

**INTELLIGENT MEDICINE BOX SYSTEM WITH  
AI-POWERED PILL DETECTION AND IOT  
INTEGRATION**

**CHAN YEE WEI**

**UNIVERSITI TUNKU ABDUL RAHMAN**

**INTELLIGENT MEDICINE BOX SYSTEM WITH AI-POWERED PILL  
DETECTION AND IOT INTEGRATION**

**CHAN YEE WEI**

**A project report submitted in partial fulfilment of the  
requirements for the award of  
Bachelor of Electronics Engineering With Honours**

**Faculty of Engineering and Green Technology  
Universiti Tunku Abdul Rahman**

**May 2025**

## DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :



Name : CHAN YEE WEI

ID No. : 20AGB02486

Date : 22 May 2025

### APPROVAL FOR SUBMISSION

I certify that this project report entitled **“INTELLIGENT MEDICINE BOX SYSTEM WITH AI-POWERED PILL DETECTION AND IOT INTEGRATION”** was prepared by **CHAN YEE WEI** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Electronic Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature

:



Supervisor

:

Ts Dr Toh Pek Lan

Date

:

22 May 2025

Signature

:

\_\_\_\_\_

Co-Supervisor

:

\_\_\_\_\_

Date

:

\_\_\_\_\_

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2025, Chan Yee Wei. All right reserved.

## **ACKNOWLEDGEMENTS**

I would like to thank everyone who had contributed to the successful completion of this project titled Intelligent Medicine Box System with AI-powered Pill Detection and IoT Integration. I would like to express my gratitude to my research supervisor, Ts. Dr. Toh Pek Lan for her invaluable advice, guidance, and her enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who have helped and given me encouragement and support throughout the process. Furthermore, I would like to thank the lab assistants for their guidance and advice.

## **INTELLIGENT MEDICINE BOX SYSTEM WITH AI- POWERED PILL DETECTION AND IOT INTEGRATION**

### **ABSTRACT**

Taking medication as prescribed is a challenging task for most patients, particularly for those with busy lifestyles. In simple words, medication adherence can be defined as taking prescriptions at the right time and in the correct dosage. Adhering to medication schedules and dosages is crucial for managing chronic conditions such as hypertension and high cholesterol. Failure to follow the medication regimen can lead to several adverse consequences, including disease progression and the deterioration of health conditions. In the long run, this can ultimately reduce the overall quality of life, leading to an increased risk of long-term health consequences. To address this issue, an Intelligent Medicine Box System that leverages the power of Artificial Intelligence (AI) and the Internet of Things (IoT) has been developed to improve the user's medication adherence. This book describes the development of an Intelligent Medicine Box System with AI-Powered Pill Detection and IoT Integration. The Intelligent Medicine Box System, which leverages IoT, provides the features of timely reminders via mobile application and the buzzer in the pill box to ensure the user takes their medication as prescribed. Besides that, the risk of running out of medicine will be reduced with the real-time pill tracker feature that automatically tracks and monitors supply levels and alerts users when refills are needed. In addition, the AI-powered pill detection and counting feature using a deep learning model can further detect foreign objects and potential missed doses, thus reducing contamination and enhancing medication adherence. By ensuring their medications are taken on time, this system can help in managing users' conditions more effectively, thereby improving their quality of life.

## TABLE OF CONTENTS

<b>DECLARATION</b>	<b>i</b>
<b>APPROVAL FOR SUBMISSION</b>	<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>iv</b>
<b>ABSTRACT</b>	<b>v</b>
<b>TABLE OF CONTENTS</b>	<b>vi</b>
<b>LIST OF TABLES</b>	<b>ix</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>	<b>xv</b>
<b>LIST OF APPENDICES</b>	<b>xvi</b>
 <b>CHAPTER</b>	
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Background	1
1.2 Problem Statement	3
1.3 Aim and Objectives	5
1.3.1 Aim	5
1.3.2 Objectives	5
<b>2 LITERATURE REVIEW</b>	<b>6</b>
2.1 Introduction	6
2.2 Literature Review	7
2.2.1 Smart Medicine Pill Box Reminder Presented by Azlan and Yahya (2023)	7
2.2.2 Design of a Smart Medical Box for Automatic Pill Dispensing and Health Monitoring Presented by Nasir <i>et al.</i> (2023)	10
2.2.3 Automated Medication Verification System (AMVS): System Based on	



	Edge Detection and CNN Classification Drug on Embedded Systems Presented by Chiu (2024)	13
	2.2.4 Development of Smart Pill Expert System Based on IoT Presented by Dayananda and Upadhya (2024)	18
<b>3</b>	<b>METHODOLOGY</b>	<b>23</b>
3.1	Introduction	23
3.2	Project Management	23
3.3	Component Description	25
3.3.1	Raspberry Pi 4 Model B	25
3.3.2	Raspberry Pi Camera Module V2	28
3.3.3	16 × 2 Liquid Crystal Display (LCD)	28
3.3.4	Light Emitting Diode (LED)	30
3.3.5	Acrylic Diffuser Sheet	30
3.3.6	RGB LED	31
3.3.7	NodeMCU ESP8266 V2	32
3.3.8	Buzzer	33
3.3.9	LC18650 Li-Ion Rechargeable Lithium Battery	34
3.3.10	18650 Battery Holder	34
3.3.11	MT3608 Step-Up Power Module	35
3.4	Software Descriptions	35
3.4.1	Cloud Firestore	36
3.4.2	Android Studio	38
3.4.3	Flutter	39
3.4.4	Visual Studio Code	40
3.4.5	Arduino IDE	40
3.5	Block Diagram	40
3.6	Arduino IDE and Raspberry Pi Setup	41
3.6.1	Setup for Arduino IDE	41
3.6.2	Setup for Raspberry Pi	43
3.7	Pill Detection	48
3.7.1	Object Detection	48

	3.7.2 YOLO (You Only Look Once)	48
	3.7.3 YOLO Model Training Workflow	49
<b>4</b>	<b>RESULTS AND DISCUSSION</b>	<b>56</b>
	4.1 Mobile Application User Interface (UI)	56
	4.2 Prototype	66
	4.2.1 Portable Pill Box	67
	4.2.2 Home Base	68
	4.3 Pill Box	70
	4.3.1 Program of NodeMCU ESP8266 V2	70
	4.3.2 Hardware Connection in Pill Box	73
	4.3.3 Light Indications and Sound Alert	74
	4.4 Home Base	75
	4.4.1 YOLOv8 Model Evaluation	75
	4.4.2 Operation of the Home Base System	77
	4.4.3 Results of the Deployment of YOLOv8 Model on Raspberry Pi	80
	4.4.4 LED and LCD Indications	82
<b>5</b>	<b>CONCLUSION AND RECOMMENDATIONS</b>	<b>85</b>
	5.1 Conclusion	85
	5.2 Future Improvements and Recommendations	86
	<b>REFERENCES</b>	<b>87</b>
	<b>APPENDICES</b>	<b>90</b>

**LIST OF TABLES**

<b>TABLE</b>	<b>TITLE</b>	<b>PAGE</b>
2.1	Comparison of Techniques Related to Intelligent Medicine Box System	22
3.1	Gantt Chart for FYP 1	24
3.2	Gantt Chart for FYP 2	25
3.3	Specifications and Description of the Raspberry Pi 4 Model B	27
3.4	Pinout for the 16×2 LCD	29
4.1	Light Indications and Sound Alert on Pill Box	74
4.2	LED and LCD Indications on Home Base	83

## LIST OF FIGURES

FIGURE	TITLE	PAGE
2.1	Flowchart of the Whole System (Azlan and Yahya, 2023)	9
2.2	The Total Weight of the Medicines (Azlan and Yahya, 2023)	10
2.3	Blynk Application (a) Total Weight of Medicines Reaches Below 50% (b) Alert Notifications Will Be Sent to the Caretaker (Azlan and Yahya, 2023)	10
2.4	The Design of the Rotating Disk Dispenser (Nasir <i>et al.</i> , 2023)	12
2.5	The Database of Patients Saved in the Module (Nasir <i>et al.</i> , 2023)	13
2.6	GUI Designed (a) New Medicine Schedules Created (b) Medicine Schedule (Nasir <i>et al.</i> , 2023)	13
2.7	The Design of Drug Verification Box (Chiu, 2024)	16
2.8	The Workflow of the System (Chiu, 2024)	17
2.9	The Comparison of the Computational Time of Edge Detection Between the Watershed Algorithm and Other Methods (Chiu, 2024)	17
2.10	The SA Average Accuracy for Classification (Chiu, 2024)	18
2.11	The Working Dispensing Mechanism of the System (Dayananda and Upadhya, 2024)	20
2.12	Accuracy and Functionality Graph of SEPC 2.0 (Dayananda and Upadhya, 2024)	20

2.13	Comparative Analysis of SPEC 2.0 with Existing System (Dayananda and Upadhyaya, 2024)	21
3.1	Raspberry Pi 4 Model B	26
3.2	Structure and Specifications of the Raspberry Pi Model B (Singh, 2021)	26
3.3	40 GPIO Pins of the Raspberry Pi 4 Model B (Singh, 2021)	27
3.4	Raspberry Pi Camera Module V2 (a) Camera Module with Ribbon Cable (b) Camera Module Connected to Raspberry Pi Board	28
3.5	16 × 2 LCD (Cd-Team, 2023)	29
3.6	Light Emitting Diode (Scully, 2019)	30
3.7	Acrylic Diffuser Sheets	31
3.8	RGB LED	31
3.9	Common Cathode RGB LED Pins (Santos, 2019)	32
3.10	NodeMCU ESP8266 V2	32
3.11	NodeMCU ESP8266 V2 Pinout	33
3.12	Buzzer (Agarwal, 2021)	33
3.13	LC18650 Li-Ion Rechargeable Lithium Battery	34
3.14	18650 Battery Holder	34
3.15	MT3608 DC-DC Boost Converter	35
3.16	Trimmer for Adjusting Voltage (Robottronic, 2020)	35
3.17	Logo of Cloud Firestore	36
3.18	Data Structures in Cloud Firestore	37
3.19	Cloud Firestore Database	37
3.20	Customized Cloud Firestore Security Rules	38
3.21	Logo of Android Studio	38
3.22	Pixel 4 API 35 Emulator	39

3.23	Logo of Flutter	39
3.24	Block Diagram of Intelligent Medicine Box System With AI-Powered Pill Detection and IoT Integration	41
3.25	URL Added in the Preferences Menu	42
3.26	Installation of ESP8266 Board Package	42
3.27	NodeMCU 1.0 (ESP-12E Module) Is Selected	43
3.28	Selections Available in Raspberry Pi Imager	44
3.29	Recommended Operating System Shown at the Top	44
3.30	Advance Settings for Wi-Fi Credentials	45
3.31	Text File Is Created	45
3.32	Settings for SSH	46
3.33	VNC Enabled in the Raspberry Pi Configuration	47
3.34	RealVNC Viewer Home Screen	47
3.35	Authentication Window in RealVNC Viewer	48
3.36	Detection System of YOLO (Redmon <i>et al.</i> , 2016)	49
3.37	YOLO Model Training Workflow	50
3.38	Output Messages When Capturing Images	51
3.39	Images Captured by the Python Script (a) Amlodipine (b) Simvastatin (c) BoxPresent (d) Unknown	51
3.40	Annotated Images (a) Amlodipine (b) Simvastatin (c) BoxPresent (d) Unknown	52
3.41	Medicine_box_v4.yaml File Is Created	53
3.42	Train.py File Is Created	54
3.43	Terminal Displays the Progress of Each Epoch	54
3.44	Model Performance Shows in Terminal	55
4.1	Flow of the Mobile Application UI	58

