_entrance:
273             # If empty, show all rows in the table and return
274             for row in range(self.tableWidget.rowCount()):
275                 self.tableWidget.showRow(row)
276             return
277
278         # Create a mapping from entrance names to their corresponding codes
279         entrance_mapping = {
280             "Zone 1": "Z01",
281             "Zone 2": "Z02",
282             "Zone 3": "Z03",
283             "Zone 4": "Z04",
284         }
285
286         # Reload the data from the database
```

Figure 4.2.5.8 main_building_record.py (8/13)

```
287         self.load_data_from_database()
288
289         # Get the corresponding code from the mapping
290         entrance_code = entrance_mapping.get(selected_entrance)
291
292         # Apply the filter and populate the table with filtered data
293         for row in range(self.tableWidget.rowCount()):
294             entrance_item = self.tableWidget.item(row, 4)
295
296             if entrance_code and entrance_code.lower() not in entrance_item.text().lower():
297                 self.tableWidget.hideRow(row)
298             else:
299                 self.tableWidget.showRow(row)
300
    Cheong Kok Siong
301     def filter_table_by_date(self):
302         selected_date = self.date_edit.date().toPython()
303         formatted_date = selected_date.strftime("%d-%m-%Y")
304         print("now: " + str(formatted_date))
305
306         # Reload the data from the database
307         self.load_data_from_database()
308
309         if formatted_date is None:
310             # If no date is selected, show all rows in the table and return
311             for row in range(self.tableWidget.rowCount()):
312                 self.tableWidget.showRow(row)
313             return
314
315         # Apply the filter and populate the table with filtered data
```

Figure 4.2.5.9 main_building_record.py (9/13)

```
316         for row in range(self.tableWidget.rowCount()):
317             item_date_str = self.tableWidget.item(row, 3).text()
318
319             if formatted_date in item_date_str:
320                 self.tableWidget.showRow(row)
321             else:
322                 self.tableWidget.hideRow(row)
323
    Cheong Kok Siong
324     def filter_table_by_status(self):
325         selected_status = self.combo_status.currentText()
326
327         # Reload the data from the database
328         self.load_data_from_database()
329
330         if selected_status.strip() == "":
331             # If empty, show all rows in the table and return
332             for row in range(self.tableWidget.rowCount()):
333                 self.tableWidget.showRow(row)
334             return
335
336         # Reload the data from the database
337         self.load_data_from_database()
338
339         # Apply the filter and populate the table with filtered data
340         for row in range(self.tableWidget.rowCount()):
341             status_item = self.tableWidget.item(row, 5)
342             if status_item.text() != selected_status:
343                 self.tableWidget.hideRow(row)
344             else:
```

Figure 4.2.5.10 main_building_record.py (10/13)

```
Cheong Kok Siong *
347    def apply_filters(self):
348        ...
353        self.load_data_from_database()
354
355        selected_entrance = self.combo_entrance.currentText().strip()
356        selected_status = self.combo_status.currentText().strip()
357        selected_date = self.date_edit.date().toPython()
358        formatted_date = selected_date.strftime("%d-%m-%Y")
359        selected_floor = self.combo_floor.currentText().strip()
360        selected_sort = self.combo_sort.currentText().strip()
361
362        # Reload the data from the database
363        self.load_data_from_database()
364
365        for row in range(self.tableWidget.rowCount()):
366            item_entrance = self.tableWidget.item(row, 4)
367            item_status = self.tableWidget.item(row, 5)
368            item_date_str = self.tableWidget.item(row, 3).text()
369            item_level = self.tableWidget.item(row, 2)
370            sorting_option = self.combo_sort.currentText()
371
372            # Filter by entrance
373            if selected_entrance and selected_entrance != " ":
374                ent = "Z0" + selected_entrance[-1]
375                if ent not in item_entrance.text():
376                    self.tableWidget.hideRow(row)
377                    continue
```

Figure 4.2.5.11 main_building_record.py (11/13)

```
379             # Filter by status
380             if selected_status and selected_status != " ":
381                 if selected_status != item_status.text():
382                     self.tableWidget.hideRow(row)
383                     continue
384
385             # Filter by date
386             if selected_date:
387                 if formatted_date not in item_date_str:
388                     self.tableWidget.hideRow(row)
389                     continue
390
391             # Filter by level
392             if selected_floor and selected_floor != " ":
393                 entrance_mapping = {
394                     "Floor 1": "LV01",
395                     "Floor 2": "LV02",
396                     "Floor 3": "LV03",
397                     "Floor 4": "LV04",
398                 }
399                 floor_code = entrance_mapping.get(selected_floor)
400                 if floor_code and floor_code.lower() not in item_entrance.text().lower():
401                     self.tableWidget.hideRow(row)
402                     continue
403
404             self.tableWidget.showRow(row)
405
406         # Sort the table by occupant_id (assuming it is in column 0)
407         self.sort_table(sorting_option)
```

Figure 4.2.5.12 main_building_record.py (12/13)

```
                Cheong Kok Siong
409     def open_occupant_detail_window(self):
410         self.occupant_detail_window = MOD()
411         self.occupant_detail_window.show()
412
                Cheong Kok Siong
413     def open_building_summarized_window(self):
414         self.building_summarized_window = SUM()
415         self.building_summarized_window.show()
            Cheong Kok Siong
416 class FontSizeDelegate(QStyledItemDelegate):
                Cheong Kok Siong
417     def __init__(self, parent=None):
418         super(FontSizeDelegate, self).__init__(parent)
419         self.font = QFont("Arial", 12)
420
                Cheong Kok Siong
421     def paint(self, painter, option, index):
422         option.font = self.font
423         option.displayAlignment = Qt.AlignCenter  # Align content to center
424         super(FontSizeDelegate, self).paint(painter, option, index)
425
426 if __name__ == "__main__":
427     app = QApplication(sys.argv)
428     window = BuildingRecord_Window()
429     window.show()
430     sys.exit(app.exec_())
431
```

Figure 4.2.5.13 main_building_record.py (13/13)

"main_building_record.py" is define a GUI application using the PySide2 library which imported "QtGui", "Qtcore", "QtSql", "sys". Some functions are imported from pre-defined classes like import "MainOccupants_DetailWindow" from "main_occupant_detail.py" and "UI_BuilidngSum_Window" from "main_building_summarized.py". The application primarily focuses on loading data from a database by executing SQL queries that join the "Detected_Occ" and "Occupants" tables based on matching OID values and retrieve them to display in table or for further processing. For ease of management, it creates a window with various UI element like buttons and combo boxes to manager those displayed building detected record in table. The code also include functionality to open additional windows for detailed occupant information which will only shown the target occupants information and building summarization. Additionally, a custom item delegate "FontSizeDelegate" is defined to adjust the font size and alignment of table cells, enhancing the user experience.

4.2.6 main_occupant_detail.py

```
from PySide2.QtCore import Qt, QDateTime, QRect
from PySide2.QtWidgets import QMainWindow, QApplication, QMessageBox, QTableWidgetItem, QHeaderView, QLabel, QTextEdit
from PySide2.QtSql import QSqlDatabase, QSqlQuery
from PySide2.QtGui import QFont, QPixmap, QImage
from UI_occupant_detail import Ui_MainWindow
from datetime import datetime

 Cheong Kok Siong *
class MainOccupants_DetailWindow(QMainWindow, Ui_MainWindow):
    Cheong Kok Siong *
    def __init__(self):
        super(MainOccupants_DetailWindow, self).__init__()

        self.setupUi(self)
        self.setFieldFont()

        self.photo_label = QLabel(self.frame)
        self.photo_label.setScaledContents(True)
        self.photo_label.setGeometry(QRect(0, 0, 240, 260))

        font = QFont()
        font.setPointSize(12)
        self.blacklist.setFont(font)

        self.btn_srch.clicked.connect(self.search_btn_clicked)
        self.btn_flt.clicked.connect(self.filter_table)
        self.blacklist.clicked.connect(self.checkboxClicked)

        self.setFixedSize(self.size())  # Set window size as fixed
        self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)  # Resize columns to fit content
```

Figure 4.2.6.1 main_occupant_detail.py (1/17)

```
29
30          self.tableWidget.setSortingEnabled(True)
31          self.tableWidget.horizontalHeader().sectionClicked.connect(self.sort_table)
32          self.tableWidget.setEditTriggers(QHeaderView.NoEditTriggers)
33
34          font = QFont()
35          font.setPointSize(12)
36          self.tableWidget.setFont(font)
37
            Cheong Kok Siong
38      def set_clicked_item(self, item_name, clicked_OID):
39          occupants_data, detected_data = self.retrieve_data_from_tables(clicked_OID)
40
41          if occupants_data is not None and detected_data is not None:
42              self.display_data(occupants_data, detected_data)
            Cheong Kok Siong
43      def search_btn_clicked(self):
44          name = self.input_NID.toPlainText()
45          print(name)
46
47          if not name:
48              # Handle the case when the name is empty
49              QMessageBox.information(None, "Invalid Input", "Please enter an available name or OID.")
50              return
51
52          # Clear the table contents
53          self.tableWidget.clearContents()
54          self.tableWidget.setRowCount(0)
55
56          occupants_data, detected_data = self.retrieve_data_from_tables(name)
```

Figure 4.2.6.2 main_occupant_detail.py (2/17)

```
57          if occupants_data is not None and detected_data is not None:
58              self.display_data(occupants_data, detected_data)
59
60              if self.check_blacklist(occupants_data[0]["OID"]):
61                  self.blacklist.setChecked(True)
62              else:
63                  self.blacklist.setChecked(False)
64
65          else:
66              # Handle the case when no data is found
67              QMessageBox.information(None, "Data Not Found", "No data found for the entered name or OID.")
68              self.clear_displayed_data()
69
            Cheong Kok Siong
70      def clear_displayed_data(self):
71          # Clear the displayed data when no data is found
72          self.id_in.clear()
73          self.name_in.clear()
74          self.gender_in.clear()
75          self.comp_in.clear()
76          self.pos_in.clear()
77          self.datereg_in.clear()
78          self.photo_label.clear()
79          self.tableWidget.clearContents()
80          self.tableWidget.setRowCount(0)
81
            Cheong Kok Siong
82      def setFieldFont(self):
83          font = QFont()
84          font.setPointSize(10)  # Set font size to 10
```

Figure 4.2.6.3 main_occupant_detail.py (3/17)

```
85
86          textEdits = self.findChildren(QTextEdit)  # Find all QTextEdit widgets
87
88          for textEdit in textEdits:
89              textEdit.setFont(font)  # Apply the font to each QTextEdit
90
91      ...
            Cheong Kok Siong
113     def retrieve_data_from_tables(self, name_or_oid):
114         # Connect to the database
115         db = QSqlDatabase.addDatabase("QSQLITE")
116         db.setDatabaseName("Occupants.db")
117         if not db.open():
118             print("Failed to connect to the database.")
119             return
120
121         # Prepare the query to retrieve data from the Occupants table
122         occupants_query = QSqlQuery()
123         #occupants_query.prepare("SELECT * FROM Occupants WHERE LOWER(Name) = :name OR LOWER(OccupantsID) = :oid")
124         occupants_query.prepare("SELECT * FROM Occupants WHERE Name = :name OR OccupantsID = :oid")
125         occupants_query.bindValue(":name", name_or_oid)
126         occupants_query.bindValue(":oid", name_or_oid)
127
128         if not occupants_query.exec_():
129             print("Failed to execute query for Occupants table.")
130             db.close()
131             return
132
133         occupants_data = []
134         while occupants_query.next():
135             # Retrieve the data from the Occupants table
```

Figure 4.2.6.4 main_occupant_detail.py (4/17)

```
132
133         occupants_data = []
134         while occupants_query.next():
135             # Retrieve the data from the Occupants table
136             oid = occupants_query.value("OccupantsID")
137             name = occupants_query.value("Name")
138             contact = occupants_query.value("Contact")
139             dob = occupants_query.value("DOB")
140             gender = occupants_query.value("Gender")
141             company = occupants_query.value("Company")
142             position = occupants_query.value("Position")
143             date_accessed = occupants_query.value("DateAccessed")
144             photo = occupants_query.value("Photo")
145             print(photo)
146
147             occupants_data.append({
148                 "OID": oid,
149                 "Name": name,
150                 "Contact": contact,
151                 "DOB": dob,
152                 "Gender": gender,
153                 "Company": company,
154                 "Position": position,
155                 "DateAccessed": date_accessed,
156                 "Photo" : photo
157             })
158
159         # Prepare the query to retrieve data from the Detected_Occ table
160         detected_query = QSqlQuery()
161         #detected_query.prepare("SELECT * FROM Detected_Occ WHERE LOWER(Name) = :name OR LOWER(OID) = :oid")
```

Figure 4.2.6.5 main_occupant_detail.py (5/17)

```
        #detected_query.prepare("SELECT * FROM Detected_Occ WHERE LOWER(Name) = :name OR LOWER(OID) = :oid")
        detected_query.prepare("SELECT * FROM Detected_Occ WHERE Name = :name OR OID = :oid")
        detected_query.bindValue(":name", name_or_oid)
        detected_query.bindValue(":oid", name_or_oid)

        if not detected_query.exec_():
            print("Failed to execute query for Detected_Occ table.")
            db.close()
            return

        detected_data = []
        while detected_query.next():
            # Retrieve the data from the Detected_Occ table
            record_id = detected_query.value("RecordID")
            oid = detected_query.value("OID")
            name = detected_query.value("Name")
            contact = detected_query.value("Contact")
            detected_dt = detected_query.value("DetectedDT")
            zone = detected_query.value("Zone")
            status = detected_query.value("Status")

            detected_data.append({
                "RecordID": record_id,
                "OID": oid,
                "Name": name,
                "Contact": contact,
                "DetectedDT": detected_dt,
                "Zone": zone,
```

Figure 4.2.6.6 main_occupant_detail.py (6/17)

```
                "DetectedDT": detected_dt,
                "Zone": zone,
                "Status": status
            })

        # Close the database connection
        db.close()

        return occupants_data, detected_data
    Cheong Kok Siong
    def display_data(self, occupants_data, detected_data):
        # Display occupants data
        if occupants_data:
            occupant = occupants_data[0]  # Assuming only one occupant is retrieved
            self.id_in.setText(str(occupant["OID"]))
            self.name_in.setText(occupant["Name"])
            self.gender_in.setText(occupant["Gender"])
            self.comp_in.setText(occupant["Company"])
            self.pos_in.setText(occupant["Position"])
            self.datereg_in.setText(occupant["DateAccessed"])

            photo_data = occupant["Photo"]
            # print(photo_data)
            image = QImage.fromData(photo_data)

            if not image.isNull():
                # Scale the image to fit the label size
                scaled_image = image.scaled(self.photo_label.size(), Qt.AspectRatioMode.KeepAspectRatio,
                                            Qt.SmoothTransformation)
```

Figure 4.2.6.7 main_occupant_detail.py (7/17)

```
                                        Qt.SmoothTransformation)
                # Create a QPixmap from the scaled image
                pixmap = QPixmap.fromImage(scaled_image)
                self.photo_label.setPixmap(pixmap)

            # Display detected data
            if detected_data:
                self.tableWidget.setRowCount(len(detected_data))
                for row, data in enumerate(detected_data):
                    date = data["DetectedDT"]
                    zone = data["Zone"]
                    status = data["Status"]
                    formatted_date = self.change_date_format(date)
                    new_date = formatted_date.split(" ")
                    #new_date = date.split(" ")
                    print(new_date)

                    self.tableWidget.setItem(row, 0, QTableWidgetItem(new_date[0]))
                    self.tableWidget.setItem(row, 1, QTableWidgetItem(new_date[1]))
                    self.tableWidget.setItem(row, 2, QTableWidgetItem(zone))
                    self.tableWidget.setItem(row, 3, QTableWidgetItem(status))


                    # Set other columns with corresponding data

                    # Set the alignment for each cell in the row
                    for col, text in enumerate(new_date):
                        item = QTableWidgetItem(text)
```

Figure 4.2.6.8 main_occupant_detail.py (8/17)

```
                    for col, text in enumerate(new_date):
                        item = QTableWidgetItem(text)
                        item.setTextAlignment(Qt.AlignCenter)
                        self.tableWidget.setItem(row, col, item)
                    zone_item = QTableWidgetItem(zone)
                    zone_item.setTextAlignment(Qt.AlignCenter)
                    self.tableWidget.setItem(row, 2, zone_item)

                    status_item = QTableWidgetItem(status)
                    status_item.setTextAlignment(Qt.AlignCenter)
                    self.tableWidget.setItem(row, 3, status_item)

    Cheong Kok Siong *
    def filter_table(self):
        filter_text = self.input_NID_2.toPlainText().strip().lower()
        print(filter_text)

        if not filter_text:
            # If the filter text is empty, show all rows in the table
            for row in range(self.tableWidget.rowCount()):
                self.tableWidget.setRowHidden(row, False)
            return
        # Get the column indices to apply the filter
        date_column = 0
        time_column = 1
        zone_column = 2
        status_column = 3

        row_count = self.tableWidget.rowCount()
```

Figure 4.2.6.9 main_occupant_detail.py (9/17)

```
267         row_count = self.tableWidget.rowCount()
268         for row in range(row_count):
269             # Get the text in each cell of the row
270             date_text = self.tableWidget.item(row, date_column).text().strip().lower()
271             time_text = self.tableWidget.item(row, time_column).text().strip().lower()
272             zone_text = self.tableWidget.item(row, zone_column).text().strip().lower()
273             status_text = self.tableWidget.item(row, status_column).text().strip().lower()
274 
275             # Check if the filter text is present in any of the columns
276             if (filter_text in date_text) or (filter_text in time_text) or (filter_text in zone_text) or (filter_text in status_text):
277                 self.tableWidget.setRowHidden(row, False)
278             else:
279                 self.tableWidget.setRowHidden(row, True)
280 
    Cheong Kok Siong
281     def sort_table(self, index):
282         # Initialize the sort order dictionary if it doesn't exist
283         if not hasattr(self, "sort_order_dict"):
284             self.sort_order_dict = {}
285 
286         # Get the current sort order of the clicked column
287         current_sort_order = self.sort_order_dict.get(index, Qt.SortOrder.AscendingOrder)
288 
289         # Determine the new sort order for the clicked column
290         if current_sort_order == Qt.SortOrder.AscendingOrder:
291             new_sort_order = Qt.SortOrder.DescendingOrder
292         else:
293             new_sort_order = Qt.SortOrder.AscendingOrder
```

Figure 4.2.6.10 main_occupant_detail.py (10/17)

```
292         else:
293             new_sort_order = Qt.SortOrder.AscendingOrder
294 
295         # Update the sort order for the clicked column in the dictionary
296         self.sort_order_dict[index] = new_sort_order
297 
298         # Update the sort order for the clicked column in the table
299         self.tableWidget.sortByColumn(index, new_sort_order)
300 
301         # Set the sort order arrow for the clicked column
302         self.tableWidget.horizontalHeader().setSortIndicator(index, new_sort_order)
303         self.tableWidget.horizontalHeader().setSortIndicatorShown(True)
304 
305         # If the column clicked is the date column (let's assume it's column 0)
306         if index == 0:
307             self.tableWidget.sortByColumn(index, new_sort_order, lambda row: self.get_sort_date(row, index))
    Cheong Kok Siong
308     def get_sort_date(self, row, column):
309         # This function returns a sort key for the date column
310         date_item = self.tableWidget.item(row, column)
311         time_item = self.tableWidget.item(row, column + 1)  # Assuming time is in the next column (column + 1)
312         if date_item and time_item:
313             date_str = date_item.text()
314             time_str = time_item.text()
315             # Combine the date and time strings into a single datetime string
316             datetime_str = f"{date_str} {time_str}"
317             # Parse the datetime string into a QDateTime object
318             datetime = QDateTime.fromString(datetime_str, "yyyy-MM-dd HH:mm:ss")
319             return datetime
```

Figure 4.2.6.11 main_occupant_detail.py (11/17)

```
318             datetime = QDateTime.fromString(datetime_str, "yyyy-MM-dd HH:mm:ss")
319             return datetime
320         return None
321
            Cheong Kok Siong
322     def change_date_format(self, input_date):
323         # Parse the input date using the original format
324         input_format = "%m/%d/%Y %H:%M:%S"
325         date_obj = datetime.strptime(input_date, input_format)
326
327         # Format the date to the desired format
328         output_format = "%d-%m-%Y %H:%M:%S"
329         formatted_date = date_obj.strftime(output_format)
330
331         return formatted_date
332
            Cheong Kok Siong
333     def create_blacklist_table(self):
334         # Connect to the database
335         db = QSqlDatabase.addDatabase("QSQLITE")
336         db.setDatabaseName("Occupants.db")
337
338         if not db.open():
339             print("Failed to connect to the database.")
340             return
341
342         # Execute a query to create the "Detected_Occ" table
343         query = QSqlQuery()
344         query.prepare(
```

Figure 4.2.6.12 main_occupant_detail.py (12/17)

```
344         query.prepare(
345             '''CREATE TABLE IF NOT EXISTS Blacklist(
346                     BKID INTEGER PRIMARY KEY AUTOINCREMENT,
347                     OID INTEGER,
348                     Name VARCHAR(255),
349                     BlacklistedDT DATETIME
350                     )
351                     ''')
352
353         if not query.exec_():
354             print("Failed to create the Blacklist table.")
355             db.close()
356             return
357
358         # Close the database connection
359         db.close()
360
            Cheong Kok Siong
361     def retrieve_blacklist_data(self):
362         # Connect to the database
363         db = QSqlDatabase.addDatabase("QSQLITE")
364         db.setDatabaseName("Occupants.db")
365
366         if not db.open():
367             print("Failed to connect to the database.")
368             return None
369
370         blacklist_data = []
```

Figure 4.2.6.13 main_occupant_detail.py (13/17)

```
372         # Execute a query to retrieve data from the "Blacklist" table
373         query = QSqlQuery()
374         query.prepare("SELECT BKID, OID, Name, BlacklistedDT FROM Blacklist")
375
376         if query.exec_():
377             while query.next():
378                 bk_id = query.value(0)
379                 oid = query.value(1)
380                 name = query.value(2)
381                 blacklist_dt = query.value(3)
382                 blacklist_data.append((bk_id, oid, name, blacklist_dt))
383         else:
384             print("Failed to retrieve data from the Blacklist table.")
385
386         # Close the database connection
387         db.close()
388         return blacklist_data
    Cheong Kok Siong
389 def checkboxClicked(self):
390     if self.blacklist.isChecked():
391         confirmation = QMessageBox.question(self, "Confirmation", "Are you sure you want to add the occupant into BLACKLIST?",
392                                             QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
393
394         if confirmation == QMessageBox.Yes:
395             occupant_id = int(self.id_in.toPlainText())
396             occupant_name = self.name_in.toPlainText()
397
398             current_datetime = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
399
```

Figure 4.2.6.14 main_occupant_detail.py (14/17)

```
400             db = QSqlDatabase.addDatabase("QSQLITE")
401             db.setDatabaseName("Occupants.db")
402
403             if not db.open():
404                 print("Failed to connect to the database.")
405                 return
406
407             query = QSqlQuery()
408             query.prepare("INSERT INTO Blacklist (OID, Name, BlacklistedDT) VALUES (:oid, :name, :blacklisted_dt)")
409             query.bindValue(":oid", occupant_id)
410             query.bindValue(":name", occupant_name)
411             query.bindValue(":blacklisted_dt", current_datetime)
412
413             if not query.exec_():
414                 print("Failed to add occupant to Blacklist table.")
415                 db.close()
416                 return
417             # Close the database connection
418             db.close()
419             print("Occupant added to Blacklist")
420
421         else:
422             self.blacklist.setChecked(False)
423             print("Action Fail")
424
425     else:
426         confirmation = QMessageBox.question(self, "Confirmation", "Are you sure you want to remove occupant from BLACKLIST?",
427                                             QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
428
429         if confirmation == QMessageBox.Yes:
```

Figure 4.2.6.15 main_occupant_detail.py (15/17)