4.2	Register Page	59
4.3	Login Page	59
4.4	Reminder Alarm Page When No Reminder Is Set	60
4.5	Reminder Alarm Page with Set Reminders	60
4.6	Pill Tracker Page	61
4.7	Add Reminder Dialog	61
4.8	Time Picker	62
4.9	Reminder Successfully Added	62
4.10	Delete Reminder Dialog	63
4.11	Reminder Successfully Deleted	63
4.12	Refill Pill Dialog	64
4.13	Successful Pill Refill	64
4.14	Reminder Notification	65
4.15	Refill Alert Notification	65
4.16	Prototype of Intelligent Medicine Box System	66
4.17	Pill Box Detected by the System	66
4.18	Front View of the Pill Box	67
4.19	Side Perspectives of the Pill Box (a) Right Side View (b) Left Side View	67
4.20	Top View of the Pill Box	68
4.21	Front View of Home Base	68
4.22	Side Perspectives of Home Base (a) Right Side View (b) Left Side View	69
4.23	Top View of the Home Base	69
4.24	Drawer in Home Base	69
4.25	Firestore Credentials and Firestore API URL Included in the Code	71

4.26	Program Flow in ESP8266	72
4.27	Conditions to Trigger Buzzer	72
4.28	Hardware Connection for Circuit Inside Pill Box	73
4.29	Circuit Connection Inside Pill Box	73
4.30	Normalized Confusion Matrix	76
4.31	Precision-Recall Curve	77
4.32	F1- Confidence Curve	77
4.33	Flowchart of the Home Base System	79
4.34	Result from Real-Time Detection Window	80
4.35	Unknown Objects Detected (a) One Unknown Object Detected (b) Two Unknown Objects Detected	81
4.36	Output Messages When No Box Detected	81
4.37	Output Messages When Pill Match	81
4.38	Output Messages When Pill Mismatch	81
4.39	Output Messages When Unknown Object Detected	82



## LIST OF SYMBOLS / ABBREVIATIONS

3D	Three Dimension
AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
EDP	Edge Detection Processing
GPIO	General-Purpose Input/Output
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
IoU	Intersection over Union
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LED	Light Emitting Diode
mAP	mean Average Precision
NCDs	Non-Communicable Diseases
NTP	Network Time Protocol
RGB	Red, Green, Blue (Additive Colour Model)
RTC	Real-Time Clock
SSH	Secure Shell
UI	User Interface
UTC	Coordinated Universal Time
VNC	Virtual Network Computing
YOLO	You Only Look Once

**LIST OF APPENDICES**

<b>APPENDIX</b>	<b>TITLE</b>	<b>PAGE</b>
A	QR Code for Demonstration Video	90
B	C++ Code for ESP8266	90
C	Python Code for Capturing Dataset	101
D	Python Code for Real-Time Pill Detection	104
E	Dart Code for Main	110
F	Dart Code for Home Screen	111
G	Dart Code for Notification Logic	118
H	Dart Code for Add Reminder	123
I	Dart Code for Button	127
J	Dart Code for Delete Reminder	128
K	Dart Code for Login Screen	130
L	Dart Code for Pill Tracker	136
M	Dart Code for Reminder Model	144
N	Dart Code for Round Text Field	145
O	Dart Code for Signup Screen	147
P	Dart Code for Switcher	155

## CHAPTER 1

### INTRODUCTION

#### 1.1 Background

Medications play a critical role in our daily life, as medicines are mainly used to treat or prevent diseases. They are not only used to manage certain disorders, but they also help in preventing the progression of illnesses, thus improving the quality of life. According to the findings from the National Health and Morbidity Survey (NHMS) 2023, it showed that 29.2% of Malaysian adults have hypertension, 33.3% have high cholesterol, and 15.6% of the population suffer from diabetes (Institute for Public Health (IPH), 2024). With the rise in prevalence of certain illnesses, medication plays a vital role in enhancing and safeguarding patients' health.

Adherence is the key element in the effectiveness of pharmacological therapies, especially in managing chronic illnesses. Treating chronic illnesses such as hypertension and high blood pressure typically requires long-term use of medication. However, the effectiveness of the medication is often compromised by non-adherence (Brown and Bussell, 2011). Poor adherence to medical treatment, such as missed doses, can lead to worsening of their conditions. In other words, failing to adhere to prescribed regimens not only exacerbates the risk of disease progression but also increases the likelihood of complications, ultimately increasing the healthcare costs (Religioni *et al.*, 2025). One of the major hurdles in healthcare is to ensure that patients take their prescribed medicine and consistently follow healthcare professionals' instructions for taking the medicine. Medicines prescribed by healthcare professionals should be taken

properly and on time to ensure effective control of chronic conditions. In simple terms, it is essential to take prescriptions according to the doctor's instructions in order to maximize the benefits of the medications. If the medications are not taken as directed, this may lead to adverse outcomes. Therefore, an effective approach that leverages technology for reminders and monitoring should be implemented.

Being tech-savvy, we have seamlessly integrated technology into our daily lives. In other words, technology has become an indispensable part of our lives. The Internet of Things (IoT) can be defined as a network of devices that are embedded with sensors to collect and share data with other devices via an internet connection. The power of IoT is not only limited to household tools but can also be utilized in the healthcare field. In fact, IoT devices will be beneficial for healthcare professionals to monitor patients as well as for patients to monitor themselves more easily. To ensure that the user adheres to the prescribed medication schedules, an Intelligent Medicine Box System can provide a timely reminder and track the medication supply. Compared to the conventional medicine box that can only serve for medicine storage, the Intelligent Medicine Box System that integrates with the mobile application enables the user to set reminders, which are stored on the cloud server. Then, the buzzer in the pill box will be triggered based on the reminders fetched from the cloud server. In this case, the patient will be reminded to take their medicine, ensuring consistent medication intake by patients. Besides that, the pill tracking feature can ensure a more efficient medication management system that allows users to keep track of their current medication quantity.

In addition, users will get a timely reminder when the scheduled time is reached. Notifications that include a 'Mark as taken' button, allowing users to confirm the consumption of medication. By having this feature, it enables real-time updates of the pill quantity in the pill tracker. In this instance, the pill tracker will always be up to date with the current medication quantity. To ensure there is always an adequate supply available in the pill box and reduce the risk of medication shortages, an alert notification will be sent to the user's phone when the pill quantity is running low. If the status of the medication quantity reaches 20% or below of the original quantity, an alert message will automatically be sent to the mobile phone to remind the user to refill the medicine. By utilizing real-time pill tracking, the system helps in ensuring timely refills and

preventing any missed doses. This feature is not only useful for pill monitoring, but also for preventing medication schedule disruptions.

On top of that, the integration of Artificial Intelligence (AI) with the Intelligent Medicine Box System has led to the development of the AI-powered pill-detecting and counting features. Based on the visual information captured by the camera, the pills remaining in the pill box can be detected using a real-time object detection algorithm, and after that, the system performs the pill counting. By comparing the counting result with the pill log stored in the Cloud Firestore, the system can identify any potential missed doses. Through this process, the system can perform pill detection and counting precisely and efficiently.

## 1.2 Problem Statement

In Malaysia, 2.5% of the population, or more than half a million adults, suffer from all four non-communicable diseases (NCDs). The four NCDs include high cholesterol, hypertension, diabetes, and obesity (Harun and Nizam, 2024). Besides that, a survey was conducted with 20 participants aged between 40 to 60. The main goal of the survey was to evaluate how well they adhere to the medication, particularly in managing chronic illnesses. The results of the survey showed that 40% of them suffer from hypertension, 45% have high cholesterol, 20% have diabetes, 10% have cardiovascular diseases, and another 10% have osteoarthritis. Among them, half of the participants struggle to remember to take medication daily and sometimes miss doses. This highlights the prevalence of chronic health conditions within this age group, emphasizing the need for the development of an Intelligent Medicine Box System with AI-powered Pill Detection and IoT Integration.

However, it is prevalent for many individuals to experience difficulties taking their medications on time, specifically those who have a hectic lifestyle, resulting in frequent non-adherence. According to Kvarnström *et al.* (2021), one of the obstacles to medication adherence is the busy schedule that results in missed doses. In the midst of

their busy schedules, it's easy for them to prioritize work over their well-being. Hence, taking medicines on time always falls by the wayside. In other words, medication adherence becomes a challenge for them. In the long run, this issue may contribute to increasing the risk of detrimental effects on their health. To combat this, it is important to develop a system with timely reminders to ensure timely and consistent medication intake.

For conventional medication management, manual pill counting and tracking are prone to errors due to forgetfulness or miscounting doses, which can result in poor adherence to treatment plans. This not only compromises the effectiveness of the medication but also exacerbates their health condition. Additionally, manually counting pills is a time-consuming task that increases their workload, particularly for those with a busy schedule. The lack of real-time tracking worsens the situation as timely reminders or alerts cannot be delivered, thereby increasing the risk of running out of medication without noticing and disrupting the treatment plan.

Besides that, regularly refilling medications is particularly crucial for individuals who have been diagnosed with chronic conditions such as hypertension or high blood pressure. It is important for them to refill their medication to ensure there is always an adequate supply in their pill box, thus maintaining their health. To manage their chronic conditions, they need to take medicine constantly and adhere to their treatment plans. Failure or delay in refilling the medications may result in running out of medication, leading to a treatment gap. In the long run, their treatment plan will be disrupted and cause health complications.

Therefore, the development of an Intelligent Medicine Box System with AI-Powered Pill Detection and IoT Integration can effectively address these problems by providing the respective solutions to the problems. This Intelligent Medicine Box System not only provides timely reminders and pill tracking but also detects discrepancies in pill count and enhances medication safety by identifying any foreign object that could lead to contamination.

### **1.3 Aim and Objectives**

#### **1.3.1 Aim**

The aim of this project is to improve the medication adherence of the user by utilizing the Intelligent Medicine Box System with AI-powered Pill Detection and IoT Integration. The mobile application allows users to schedule reminders and track their pill quantity, while the audible alert from the buzzer helps ensure that they take their medication on time. Additionally, the object detection model employed can help in object detection and pill counting accurately, thus ensuring the user's medication adherence.

#### **1.3.2 Objectives**

The primary objective of this project is to design an Intelligent Medicine Box System integrated with a mobile application for setting reminders and pill tracking. The pill tracking function is like a small-scale inventory system. Besides that, the second objective is to integrate IoT technology into a pill box for timely reminders through buzzer alert based on scheduled reminders in the mobile application. The user is allowed to customize their medication schedule via the mobile application. By triggering the audible alert in the pill box, they can receive timely reminders and prevent missed doses. In addition, the objective of this project is to develop an AI-powered pill detection and counting system using the Raspberry Pi. This feature ensures accuracy and convenience, minimizing the likelihood of running out of medication supplies. As a result, it promotes better adherence to prescribed treatment plans, improving health outcomes.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 Introduction**

In this chapter, literature reviews related to the Intelligent Medicine Box System are conducted. The goal of conducting the literature review is to collect relevant research and clarify the project's concepts. Furthermore, the literature review serves to broaden our knowledge base and provides a comprehensive understanding of the topic that is related to this project. In other words, conducting a thorough literature review is paramount for enhancing knowledge and improving research methods. By analysing the existing studies, it can provide valuable insights into the most effective research method, thereby enhancing the project's credibility. This project has reviewed various research on the integration of IoT technology with the medicine box and the implementation of AI in the detection and counting process. Additionally, this part examines the pros and cons of each research, while also discussing the areas for future improvements.



## 2.2 Literature Review

### 2.2.1 Smart Medicine Pill Box Reminder Presented by Azlan and Yahya (2023)

Azlan and Yahya (2023) created a Smart Medicine Pill Box Reminder to tackle the problem of forgetting to take medication by providing timely reminders. The Smart Medicine Pill Box not only can carry out the function of alerting patients to take their medicine but also is able to notify the caretaker when the patient has taken their medicines based on the weight of the medicine box. When the time to take the medication arrives, the Smart Medicine Pill Box will notify the patient to take the medicine visually as well as audibly by using LEDs and a buzzer. This can ensure that the patients consistently receive notifications, thus improving medication adherence.

One of the core elements of this project is the presence of a weight sensor. A weight sensor is used to determine if the patient has taken their medication by measuring the weight of the remaining medicine. As the patient took the medicine, the removal of pills will directly cause changes in the weight measured by the sensor. By doing so, the caretaker can monitor and ensure that the patient adheres to the medication schedule. This project is suitable for both elderly and young people to ensure they take the right doses on time.

The authors utilize Arduino Uno to control the overall operation of the Smart Medicine Pill Box. This system is also equipped with a 1kg load cell to perform the function of measuring the weight of the medications in the medication box. To connect the Smart Medicine Pill Box with the BLYNK application on the phone, an ESP8266 Wi-Fi module is used. Besides that, this system includes a real-time clock (RTC) DS3231 that ensures accurate timekeeping. The LCD, LEDs, and buzzer serve as the outputs.

Figure 2.1 illustrates the flowchart of the operation of the whole system. This project included the date and time setup by the patient at the beginning. By pressing the “Set Time / Date” button provided on the casing of the medicine box, patients are allowed to adjust the time and date via the “Up” and “Down” buttons. Next, the patient

can proceed with the alarm setup so that they can set an alarm to remind them to take the medicines according to the medication schedules. To ensure accurate weight measurement, the load cell sensor should be calibrated.

The load cell measurement data will be sent directly to the BLYNK application. After that, the caretaker will receive the notification depending on the weight of the remaining medicines. When the load cell detects that the weight of the pill box reaches below 50% of the total weight, a notification will be sent to the BLYNK application and received by the caretaker. It will be extremely helpful for the caretaker to track the patient's medication intake. If the weight of the pill box decreases, it indicates that the patient has been taking their medication on time. As shown in Figure 2.2 and Figure 2.3, when the total weight of the remaining medicine reaches below 6.9g (below 50% of the total weight of 13.8g), the caretaker will receive notification that the patient has taken the medicine from the BLYNK IoT application.

One of the drawbacks of this project is that the patient needs to take the prescribed doses four times to reach 50% of the total weight of the medicines. This feature will only happen after the patient consumes medicines four times, making it difficult for daily monitoring of medication intake.

In this scenario, the caretaker will only receive notification after the patient has taken their medication for four days. This implies that the daily consumption of the patient cannot be tracked accurately. In this context, the patient is likely to miss doses without any reminders. If the caregiver cannot monitor whether the patient is taking the medication follows prescribed dose and timing, this can lead to inconsistent medication adherence and consequently cause negative impacts on their health. Besides that, this medicine box has medication shape and size constraints, as it is only suitable for medicines that have the same shape and size. Therefore, further improvement can be achieved by changing the working system of the project to ensure that the daily changes in the weight of the medications can be accurately monitored.

In a nutshell, this Smart Medicine Pill Box Reminder project has met its objective of designing a system that will remind the patient to take medication and has

developed a mobile application that is able to notify the caretaker if the patient has taken the medicine by measuring the weight of the pill box. Overall, this system effectively addresses the concern of missed doses by providing timely reminders and tracking adherence to medication schedules.

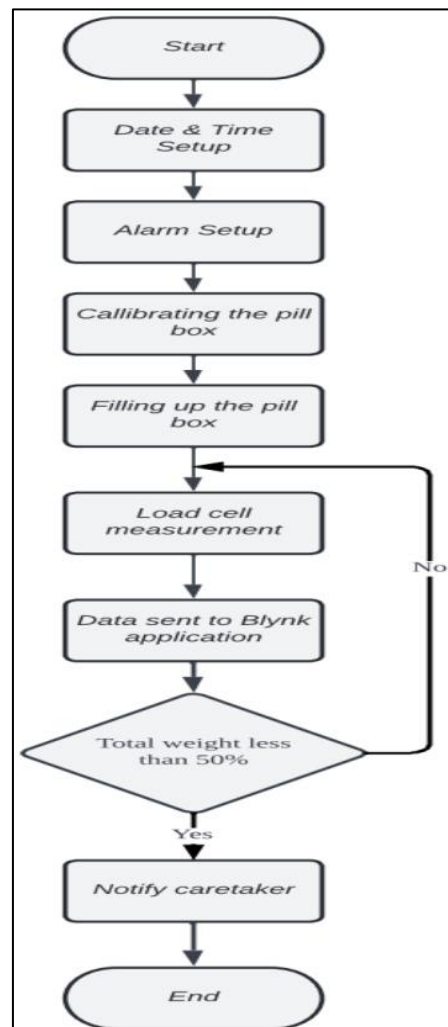


Figure 2.1: Flowchart of the Whole System (Azlan and Yahya, 2023)



Figure 2.2: The Total Weight of the Medicines (Azlan and Yahya, 2023)

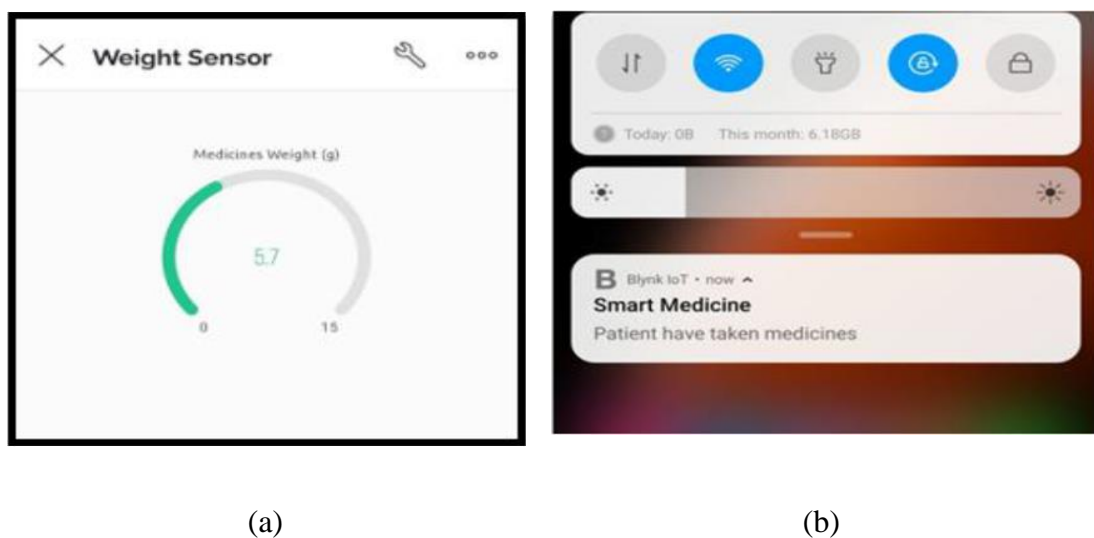


Figure 2.3: Blynk Application (a) Total Weight of Medicines Reaches Below 50% (b) Alert Notifications Will Be Sent to the Caretaker (Azlan and Yahya, 2023)

### 2.2.2 Design of a Smart Medical Box for Automatic Pill Dispensing and Health Monitoring Presented by Nasir *et al.* (2023)

Nasir *et al.* (2023) proposed a Smart Medical Box for Automatic Pill Dispensing that includes a health monitoring system. The health monitoring system includes measuring temperature, heart rate, and oxygen level. In order to enhance security and ensure correct patient access, the Smart Medical Box is equipped with biometric recognition. Furthermore, the user is able to receive notifications through SMS when their medicine has been dispensed. This function not only informs the patient to collect their medicine

on time but also improves their adherence to the medication schedules. The goal of this project is to develop a simple, reliable, easy-to-use system that will be beneficial for older users.

The proposed Smart Medical Box has three compartments for three time periods, mainly morning, afternoon, and night, each operated by its own stepper motor. Once the ultrasonic sensor detects the patient's hand, it will trigger the stepper motor to operate and dispense the medicine.

The author builds the basic health monitoring system by using a wide variety of sensors to perform their specific task. For instance, a biometric verification is performed by the biometric sensor, R307, while heart rate and oxygen are measured by MAX30102. For the temperature measuring and hand detection, the system utilizes the DS18B20 and Ultrasonic sensor, HC-SR04. All these sensors are integrated with a Raspberry Pi, which processes the data and allows for real-time monitoring. Subsequently, the respective readings will be displayed on the LCD.

When the time for medication is reached, a SIM800L GSM (Global System for Mobile Communication) serves the purpose of sending the reminder message to the patient and caretakers at the same time. This function will be particularly beneficial for those who may forget to take their medication and hence, ensures they follow the medication schedules properly. Moreover, the system is even equipped with a touch-sensitive LED to enable user interaction with the Smart Medical Box.

The author uses 'AutoCAD' to design a rotating disk dispenser with a hole that only allows only single pill to pass through at a time, ensuring only single pill dispensing at a time as shown in Figure 2.4.

On top of that, the biometric data, medicine schedule, and health conditions can be saved for up to 1000 patients, as shown in Figure 2.5. For the data stored, an existing user can modify and update the medicines. This is also applicable for a new user to enter the respective details and save them in the database. Upon receiving the alerting

notification, patients authenticate their identity, and then the automatic dispensing of the specific medication will be triggered.

The most notable advantage of this project is the biometric verification for granting access to ensure that only authorized patients can retrieve the medication. This function can prevent unauthorized access to the medicine, leading to medication misuse. In terms of software, the user-friendly graphical user interface (GUI) will be convenient for the elderly, as shown in Figure 2.6. Hence, the task of setting the medication schedule can be done effectively and independently. Saving the patient data in a systematic way is also the key aspect of enhancing accuracy and preventing medication errors.

However, one of the limitations of this project is that it is not suitable for managing non-oral medication. Creams, eye drops, and inhalers are examples of non-oral medication that the medicine box cannot accommodate. Thus, future improvements can be made to solve this problem. Besides that, this project can be improved by adding more compartments for the medicine storage. Another improvement that can be made in the future is creating an application that allows access for both the patient and the caretaker. In this case, not only is the patient able to manage the medication schedules, but the caretaker can also track and monitor the patient's health data.

In conclusion, the Smart Medical Box that was developed can effectively address the issue of forgetting to take the right medication at the right time. It is not the conventional medical box that is only used for medicine storage, but it also provides a reminder function as well as basic health monitoring. The multi-functionality of the medical box makes it suitable for everyone.