```
429             if confirmation == QMessageBox.Yes:
430                 occupant_id = int(self.id_in.toPlainText())
431 
432                 db = QSqlDatabase.addDatabase("QSQLITE")
433                 db.setDatabaseName("Occupants.db")
434 
435                 if not db.open():
436                     print("Failed to connect to the database.")
437                     return
438 
439                 query = QSqlQuery()
440                 query.prepare("DELETE FROM Blacklist WHERE OID = :oid")
441                 query.bindValue(":oid", occupant_id)
442 
443                 if not query.exec_():
444                     print("Failed to remove occupant from Blacklist table.")
445                     db.close()
446                     return
447 
448                 db.close()
449                 print("Occupant removed from Blacklist")
450             else:
451                 self.blacklist.setChecked(True)
452                 print("Action Remove Fail")
453 
        Cheong Kok Siong
454     def check_blacklist(self, oid):
455         db = QSqlDatabase.addDatabase("QSQLITE")
```

Figure 4.2.6.16 main_occupant_detail.py (16/17)

```
454     def check_blacklist(self, oid):
455         db = QSqlDatabase.addDatabase("QSQLITE")
456         db.setDatabaseName("Occupants.db")
457         if not db.open():
458             print("Failed to connect to the database.")
459             return False
460 
461         query = QSqlQuery()
462         query.prepare("SELECT COUNT(*) FROM Blacklist WHERE OID = :oid")
463         query.bindValue(":oid", oid)
464 
465         if not query.exec_():
466             print("Failed to check if occupant is in Blacklist.")
467             db.close()
468             return False
469 
470         query.next()
471         count = query.value(0)
472 
473         db.close()
474         return count > 0
475 
476 if __name__ == "__main__":
477     app = QApplication([])
478     window = MainOccupants_DetailWindow()
479     window.show()
480     app.exec_()
```

Figure 4.2.6.17 main_occupant_detail.py (17/17)

"MainOccupants_DetailWindow.py" is a Python code file that primarily focuses on managing occupant details and incorporates blacklisting functionality. This application provides various features, including occupant search, detection record filtering based on selected criteria from combo boxes, table sorting, and occupant blacklisting and unblacklisting. To interact with data related to occupants and their detection records, the program interfaces with an SQLite database.

Users can input search criteria to filter detection records and display occupant data in both labels and tables. Additionally, this code includes two essential functions: "create_blacklist_table" is responsible for creating the "Blacklist" table within the database, and "retrieve_blacklist_data" retrieves data from this table.

Furthermore, the "check_black" function is defined to determine whether an occupant with a specific OID is already present in the "Blacklist" table. If an occupant is blacklisted, the corresponding checkbox is checked; otherwise, it remains unchecked. This application offers user-friendly functionality, allowing the addition and removal of tenants from the blacklist and enabling users to arrange detection records by clicking on column headers for ease of management.

4.2.7 main_report.py

```python
import os
import sys
import re
from io import BytesIO
from PIL import Image
from PySide2.QtGui import QPixmap, QImage, QFontMetrics, QFont, QColor
from PySide2.QtWidgets import QApplication, QMainWindow, QLabel, QFrame, QMessageBox, QAction, QListWidgetItem, \
    QListWidget, QVBoxLayout, QWidget, QGraphicsScene, QDateEdit, QCalendarWidget
from PySide2.QtCore import Qt, QTimer, QRect, QCoreApplication, QDate, QDateTime, QTime
from PySide2.QtSql import QSqlDatabase, QSqlQuery
from datetime import datetime
from main_occupant_detail import MainOccupants_DetailWindow
import csv
import matplotlib
matplotlib.use('Qt5Agg')
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
import pandas as pd

from UI_report import UI_ReportWindow

Cheong Kok Siong
class MatplotlibGraph(QWidget):
    Cheong Kok Siong
    def __init__(self):
        super().__init__()

        layout = QVBoxLayout(self)
        self.canvas = FigureCanvas(plt.figure())
        layout.addWidget(self.canvas)
```

Figure 4.2.7.1.1 MatplotlibGraph (1/5)

```python
        layout.addWidget(self.canvas)


    Cheong Kok Siong
    def resizeEvent(self, event):
        super().resizeEvent(event)
        self.update_figure_size()
    Cheong Kok Siong
    def update_figure_size(self):
        self.canvas.figure.set_size_inches(5,3.8)
        self.canvas.draw()
    Cheong Kok Siong
    def clear(self):
        self.canvas.figure.clear()
        self.canvas.draw()
    Cheong Kok Siong *
    def plot_bar(self, data, report_type):
        self.clear()
        df = pd.DataFrame(data, columns=['Name', 'LastDetectedDT', 'Status'])
        df['LastDetectedDT'] = pd.to_datetime(df['LastDetectedDT'])
        df = df[df['Status'] == 'IN']

        if report_type == 'day':
            df['Date'] = df['LastDetectedDT'].dt.strftime('%d/%m')
            df['Date'] = df['LastDetectedDT'].dt.date
            min_date = df['Date'].min()
            max_date = df['Date'].max()
            complete_date_range = pd.date_range(min_date, max_date, freq='D').date
        elif report_type == 'month':
            df['Date'] = df['LastDetectedDT'].dt.strftime('%B')
```

Figure 4.2.7.1.2 MatplotlibGraph (2/5)

```
ons
52          elif report_type == 'month':
53              df['Date'] = df['LastDetectedDT'].dt.strftime('%B')
54          elif report_type == 'year':
55              df['Date'] = df['LastDetectedDT'].dt.strftime('%Y')
56
57          df = self.drop_duplicates_by_name_and_date(df)
58
59          grouped_data = df.groupby('Date')['Name'].count()
60
61          if report_type == 'day':
62              grouped_data = grouped_data.reindex(complete_date_range, fill_value=0)
63
64          ax = self.canvas.figure.add_subplot(111)
65          grouped_data.plot(kind='bar', ax=ax)
66          ax.set_title('Counted Occupancies in Building')
67          ax.set_xlabel('Date')
68          ax.set_ylabel('Count')
69          self.update_figure_size()
            Cheong Kok Siong *
70      def plot_line(self, data, selected_date):
71          self.clear()
72          df = pd.DataFrame(data, columns=['Name', 'DetectedDT', 'Status'])
73          df['DetectedDT'] = pd.to_datetime(df['DetectedDT'])
74          df = df[df['Status'] == 'IN']
75          print(df)
76          df['Date'] = df['DetectedDT'].dt.date  # Create the 'Date' column
77          df = df[df['Date'] == selected_date]
78
79          print("Today Data Befor Drop\n",df )
```

Figure 4.2.7.1.3 MatplotlibGraph (3/5)

```
80          df = self.drop_duplicates_by_name_and_hour(df)
81          # Filter data for the selected date
82          df_selected_date = df[df['Date'] == selected_date]
83          print("After DROP___\n",df_selected_date)
84          print(selected_date)
85
86          if not df_selected_date.empty:
87              hourly_count = df_selected_date.groupby(df_selected_date['DetectedDT'].dt.hour)['Name'].count()
88
89              fig, ax = plt.subplots()
90              ax.xaxis.set_major_locator(plt.MultipleLocator(base=1))  # Set locator for each hour
91              ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'{int(x):02}:00'))  # Format to 24-hour tim
92
93              unique_hours = df_selected_date['DetectedDT'].dt.hour.unique()
94              y_values = [int(hourly_count.get(hour, 0)) for hour in unique_hours]  # Ensure integer value
95
96              ax.plot(unique_hours, y_values, marker='o')
97
98              ax.set_title('Occupants Availability')
99              ax.set_xlabel('Time')
100             ax.set_ylabel('Number of Occupants')
101
102             self.canvas.figure = fig
103             self.canvas.draw()
```

Figure 4.2.7.1.4 MatplotlibGraph (4/5)

```
111     def drop_duplicates_by_name_and_date(self, df):
112         unique_records = {}
113         for index, row in df.iterrows():
114             key = (row['Name'], row['Date'])
115             if key not in unique_records:
116                 unique_records[key] = index
117 
118         return df.loc[unique_records.values()]
119 
            Cheong Kok Siong
120     def drop_duplicates_by_name_and_hour(self, df):
121         unique_records = {}
122         for index, row in df.iterrows():
123             key = (row['Name'], row['DetectedDT'].hour)
124             if key not in unique_records:
125                 unique_records[key] = index
126 
127         return df.loc[unique_records.values()]
128 
```

Figure 4.2.7.1.5 MatplotlibGraph (5/5)

The “MatplotlibGraph” class is serving as important component for data visualization and analysis within the GUI application, enhancing the user experience and provide valuable insights about the data being displayred. “plot_bar” function is defined to display a bar chart using Matplotlib which takes data and selected date as input, those data firstly will converted into a Pandas DataFrame, and then it is processed in accordance with the type of report. It then plots the data as a bar chart on the Matplotlib canvas. “plot_line” function is created and display a line chart using Matplotlib, and the data is processed to filter records for selected date, and then the hourly count of occupants is plotted as line chart. “drop_duplicates_by_name_and_date” and “drop_duplicates_bu_name_and_hour” are define to serve as data preprocessing which remove duplicate records from a Pandas DataFrame.

```
Cheong Kok Siong *
class MainReport_Window(QMainWindow, UI_ReportWindow):
    Cheong Kok Siong *
    def __init__(self):
        super(MainReport_Window, self).__init__()

        self.setupUi(self)
        self.setFixedSize(self.size())
        self.detail_window = None
        self.graph_widget = MatplotlibGraph()
        self.att_view.setScene(QGraphicsScene(self))
        self.att_view.scene().addWidget(self.graph_widget)
        self.att_view.setFixedSize(550, 450)

        self.graph_widget2 = MatplotlibGraph()
        self.daytime_view.setScene(QGraphicsScene(self))
        self.daytime_view.scene().addWidget(self.graph_widget2)
        self.daytime_view.setFixedSize(550, 450)

        self.fromLabel.hide()
        self.fromDateEdit.hide()
        self.toLabel.hide()
        self.toDateEdit.hide()
        self.nameLabel.hide()
        self.nameComboBox.hide()
        self.btn_export.hide()

        self.selectDataComboBox.currentTextChanged.connect(self.selectdata)

```

Figure 4.2.7.2.1 Main_ReportWindow(1/18)

```
        self.textBrowser_2.setFixedHeight(40)
        self.att_view.setFixedSize(550, 430)
        self.daytime_view.setFixedSize(550,430)

        current_date = datetime.now().date()

        self.retrieve_data_from_detected_occ_database()
        self.dateEdit.dateChanged.connect(self.show_line_plot)

        self.dateEdit.setCalendarPopup(True)
        self.dateEdit.setDate(QDate.currentDate())

        self.fromDateEdit.setCalendarPopup(True)
        self.fromDateEdit.setDate(QDate.currentDate())

        self.toDateEdit.setCalendarPopup(True)
        self.toDateEdit.setDate(QDate.currentDate())

        self.day_report()
        self.r_day.setChecked(True)
        self.r_day.toggled.connect(self.day_report)
        self.r_month.toggled.connect(self.month_report)
        self.r_year.toggled.connect(self.year_report)
        self.listWidget.itemClicked.connect(self.click_list_item)

        self.retrieve_data_from_detected_trans_database()
        self.time_timer = QTimer()
        self.time_timer.timeout.connect(self.report_notification)
        self.time_timer.start(1000)
        #self.generate_report()
```

Figure 4.2.7.2.2 Main_ReportWindow(2/18)

```
186
187         ...
            Cheong Kok Siong
192         def selectdata(self):
193             self.fromLabel.hide()
194             self.fromDateEdit.hide()
195             self.toLabel.hide()
196             self.toDateEdit.hide()
197             self.nameLabel.hide()
198             self.nameComboBox.hide()
199             self.btn_export.hide()
200             sltitem = self.selectDataComboBox.currentText()
201             if sltitem == " ":
202                 return
203
204             elif sltitem == "Occupant Detail":
205                 self.nameLabel.show()
206                 self.nameComboBox.show()
207                 self.btn_export.show()
208                 self.add_name_to_combobox()
209                 self.btn_export.clicked.connect(self.export_occ_det_record)
210
211             elif sltitem == "Building's Occupants":
212                 self.btn_export.show()
213                 self.btn_export.clicked.connect(self.export_building_data_to_csv)
214
215             elif sltitem == "Detected Occupants Record":
216                 self.toLabel.show()
217                 self.toDateEdit.show()
218                 self.fromDateEdit.show()
```

Figure 4.2.7.2.3 Main_ReportWindow(3/18)

```
219                 self.fromLabel.show()
220                 self.btn_export.show()
221                 self.btn_export.clicked.connect(self.export_building_record)
222             ...
            Cheong Kok Siong
230         def day_report(self):
231             data = self.retrieve_data_from_detected_trans_database()
232             if data:
233                 self.graph_widget.plot_bar(data, 'day')
            Cheong Kok Siong
234         def month_report(self):
235             if self.r_month.isChecked():
236                 data = self.retrieve_data_from_detected_trans_database(monthly=True)
237                 if data:
238                     self.graph_widget.plot_bar(data, 'month')
            Cheong Kok Siong
239         def year_report(self):
240             if self.r_year.isChecked():
241                 data = self.retrieve_data_from_detected_trans_database(yearly=True)
242                 if data:
243                     self.graph_widget.plot_bar(data, 'year')
244         ...
            Cheong Kok Siong
276         def retrieve_data_from_detected_trans_database(self, monthly=False, yearly=False):
277             try:
278                 # Connect to the database
279                 conn = QSqlDatabase.addDatabase("QSQLITE")
280                 conn.setDatabaseName("Occupants.db")
281                 if not conn.open():
282                     print("Failed to connect to the database.")
```

Figure 4.2.7.2.4 Main_ReportWindow(4/18)

```
283             return []
284
285         # Execute a SELECT query to retrieve the data from Detected_Trans table
286         query = QSqlQuery()
287         if monthly:
288             query.prepare("SELECT Name, LastDetectedDT, Status, strftime('%Y-%m', LastDetectedDT) AS MonthYear FROM Detected_Trans")
289         elif yearly:
290             query.prepare("SELECT Name, LastDetectedDT, Status, strftime('%Y', LastDetectedDT) AS Year FROM Detected_Trans")
291         else:
292             query.prepare("SELECT Name, LastDetectedDT, Status FROM Detected_Trans")
293
294         if not query.exec_():
295             print("Failed to execute the query:", query.lastError().text())
296             conn.close()
297             return []
298
299         data = []
300         while query.next():
301             name = query.value(0)
302             detected_dt = query.value(1)
303             status = query.value(2)
304             data.append((name, detected_dt, status))
305
306         # Close the database connection
307         conn.close()
308
309         return data
310     except Exception as e:
311         print("Error retrieving data from database:", e)
```

Figure 4.2.7.2.5 Main_ReportWindow(5/18)

```
313         return []
            Cheong Kok Siong
314     def retrieve_data_from_detected_occ_database(self):
315         try:
316             # Connect to the database
317             conn = QSqlDatabase.addDatabase("QSQLITE")
318             conn.setDatabaseName("Occupants.db")
319             if not conn.open():
320                 print("Failed to connect to the database.")
321                 return []
322
323             # Execute a SELECT query to retrieve the data from Detected_Occ table
324             query = QSqlQuery()
325             query.prepare("SELECT Name, DetectedDT, Status FROM Detected_Occ")
326             if not query.exec_():
327                 print("Failed to execute the query:", query.lastError().text())
328                 conn.close()
329                 return []
330
331             data = []
332             while query.next():
333                 name = query.value(0)
334                 detected_dt = query.value(1)
335                 status = query.value(2)
336                 data.append((name, detected_dt, status))
337
338             # Close the database connection
339             conn.close()
340
341             self.detected_occ_data = data  # Replace 'data' with the actual data you retrieve
```

Figure 4.2.7.2.6 Main_ReportWindow(6/18)

```
340
341                 self.detected_occ_data = data  # Replace 'data' with the actual data you retrieve
342             except Exception as e:
343                 print("Error retrieving data from database:", e)
344                 self.detected_occ_data = []
345
    Cheong Kok Siong
346     def show_line_plot(self, selected_date):
347         # if self.detected_occ_data:
348         self.graph_widget2.plot_line(self.detected_occ_data, selected_date.toPython())
349         ...
352
353     ...
    Cheong Kok Siong *
375     def report_notification(self):
376         occ_status_data = self.retrieve_occ_status()
377
378         self.listWidget.clear()
379
380         for data in occ_status_data:
381             oid = data["OID"]
382             name = data["Name"]
383             zone = data["Zone"]
384             lastdt = data["LastDT"]
385             status = data["Status"]
386
387             is_blacklisted = self.check_blacklist(name)
388
389             if status == "IN":
```

Figure 4.2.7.2.7 Main_ReportWindow(8/18)

```
388
389             if status == "IN":
390                 #print(oid)
391                 lastdt_obj = datetime.strptime(lastdt, "%m/%d/%Y %H:%M:%S")  # Use the correct format here
392
393                 current_datetime = datetime.now()
394                 time_difference = current_datetime - lastdt_obj
395                 #print(time_difference)
396
397                 if time_difference.days > 2:
398                     item_text = f"Warning!!!\nOID: {oid} {name} was IN the building more than 2 days !\n    Last Detected Date: {lastdt}"
399                     item = QListWidgetItem(item_text)
400                     item.setBackground(QColor(255, 165, 0))
401                     self.listWidget.addItem(item)
402
403             if is_blacklisted:
404                 item_text = f"Warning!!! Blacklist Detected!!!\n BlackListed Person OID: {oid} {name} was IN the building !\n    Last Detected Date: {lastdt}"
405                 item = QListWidgetItem(item_text)
406                 item.setBackground(QColor(255, 200, 200))
407                 self.listWidget.addItem(item)
    Cheong Kok Siong
408     def retrieve_occ_status(self):
409         db = QSqlDatabase.addDatabase("QSQLITE")
410         db.setDatabaseName("Occupants.db")
411         if not db.open():
412             print("Failed to connect to the database.")
413             return []
414
415         query = QSqlQuery()
```

Figure 4.2.7.2.8 Main_ReportWindow(8/18)

```
            query.prepare("SELECT * FROM occ_status")
            if not query.exec_():
                print("Failed to fetch records from occ_status table.")
                db.close()
                return []

            occ_status_data = []

            while query.next():
                oid = query.value(0)
                name = query.value(1)
                lastdt = query.value(2)
                status = query.value(3)
                zone = query.value(4)
                occ_status_data.append({"OID": oid, "Name": name, "LastDT": lastdt, "Status":status, "Zone": zone})

            db.close()
            return occ_status_data

    Cheong Kok Siong
    def check_blacklist(self, facename):
        db = QSqlDatabase.addDatabase("QSQLITE")
        db.setDatabaseName("Occupants.db")

        if not db.open():
            print("Failed to connect to the database.")
            return False

        query = QSqlQuery()
        query.prepare("SELECT COUNT(*) FROM Blacklist WHERE NAME = :name")
```

Figure 4.2.7.2.9 Main_ReportWindow(9/18)

```
        query = QSqlQuery()
        query.prepare("SELECT COUNT(*) FROM Blacklist WHERE NAME = :name")
        query.bindValue(":name", facename)

        if not query.exec_():
            print("Failed to check if occupant is in Blacklist.")
            db.close()
            return False

        query.next()
        count = query.value(0)

        db.close()
        return count > 0

    Cheong Kok Siong *
    def click_list_item(self, item):
        # Access the clicked item
        clickedItem = item.text()

        print(clickedItem)
        match = re.search(r"\b\d+\b", clickedItem)
        if match:
            clicked_OID = match.group()
            print(clicked_OID)
        # If the detail window doesn't exist or has been closed, create a new instance
        if not self.detail_window or not self.detail_window.isVisible():
            self.detail_window = MainOccupants_DetailWindow()
        # Set the clicked item name in the detail window
```

Figure 4.2.7.2.10 Main_ReportWindow(10/18)

```
470         # Set the clicked item name in the detail window
471         self.detail_window.set_clicked_item(clickedItem, clicked_OID)
472         # Show the detail window
473         self.detail_window.show()
474
            Cheong Kok Siong
475     def export_building_data_to_csv(self):
476         csv_filename = "occupants_data.csv"
477         db = QSqlDatabase.addDatabase("QSQLITE")
478         db.setDatabaseName("Occupants.db")
479         if not db.open():
480             print("Failed to connect to the database.")
481             return
482
483         query = QSqlQuery()
484         query.prepare("SELECT * FROM Occupants")
485         if not query.exec_():
486             print("Failed to fetch records.")
487             db.close()
488             return
489
490         image_directory = "image_exports"
491         if not os.path.exists(image_directory):
492             os.makedirs(image_directory)
493
494         # Fetch data and write to CSV
495         with open(csv_filename, "w", newline="") as csv_file:
496             csv_writer = csv.writer(csv_file)
```

Figure 4.2.7.2.11 Main_ReportWindow(11/18)

```
496         csv_writer = csv.writer(csv_file)
497
498         # header
499         field_names = [query.record().fieldName(i) for i in range(query.record().count())]
500         csv_writer.writerow(field_names)
501         # data
502         while query.next():
503             row = [query.value(i) for i in range(query.record().count())]
504
505             # Convert image blob data to image and save temporarily
506             image_blob_data = query.value(field_names.index("Photo"))
507             if image_blob_data is not None:
508                 image = Image.open(BytesIO(image_blob_data))
509                 image_name = query.value(field_names.index("Name"))
510                 image_filename = f"{image_name}.jpg"
511
512                 image_path = os.path.join(image_directory, image_filename)
513                 # Convert RGBA image to RGB mode
514                 if image.mode == "RGBA":
515                     image = image.convert("RGB")
516
517                 image.save(image_path)
518                 row[field_names.index("Photo")] = image_filename
519
520             csv_writer.writerow(row)
521
522     print(f"Data exported to {csv_filename} successfully.")
523
```

Figure 4.2.7.2.12 Main_ReportWindow(12/18)

```
524             msg_box = QMessageBox()
525             msg_box.setIcon(QMessageBox.Information)
526             msg_box.setWindowTitle("Export Successful")
527             msg_box.setText(f"Data exported to {csv_filename} successfully.")
528             msg_box.exec_()
529
530             db.close()
531
                Cheong Kok Siong
532         def export_building_record(self):
533             csv_filename = "building_detected_record.csv"
534
535             # Get start and end dates from the QDateEdit widgets
536             start_date = self.fromDateEdit.date().toPython()
537             end_date = self.toDateEdit.date().toPython()
538
539             try:
540                 # Connect to the database
541                 db = QSqlDatabase.addDatabase("QSQLITE")
542                 db.setDatabaseName("Occupants.db")
543                 if not db.open():
544                     print("Failed to connect to the database.")
545                     return
546
547                 start_date = QDate(start_date).toPython()
548                 end_date = QDate(end_date).toPython()
549
550                 start_datetime = QDateTime(start_date, QTime(0, 0, 0))
551                 end_datetime = QDateTime(end_date, QTime(23, 59, 59))
552                 db_date_format = "MM/dd/yyyy"
```

Figure 4.2.7.2.13 Main_ReportWindow(13/18)