Figure 2.4: The Design of the Rotating Disk Dispenser (Nasir *et al.*, 2023)

Name	Gender	Age	Weight	M1 Time	M2 Time	M3 Time	M1 Quantity	M2 Quantity	M3 Quantity	Temp	Heart Rate	SPO2
M. Nawaz	Male	22	70	0900	1400	2100	1	2	1	97	98	96
M. Ali	Male	21	60	0900	1400	2100	2	1	1	96	97	98
Zara	Female	44	65	0900	1400	2100	1	1	1	98	96	97
Shehroz	Male	23	80	0900	1400	2100	2	1	2	96	94	94
Mughees	Male	45	55	0900	1400	2100	2	2	2	95	97	99
Mustabeen	Male	30	67	0900	1400	2100	1	2	1	98	98	96
Usman	Male	20	54	0900	1400	2100	1	1	1	99	99	97
Zain	Male	48	76	0900	1400	2100	1	2	1	100	95	98
Hasnain	Male	50	56	0900	1400	2100	2	1	1	94	94	97
Uzair	Male	55	76	0900	1400	2100	1	1	2	91	98	95
Hassan	Male	67	80	0900	1400	2100	1	2	2	92	95	90
Fazeen	Female	40	60	0900	1400	2100	2	1	2	96	97	98
Ayesha	Female	44	70	0900	1400	2100	1	1	1	97	95	99

Figure 2.5: The Database of Patients Saved in the Module (Nasir *et al.*, 2023)

**Medicine Schedule**

■ Medicine = Panadol      No of Times      1

Times :      0900      1400      2100

■ Medicine = Azomax      No of Times      2

Times :      0900      1400      2100

■ Medicine = Surbex Z      No of Times      1

Times :      0900      1400      2100

(a)

**Medicine Schedule**

Medicines :      Time

Panadol      0900      1400      2100

Azomax      0900      1400      2100

Surbex Z      0900      1400      2100

(b)

Figure 2.6: GUI Designed (a) New Medicine Schedules Created (b) Medicine Schedule (Nasir *et al.*, 2023)

### 2.2.3 Automated Medication Verification System (AMVS): System Based on Edge Detection and CNN Classification Drug on Embedded Systems Presented by Chiu (2024)

In the medical field, medication dispensing errors may cause adverse outcomes, such as hospitalization or even death. Contributing factors to the healthcare professionals include exhaustion from high workload and an insufficient nurse-to-patient ratio. Thus, the function of advanced technology can be widely used to address these issues.

According to Chiu (2024), an Automated Medication Verification System (AMVS) has been developed based on edge detection and Convolutional Neural Network (CNN) classification of drugs on embedded systems. In simpler words, the system automates the verification process by employing edge detection techniques combined with deep learning models, particularly CNN. To tackle the challenges posed by manual medication verification, this project can minimize medication errors in hospital settings. In the traditional approach, nursing professionals used to verify the medication manually multiple times, significantly increasing their workload. Therefore, the system proposed by the author can guarantee higher accuracy in medication administration and enhance patient safety.

For the verification and training datasets, there are a total of 300 images from 10 drug categories were captured by the drug verification box. To reduce the effect of optical changes during the image capturing process, the drug verification box is designed to be fully sealed and light tight. In this case, LED light sources play a fundamental role in providing stable and consistent lighting because all the system operations are carried out in closed spaces. This setup will minimize the variations in pixel values that could be caused by external light sources. Ensuring high resolution of images captured, the distance between the camera and the 3D-printed medication tray is considered during the design process. Furthermore, the medication tray features outer walls with an angled design to minimize the segmentation errors. One of the special features of this medication box is the presence of an electromagnetic valve. As shown in Figure 2.7, the electromagnetic valve is utilized in the medication box to enhance user convenience by allowing for easy access.

In terms of software, it is designed to control the state of the LED, brightness, image capture size, and even the exposure time of the camera. Besides that, the edge detection analysis is carried out using the OpenCV package, while the PyTorch package is chosen for building the deep learning model. For this system, the author uses pre-trained deep learning models such as ResNet, VGG, AlexNet, MobileNet, and SqueezeNet to perform the medication classification.



The author also explained the workflow of the system in Figure 2.8, which consists of 2 steps, including classification and verification process. The CNN-based model is trained using the single pill images in step 1, while the same algorithm is used to identify the pill region and the predict the pill type.

By comparing the results of the Watershed algorithm and the Canny algorithm, the Watershed algorithm can perform edge detection effectively, while the Canny algorithm achieved an accuracy of 60%, but may result in error when identifying regions for different drug types. In terms of processing time for edge detection, the Watershed algorithm takes the lowest time consumption compared to other methods, as shown in Figure 2.9.

To verify the model accuracy, VGG11 is employed to evaluate the effect of Edge Detection Processing (EDP) with and without padding. As a result, there is a successful pill prediction when applying EDP with the padding method. Out of the eight classification models using EDP, VGG11 attained the greatest SA average accuracy of around 93%, as depicted in Figure 2.10. Additionally, VGG11 also showcased an impressive accuracy of approximately 96% when analysing images less than 10 pills. For other models, the SA average accuracy is around 60% to 80%. Thus, it can be proved that the VGG11 achieves a high level of accuracy compared to other models. Therefore, the VGG11 model is chosen as the deep learning model in this system. Ultimately, the watershed algorithm and pre-trained VGG11 model offer superior performance as compared to other edge detection methods and pre-trained models. This is due to the fact that the Watershed algorithm excels at the lowest time consumption, while the VGG11 model is able to achieve a high level of accuracy, effectively discerning the images.

By comparing among few models and techniques, the author chooses the edge detection techniques and pre-trained models with a high overall performance. Therefore, the system not only can save a lot of time but also reduce medication error that occurs in the future. Another benefit of this system is the drug verification box that is fully enclosed. This means that it will not be influenced by the external factors, thus improving the accuracy of the detection and classification process. In addition, this

system achieves an accuracy rate of 93% is one of the key benefits of this system. The high level of precision can enhance the efficiency of healthcare professionals, thus reducing the risk of medication dispensing errors occurring.

On the contrary, to deal with the problem of long training time using Raspberry Pi, the future improvement is to integrate IoT technology with Raspberry Pi for more effective and accurate performance in medication classification.

Overall, this AMVS can definitely help reduce medication dispensing errors through the integration of edge detection techniques with CNNs. At the same time, this study highlights the importance of integrating conventional edge detection techniques with CNNs to optimize the performance of the system.

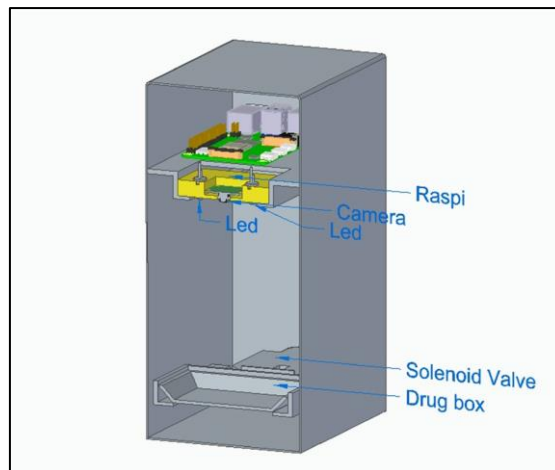


Figure 2.7: The Design of Drug Verification Box (Chiu, 2024)

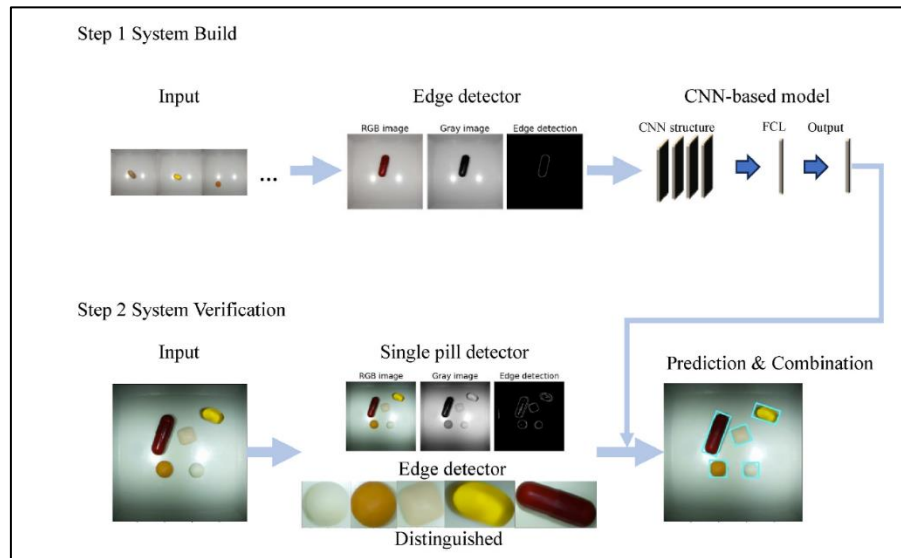


Figure 2.8: The Workflow of the System (Chiu, 2024)

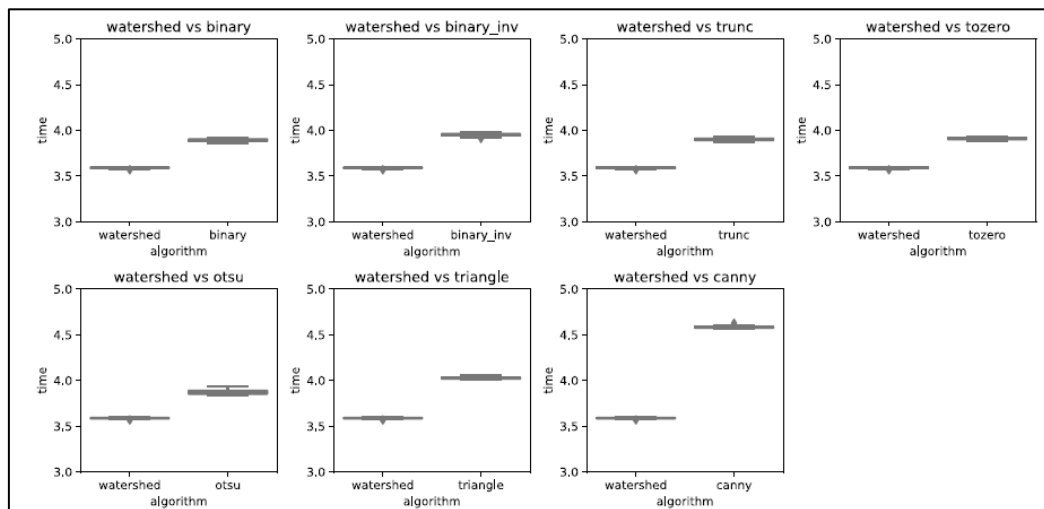


Figure 2.9: The Comparison of the Computational Time of Edge Detection Between the Watershed Algorithm and Other Methods (Chiu, 2024)

Model		Batch Size		
		4	16	32
resnet18	All	80 %	64 %	63 %
	Less than 10 pills	85 %	69 %	68 %
	10 pills	70 %	57 %	52 %
resnet50	All	79 %	70 %	78 %
	Less than 10 pills	85 %	75 %	84 %
	10 pills	65 %	57 %	59 %
vgg11	All	95 %	92 %	91 %
	Less than 10 pills	96 %	96 %	95 %
	10 pills	93 %	83 %	84 %
alexnet	All	91 %	92 %	92 %
	Less than 10 pills	96 %	97 %	97 %
	10 pills	77 %	82 %	80 %
mobilenet_v3_small	All	64 %	64 %	74 %
	Less than 10 pills	71 %	71 %	86 %
	10 pills	46 %	46 %	46 %
googlenet	All	73 %	72 %	76 %
	Less than 10 pills	76 %	77 %	82 %
	10 pills	66 %	60 %	64 %
squeezeNet	All	76 %	87 %	87 %
	Less than 10 pills	80 %	91 %	90 %
	10 pills	68 %	78 %	80 %
shufflenet	All	70 %	66 %	64 %
	Less than 10 pills	76 %	78 %	74 %
	10 pills	58 %	38 %	40 %

Figure 2.10: The SA Average Accuracy for Classification (Chiu, 2024)

#### 2.2.4 Development of Smart Pill Expert System Based on IoT Presented by Dayananda and Upadhya (2024)

According to Dayananda and Upadhya (2024), an enhanced version of the Smart Pill Expert System Based on IoT called SPEC 2.0, is designed. In order to minimize the improper dosage due to the medication system error, the goal of this system is to propose a user-friendly device. It aims to provide users with control and monitoring features through an Android application without any in-app purchases or subscriptions needed. On top of that, the system has the capability to send notifications and SMS messages to remind the user. From the perspective of the user, they can receive reminders based on their prescribed schedule. The standout feature of this system is the management of overdose. This system not only can remind the user to consume the medication on schedule but is also useful in avoiding excessive medication consumption. The usage of IoT in this system can enhance the efficiency of the pill box through a user-friendly interface.