```
553             formatted_start_date = start_datetime.toString(db_date_format + " hh:mm:ss")
554             formatted_end_date = end_datetime.toString(db_date_format + " hh:mm:ss")
555             # Execute a SELECT query to retrieve the data from Detected_Occ table within the date range
556             query = QSqlQuery()
557             query.prepare("SELECT * FROM Detected_Occ WHERE DetectedDT BETWEEN :start_date AND :end_date")
558             query.bindValue(":start_date", formatted_start_date)
559             query.bindValue(":end_date", formatted_end_date)
560             if not query.exec_():
561                 print("Failed to execute the query:", query.lastError().text())
562                 db.close()
563                 return
564
565             data = []
566             while query.next():
567                 record_id = query.value(0)
568                 oid = query.value(1)
569                 name = query.value(2)
570                 contact = query.value(3)
571                 detected_dt = query.value(4)
572                 photo_blob = query.value(5)
573                 zone = query.value(6)
574                 status = query.value(7)
575                 data.append((record_id, oid, name, contact, detected_dt, zone, status))
576
577             # Close the database connection
578             db.close()
579
580             if not data:
581                 print("No data found within the specified date range.")
582                 return
```

Figure 4.2.7.2.14 Main_ReportWindow(14/18)

```
584             # Fetch data and write to CSV
585             with open(csv_filename, "w", newline="") as csv_file:
586                 csv_writer = csv.writer(csv_file)
587 
588                 # Write header
589                 field_names = ["RecordID", "OID", "Name", "Contact", "DetectedDT", "Zone", "Status"]
590                 csv_writer.writerow(field_names)
591 
592             # Write data
593                 for row in data:
594                     csv_writer.writerow(row)
595 
596             print(f"Data exported to {csv_filename} successfully.")
597             msg_box = QMessageBox()
598             msg_box.setIcon(QMessageBox.Information)
599             msg_box.setWindowTitle("Export Successful")
600             msg_box.setText(f"Data exported to {csv_filename} successfully.")
601             msg_box.exec_()
602 
603         except Exception as e:
604             print("Error retrieving and exporting data:", e)
605 
        Cheong Kok Siong
606     def add_name_to_combobox(self):
607         try:
608             db = QSqlDatabase.addDatabase("QSQLITE")
609             db.setDatabaseName("Occupants.db")
610             if not db.open():
611                 print("Failed to connect to the database.")
```

Figure 4.2.7.2.15 Main_ReportWindow(15/18)

```
612                 return
613 
614             query = QSqlQuery()
615             query.prepare("SELECT Name FROM Occupants")
616             if not query.exec_():
617                 print("Failed to execute the query:", query.lastError().text())
618                 db.close()
619                 return
620 
621             self.nameComboBox.clear()
622 
623             while query.next():
624                 name = query.value(0)
625                 self.nameComboBox.addItem(name)
626 
627             db.close()
628         except Exception as e:
629             print("Error retrieving 'Name' values from database:", e)
630 
        Cheong Kok Siong
631     def export_occ_det_record(self):
632         selected_name = self.nameComboBox.currentText()
633         if not selected_name:
634             print("No name selected.")
635             return
636 
637         csv_filename = f"{selected_name}.csv"
638         try:
639             db = QSqlDatabase.addDatabase("QSQLITE")
640             db.setDatabaseName("Occupants.db")
```

Figure 4.2.7.2.16 Main_ReportWindow(16/18)

```
            db = QSqlDatabase.addDatabase("QSQLITE")
            db.setDatabaseName("Occupants.db")
            if not db.open():
                print("Failed to connect to the database.")
                return

            # Execute a SELECT query to retrieve the data from Detected_Occ table within the date range
            query = QSqlQuery()
            query.prepare("SELECT * FROM Detected_Occ WHERE Name = :selected_name")
            query.bindValue(":selected_name", selected_name)
            if not query.exec_():
                print("Failed to execute the query:", query.lastError().text())
                db.close()
                return

            data = []
            while query.next():
                record_id = query.value(0)
                oid = query.value(1)
                name = query.value(2)
                contact = query.value(3)
                detected_dt = query.value(4)
                photo_blob = query.value(5)
                zone = query.value(6)
                status = query.value(7)
                data.append((record_id, oid, name, contact, detected_dt, zone, status))

            # Close the database connection
            db.close()
```

Figure 4.2.7.2.17 Main_ReportWindow(17/18)

```
            if not data:
                print("No data found within the specified date range.")
                return

            # Fetch data and write to CSV
            with open(csv_filename, "w", newline="") as csv_file:
                csv_writer = csv.writer(csv_file)

                # Write header
                field_names = ["RecordID", "OID", "Name", "Contact", "DetectedDT", "Zone", "Status"]
                csv_writer.writerow(field_names)

                # Write data
                for row in data:
                    csv_writer.writerow(row)

            print(f"Data exported to {csv_filename} successfully.")
            msg_box = QMessageBox()
            msg_box.setIcon(QMessageBox.Information)
            msg_box.setWindowTitle("Export Successful")
            msg_box.setText(f"Data exported to {csv_filename} successfully.")
            msg_box.exec_()