To address the problem of overdose, the system includes a container box that serves as the pill box, which features a flap mechanism driven by a DC motor as illustrated in Figure 2.11. Besides being the container that holds the dispensed medication, the pill box also functions to drop the untaken medication into an overdose

pill container if not taken within a specific time limit. According to the predefined time duration threshold of 10 seconds, the dispensed medication is considered an overdose if the user does not take the medication from the pill box. In fact, the pill's detection is carried out by an infrared obstacle sensor that serves to verify if the user has taken the medication. In this system, the buzzer and LCD are used to serve as the alert notification and display component.

For the software implementation, an Android mobile application is built using Java Development Kit (JDK), and a common user interface (UI) is built using Eclipse. Furthermore, the system incorporates both software and hardware components, ensuring seamless connectivity between the GSM module and the mobile application. This integration allows the user to receive the SMS notifications directly from the system, so that the SMS can be received by the user from the system, carrying out the function of real-time tracking.

For this system, there are some pros and cons. First of all, it will be beneficial for the elderly user due to its usability and user-friendly design, making it easier for them to understand and operate. Next, the overdosage prevention is another advantage of the system that can monitor the medication intake of the user. However, one major limitation of the system is that it only provides for a single user interface, which means that it can only accommodate the needs of one user at a time. Furthermore, this system has the limitation of only restricted to solid pharmaceutical forms and is not suitable for non-oral medication. Lacking a backup battery is another drawback of this system. In this context, it is likely to lead to unexpected power outages.

To enhance the accuracy and ensure the functionality of the system, 8 test cases are introduced in the testing process as illustrated in Figure 2.12. Figure 2.13 shows the comparative analysis of SPEC 2.0 with the existing system. By referring to the comparative analysis, the system achieves 90% accuracy, making a 5% improvement over existing systems. The improvement can be done is to designing using 3D printing in order to enhance its reliability. Furthermore, the sudden power outage problem can be improved by including a backup battery to ensure the system continues to function and prevent missed doses in this situation. Ultimately, the SPEC 2.0 can be improved

by increasing the time slots, enabling the generation of reports for dispensing events, and incorporating enhanced security features such as a fingerprint feature. In conclusion, SPEC 2.0 is effectively tackling challenges like unreliable performance and improper medication dosage. By incorporating advanced technologies and features, medication adherence can be improved, and the risk of overdose is significantly minimized.

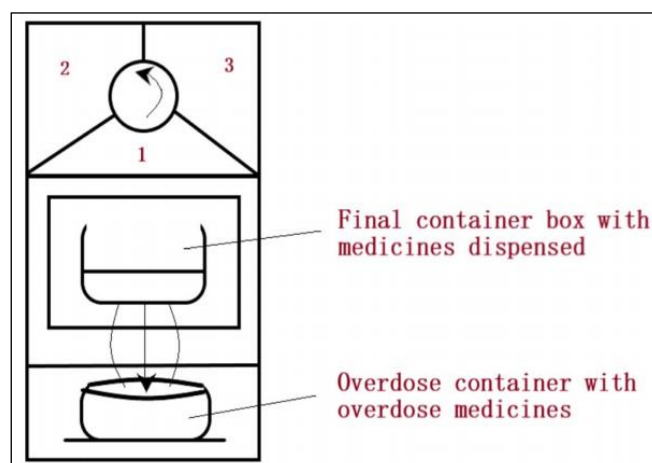


Figure 2.11: The Working Dispensing Mechanism of the System (Dayananda and Upadhya, 2024)

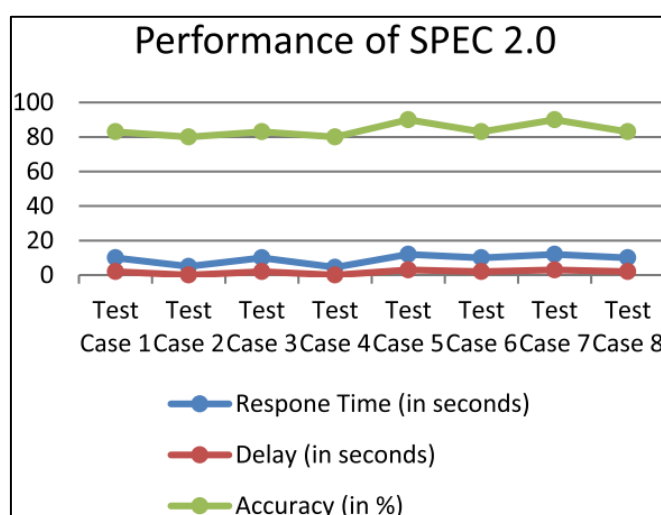


Figure 2.12: Accuracy and Functionality Graph of SEPC 2.0 (Dayananda and Upadhya, 2024)

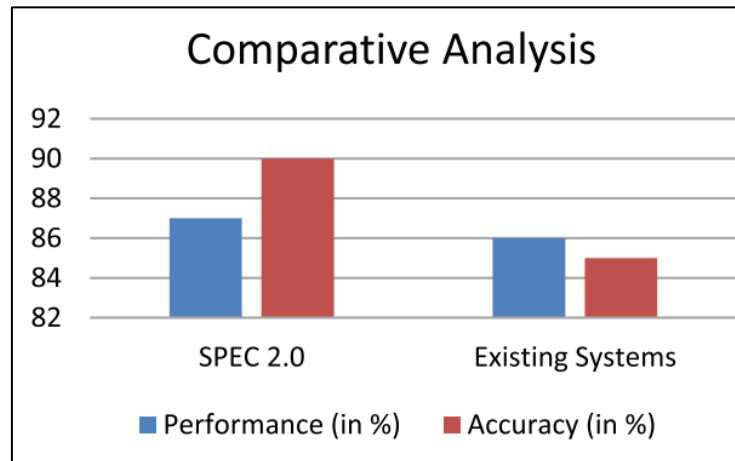


Figure 2.13: Comparative Analysis of SPEC 2.0 with Existing System (Dayananda and Upadhya, 2024)

Table 2.1: Comparison of Techniques Related to Intelligent Medicine Box System

Author	Year	Title and Details	Advantages/ Disadvantages
Chiu	2024	<u>Automated Medication Verification System (AMVS)</u> <ul style="list-style-type: none"> <li>Automates the classification of medications using a combination of edge detection methods with deep learning models</li> </ul>	Advantages: <ul style="list-style-type: none"> <li>Low time consumption</li> <li>Achieved over 93% accuracy</li> </ul>
Dayananda and Upadhya	2024	<u>Development of Smart Pill Expert System Based on IoT</u> <ul style="list-style-type: none"> <li>Developed an enhanced version of the Smart Pill Expert System Based on IoT called SPEC 2.0</li> </ul>	Advantages: <ul style="list-style-type: none"> <li>User-friendly interface</li> <li>Overdosage prevention</li> </ul> Disadvantages: <ul style="list-style-type: none"> <li>Only one user interface</li> <li>Without a backup battery for power</li> </ul>
Azlan and Yahya	2023	<u>Smart Medicine Pill Box Reminder</u> <ul style="list-style-type: none"> <li>Smart Medicine Pill Box for reminder purposes and equipped with a weight sensor</li> </ul>	Advantages: <ul style="list-style-type: none"> <li>The status of the pill box can be monitored</li> </ul> Disadvantages: <ul style="list-style-type: none"> <li>Difficult for daily monitoring</li> <li>Medication shape and size constraint</li> </ul>
Nasir <i>et al.</i>	2023	<u>Design of a Smart Medical Box for Automatic Pill Dispensing and Health Monitoring</u> <ul style="list-style-type: none"> <li>Smart Medical Box with health monitoring system, including measuring for temperature, heart rate, and oxygen level</li> </ul>	Advantages: <ul style="list-style-type: none"> <li>Enhance security with biometric recognition</li> <li>User-friendly GUI</li> </ul> Disadvantages: <ul style="list-style-type: none"> <li>Not suitable for non-oral medication</li> <li>Limited compartments</li> </ul>



## **CHAPTER 3**

### **METHODOLOGY**

#### **3.1 Introduction**

Methodology is an important aspect of any project because it outlines the components and techniques employed throughout the project. This chapter covers the detailed descriptions of hardware and software components used in developing the smart medicine box system for health monitoring. It provides a detailed description of components utilized and outlines the research design and methodology employed. The chapter is divided into three sections, which are project timeline, component description and block diagram. Together, these sections provide a clear framework for the system's development process.

#### **3.2 Project Management**

A Gantt chart is a helpful tool for project management as it can help in planning and scheduling the project timeline effectively. This project is scheduled using a Gantt chart to ensure that all the tasks will be completed on time without delay. It offers a visual timeline for each task and also helps to track the progress. Table 3.1 shows the Gantt chart for Final Year Project 1, while Table 3.2 illustrates the Gantt chart for Final Year Project 2.



Table 3.2: Gantt Chart for FYP 2

Details/ Weeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Model Training														
Assembling Circuit														
Building the Mobile Application														
3D Printing of Prototype														
Debugging & Testing														
Reporting														
FYP 2 Oral Presentation														
FYP 2 Report Submission														

### 3.3 Component Description

#### 3.3.1 Raspberry Pi 4 Model B

As shown in Figure 3.1, Raspberry Pi is a tiny single-board computer (SBC) that is equipped with all the essential components necessary for a computer to operate. Compared to the other models, Raspberry Pi 4 comes with a more powerful processor and provides a choice of RAM configurations. The structure of the Raspberry Pi Model B is illustrated in Figure 3.2. As shown in Figure 3.3, it includes a total of 40 general-purpose input/output (GPIO) pins in order to interface and communicate with other electronic circuits or components. There are dual-HDMI ports to support multiple monitors. Thus, a micro-HDMI to HDMI cable is needed to connect from the micro-

HDMI ports to the screen. Besides that, Raspberry Pi Model B also features a Wi-Fi port, four USB ports, a USB Type-C power jack, and a Gigabit Ethernet port for quicker stressed network connectivity. The micro-SD card slot also plays a fundamental role in the Raspberry Pi board as it serves as the storage medium for files and the Operating System for turning the system on. Basically, the Raspberry Pi is a programmable device that supports various types of programming languages, like Python, Java, and C. In short, it is similar to a minicomputer that can be used to perform numerous applications, ranging from simple programming projects to complex home automation projects (Saddam, 2016). All the specifications and respective descriptions are listed in Table 3.3.

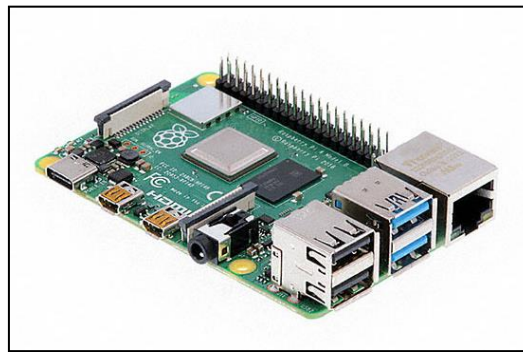


Figure 3.1: Raspberry Pi 4 Model B

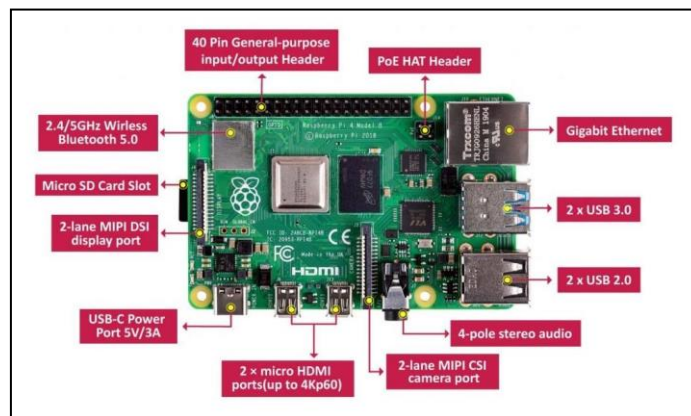


Figure 3.2: Structure and Specifications of the Raspberry Pi Model B (Singh, 2021)

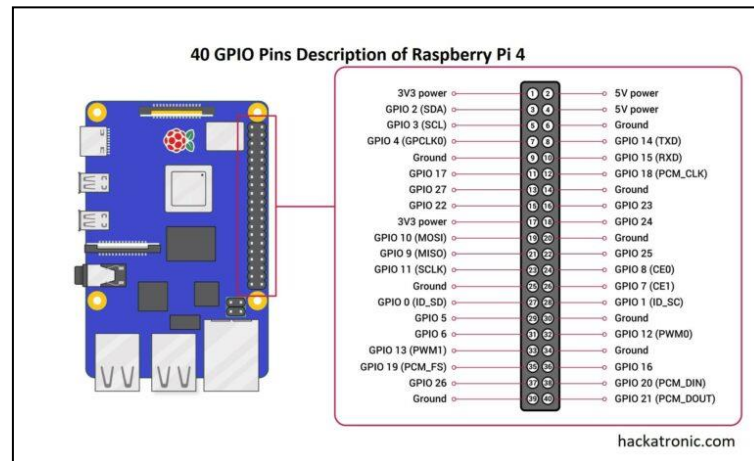


Figure 3.3: 40 GPIO Pins of the Raspberry Pi 4 Model B (Singh, 2021)

Table 3.3: Specifications and Description of the Raspberry Pi 4 Model B

Specifications	Descriptions
Processor	quad-core Broadcom BCM2711B0 (Cortex A-72) 64-bit SoC @ 1.5GHz
Memory (RAM)	1GB, 2GB, 4GB or 8GB (depends on model)
Camera Interface	2-lane MIPI CSI camera port
Display Interface	2-lane MIPI DSI display port
Power over Ethernet (PoE)	enabled
GPIO	40 pins
USB	2 × USB 2.0 ports, 2 × USB 3.0 ports
HDMI	2 × micro-HDMI ports
Input Power	5V via USB Type C connector 5V via GPIO header PoE capability via separate POE HAT (add-on)
Operating System	Raspberry Pi OS, as well as Linux and Windows 10
WIFI	Available
Bluetooth	5.0, BLE
Operating Voltage	3.3V
Operating Temperature Range	0 to 50 degrees Celsius

### 3.3.2 Raspberry Pi Camera Module V2

The Raspberry Pi Camera Module V2 as illustrated in Figure 3.4, plays a fundamental role in this project by capturing the images. It is equipped with an ultra-high quality 8-megapixel Sony IMX219 image sensor and consists of a fixed focus lens on-board. The camera module is connected to the 15-pin Raspberry Pi Camera Serial Interface (CSI) camera connector on the top of the Raspberry Pi board using a ribbon cable. Furthermore, this camera module is capable of capturing  $3280 \times 2464$ -pixel static images. It is also able to support video recording at various resolutions, including 1080p at 30fps, 720p at 60 fps, and  $640 \times 480$ p at 90 fps. One of the benefits of this camera module is its small size and lightweight design.

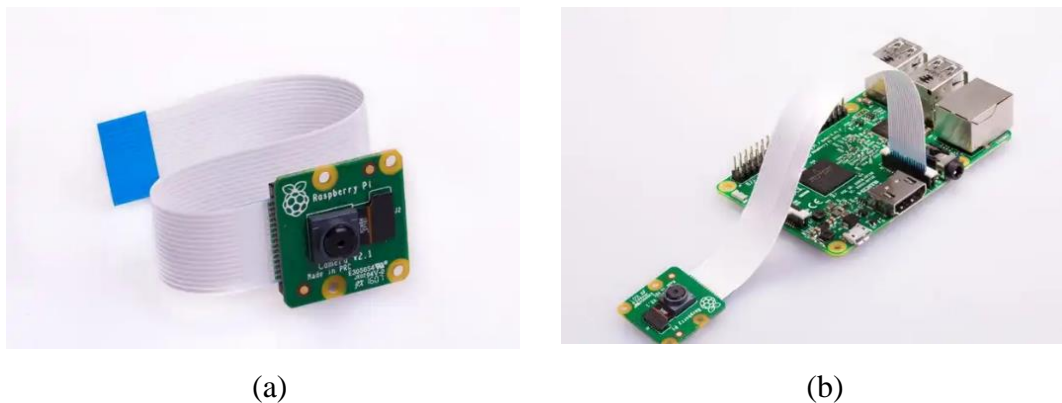


Figure 3.4: Raspberry Pi Camera Module V2 (a) Camera Module with Ribbon Cable (b) Camera Module Connected to Raspberry Pi Board

### 3.3.3 $16 \times 2$ Liquid Crystal Display (LCD)

A  $16 \times 2$  LCD is an electronic display module that supports diverse applications. Basically, the main function of an LCD is to display information or data. As shown in Figure 3.5, this display can show 32 characters, with 16 columns and 2 rows, while the characters can be displayed in a  $5 \times 7$ -pixel matrix. There are a total of 16 pins that carry out different tasks, including power supply, data communication, and signal control. Table 3.4 shows the pinout of a  $16 \times 2$  LCD, DB0 to DB7 refer to Data Bus lines 0 to 7.

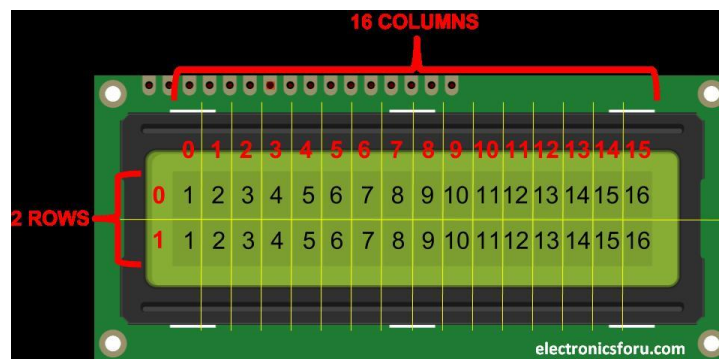


Figure 3.5: 16 × 2 LCD (Cd-Team, 2023)

Table 3.4: Pinout for the 16×2 LCD

Pin No.	Pin Name
1	Ground
2	Vcc
3	Vo/V <sub>EE</sub>
4	RS (Register Select)
5	Read/Write
6	Enable
7	DB1
8	DB2
9	DB3
10	DB4
11	DB5
12	DB6
13	DB7
14	DB8
15	Led+ (LED backlight Vcc)
16	Led- (LED backlight ground)

### 3.3.4 Light Emitting Diode (LED)

LED is one of the most common hardware components used in embedded systems. The main function of an LED is to emit light when an electric current is applied and flows through it. It performs this specific function by ensuring that the current flows forward while blocking the current in the reverse direction. Due to their small size and minimal power usage, they are suitable for many applications. An LED is used to provide stable and consistent illumination. In this context, it enhances the accuracy of the system in detecting and counting the medication in the medicine box. As shown in Figure 3.6, the LEDs have different colours.



Figure 3.6: Light Emitting Diode (Scully, 2019)

### 3.3.5 Acrylic Diffuser Sheet

Figure 3.7 shows acrylic diffuser sheets that are used to diffuse light evenly while maintaining the optimal brightness (Jinbao Plastic, 2025). It helps in spreading and softening the light from sources like LED, ensuring the light is evenly distributed without bright spots or glare.



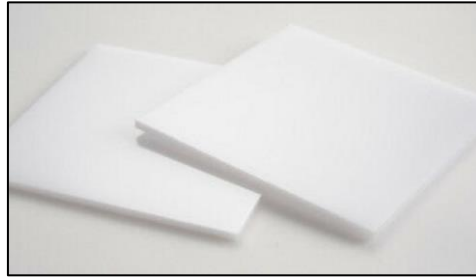


Figure 3.7: Acrylic Diffuser Sheets

### 3.3.6 RGB LED

The RGB LED in Figure 3.8 is an LED module that uses three primary colours: red (R), green (G), and blue (B) to generate different colours. By combining these colours in different intensities, it can produce almost any colour. In a common cathode RGB LED, all three LEDs share the same cathode. As shown in Figure 3.9, an RGB LED has four legs, and the longest leg represents the common cathode and the other colours are following in the sequence of red, cathode, green, and blue.



Figure 3.8: RGB LED

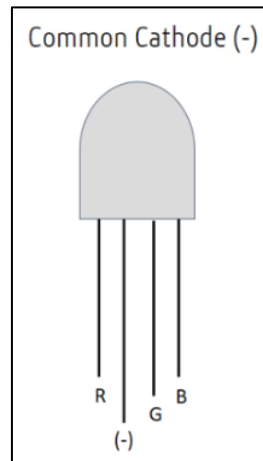


Figure 3.9: Common Cathode RGB LED Pins (Santos, 2019)

### 3.3.7 NodeMCU ESP8266 V2

NodeMCU ESP8266 V2, shown in Figure 3.10, is an IoT development board that is built around a System-on-Chip called ESP8266. It comes with built-in Wi-Fi features that allow access to online services. Therefore, it is suitable for IoT-based applications. It featured with 4 MB of Flash memory with a clock speed of 80 MHz. There are several peripherals included in the ESP8266, such as 17 GPIO, SPI, I2C, UART, and 10-bit ADC. Figure 3.11 illustrates the pinout of the NodeMCU ESP8266 V2.