        except Exception as e:
            print("Error retrieving and exporting data:", e)
```

Figure 4.2.7.2.18 Main_ReportWindow(18/18)

“Main_ReportWindow” class that inherits from both “QMainWindow” and “UI_ReportWindow” is defined to create and manage GUI window for generating and displaying report related to building occupants and their behaviour and activity.

In this class, two instances of “MatplotlivGraph” class are created for displaying graphical plots and charts in GUI. Then, data retrieval is always the function we should run at initially since the data is used for generating report and graph. “selectdata” function is defined to control the show or hides various UI elements those are connected to specific button to their corresponding function. “day_report”, “month_report”, “year_report” are created for generating different types of report based on retrieved data, and those reports are displayed as bar chart.

“retrieve_data_from_detected_trans_database(self, monthly=False, yearly=False” which will depending on the parameters ‘monthly’ and ‘yearly’ and retrieve data for monthly or yearly report. “report_notification” is to generate notifications and populate a list widget based on retrieved data and checks for conditions where notification should be displayed like occupants being blacklisted or being in the building for more than 2 days. “click_list_item” handle event when item in list widget is clicked will extracts OID and open a occupant’s detail window to display related detail.

“add_name_to_combobox” function defined to modify the combo box elements which will display the registered occupants’ name for selection. “CSV” module is import for those functions including “export_building_daya_to_csv”, “export_building_record” and “export_occ_det_record”.

CHAPTER 5

System Implementation

5.1 Hardware Setup

Description	Specifications
Model	Acer Nitro AN515-52
Processor	Intel Core i5-8300H
Operating System	Windows 10
Graphic	NVIDIA GeForce GTX1050
Memory	12GB DDR4 RAM
Storage	500GB SATA HDD

Table 5.1.1 Specifications of laptop

This project will develop on Acer Nitro AN515-52 laptop with processor Intel Core i5-8300H, Window 10 operating system, NVIDIA GeForce GTX1050 GPU, 12GM RAM and 500GB SATA HDD. Another than that, personal smartphone – Iphone 14 also been used as second camera for capturing occupants' activities.

5.2 Software Setup

FRLD system is accompanied by a number of software programmes to enable a seamless development process.:

Software	**Specification**
Operation System	Microsoft Window 10
Integrated Development Environment (IDE)	Pycharm 2023.1 Community, IntelliJ IDEA Community Edition
Programming Languages	Python 3.9.7
Front-end	QTdesigner
Database	SQL
Camera Connection	iVCam

Table 5.2.1 Software List

5.3 Setting and Configuration

5.3.1 Additional Module and Library

- **Pillow 10.0.0**

  Pillow is a Python Image Library (PIL) that provide a comprehensive set of libraries for working with image. It facilitates the tasks such as opening, processing and saving various file format in python, below is the command for install the library:

  $ pip install Pillow

- **Dlid 19.24.2**

  Dlib is a versatile cross-platform C++ toolkit that provide in-depth support for numerous operating systems. It offers tool for creating complex C++ programmes to handle a variety of problems as well as machine learning techniques. Dlib also provides a Python interface, allowing Python developers to take advantage of its robust capabilities. Typically, it is utilized for computer vision applications:

  $ pip install dlib

- **Face-recognition 1.3.0**

  Utilising the strength of deep learning models, the "face_recognition" package is a Python library created with Dlib that enables facial feature identification in both still photos as well as streaming videos through the process of face detection and face recognition. Command used to download face_recognition:

  $ pip install face-recogntion

- **Face-recognition-models 0.3.0**

  Face-recognition-models refers to a Python package that complements the face-recogntion 1.3.0 library. Developed by dlib, offers a collection of models that have already been trained to enable face recognition procedures. There are several models included "dlib_face_recognition_resnet_model," "mmod_human_face_detector," "shape_predictor_5_face_landmark," and

"shape_predictor_68_face_landmark." These models have been trained using large datasets, giving them the ability to recognise faces with astounding precision.

- **Opencv-python 4.8.0.76**

  OpenCV-python is a Python package that offers a variety of tools for image and video processing. Additionally, this package has tools for object detection, camera calibration, and machine learning. Use the following command in the terminal to install OpenCV in PyCharm:

  $ pip install opencv-python

  NOTICE : python version should 2.7 or above !!

- **Numpy 1.25.2**

  Numpy is an open-source Python library for numerical data manipulation in matrices and arrays. Due to its powerful N-dimensional array capabilities, it is frequently utilised in data analysis, machine learning, and scientific computing.

- **Matplotlib**

  Matplotlib is a powerful Python package that can be used to create a variety of data visualizations.

- **Pandas**

  Pandas is a Python library that simplifies data manipulation and analysis. Data cleaning, transformation, and analysis are made easier in Python by the fact that it offers data structures and functions to effectively manage and analyse structured data, such as spreadsheets or databases.

5.3.2 IDE Setting and Configuration

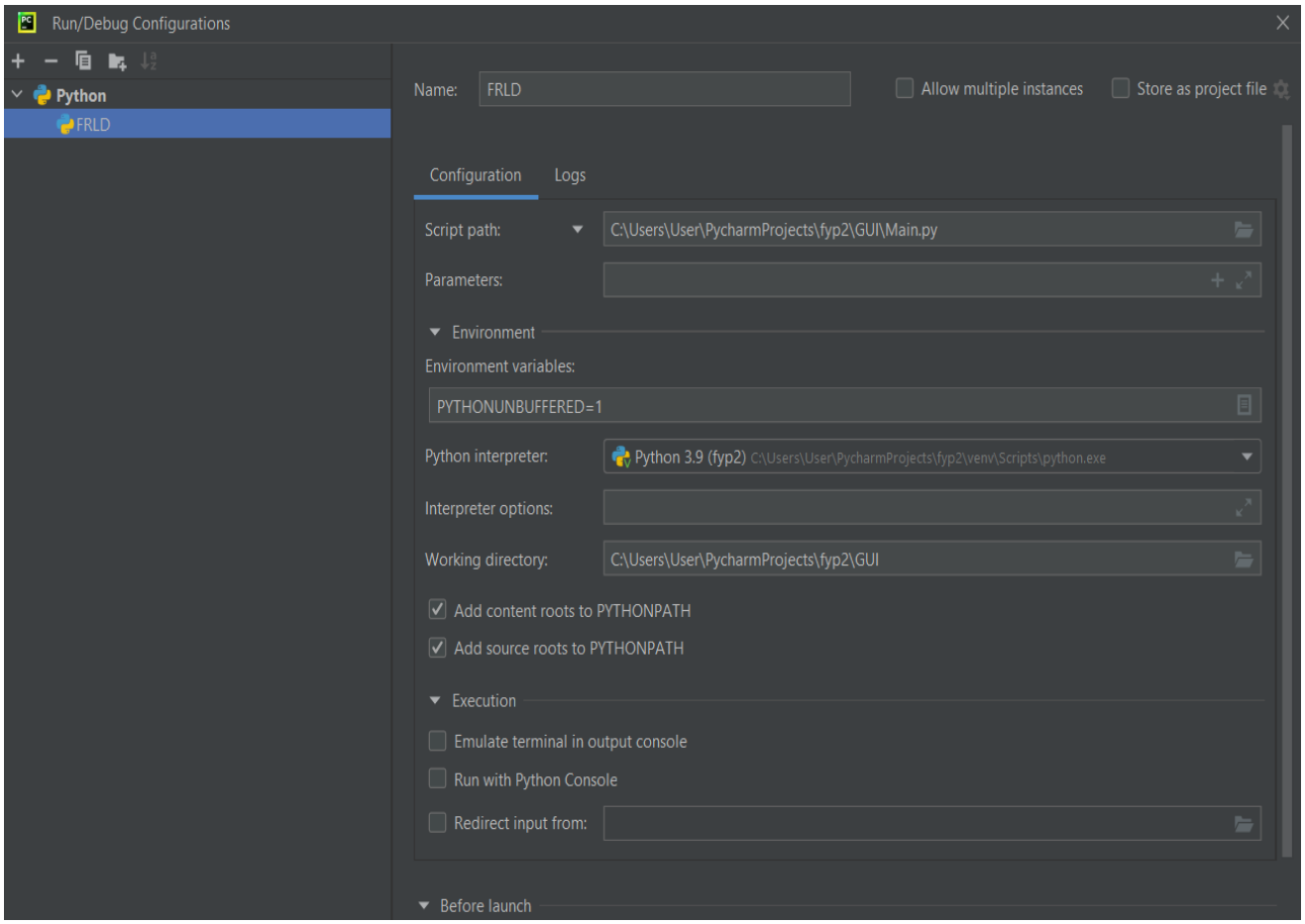


Figure 5.3.2.1 IDE Setting

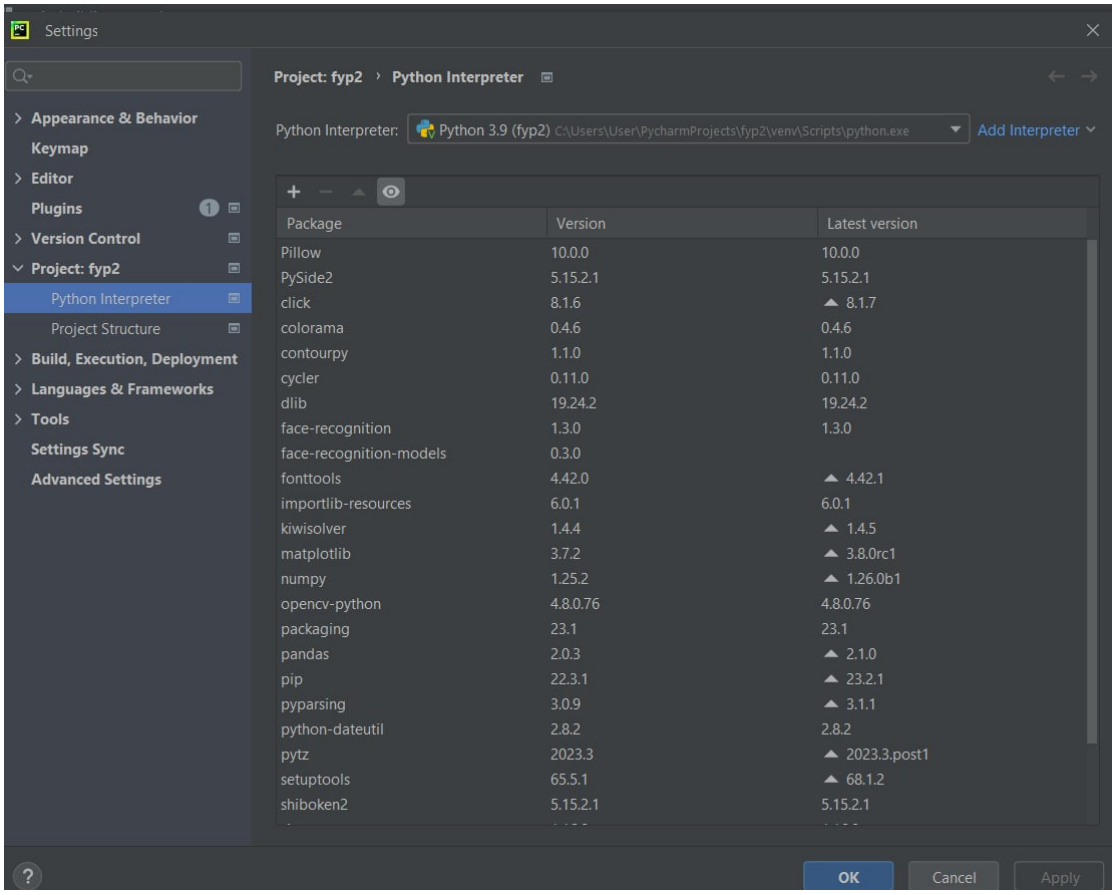


Figure 5.3.2.2 Python Installed Modules (1/2)

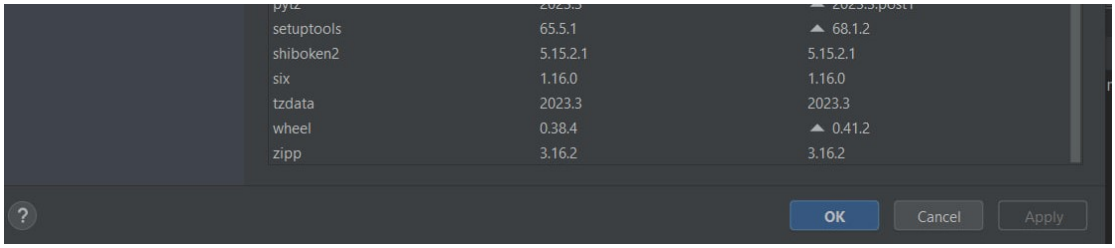


Figure 5.3.2.3 Python Installed Modules (2/2)

5.4 System Operation

Since it directly influences user experience and has a big impact on a product's success, user interface (UI) design is an essential part of developing software or systems. There are going to demonstrate the design and implementation of the user interface (UI) for the software application intended in this report but currently had successfully done development in our proposed system, with a focus on the crucial ideas of usability, aesthetics, and usefulness.

The UI for the proposed report is straightforward, easy to use, visually appealing, and consistent with the overall brand and design language. The UI is created with care, taking into account the needs and actions of the user as well as industry best practises, to satisfy these goals and enhance the overall user experience of the application. In the part that follows, the system will be shown, along with a thorough breakdown of the design choices and components that went into the finished item.

5.4.1 Login

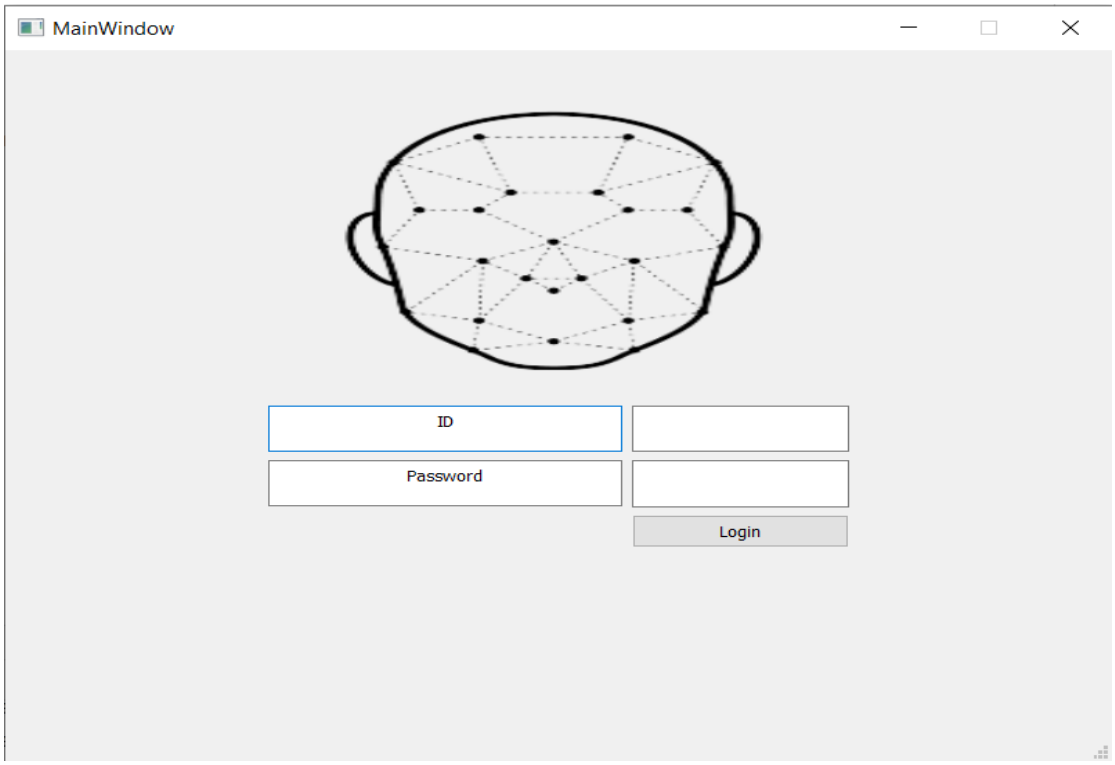


Figure 5.4.1 Login window

The login window is displayed in Figure 4.2.1. Our system does not support manual sign-up or the "forgot password" feature due to security concerns. The

system's main goal is to give authenticated users and authorised administrators access so they may administer, monitor, and modify the system.

5.4.2 MainCam

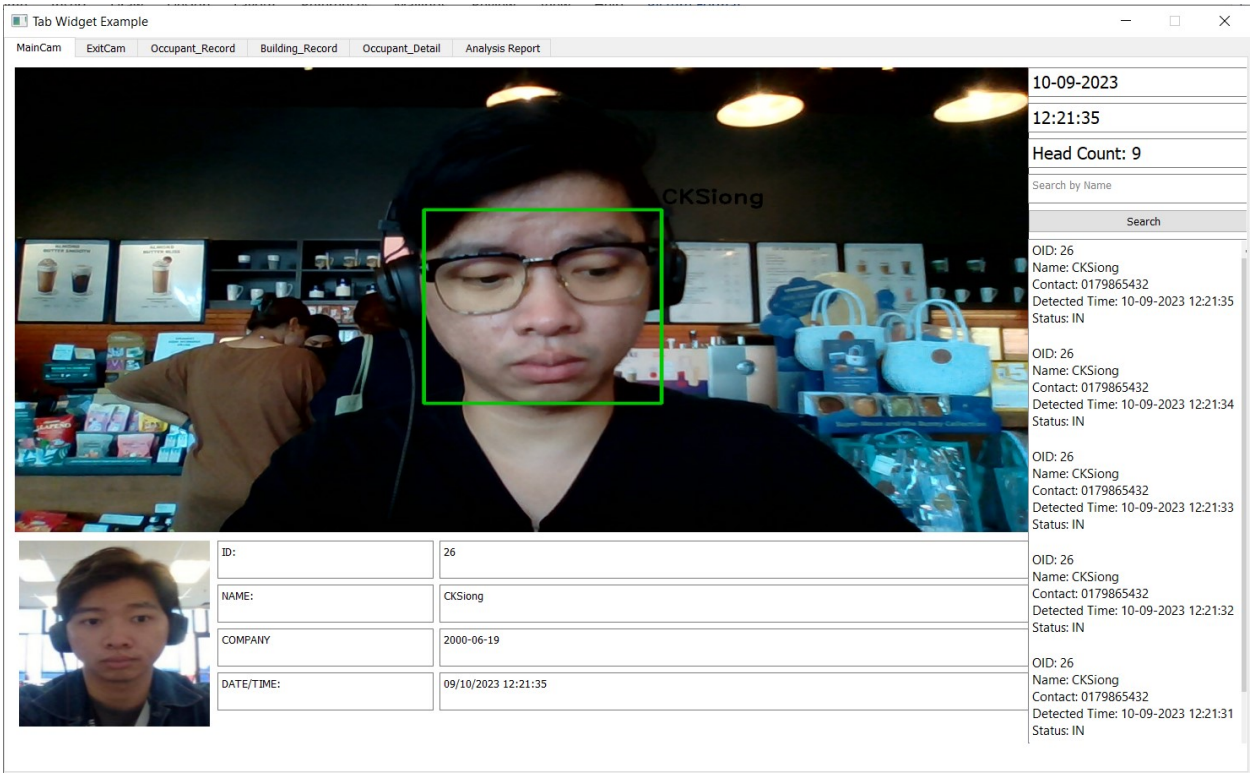


Figure 5.4.2.1 MainCam (1/3)

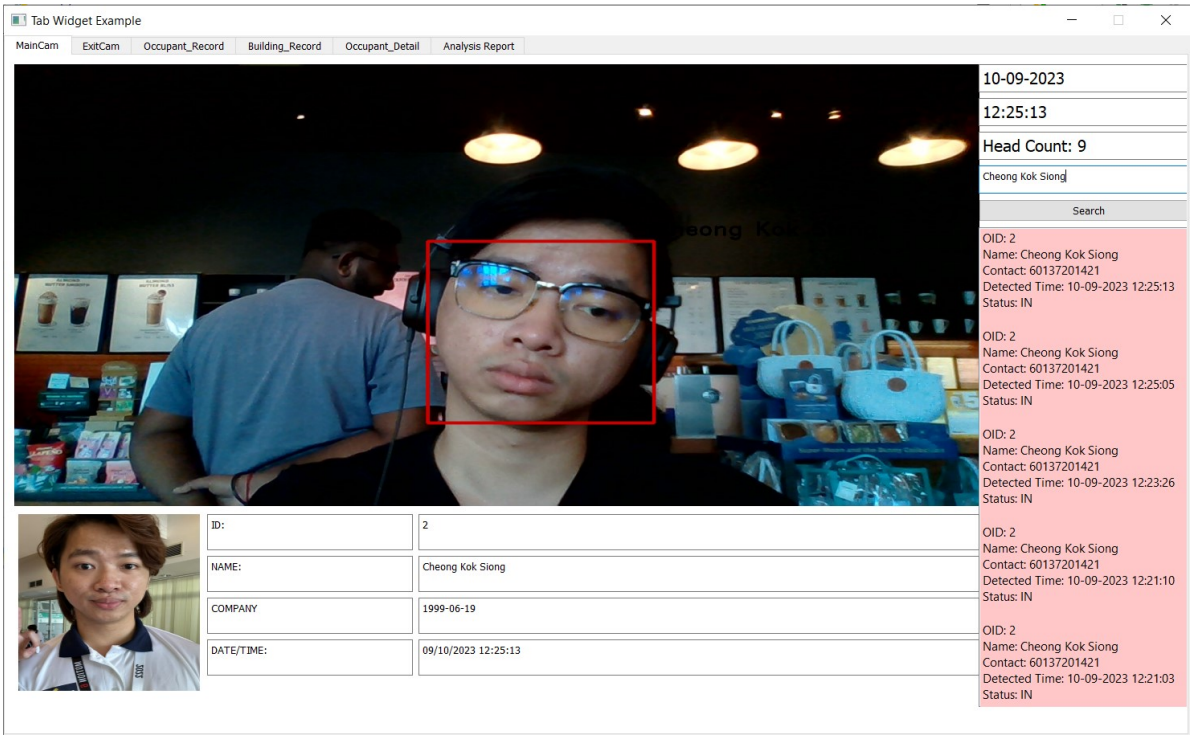


Figure 5.4.2.2 MainCam (2/3)

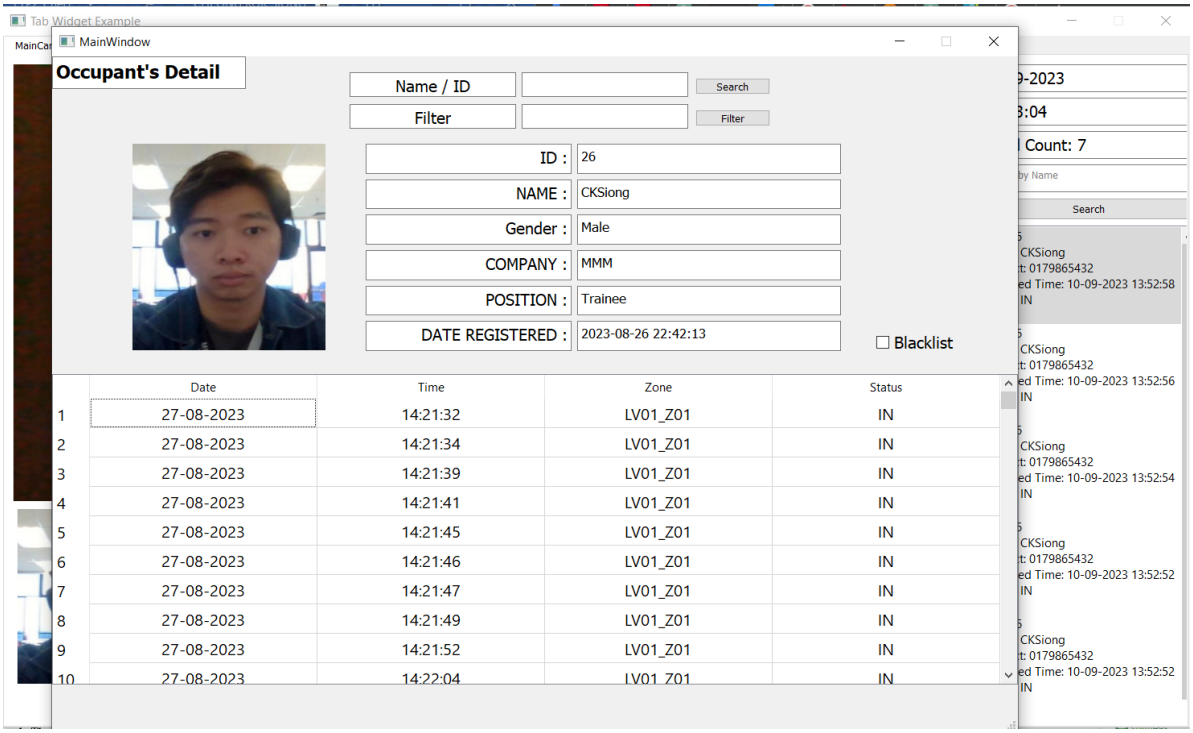


Figure 5.4.2.3 MainCam (3/3)

After a successful login, the system's main window is shown in Figure 5.4.2. This window gives you an extensive understanding of how the system works. Its main component is the entrance-focused main camera stream, with the identity of any detected occupant shown at the bottom. As occupants are identified, the occupant information is updated continuously to guarantee accurate real-time data. The window also displays the date, time, and current camera view at the top.

Accessible information is conveniently located to the right of the window. This includes a list of all detected occupants, listed historically with recent arrivals at the top, as well as the overall number of residents in that area. Notably, the system draws attention to people who have been blacklisted, alerting users to potential security issues and unauthorised entry. This rigorous attention to unregistered people strengthens security precautions and enables the quick and accurate detection of potential hazards in real-time.

The window also has a convenient search feature that enables users to quickly find residents by name or ID number. This function comes in quite handy in emergency situations because it makes it possible to quickly find out someone's information and make an emergency call. The clickable list widget on the right increases the system's usability and accessibility by bringing up a pop-up window with further information on the selected individual.

5.4.3 ExitCam

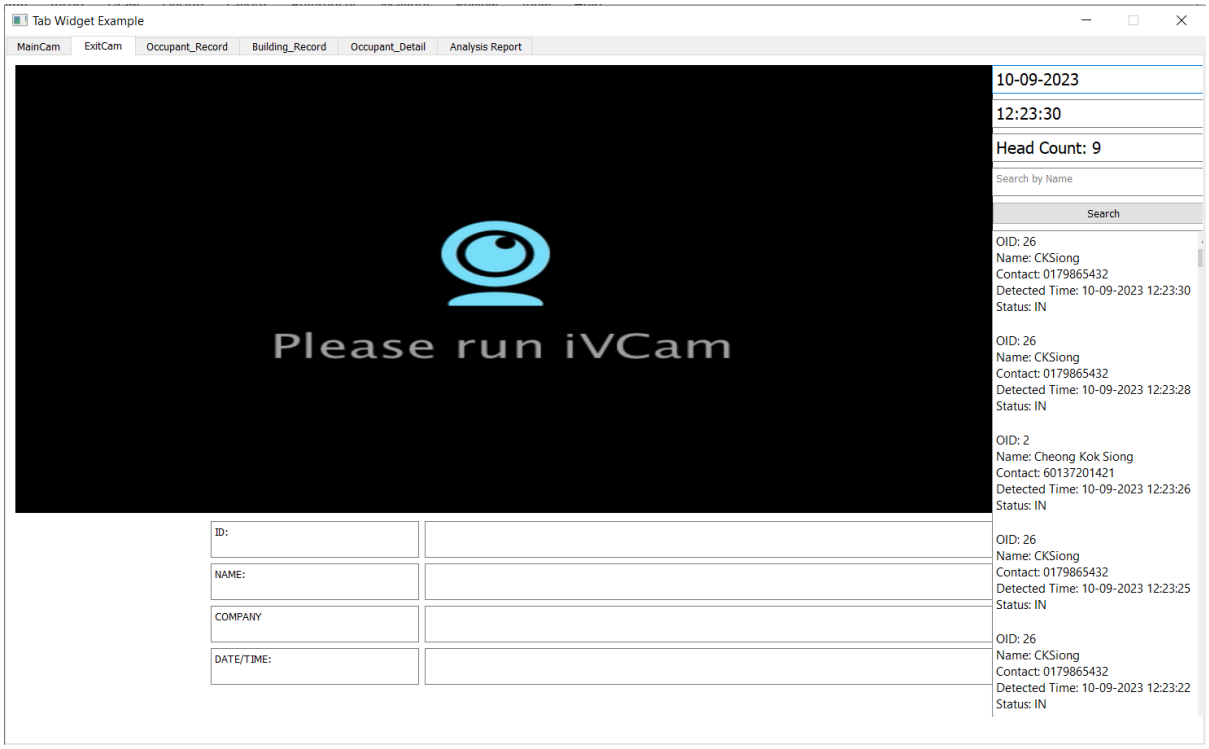


Figure 5.4.3.1 ExitCam (1/2)

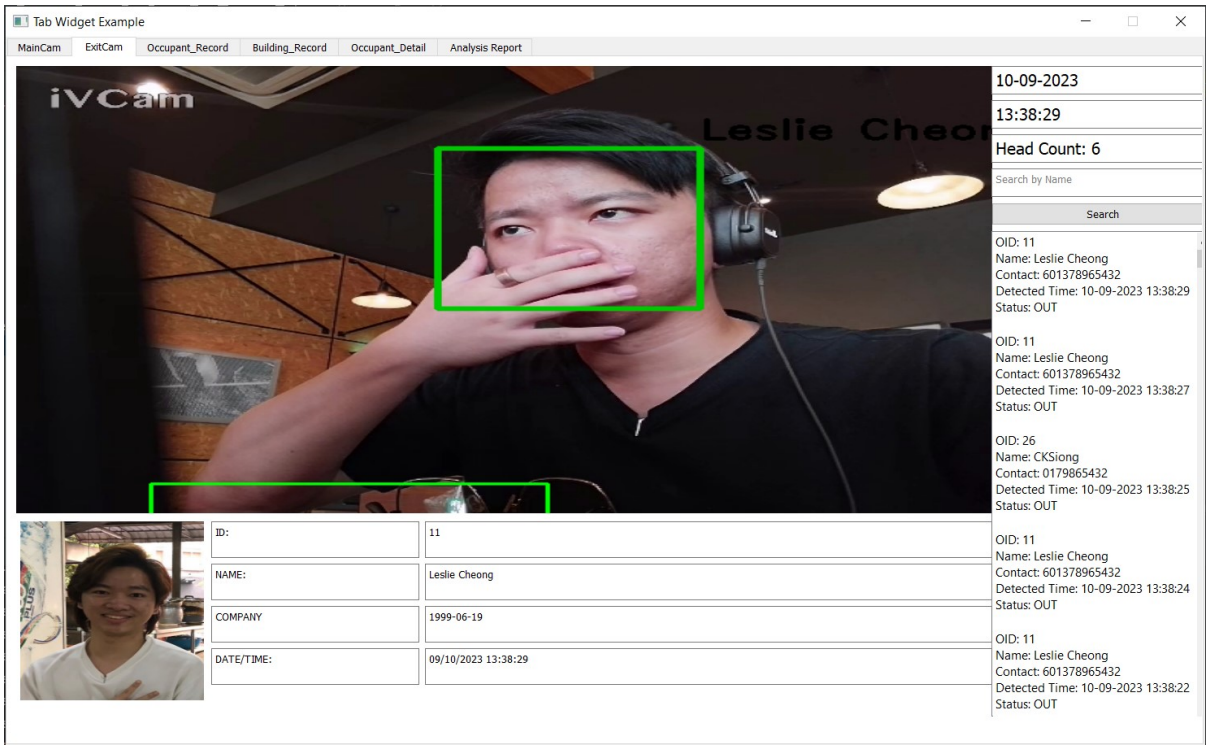


Figure 5.4.3.2 ExitCam (2/2)

Figure 4.2.3 is showing the “ExitCam” window, and it is important for comprehensive security and monitoring system. Its purpose is very similar to previous described window “MainCam”, it also displays a live video feed, but its focus is directed towards the exit area. When occupants are detected leaving through the exit camera, the system will promptly update their status as “OUT”, thus the movement of occupants inside the building can be tracked to this real-time status update.

5.4.4 Occupant_Record

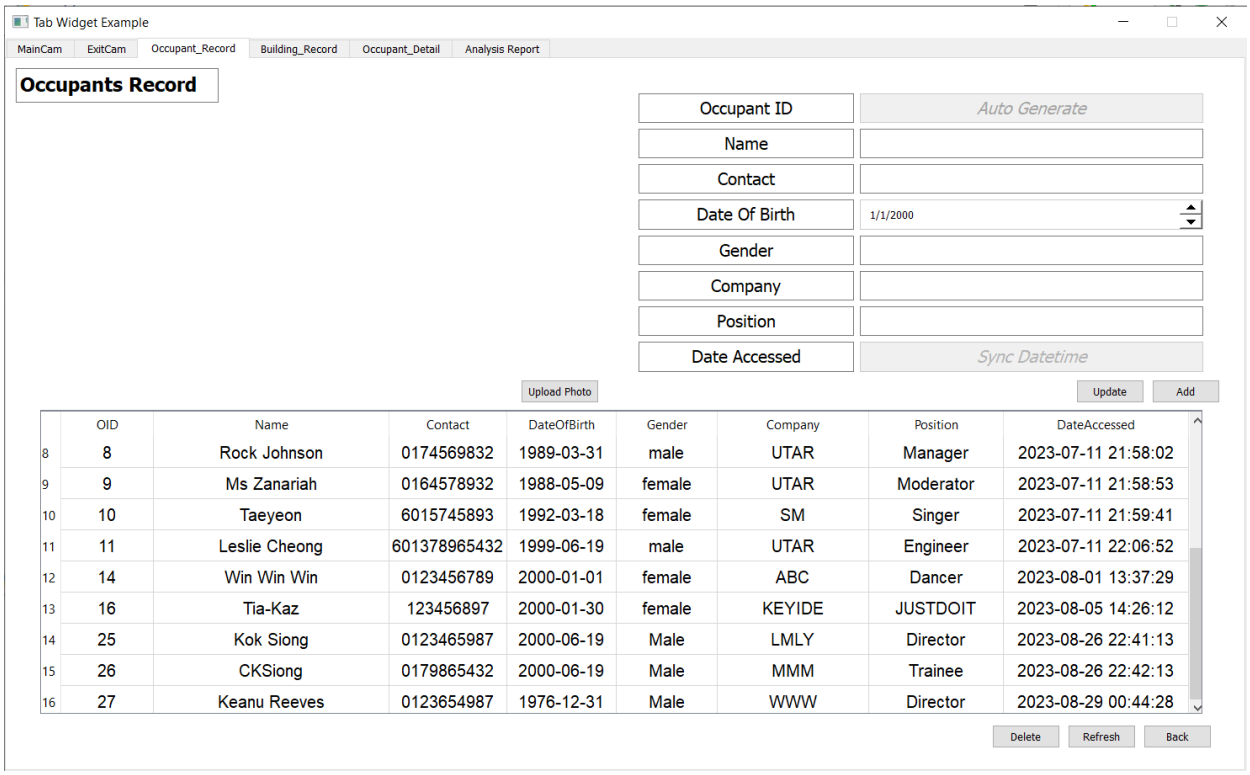


Figure 5.4.4.1 Occupant_Record (1/5)

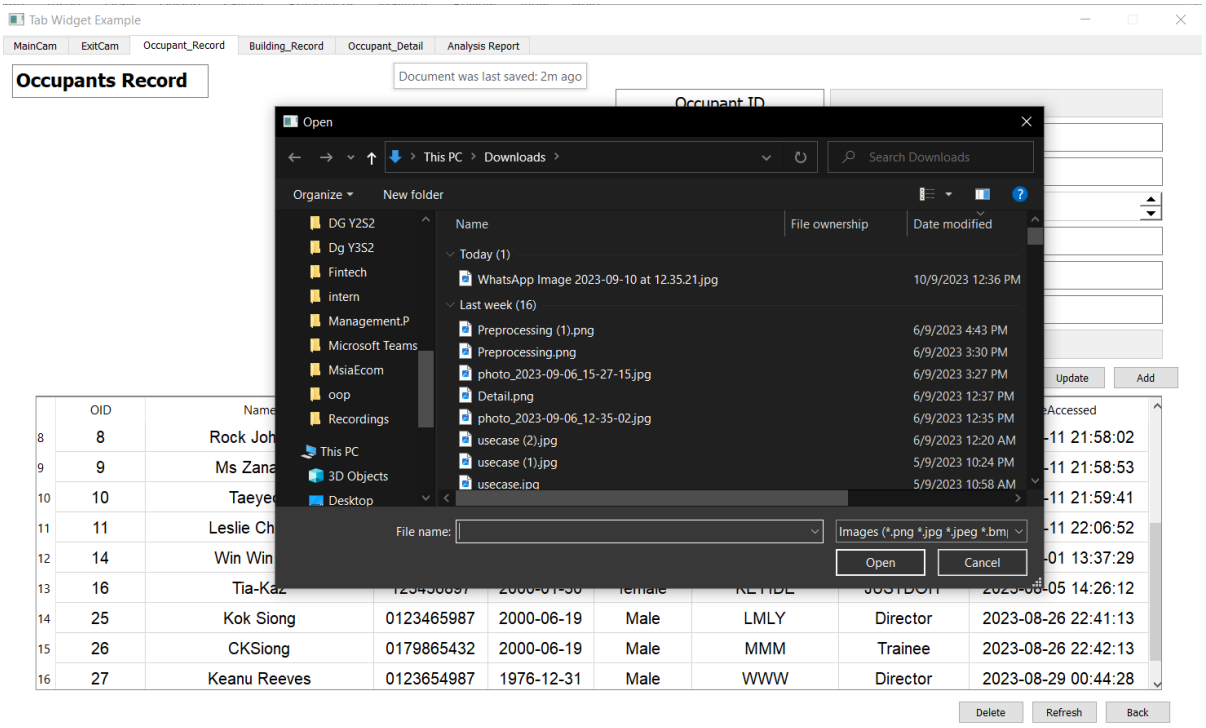


Figure 5.4.4.2 Occupant_Record (2/5)

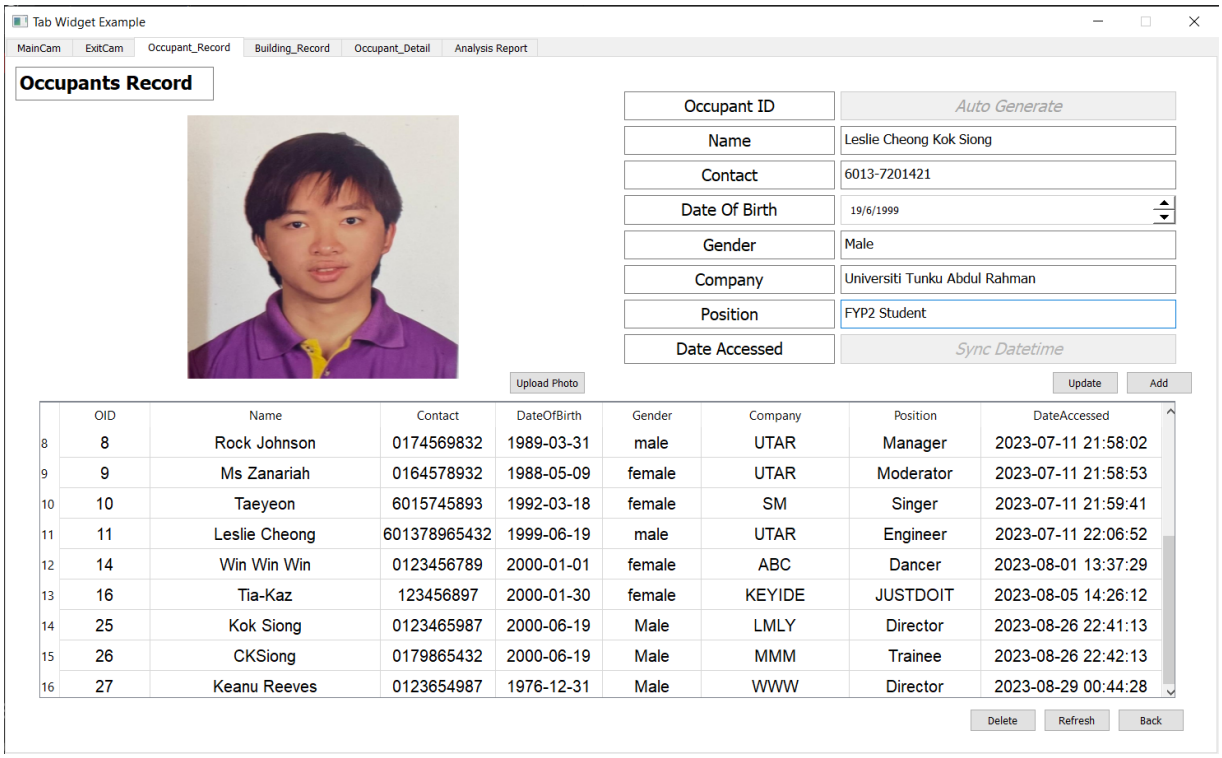


Figure 5.4.4.3 Occupant_Record (3/5)

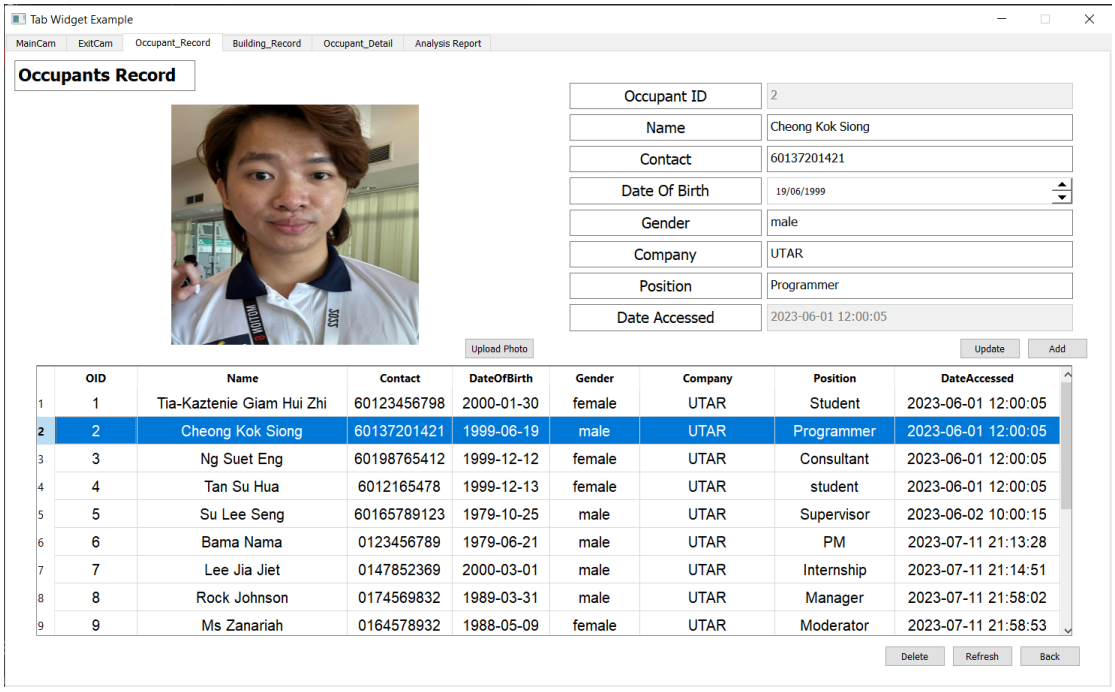


Figure 5.4.4.4 Occupant_Record (4/5)

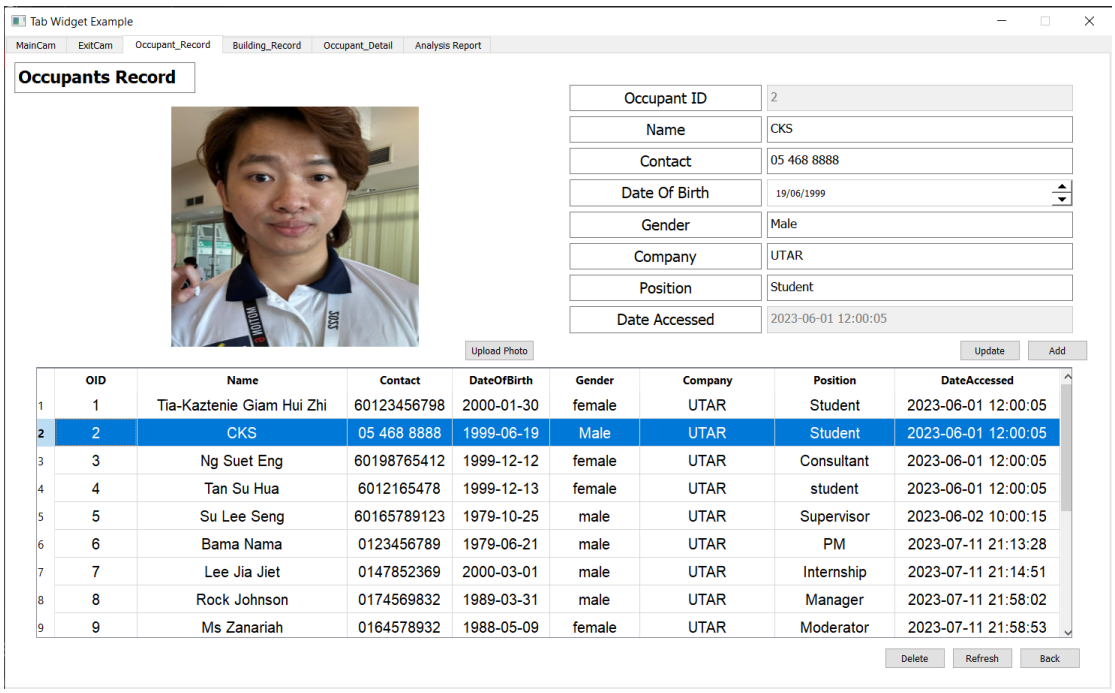


Figure 5.4.4.5 Occupant_Record (5/5)

In figure 5.4.4 features the “Occupant_Record” window, a versatile tool for administrators to add, update and delete occupant records. Admin able to upload occupant photo from his local directories. Once image is selected, admin can proceed to fill in the relevant occupant information. Once completing the details, a simple click on the “Add” button triggers the system to save the new record and update the table widget at below. The record in table widget is clickable, when click at particular record, the corresponding data is populated into the form above. This feature streamlines the process of modifying occupant details and finalize update by clicking “Update”. If willing to remove the occupant record, admin need only click the OID in the table, follow by clicking “Delete” button.