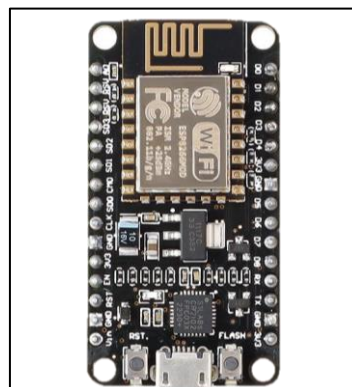


Figure 3.10: NodeMCU ESP8266 V2

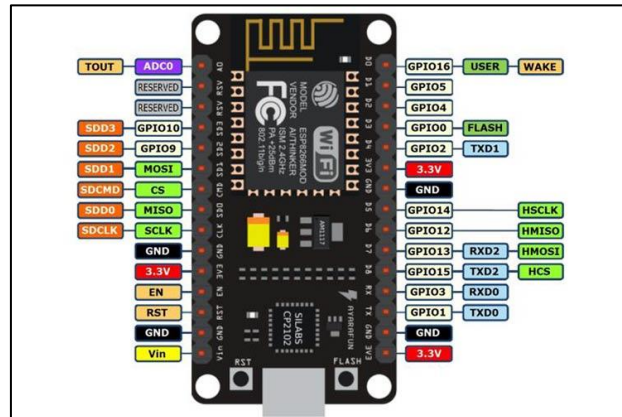


Figure 3.11: NodeMCU ESP8266 V2 Pinout

### 3.3.8 Buzzer

A buzzer is an audio signalling device that uses electrical signals to generate sound, as shown in Figure 3.12. It is commonly used for delivering an audible alert or notification by converting audio signals into sound signals. It consumes less energy and has a smaller size.

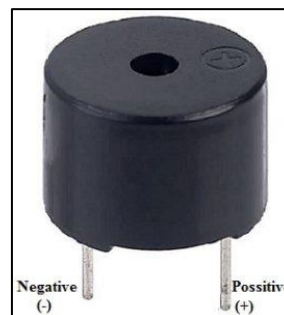


Figure 3.12: Buzzer (Agarwal, 2021)

### 3.3.9 LC18650 Li-Ion Rechargeable Lithium Battery

The LC18650 3.7V Li-ion rechargeable battery (3800mAh), as illustrated in Figure 3.13, is a lithium-ion battery that can be recharged using a rechargeable battery charger when its power is depleted. The 3800mAh represents the capacity of the battery. This highlights that the battery can supply 3800 milliamperes for one hour. Compared to the normal non-rechargeable battery (primary battery), this type of battery is known for its rechargeability and eco -friendly.



Figure 3.13: LC18650 Li-Ion Rechargeable Lithium Battery

### 3.3.10 18650 Battery Holder

The 18650 battery holder is designed to hold the 18650 lithium-ion battery and connect it to electronic circuits or devices. It serves as a place to make electrical contact with battery terminals. Figure 3.14 shows the single cell holder for a 18650 battery.



Figure 3.14: 18650 Battery Holder

### 3.3.11 MT3608 Step-Up Power Module

The MT3608 power module shown in Figure 3.15 is a step-up (boost) converter that is mainly used to power up devices by generating a higher output voltage from a lower input voltage. Therefore, it is suitable for devices that have limited power. This module can regulate the output voltage up to 28V and the output current of 2A. There is a trimmer potentiometer called trimpot on the boost converter that can be adjusted using a screwdriver to adjust the desired output voltage, as shown in Figure 3.16.

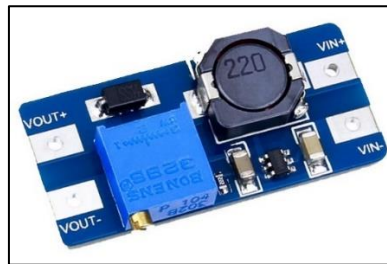


Figure 3.15: MT3608 DC-DC Boost Converter

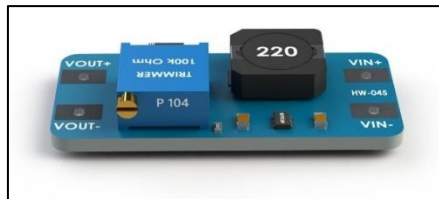


Figure 3.16: Trimmer for Adjusting Voltage (Robottronic, 2020)

## 3.4 Software Descriptions

In developing the mobile application for the Intelligent Medicine Box System, several development tools have been used. Cloud Firestore serves as the cloud database for data storage and synchronization. Android Studio Emulator and Flutter are used to build and test the mobile application. Visual Studio Code was utilized as the main Integrated Development Environment (IDE) for writing and debugging code. To program a NodeMCU ESP8266 V2, an Arduino IDE is used.

### 3.4.1 Cloud Firestore

When developing a modern application, the database is stored in the cloud to enable seamless synchronization across all devices. Cloud Firestore is a NoSQL (Not Only SQL) document database offered by Google Firebase to help in optimizing the application development process. The logo of Cloud Firestore is shown in Figure 3.17. It provides the functions to store, query, and synchronize data for both mobile and web applications. In contrast to the conventional SQL (Structured Query Language) that stores data with tables and rows, Cloud Firestore organizes data as collections of documents, making the data stored can be accessed more quickly. In Cloud Firestore, data is stored in collections where each collection contains multiple documents, and each document holds the data (Abba, 2021). This structure enables the retrieval of data to be more flexible. Therefore, it can improve data management by organizing complex databases efficiently. Based on the diagram in Figure 3.18, it shows the visual structure of the Cloud Firestore database which the data are store of the Intelligent Medicine Box System stored in collections and documents. Figure 3.19 shows a screenshot of Cloud Firestore, displaying the data stored for the Intelligent Medicine Box System.

In terms of security, Cloud Firestore integrates with Firebase Authentication, allowing developers to build a secure and efficient mobile application. With the Cloud Firestore security rules, it ensures users' data is protected by implementing strict access control and enforcing data validation. The rules can be customized based on the developers' preferences. As shown in Figure 3.20, the customized Cloud Firestore security rules for time-limited access to the data highlight that users are allowed to read or write to the database only if the request is before 31 December 2025 (Coordinated Universal Time, UTC). Furthermore, Cloud Firestore also offers offline support, meaning the application can still function and can perform read/write data even without connecting to an internet connection (Lee, 2019). Once connectivity is restored, it synchronizes any changes back to the Cloud Firestore automatically.



Figure 3.17: Logo of Cloud Firestore

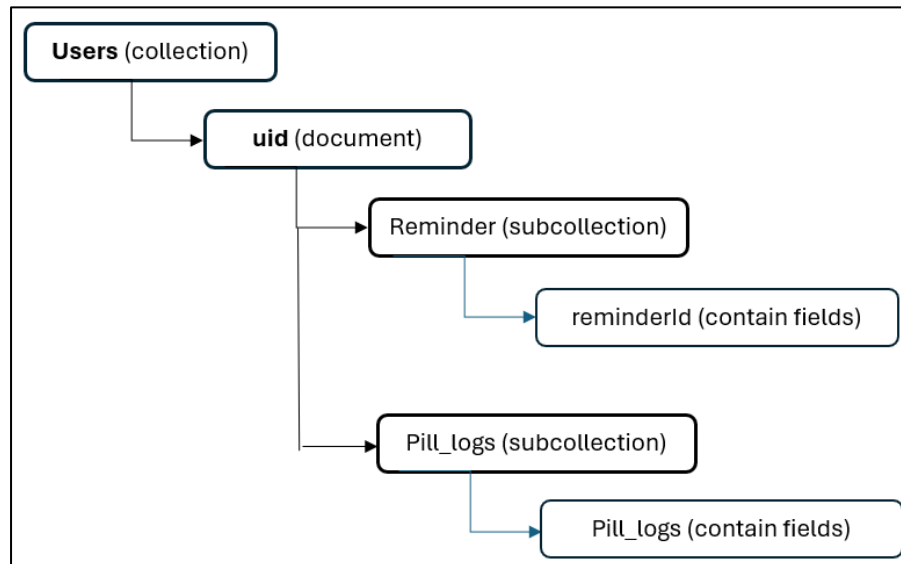


Figure 3.18: Data Structures in Cloud Firestore

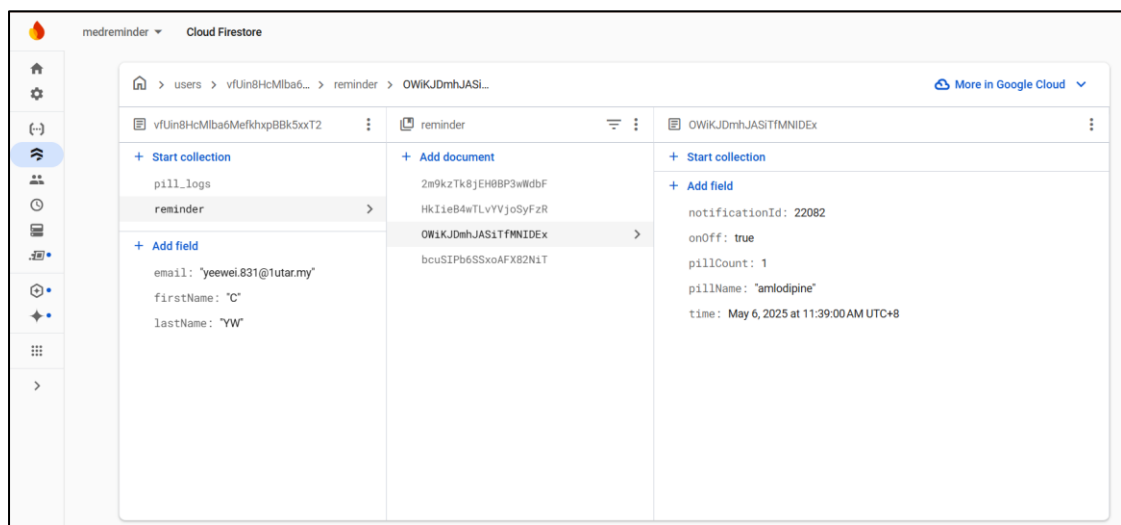


Figure 3.19: Cloud Firestore Database

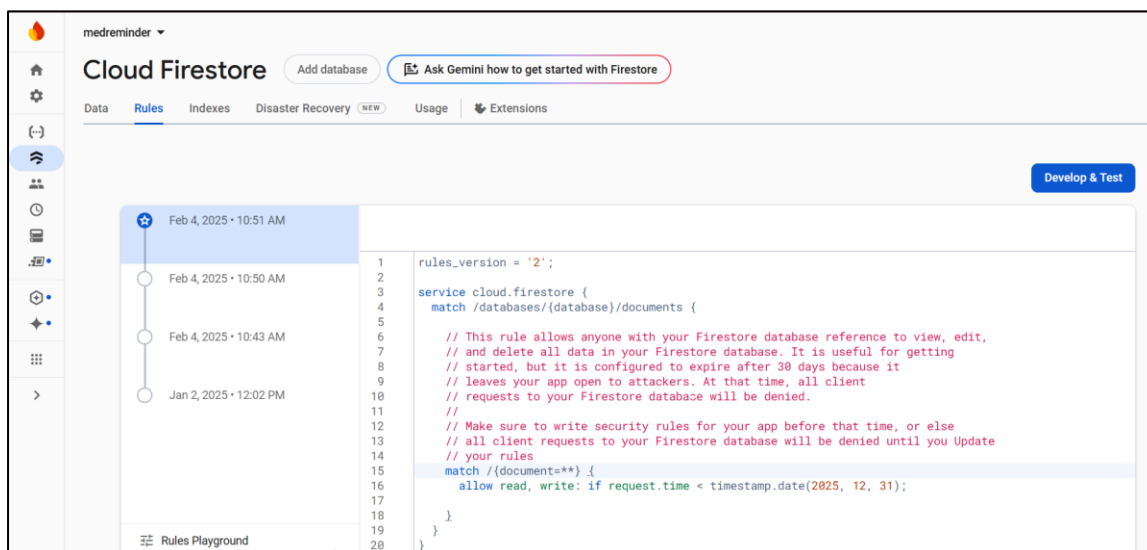


Figure 3.20: Customized Cloud Firestore Security Rules

### 3.4.2 Android Studio

Android Studio is designed mainly for developing Android applications. The logo of Android Studio is shown in Figure 3.21. It is Google's official IDE for Android app development. It offers features that enhance productivity when building apps. It provides everything that developers need to design, build, test, and debug. Android applications, such as layout editor, code editor, and debugging tools, make it easier to develop and test Android apps (Harwani, 2024). On top of that, a virtual android device called an emulator is also provided to allow the developers to test their apps without needing physical hardware. Figure 3.22 shows that the Pixel 4 API 35 emulator is generated using Android Studio.