5.4.5 Building_Record

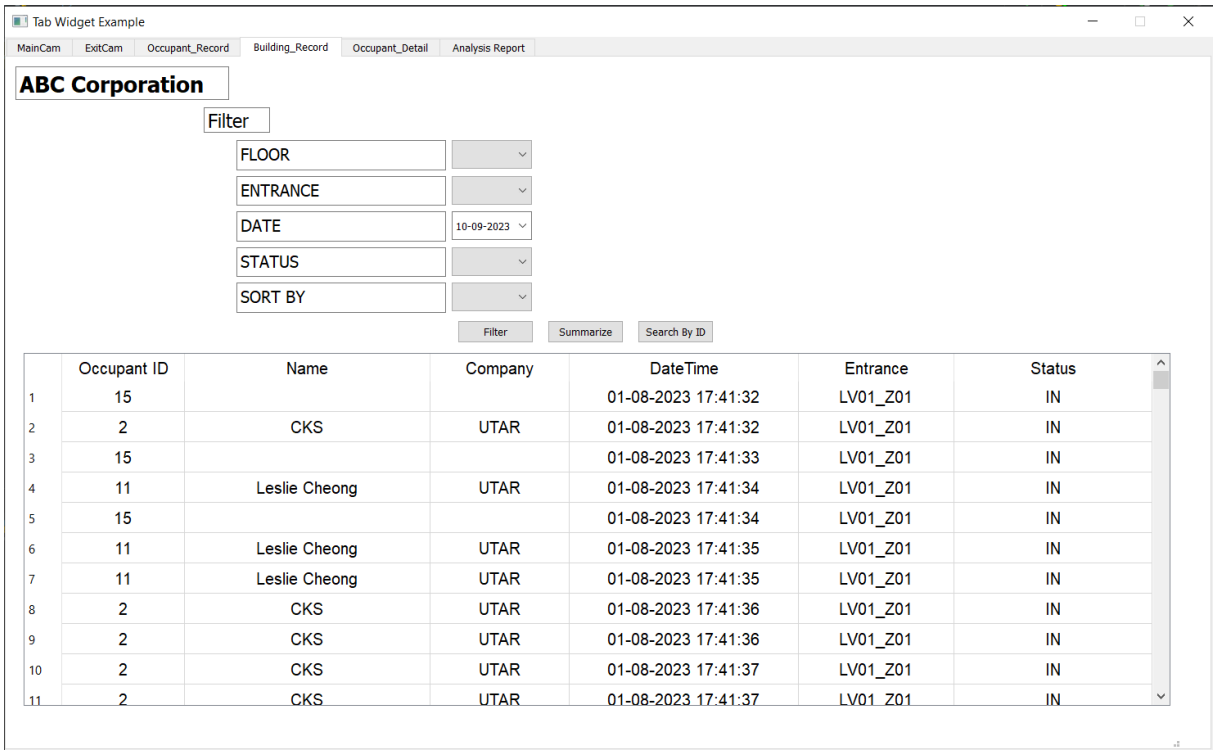
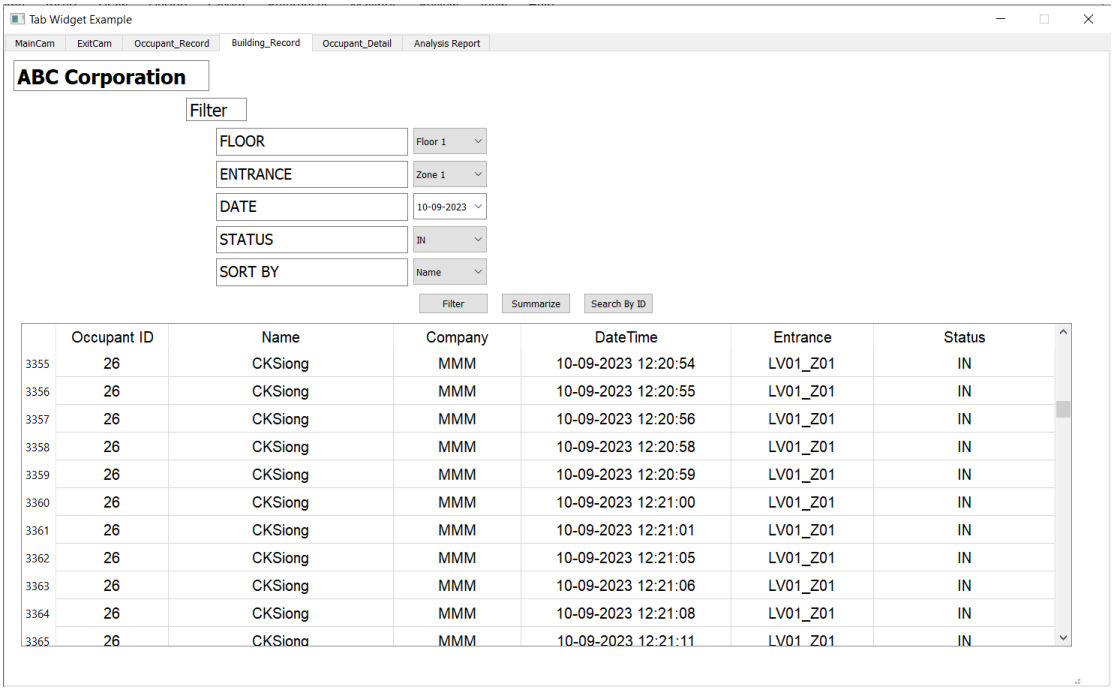
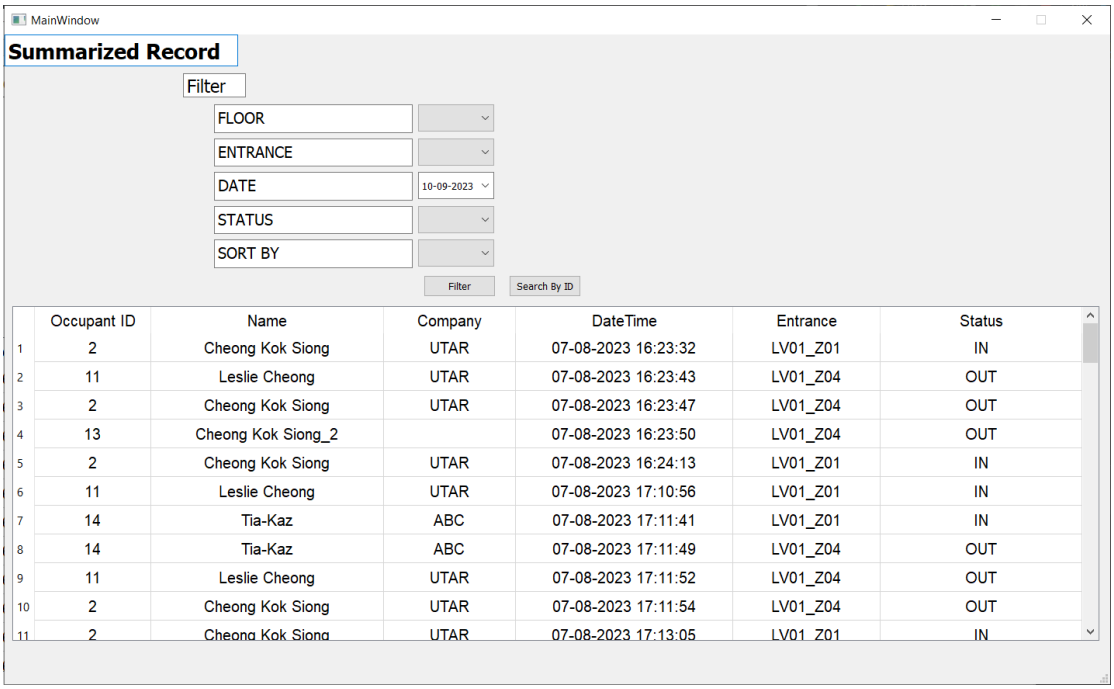


Figure 5.4.5.1 Building_Record (1/4)



	Occupant ID	Name	Company	DateTime	Entrance	Status
3355	26	CKSiong	MMM	10-09-2023 12:20:54	LV01_Z01	IN
3356	26	CKSiong	MMM	10-09-2023 12:20:55	LV01_Z01	IN
3357	26	CKSiong	MMM	10-09-2023 12:20:56	LV01_Z01	IN
3358	26	CKSiong	MMM	10-09-2023 12:20:58	LV01_Z01	IN
3359	26	CKSiong	MMM	10-09-2023 12:20:59	LV01_Z01	IN
3360	26	CKSiong	MMM	10-09-2023 12:21:00	LV01_Z01	IN
3361	26	CKSiong	MMM	10-09-2023 12:21:01	LV01_Z01	IN
3362	26	CKSiong	MMM	10-09-2023 12:21:05	LV01_Z01	IN
3363	26	CKSiong	MMM	10-09-2023 12:21:06	LV01_Z01	IN
3364	26	CKSiong	MMM	10-09-2023 12:21:08	LV01_Z01	IN
3365	26	CKSiong	MMM	10-09-2023 12:21:11	LV01_Z01	IN

Figure 5.4.5.2 Building_Record (2/4)



	Occupant ID	Name	Company	DateTime	Entrance	Status
1	2	Cheong Kok Siong	UTAR	07-08-2023 16:23:32	LV01_Z01	IN
2	11	Leslie Cheong	UTAR	07-08-2023 16:23:43	LV01_Z04	OUT
3	2	Cheong Kok Siong	UTAR	07-08-2023 16:23:47	LV01_Z04	OUT
4	13	Cheong Kok Siong_2		07-08-2023 16:23:50	LV01_Z04	OUT
5	2	Cheong Kok Siong	UTAR	07-08-2023 16:24:13	LV01_Z01	IN
6	11	Leslie Cheong	UTAR	07-08-2023 17:10:56	LV01_Z01	IN
7	14	Tia-Kaz	ABC	07-08-2023 17:11:41	LV01_Z01	IN
8	14	Tia-Kaz	ABC	07-08-2023 17:11:49	LV01_Z04	OUT
9	11	Leslie Cheong	UTAR	07-08-2023 17:11:52	LV01_Z04	OUT
10	2	Cheong Kok Siong	UTAR	07-08-2023 17:11:54	LV01_Z04	OUT
11	2	Cheong Kok Siong	UTAR	07-08-2023 17:13:05	LV01_Z01	IN

Figure 5.4.5.3 Building_Record (3/4)

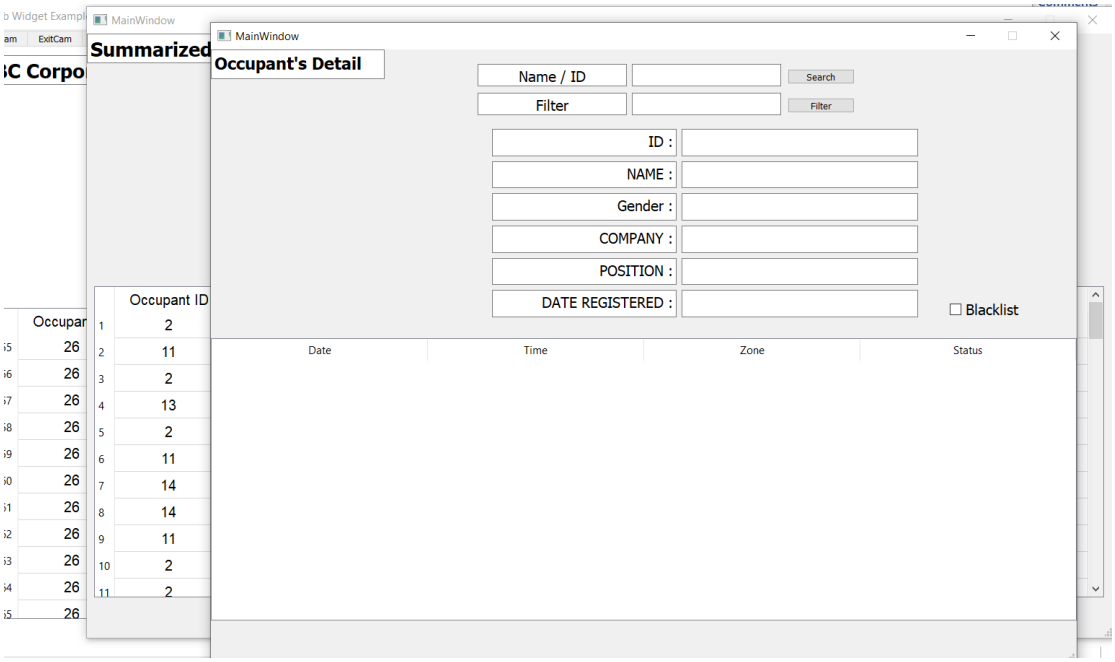


Figure 5.4.5.4 Building_Record (4/4)

Figure 5.4.5 showcases the "Building_Record" window, offering a comprehensive overview of all detected occupants within the building. This window provides filtering options for streamlined data access and summarized building records for quick insights. Furthermore, a search function empowers administrators to access detailed information on specific individuals in a separate window, enhancing data management capabilities.

5.4.6 Occupant's Detail

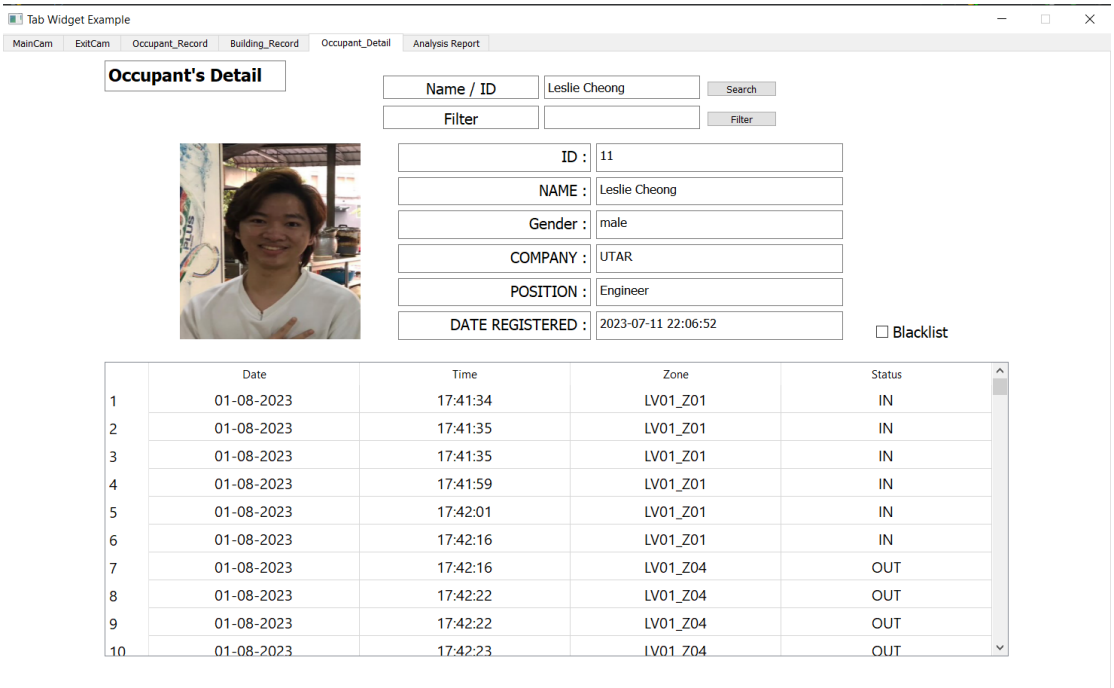


Figure 5.4.6.1 Occupant's Detail (1/3)

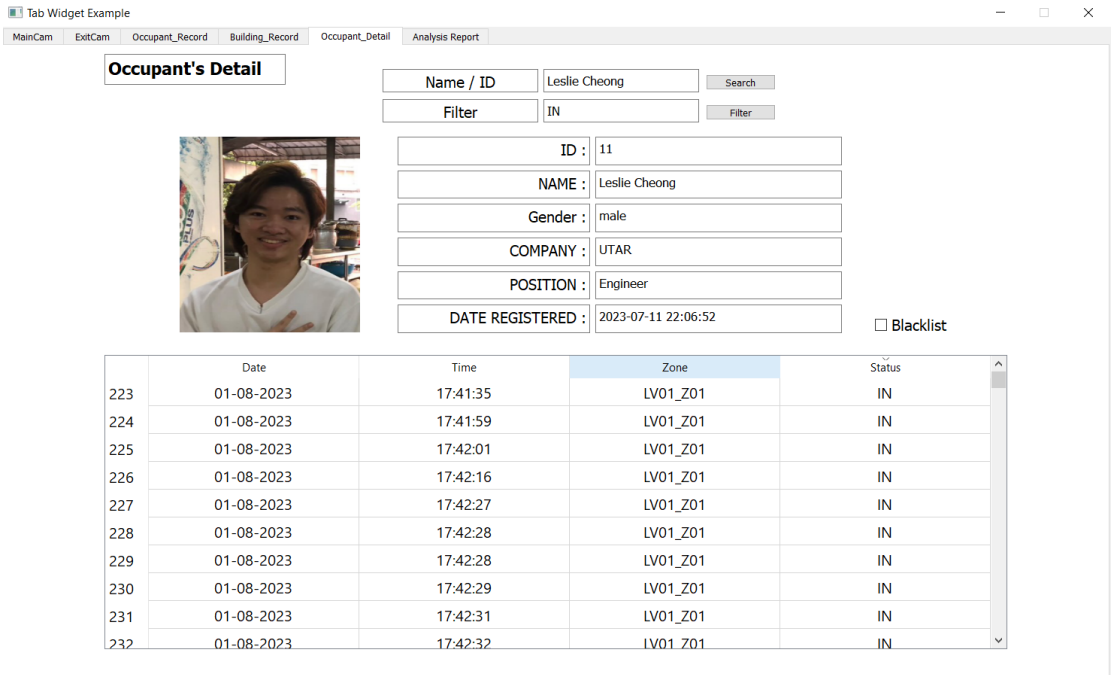


Figure 5.4.6.2 Occupant's Detail (2/3)

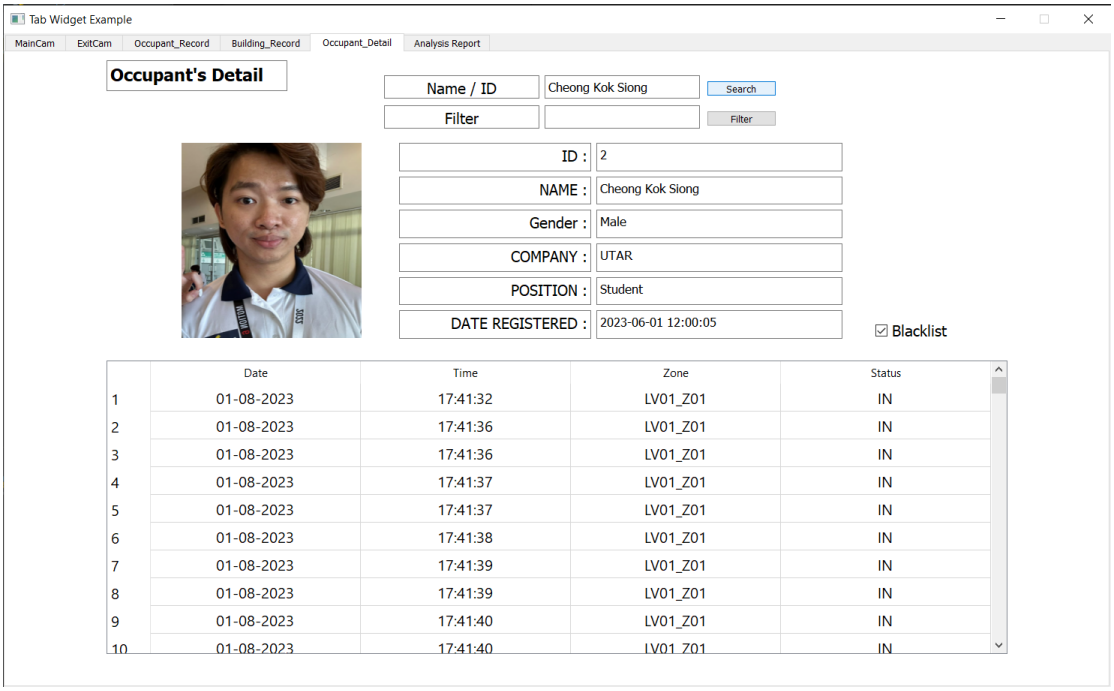


Figure 5.4.6.3 Occupant's Detail (3/3)

Figure 5.4.6 displays the "Occupant_Detail" window that enables administrators to access occupant details swiftly. The system retrieves and displays any relevant occupant's information when the user just enters the occupant's name or ID and clicks the "Search" button. Administrators can efficiently filter documents using keywords to better streamline their search. It also allow admin to blacklist occupants. Administrators can mark an occupant for blacklisting by checking a dedicated checkbox. A confirmation notification confirming this action is purposeful and authorised displays after you do so. This feature is especially useful for security concerns because it enables administrators to take action as needed.

5.4.7 Analysis Report

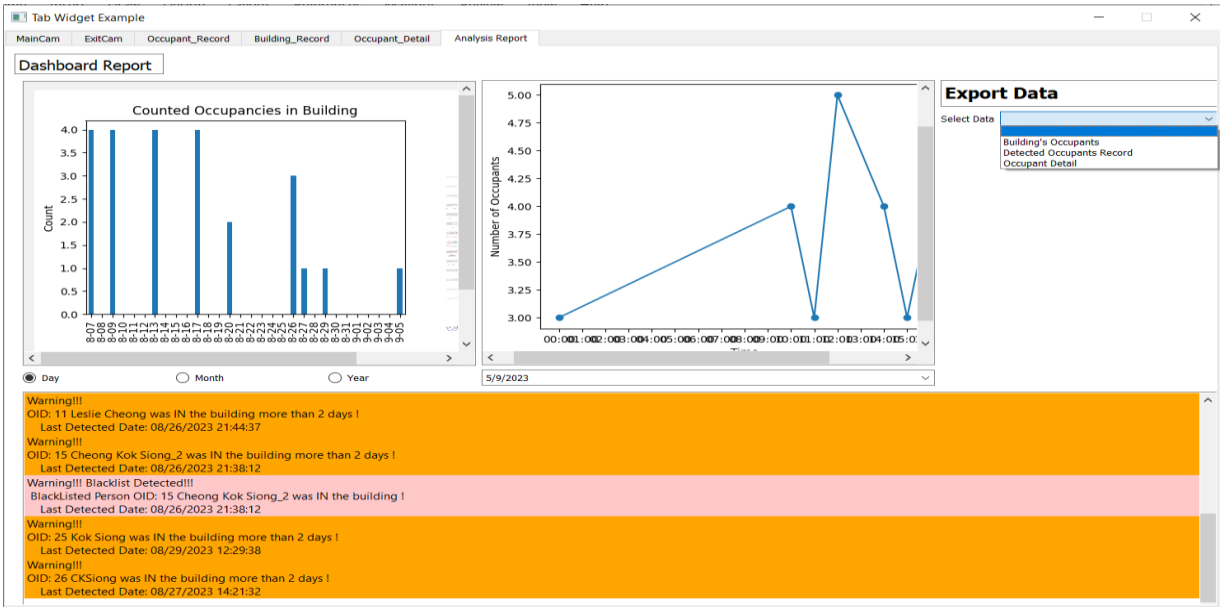


Figure 5.4.7.1 Analysis Report (1/2)

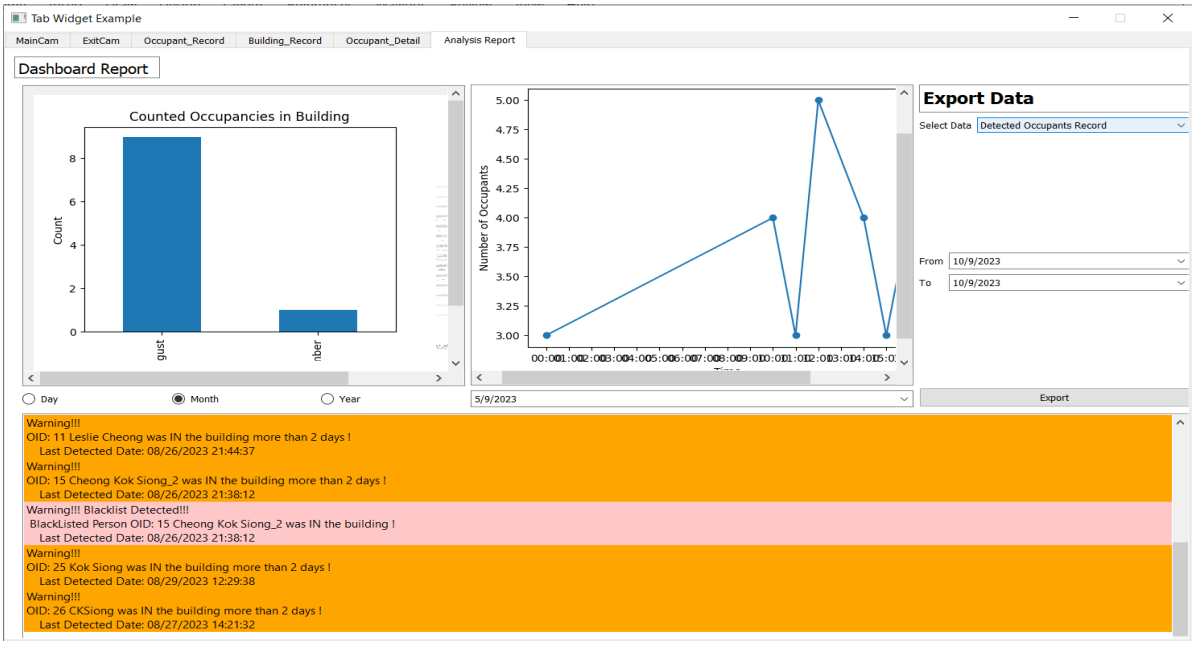


Figure 5.4.7.2 Analysis Report (2/2)

Figure 5.4.7 present a comprehensive overview of the analysed data, thoughtfully presented in both graphical and list widget format. This feature is designed to empower admin with valuable insights into occupant-related information. The system also equips admin with the ability to export this data with ease, depending on the selected dataset and generate CSV files, streamlining the process of archiving, sharing or further analysis.

5.5 Implementation Issues and Challenges

There are a number of challenges to development that slow down the process. First and foremost, the system's quality will be impacted and lowered by the accuracy of the facial recognition. Even though numerous algorithms have been developed to support the face recognition process, no one has been able to achieve the greatest performance of the suggested system expectations; it's either a speedy or high accuracy process.

Next, in the initial proposed work, all the data should be saving a cloud-based SQL database hosted on a remote VM server, and there are some issues may have stemmed from the network configuration and authentication then lead to connection problem. So, during the development process has change to local database due to simplified setup and reduced latency.

Additionally, in an effort to enhance accuracy, FRLD decided to employ multiple algorithms for the classification of multi-classes in face recognition. However, attempts to combine these algorithms for the most accurate results were unsuccessful. Running these algorithms simultaneously also necessitated a significant amount of RAM to support their execution.

Furthermore, web applications are more adaptable than computer applications because users may use them on a variety of digital devices, such as a laptop, tablet, and smartphone, to do tasks anywhere. Despite having a web server set up, creating a workable online application can create difficulties, such as trouble integrating Python server functions into the website. Django is a Python library that was used to create the web server. It can be difficult for a novice developer to do this assignment because it requires dealing with various programming languages, including HTML, JavaScript, CSS, and Python. The complexity of the work could cause misunderstandings and mistakes during the development phase. Thus, the development had changed to computer application since the connection to web applications is still a biggest challenge during the development phase.

CHAPTER 6

System Evaluation and Discussion

6.1 System Survey and Evaluation Result

To gain a better understanding of the system's performance, we invited up to 30 individuals to test the system and share their opinions through a prepared questionnaire. The analysis of the system's performance and evaluation results is based on the feedback we received from them, as outlined below:

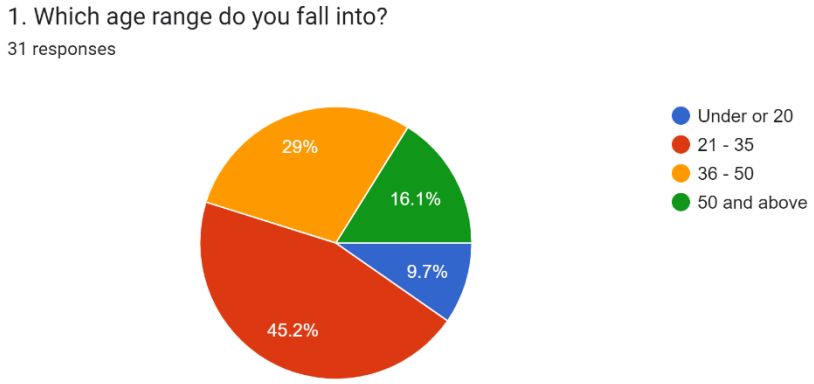


Figure 6.1.1 Survey Result (1/14)

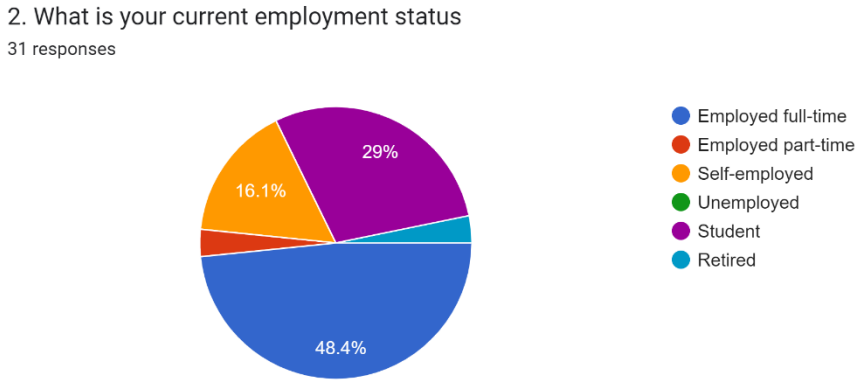


Figure 6.1.2 Survey Result (2/14)

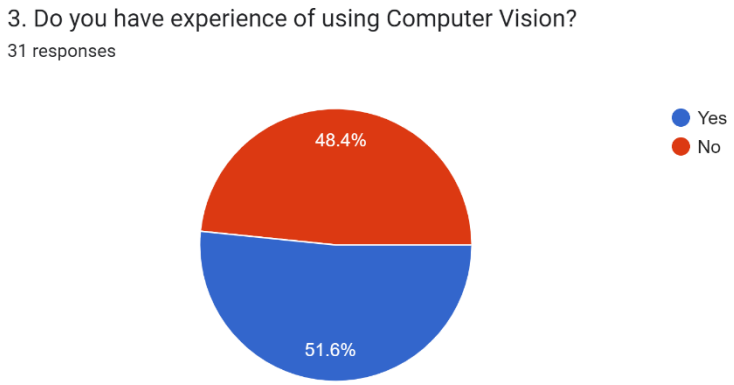


Figure 6.1.3 Survey Result (3/14)

Based on the responses to questions 1, 2, and 3, it can be observed that the majority of the respondents fall within the age range of 21 to 50. Additionally, a significant portion of these respondents are employed full-time. However, it's worth noting that only approximately 50% of the respondents reported having prior experience with Computer Vision technologies. By surveying those respondents, their opinions are making informed decisions about the future developments and updates based on the user demographics and experience.

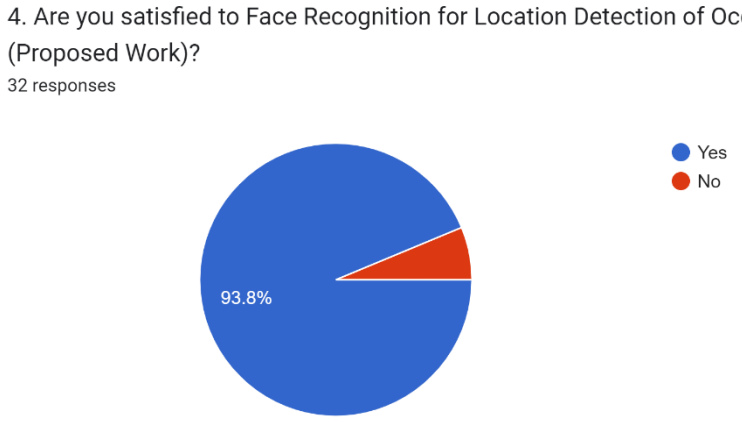


Figure 6.1.4 Survey Result (4/14)

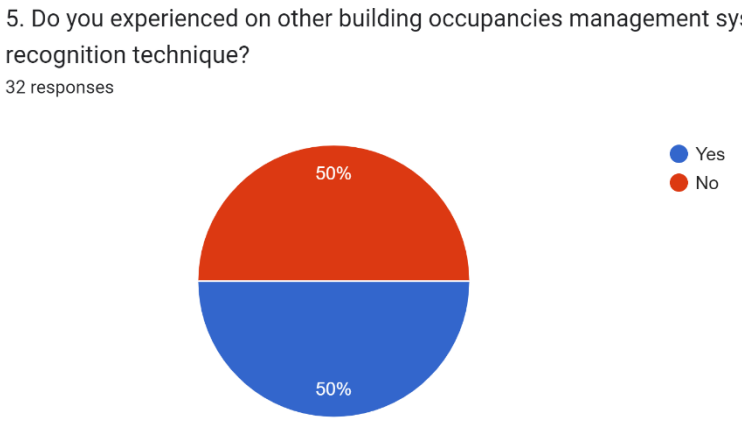


Figure 6.1.5 Survey Result (5/14)

Most of the respondents are satisfied on current developed system. However, there still have 2 respondents not satisfied on the system functions and features since two of them had experienced more expert system. Which mean two of them are able to provide more practical suggestions to improve the system.

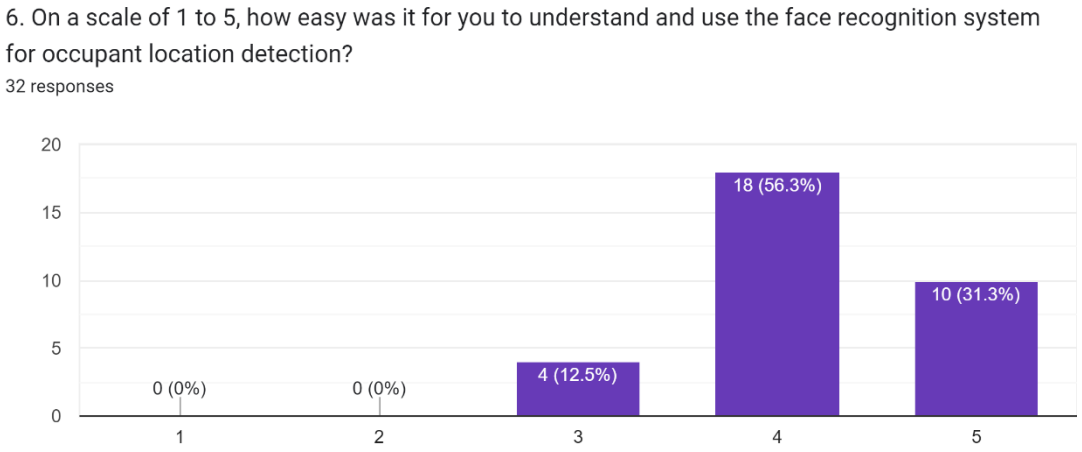


Figure 6.1.6 Survey Result (6/14)

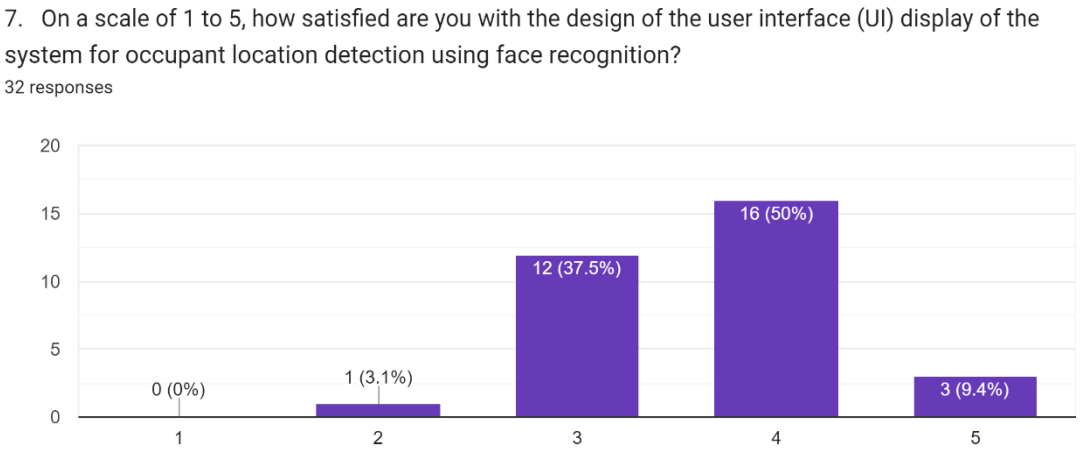


Figure 6.1.7 Survey Result (7/14)

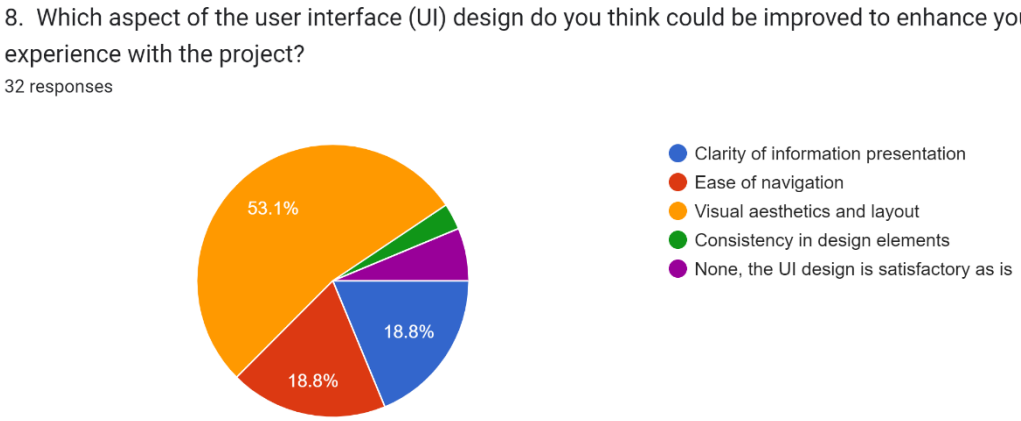


Figure 6.1.8 Survey Result (8/14)

Based on the findings presented in figures 6.1.6, 7, and 8, it's evident that a majority of users express satisfaction with the current User Interface (UI) design. However, valuable feedback and suggestions for further enhancement have been provided by users which are visual aesthetics and layout, ease of navigation and clarity of information presentation. These observations show how user feedback is used to drive continual improvement in a user-centric approach to system development. These suggestions can help the system develop so that it can give a better user experience and better match user expectations.

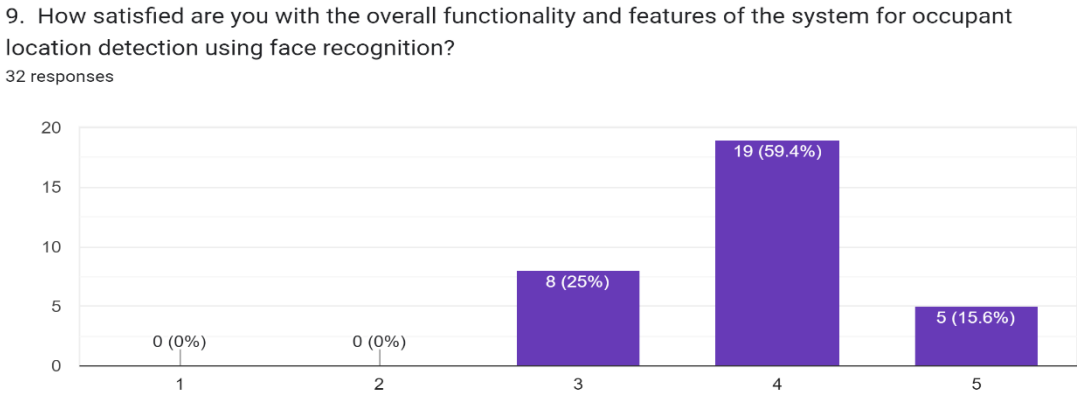


Figure 6.1.9 Survey Result (9/14)

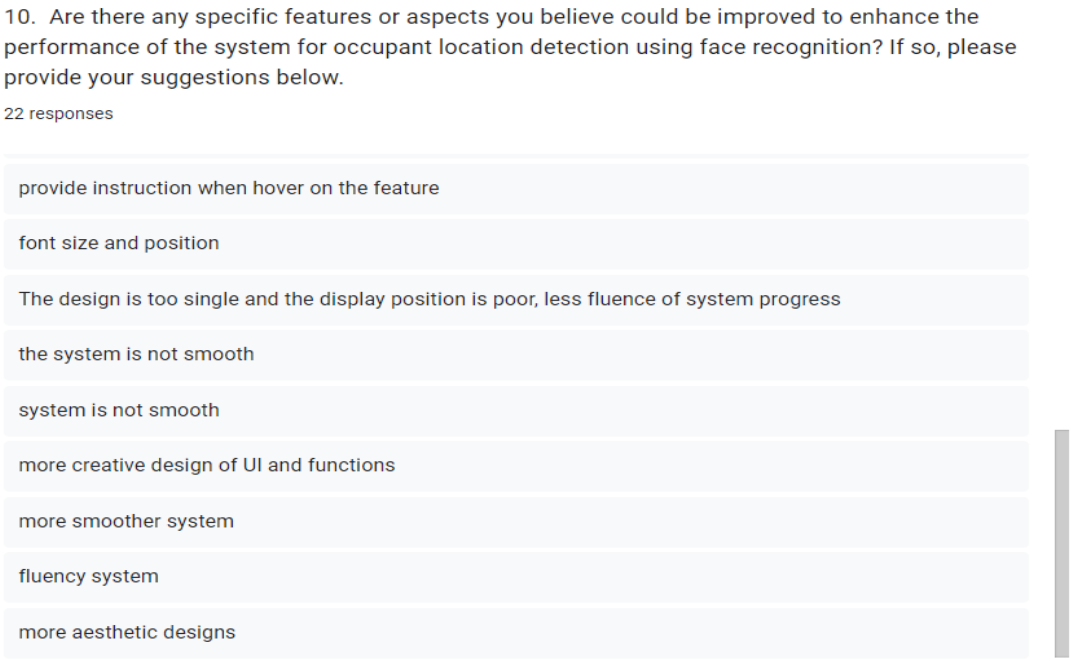


Figure 6.1.10 Survey Result (10/14)

Refer to figure 6.1.10 and 6.1.11, there are 75% of respondents are satisfied on overall functionality and features of the system, while the remaining 25% hold a neutral stance. Even they are satisfied the system but still interested on implementing more smooth system. It's interesting to notice that even among those who are satisfied there is a shared desire to improve the system even further in order to guarantee its smooth operation. This demonstrates the users' proactive use of the system's features and their willingness to accept ongoing improvement.

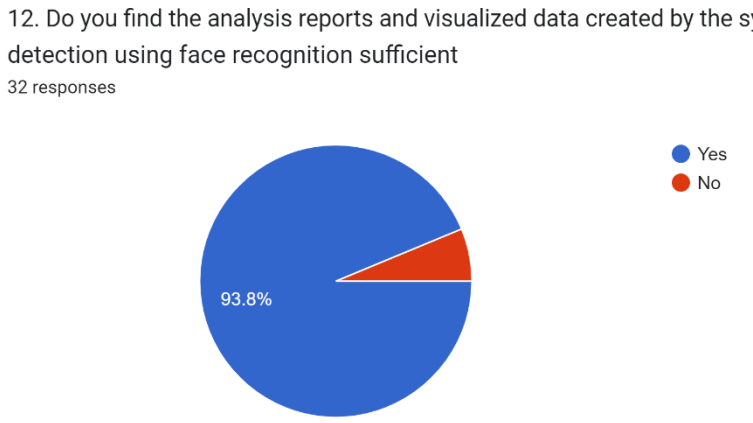


Figure 6.1.11 Survey Result (11/14)

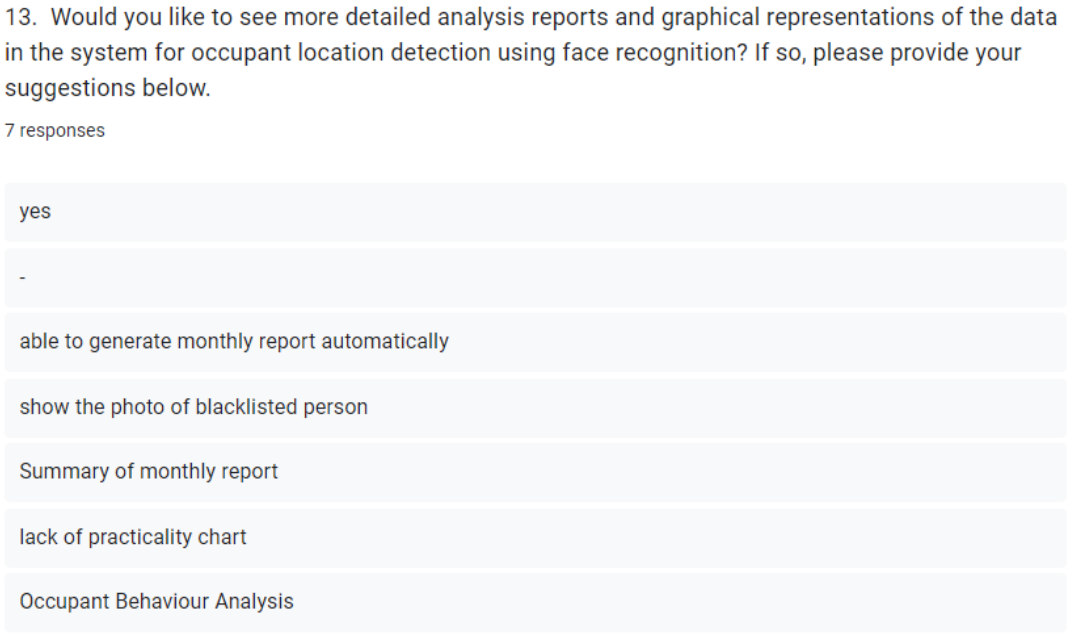


Figure 6.1.12 Survey Result (12/14)

Figure 6.1.11 finds the analysis report and data visualization provided by the system to be sufficient which mean the current reporting and data representation mechanism align well with their expectations and needs. However, in Figure 6.1.12 are showing that respondents express their willingness to view more specific reports such as summaries of monthly reports, occupant behaviour analysis and display of blacklisted individuals with accompanying photos.

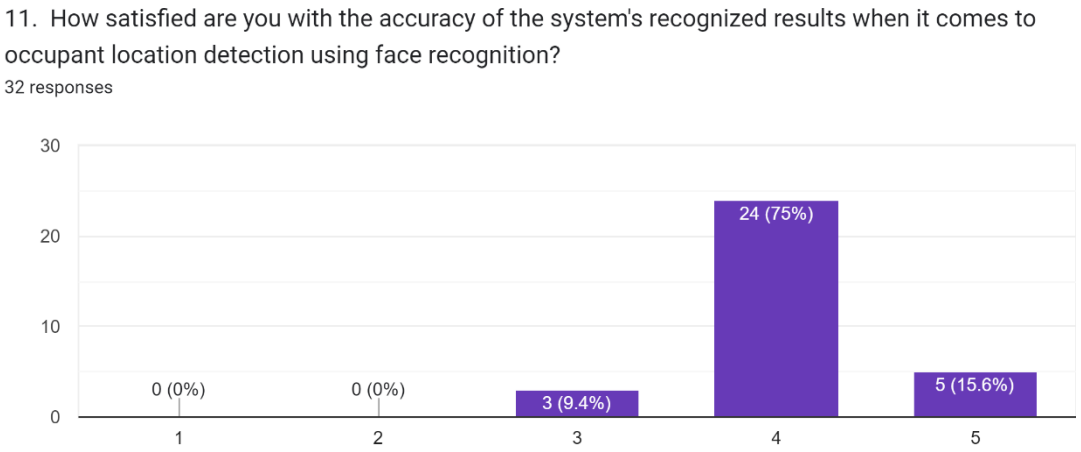


Figure 6.1.13 Survey Result (13/14)

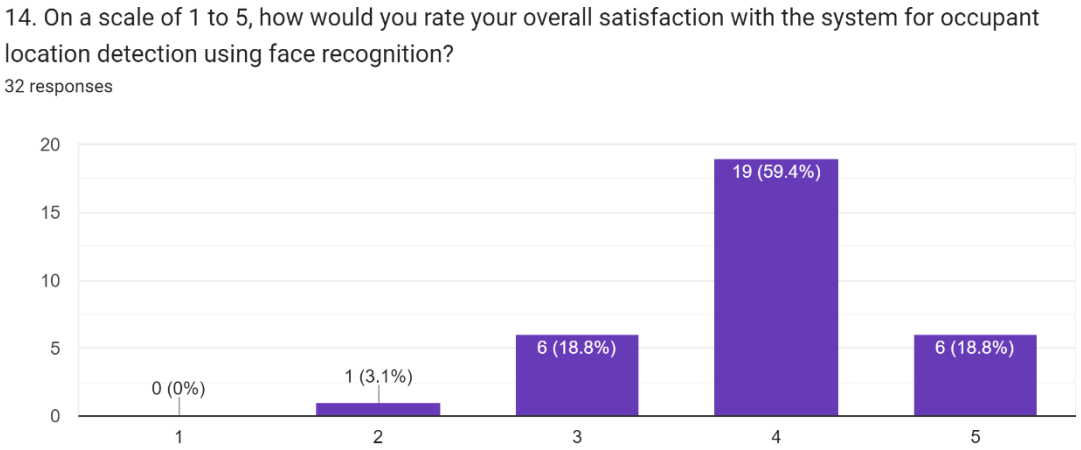


Figure 6.1.14 Survey Result (14/14)

Figure 6.1.13 and 6.1.14 are presenting the encouraging feedbacks on accuracy and general satisfaction point to the system's potential for success and user acceptability. To maintain and improve satisfaction with the product over time, it's important to keep focus on the user feedback and address any potential areas of improvement.

6.2 Testing Setup and Result

6.2.1 Testing Module 1 – Main.py

Objective : To ensure login function is success and will redirect to main_cam1 window.

Input	Expected Output	Actual Output
Login with pair predefined login credential	Success login and redirect to "main_cam1" window	User success to login notification and redirect "main_cam1" window
Login with blank input	Login fail notification pop out	Login fail notification pop out and stay at login window
Login with incorrect login credential	Login fail notification pop out	Login fail notification pop out and stay at login window

Table 6.2.1 Main.py

6.2.2 Testing Module 2 – main_cam1.py

Objective : To ensure main_cam1 able to capture recognized face and record in database.

Input	Expected Output	Actual Output
Camera record video feed	Detect and recognize registered occupants and show in table widget and list widget	Detect and recognize registered occupants and show in table widget and list widget, save in database and status is "IN"
Click item in list widget	Display occupant's detail in separate window	Display occupant's detail in separate window
Retrieve data from database and count the total available occupants within building	Total number of occupants where status is "IN"	Count correctly the total number of occupants where status is "IN"

Filter occupant in the list widget by using occupant's name	Only display occupant's detail where name matched	Only display occupant's detail where name matched
Check blacklisted occupants	Displayed occupant's item in list widget will be red background	Displayed occupant's item in list widget will be red background

Table 6.2.2 main_cam1.py

6.2.3 Testing Module 3 – main_cam2.py

Objective : To ensure module main_cam2 is able to update the occupants' status as "OUT".

Input	Expected Output	Actual Output
Camera record video feed	Detect and recognize registered occupants and show in table widget and list widget	Detect and recognize registered occupants and show in table widget and list widget, save in database and status is "OUT"

Table 6.2.3 main_cam2.py

6.2.4 Testing Module 4 – main_occupants.py

Objective : To ensure login function is success and will redirect to main_cam1 window.

Input	Expected Output	Actual Output
Retrieve data in table "Occupants" from database	Retrieved data will display at table widget below.	Retrieved data displayed at table widget below.
Click the item in table	Data in clicked row will fetch into form at above	Data fetched in the defined form
Upload photo	Photo in local storage able to upload and display in the window	Photo is displayed and saved at the frame.

Add new occupant	Detail in form and photo in frame will save inside database.	The database will save the details from the form and the picture in the frame.
Update the occupant	Update the occupant's detail in database	The occupant's detail in database is updated with new value.
Delete the selected occupant	Remove the selected occupant from the table in database.	Remove the selected occupant from the table in database.

Table 6.2.4 main_occupants.py

6.2.5 Testing Module 5 – main_building_record.py

Objective : To ensure the module able to manage and display data related to building occupants.

Input	Expected Output	Actual Output
Filter the displayed data by select certain condition	System will only display the data where matched the selected condition.	System will only display the data where matched the selected condition.
Sorting the displayed data by selecting the table header	The data will display in ascending based on selected header	Based on the selected header, the data will display in ascending order. But will have sorting error when "OccupantID" selected
Clicked "Summarize" button	Summarized data which is transaction changed will only displayed in separate window	Summary information that has undergone a transaction change will only be shown in a separate window.
Clicked "Search by ID" button	Occupant's detail window will pop out	Occupant's detail window will pop out

Table 6.2.5 main_building_record.py

6.2.6 Testing Module 6 – main_occupant_detail.py

Objective : To allow user to search for and display occupant's detail information and add/remove them to backlist if needed.

Input	Expected Output	Actual Output
Fill in the occupant's name and click search button	System will retrieve data where name is matched in form and table	System successfully retrieve data where name is matched in form and table
Fill in the keyword at filter text edit place and click the filter button	Only the relevant row of data will be show in table.	The table only display the row of data where include the keyword.
Check the blacklist checkbox	Occupant will add inside the blacklist table in database	Occupant had been update as blacklisted inside database.

Table 6.2.6 main_occupant_detail.py

6.2.7 Testing Module 7 – main_report.py

Objective : To ensure those data are visualized in window to provide insights to user.

Input	Expected Output	Actual Output
Retrieve data from database and visualize in window	Data will visualize in line graph and bar graph	The line graph and bar graph based on data in database generated correctly
Click the daily ratio button	Bar graph will display data in daily	Daily counted occupancies in building generated in bar graph
Click the month ratio button	Bar graph will display data in monthly	Monthly counted occupancies in building generated in bar graph

Click the year ratio button	Bar graph will display data in yearly	Yearly counted occupancies in building generated in bar graph
Select the date for occupants availability	Trend graph about occupants availability on selected date will generate	A trend graph about the chosen date occupants availability will generated.
Select "Building's Occupant" from combo boxes at ExportData	Predefined Building's Occupants form display	Predefined Building's Occupants form display
Export Building's Occupant data	A excel file with building's occupant data and path with all occupant images generated	A excel file with building's occupant data and path with all occupant images generated
Select "Detected Occupants Record" from combo boxes at ExportData	Predefined Detected Occupants Record form display	Predefined Detected Occupants Record form display
Export Detected Occupants record	A excel file with detected occupants record within the selected date range will generate.	A excel file with detected occupants record within the selected date range is generated.
Select "Occupant Detail" from combo boxes at ExportData	Predefined Occupant Detail form display	Predefined Occupant Detail form display
Export Occupant Detail	Selected occupants detail data will export in CSV format	Selected occupants detail data is exported in CSV format

Table 6.2.67 main_report.py

6.3 Project Challenges

The quality of camera will also affect the accuracy of the result since low-quality camera may face challenges such as pixelation, blurred images and poor low-light performance which can hinder the accuracy of reliability of FRLD. Additionally, face recognition system has high hardware requirements for computer based on the system's core practical data mining, machine learning and deep learning. A normal household hardware computer is difficult to support smooth system operation. In addition, in our pursuit of optimizing system performance and enhancing its overall functionality, obtaining and utilizing a large amount of data has become crucial. Obtaining and using a lot of data has also become essential in our quest to improve the functioning of the system overall and optimise system performance. However, if data needs to be obtained in real life, the system must be operated in the market and installed in a place with pedestrian flow in order to collect useful data and the development process and tools are ordinary computers, which cannot support the system to operate for a long time.

Developing facial recognition technology requires a deep understanding of data mining and algorithm in order to innovate and achieve a higher probability of recognition success. Due to a limited understanding of data mining and computing, it can only be flexibly applied and extended for usability on the technology of other developer.

6.4 Objective Evaluation

We have conducted up to 30 responses survey to evaluate the satisfaction of about the developed system. Most of the respondents are satisfied in current development which able to satisfy and implement in actual building for testing. Based on the results of surveys, there are 90% of responses satisfied with the accuracy of the system's recognized result and 15% of them are very satisfied, and there are no responses are below the neutral value.

The proposed system is aimed at introducing innovative soft biometric features and efficient face recognition algorithms for the precise location detection of building occupants. In today's tech-driven word, these advancements not only improve our quality of life but also minimize human errors and actions in various tasks. It is leverages face detection and recognition within building and corporate management systems, enhancing their performance through application of machine learning and deep learning in location detection processes.

To review, the project objectives are:

- Create a smart building environment using modern face recognition technology to reduce the interaction with occupants while streamlining data automation and integration for seamless management.
- Utilizing modern face recognition technology to significantly enhance the speed, precision and comprehensiveness of real-time data capture.
- Utilise modern face recognition technology to efficiently reduce security concerns and streamline the data collection process.

In conclusion, the developed system is mostly meet those objectives. But those objectives are still can do it better and better especially the first objective. The concept of a smart building environment should function as a reinforcement learning machine. Its primary purpose is to continuously train on recorded data and learn from occupants' behavior. This learning process empowers the system to efficiently manage resources, such as electricity, in order to optimize overall building utility and functionality. However, the developed system is only capable of visualizing recorded data, and certain functions still require manual human control, such as uploading an occupant's photo during registration.

CHAPTER 7

Conclusion and Recommendations

7.1 Conclusion

This project is a building occupant position detection system that applies facial recognition technology to improve performance. This project also has explored the development of innovative soft biometric features and efficient algorithms to enhance the precision and efficiency of such systems to improve human living standard an reduce human error.

Face recognition is an advanced technology that has improved the quality of human life since the development of digital cameras. The existing location detection system requires additional equipment to perform tasks or be affected by the environment. However, the combination of the two can maximize each other's practicality and complement each other's shortcomings and weaknesses, creating a more valuable system.

The developed system only able to create smart building environments or reduce the interactions between occupants and building operation, but also find critical applications in specialized setting such as hospital or educate institution. For instance, locating doctors, nurses and patients is important for efficient patient care and reduce unnecessary medical accidents.

Furthermore, the developed system also improves the speed, precision and comprehensive of real-time data capture by identifying occupants. Once success on implementing reinforcement learning in the system and the valuable insight provided by system into occupancy pattern and trends will leading to more effective and efficient building management.

In conclusion, the study described in this proposed project not only makes improvement to the location detection for building’s occupants but also highlights how face recognition technology has the potential to completely change how we manage and protect our environments. The combination of advanced solutions presented is expected to change the future of building management and occupant safety as technology advances.

7.2 Recommendation

The suggested system's potential for improvement is found via the system evaluation survey. The suggested system was an early finished prototype that offered fundamental features. Refer to the survey, there are few requests for improvement are about the UI design which is more aesthetic design and smoother fluency of system. By improving the UI design will be able to improve the user experience and more interesting on developed system.

Furthermore, for smoother fluency of system is required more powerful computers or server to handle the computation demand of the system since those practices such as data mining, machine learning and deep learning are great demand for output power.

In addition, to enhance system flexibility, a web application is essential since web applications can be accessed from anywhere with a network connection. Django is one of the Python libraries that supports web application development, so we should continue to explore it.

Although cloud storage has demonstrated impressive performance in the advancement of technology, it is advised to reinforce network infrastructure given the network setup and authentication issues that cloud based storage faces. These issues might be resolved, and the system would then be able to take use of the scalability and remote accessible benefits of cloud-based databases.

By implementing these recommendations, the FRLD development can overcome its obstacles and keep improving the precision, speed, and reliable, ultimately accomplishing its intended objectives in location detection for building occupants.

References

[1] R. Rahim, "AI-based traffic cams not new, set up as proof of concept, says Loke," The Star, 12 July 2023. [Online]. Available: https://www.thestar.com.my/news/nation/2023/07/12/ai-based-traffic-cams-not-new-set-up-as-proof-of-concept-says-loke. [Accessed 27 August 2023].

[2] C. Bernstein, "Face Detection," TechTarget, Feb 2020. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/face-detection. [Accessed 1 April 2023].

[3] B. L. M. S. Brandon Amos, "OpenFace: A general-purpose face recognition library with mobile applications," June 2016. [Online]. Available: https://elijah.cs.cmu.edu/DOCS/CMU-CS-16-118.pdf. [Accessed 30 November 2022].

[4] A. S. Gillis, "Facial Recognition," TechTarget, June 2019. [Online]. Available: http://whatis.techtarget.com/definition/facial-recognition [Accessed 25 Mar. 2018].. [Accessed 26 November 2022].

[5] S. R. M. S. S. P. P. J. K. Chanchal S. Khandelwal, "Tracking of Unauthorized Access Using Face Recognition," 1 January 2013. [Online]. Available: https://www.ijert.org/research/tracking-of-unauthorized-access-using-face-recognition-IJERTV2IS1381.pdf. [Accessed 28 March 2023].

[6] T. C. A. P. J. A.-J. N. M. Sawyer Clever, "Ethical Analyses of Smart City Applications," 20 September 2018. [Online]. Available: https://doi.org/10.3390/urbansci2040096. [Accessed 26 March 2023].

[7] F. C. Commission, "Accurate Location Detection," 2015. [Online]. Available: https://hai.stanford.edu/sites/default/files/2020-11/HAI_FacialRecognitionWhitePaper_Nov20.pdf. [Accessed 25 March 2023].

[8] B. M. Hegnes, "Location Tracking of WIFI Access Point and Bluetooth Devices," Hegnes Tech, 1 September 2021. [Online]. Available: https://www.hegnes.tech/2021/09/01/location-tracking-of-wifi-access-point-and-bluetooth-devices/. [Accessed 2023 March 25].

[9] S. H. Hyunsoo Kim, "Accuracy Improvement of Real-Time Location Tracking for Construction Workers," ResearchGate, May 2028. [Online]. Available: https://www.researchgate.net/publication/325066757_Accuracy_Improvement_of_Real-Time_Location_Tracking_for_Construction_Workers. [Accessed 25 March 2023].

[10] R. Turner, "4 Location Tracking Technologies That Every Business Should Invest In," TechDay, [Online]. Available: https://techdayhq.com/community/articles/4-location-tracking-technologies-that-every-business-should-invest-in. [Accessed 26 March 2023].

[11] INFORMATION ASSURANCE DIRECTORATE, "IAD Best Practices for Securing Wireless Devices and Networks in NSS," 13 October 2015. [Online]. Available: https://www.nsa.gov/portals/75/documents/news-features/news-stories/2015/iad-wireless-security-recommendations/iad-wireless-security-recommendations.pdf.

[12] Clare Stouffer, a NortonLifeLock employee, "Public Wi-Fi: An ultimate guide on the risks + how to stay safe," NortonLifeLock, 15 September 2022. [Online]. Available: https://us.norton.com/blog/privacy/public-wifi#. [Accessed 26 March 2023].

[13] A. A. A. D. S. J. H. T. H. A. A.-H. Aly Khalifa, "Face Recognition and Tracking Framework for Human–Robot Interaction," MDPI, 30 May 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/11/5568. [Accessed 29 August 2023].

[14] G. Sharma, "Big Data & Cloud Computing: The Roles & Relationships," IEEE Computer Society, [Online]. Available: https://www.computer.org/publications/tech-news/trends/big-data-and-cloud-computing. [Accessed 28 March 2023].

[15] D. S. Varsha Gupta, "A Study of Various Face Detection Methods," *International Journal of Advanced Research in Computer and Communication Engineering,* vol. 3, no. 5, pp. 6694-6697, May 2014.

[16] J. M. Frockman, "AI Superpowerd and Human+Machine a Visionary Revolution in Finance," in *Artificial Intelligence and Machine Learning*, Independently Published, 2019, p. 95.

[17] A. Ciocan, "AI for Facial Recognition Technology," Academia, 2021. [Online]. Available: https://www.academia.edu/50976731/AI_for_Facial_Recognition_Technology. [Accessed 28 November 2022].

[18] R. Raj, "Supervised, Unsupervised and Semi-supervised Learning with Real-life Usecase," Enjoy Algorithms, [Online]. Available: https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning. [Accessed 30 March 2023].

[19] D. R. J. S.Aanjanadevi, "A Study on Secure Online Voting System using Biometrics Face Detection and Recognition Algorithms," *International Journal for Modern Trends in Science and Technology,* vol. 03, no. 08, pp. 68-76, August 2017.

[20] V. B. Dipti Trivedi, "Occupancy detection systems for indoor environments: A survey of approaches and methods," *SAGE Journals,* vol. 29, no. 8, pp. 1053-1069, October 2020.

[21] P. W. C. H. J. O. Shushan Hu, "Building Occupancy Detection and Localization Using CCTV Camera and Deep Learning," *IEEE INTERNET OF THINGS JOURNAL ,* vol. 10, no. 1, pp. 597-608, 2023.

[22] "FaceMe Security," Cyberlink, [Online]. Available: https://www.cyberlink.com/faceme/solution/security/overview. [Accessed 2 September 2023].

[23] "Introduction to System Analysis and Design," in *UCCD 2003 OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN*, UTAR, 2022.

[24] A. Rosebrock, "Face detection with dlib (HOG and CNN)," pyimagesearch, 19 April 2021. [Online]. Available: https://pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/.

[25] K. S. S. K. R. Sharma Saravanan, "FAREC — CNN based efficient face recognition technique using Dlib," ResearchGate, May 2016. [Online]. Available: https://www.researchgate.net/publication/312963881_FAREC_-_CNN_based_efficient_face_recognition_technique_using_Dlib. [Accessed 7 September 2023].

[26] G. R.-R. Bridget Robinson-Riegler, "Cognitive psychology : applying the science of the mind," Boston, Allyn & Bacon, 2012, 2012.

[27] X. M. S. L. H. P. LiXiang Li, "A Review of Face Recognition Technology," July 2020. [Online]. Available: https://www.researchgate.net/publication/343118558_A_Review_of_Face_Recognition_Technology.

[28] Y. Li, "FACE RECOGNITION SYSTEM," Harrisburg University, 2019. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1901/1901.02452.pdf. [Accessed 26 November 2022].

[29] G. C. K. G. K. K. A. R. L. S. D. U. Z. V. Michelle Stephens, "National Institute of Standards and Technology Special Publication 1268," May 2021. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1268.pdf. [Accessed 26 March 2023].

APPENDIX

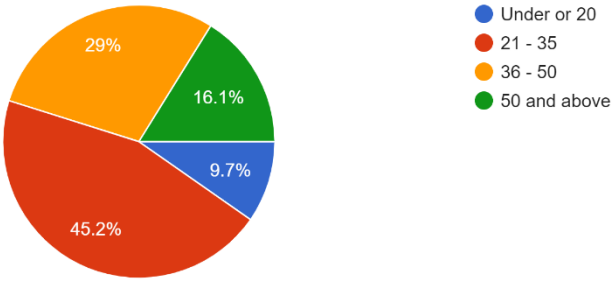
System Survey and Evaluation Result

Acknowledgement By participating in this questionnaire, I agree that my involvement is voluntary. I understand that my responses will be used exclusive...revealed in any resulting publications or reports.

32 responses

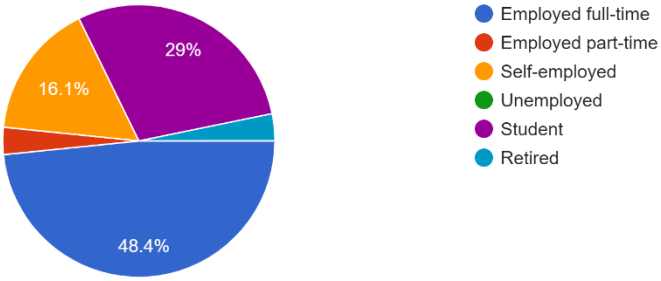
1. Which age range do you fall into?

31 responses



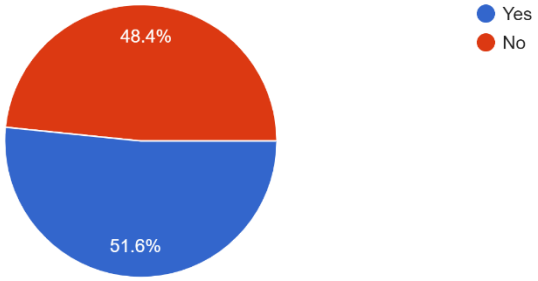
2. What is your current employment status

31 responses



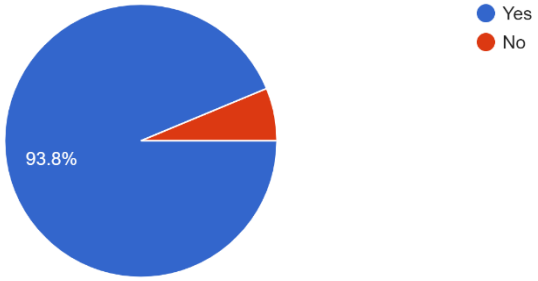
3. Do you have experience of using Computer Vision?

31 responses



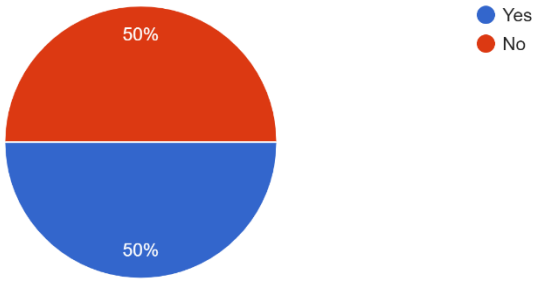
4. Are you satisfied to Face Recognition for Location Detection of Occupants Inside Building (Proposed Work)?

32 responses



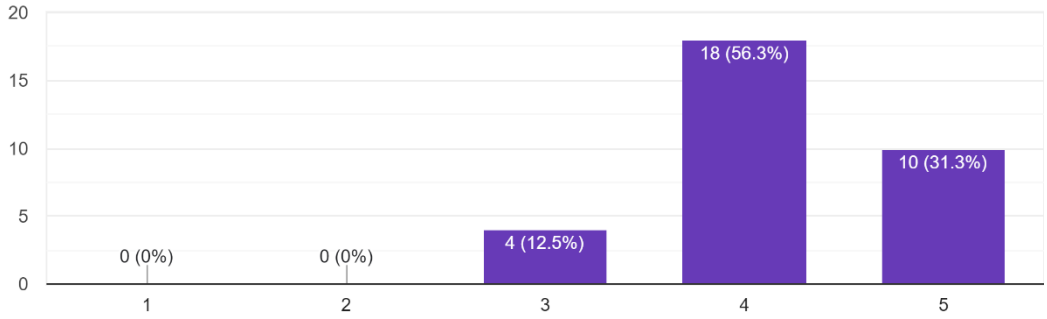
5. Do you experienced on other building occupancies management system with implemented face recognition technique?

32 responses



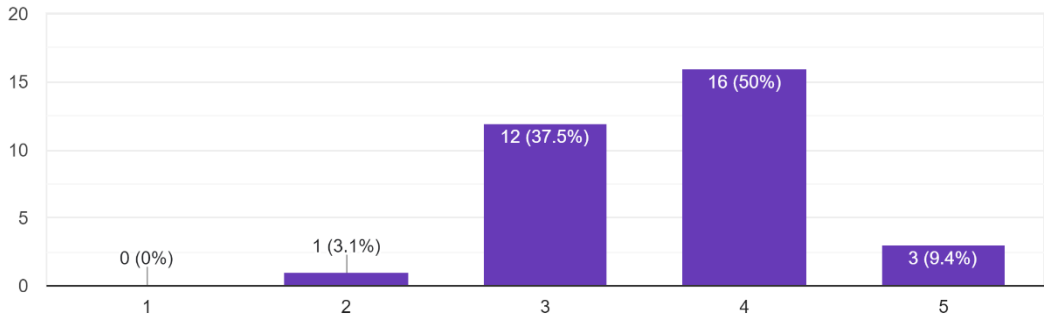
6. On a scale of 1 to 5, how easy was it for you to understand and use the face recognition system for occupant location detection?

32 responses



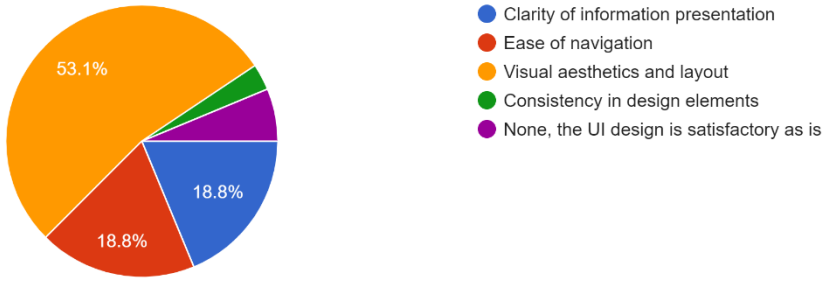
7. On a scale of 1 to 5, how satisfied are you with the design of the user interface (UI) display of the system for occupant location detection using face recognition?

32 responses



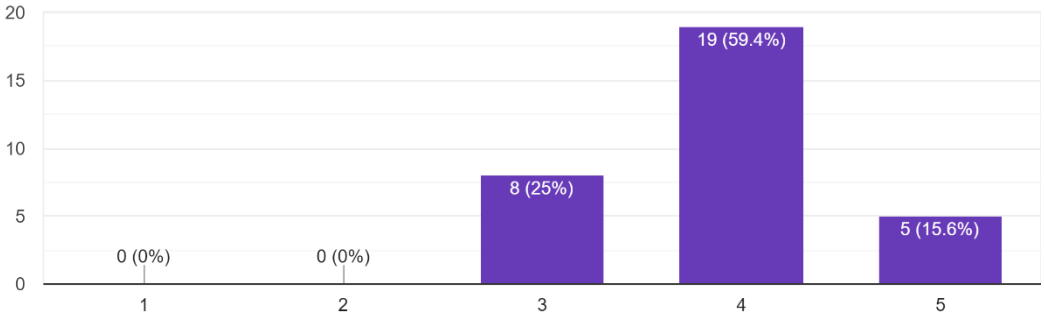
8. Which aspect of the user interface (UI) design do you think could be improved to enhance your experience with the project?

32 responses



9. How satisfied are you with the overall functionality and features of the system for occupant location detection using face recognition?

32 responses



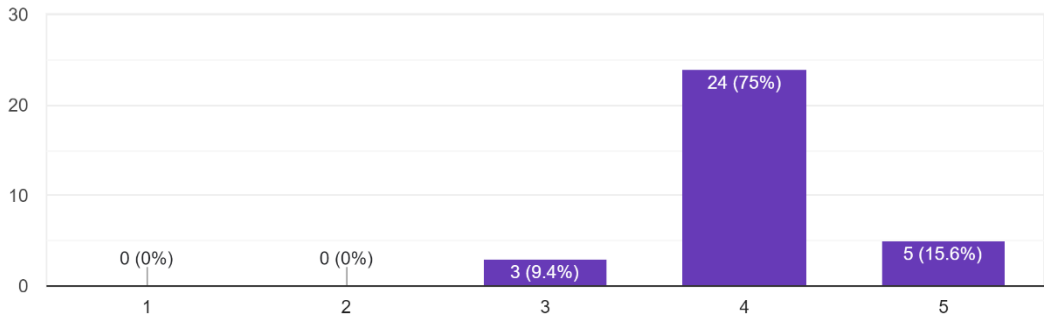
10. Are there any specific features or aspects you believe could be improved to enhance the performance of the system for occupant location detection using face recognition? If so, please provide your suggestions below.

22 responses

- improving the overall user experience and system usability.
- add more aesthetic designs
- system a bit lagging
- missing application guide
- more simple design
- provide instruction when hover on the feature
- font size and position
- The design is too single and the display position is poor, less fluence of system progress
- the system is not smooth

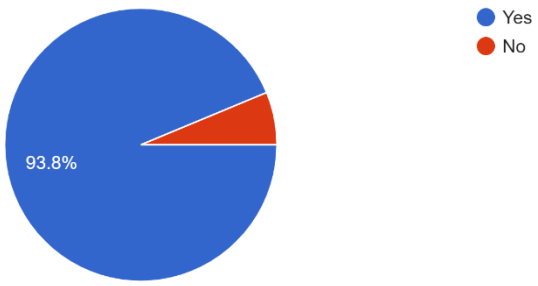
11. How satisfied are you with the accuracy of the system's recognized results when it comes to occupant location detection using face recognition?

32 responses



12. Do you find the analysis reports and visualized data created by the system for occupant location detection using face recognition sufficient

32 responses



13. Would you like to see more detailed analysis reports and graphical representations of the data in the system for occupant location detection using face recognition? If so, please provide your suggestions below.

7 responses

yes

-

able to generate monthly report automatically

show the photo of blacklisted person

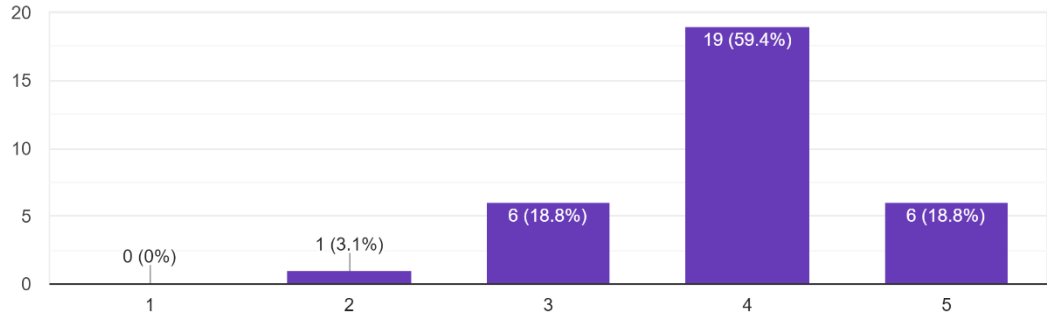
Summary of monthly report

lack of practicality chart

Occupant Behaviour Analysis

14. On a scale of 1 to 5, how would you rate your overall satisfaction with the system for occupant location detection using face recognition?

32 responses



Weekly Log

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.: 1**
Student Name & ID: Cheong Kok Siong 19ACB06441	
Supervisor: Mr Su Lee Seng	
Project Title: Face Recognition for Location Detection of Occupants in a Building	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Project timeline planning and discussion the system features.

2. WORK TO BE DONE

- Research and explore those system features' designs and python coding.

3. PROBLEMS ENCOUNTERED

- No issues for now.

4. SELF EVALUATION OF THE PROGRESS

- Have to put more effort for developing system.

SuLee

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.: 3**
Student Name & ID: Cheong Kok Siong 19ACB06441	
Supervisor: Mr. Su Lee Seng	
Project Title: Face Recognition for Location Detection of Occupants in a Building	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Literature review analyzing and summarizing.
- System UI design

2. WORK TO BE DONE

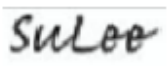
- Execute the designed UI
- Develop system functions

3. PROBLEMS ENCOUNTERED

- Connect designed UI with system functions

4. SELF EVALUATION OF THE PROGRESS

- Project is on scheduled



Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.: 5**
Student Name & ID: Cheong Kok Siong 19ACB06441	
Supervisor: Mr. Su Lee Seng	
Project Title: Face Recognition for Location Detection of Occupants in a Building	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Successful execute system UI

2. WORK TO BE DONE

- Define system's classes

3. PROBLEMS ENCOUNTERED

- System classes design quite messy

4. SELF EVALUATION OF THE PROGRESS

- Have to refer the existing published system and get idea from them.

SuLee

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.: 7**
Student Name & ID: Cheong Kok Siong 19ACB06441	
Supervisor: Mr. Su Lee Seng	
Project Title: Face Recognition for Location Detection of Occupants in a Building	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Completed the system draft design.
- Completed the part of the system functions define.

2. WORK TO BE DONE

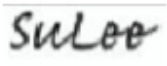
- Continue developing the system features.

3. PROBLEMS ENCOUNTERED

- Lack of practice on developing using multiple packages.

4. SELF EVALUATION OF THE PROGRESS

- Progress is completed on time that planned.
- Have to explore and study on Python packages.



Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.: 9**
Student Name & ID: Cheong Kok Siong 19ACB06441	
Supervisor: Mr. Su Lee Seng	
Project Title: Face Recognition for Location Detection of Occupants in a Building	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Have a better understanding and practices
- Done develop 75% of system functions
- Successful save data in database by using SQL query

2. WORK TO BE DONE

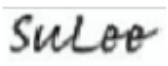
- Complete develop the system

3. PROBLEMS ENCOUNTERED

- Lack of experience on system development with multiclasses.

4. SELF EVALUATION OF THE PROGRESS

- Developing progress been delayed from my expectation.
- Have to more practice on coding.



Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.: 11**
Student Name & ID: Cheong Kok Siong 19ACB06441	
Supervisor: Mr. Su Lee Seng	
Project Title: Face Recognition for Location Detection of Occupants in a Building	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Successfully done the system development
- Done part of the system testing

2. WORK TO BE DONE

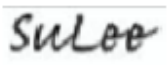
- Continue testing on developed system
- Complete the report

3. PROBLEMS ENCOUNTERED

- Developed system is having bugs and errors

4. SELF EVALUATION OF THE PROGRESS

- Developing progress been delayed from my expectation.



Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.: 13**
Student Name & ID: Cheong Kok Siong 19ACB06441	
Supervisor: Mr. Su Lee Seng	
Project Title: Face Recognition for Location Detection of Occupants in a Building	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- System done testing
- Complete part of the report

2. WORK TO BE DONE

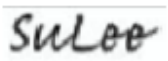
- Complete the report
- Finalize system and submit report

3. PROBLEMS ENCOUNTERED

- No issue

4. SELF EVALUATION OF THE PROGRESS

- Project is on scheduled



Supervisor's signature

Student's signature

POSTER

FACE RECOGNITION FOR LOCATION DETECTION OF OCCUPANTS IN BUILDING

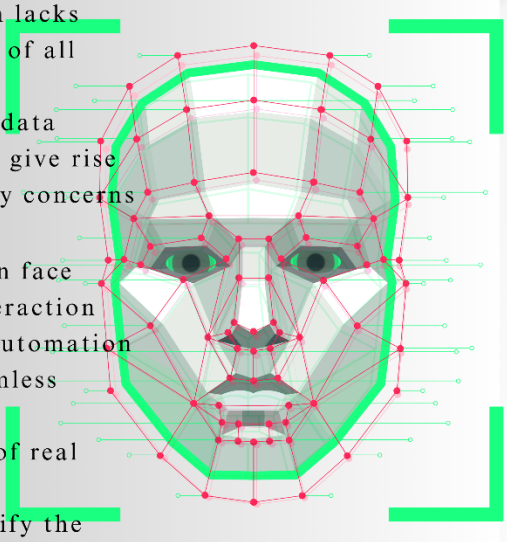
WHAT IS FACE RECOGNITION ?

- The process of identifying or verifying the identity of a person using their facial features.
- Its high accuracy and non-intrusive nature compared to other biometric technologies.

SECURE YOUR BUILDING WITH A GLANCE: FACE RECOGNITION FOR LOCATION DETECTION

PROBLEM STATEMENT

1. Existing building management system lacks effectiveness in tracking the location of all occupants
2. Delayed and Inaccuracy of real-time data
3. The existing organization procedures give rise to potential security risks and privacy concerns



PROJECT OBJECTIVES

1. Create a smart environment using modern face recognition technology to reduce the interaction with occupants while streamlining data automation data automation and integration for seamless management
2. Increase the accuracy and extensiveness of real time data
3. Prevent the cyber security risk and simplify the process of record collecting?

WHY IS NEEDED?

- IMPROVE SECURITY LEVEL
- IMPROVE HUMAN LIFE QUALITY
- EMERGENCY RESPONSE
- BUILDING MANAGEMENT
- PRIVACY PROTECTION

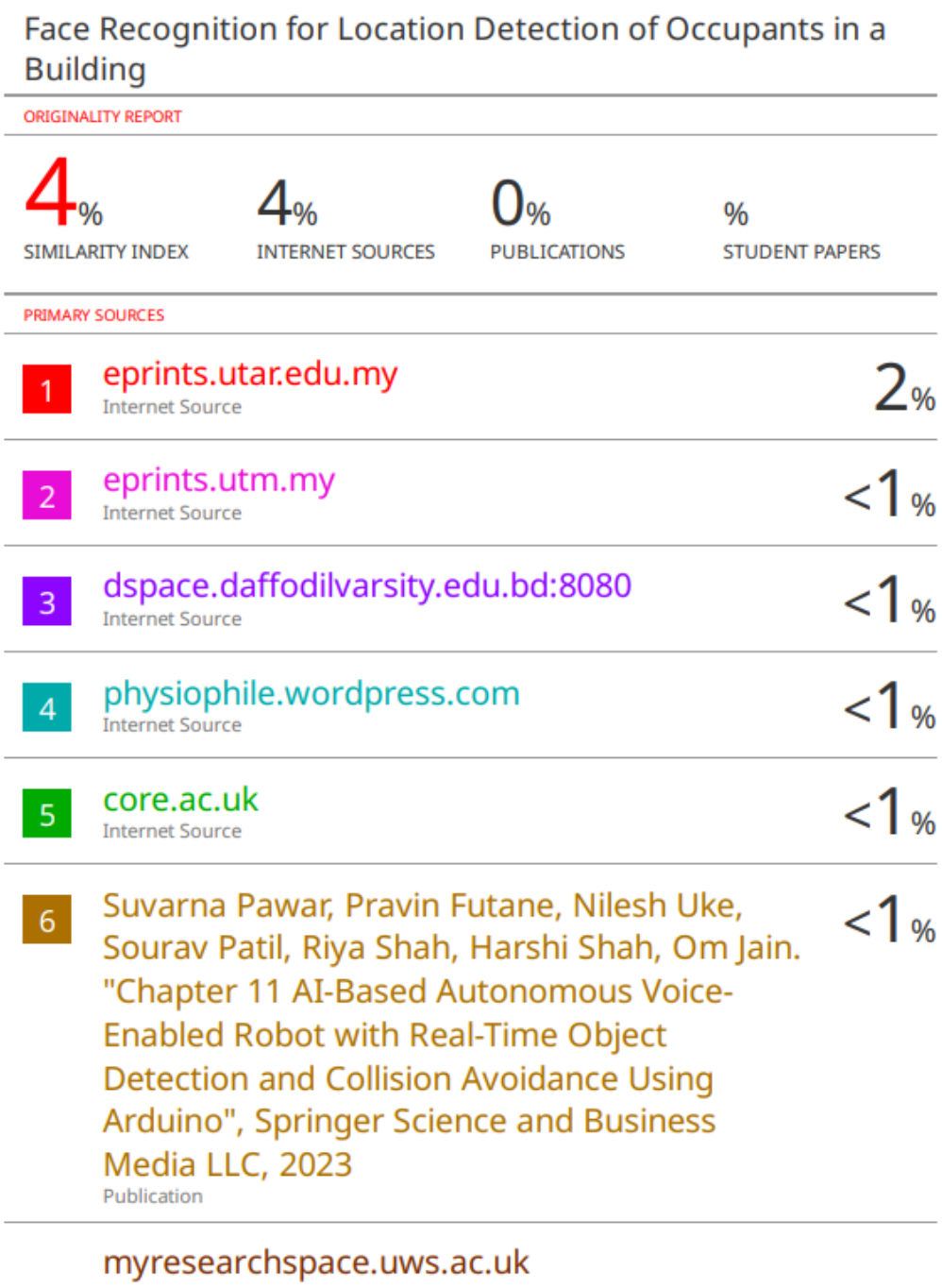
WHAT WE OFFER?

- IDENTIFY OCCUPANT'S IDENTITY
- TRACKING OCCUPANT'S LOCATION
- BLACKLISTED DETECTION
- REAL-TIME MONITORING

CHEONG KOK SIONG
BACHELOR OF INFORMATION SYSTEMS (HONS) BUSINESS INFORMATION SYSTEMS

SUPERVISOR:
MR SU LEESENG

PLAGIARISM CHECK RESULT



Face Recognition for Location Detection of Occupants in a Building

ORIGINALITY REPORT

4%	4%	0%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eprints.utar.edu.my Internet Source	2%
2	eprints.utm.my Internet Source	<1%
3	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1%
4	physiophile.wordpress.com Internet Source	<1%
5	core.ac.uk Internet Source	<1%
6	Suvarna Pawar, Pravin Futane, Nilesh Uke, Sourav Patil, Riya Shah, Harshi Shah, Om Jain. "Chapter 11 AI-Based Autonomous Voice-Enabled Robot with Real-Time Object Detection and Collision Avoidance Using Arduino", Springer Science and Business Media LLC, 2023 Publication	<1%

myresearchspace.uws.ac.uk

PLAGIARISM CHECK RESULT

7	Internet Source	<1%
8	www.pyimagesearch.com Internet Source	<1%
9	www.intechopen.com Internet Source	<1%
10	docplayer.net Internet Source	<1%
11	pubmed.ncbi.nlm.nih.gov Internet Source	<1%
12	www.researchgate.net Internet Source	<1%
13	www.giiresearch.com Internet Source	<1%

PLAGIARISM CHECK RESULT

Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

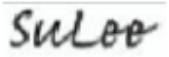
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Cheong Kok Siong
ID Number(s)	19ACB06441
Programme / Course	Bachelor of Information Systems (Honours) Business Information Systems
Title of Final Year Project	Face Recognition for Location Detection of Occupants in a Building

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)**
Overall similarity index: 4 % **Similarity by source** Internet Sources: 4 % Publications: 0 % Student Papers: 0 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required, and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.



Signature of Supervisor

Name: Su Lee Seng

Date: 13 September 2023

Signature of Co-Supervisor

Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	19ACB06441
Student Name	Cheong Kok Siong
Supervisor Name	Su Lee Seng

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

_____

(Signature of Student)
Date: 13 September 2023