Figure 3.21: Logo of Android Studio



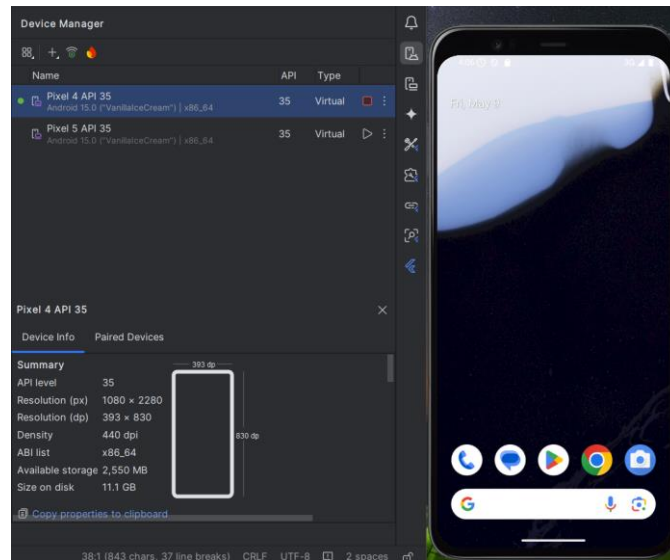


Figure 3.22: Pixel 4 API 35 Emulator

### 3.4.3 Flutter

Flutter is an open-source UI toolkit developed by Google, which is primarily intended to build multi-platform applications using a single codebase. The logo of Flutter is shown in Figure 3.23. The single codebase implies that its developers can write code and deploy it on several platforms. It is beneficial for developers in building web, desktop, and mobile applications more productively. The programming language used in Flutter is Dart. The Dart language is optimized for building UIs. In simpler terms, Flutter apps are designed using UI building blocks, which are known as widgets, that are suitable for developers to build their customizable interfaces. In simpler terms, it builds with widgets inside widgets. Flutter also provides developer-friendly tools like the hot reload feature, which is important for developers to view the changes in the apps faster and more easily without fully restarting the applications.



Figure 3.23: Logo of Flutter

### 3.4.4 Visual Studio Code

Visual Studio Code is an IDE that can be utilized to develop a Flutter app. It is a code editor to build and debug apps with extensibility that is compatible with programming languages like Python, Java, and C++. Visual Studio Code is equipped with a vast array of extensions, including plugins for Flutter and Dart, making it suitable for Flutter app development. Compared with Android Studio, Visual Studio Code is more lightweight and faster when startup.

### 3.4.5 Arduino IDE

Arduino IDE is an open-source IDE that is used to write code, compile, and then upload it to the board. It consists of a text editor and compiler for code writing and compilation, allowing users to write, test, and upload the program to the board without an internet connection. The Arduino software is beginner-friendly software that supports programming languages like C or C++. In addition, it has a serial monitor that shows the output from the board by selecting the corresponding baud rate (Ehsan *et al.*, 2018).

## 3.5 Block Diagram

Figure 3.24 shows the block diagram of the Intelligent Medicine Box System, consisting of the mobile application, Home Base, and Pill Box. The mobile app is built using Flutter in Visual Studio Code and tested using an emulator. In the Pill Box, the ESP8266 microcontroller, which is powered by a rechargeable battery, is used to fetch reminders from Cloud Firestore and trigger the buzzer. An RGB LED inside the Pill Box shows different colours according to the current task when the program is running. In the Home base, a Raspberry Pi runs the YOLO model to process images captured by the Raspberry Pi camera module. Since the operation takes place in an enclosed environment, consistent lighting is essential. Therefore, LEDs are mounted on diffuser sheets and

positioned on both sides of the camera to diffuse light evenly while maintaining the optimal brightness. Moreover, the LEDs and an LCD attached to the Home Base serve as visual outputs.

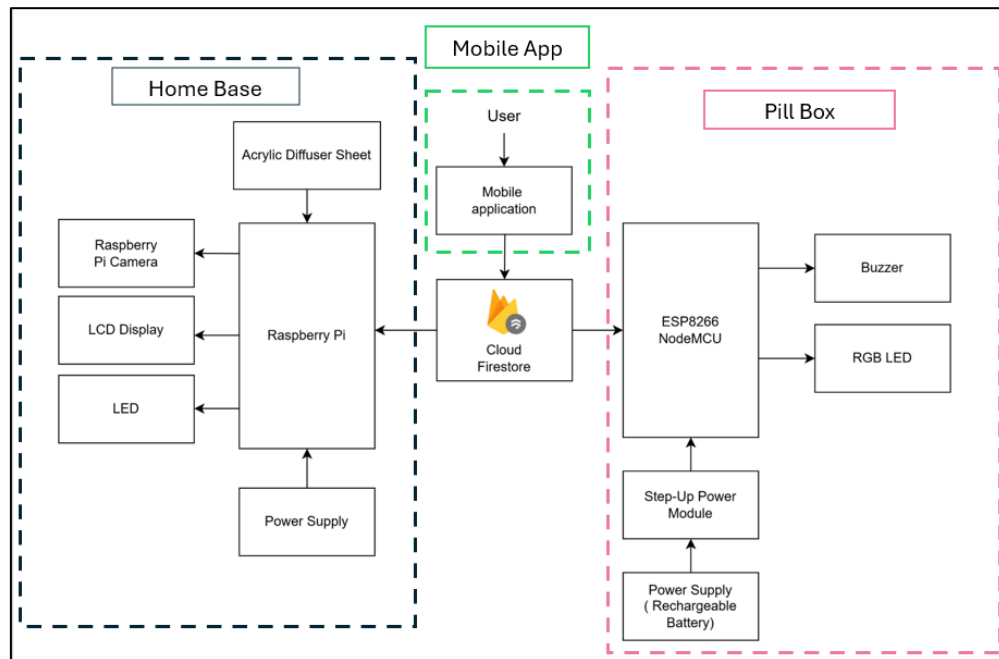


Figure 3.24: Block Diagram of Intelligent Medicine Box System With AI-Powered Pill Detection and IoT Integration

## 3.6 Arduino IDE and Raspberry Pi Setup

### 3.6.1 Setup for Arduino IDE

In this project, the NodeMCU ESP8266 V2 is programmed using the Arduino IDE. To begin, the development environment should be properly set up to support the ESP8266 board family. First and foremost, the following URL “[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)” is entered into the Additional Boards Manager URLs field found in the Preferences menu. Figure 3.25 depicts the Preferences menu with the URL added to enable access to the ESP8266 board. After that, “esp8266 by ESP8266 Community” is installed from the Boards Manager as shown in Figure 3.26. Once installed, the NodeMCU ESP8266 V2 is

connected to the laptop via a USB cable. Next, NodeMCU 1.0 (ESP-12E Module), which corresponds to the NodeMCU ESP8266 V2, is selected from the Tools options as illustrated in Figure 3.27. At this point, the board is ready to be programmed using the Arduino IDE.

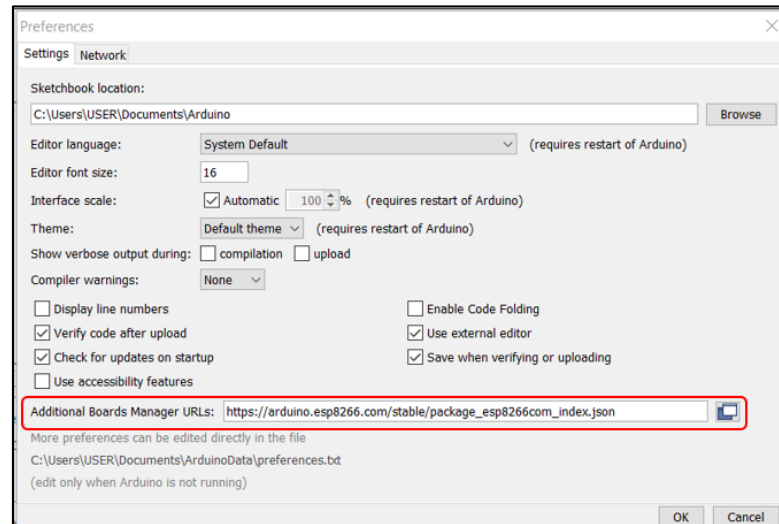


Figure 3.25: URL Added in the Preferences Menu



Figure 3.26: Installation of ESP8266 Board Package

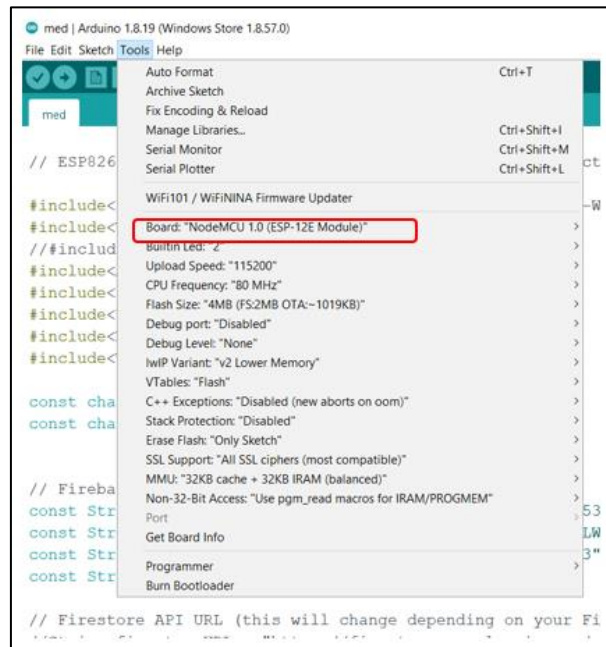


Figure 3.27: NodeMCU 1.0 (ESP-12E Module) Is Selected

### 3.6.2 Setup for Raspberry Pi

To start working with a Raspberry Pi, a microSD card with the Raspberry Pi OS installed is required. For Raspberry Pi 4 Model B, it requires a USB-C power supply with the specifications of a voltage of 5V and a current of 3A. When operating the Raspberry Pi directly, additional peripherals such as a display, a display cable that connects the Raspberry Pi to the display, a keyboard, and a mouse are required to enable full interaction and control the Raspberry Pi. Installing the operating system on the microSD card is an essential step, and it can be done by using Raspberry Pi Imager. Once the Imager is installed, it allows the user to select the Raspberry Pi Device, preferred operating system, and storage as shown in Figure 3.28. At the top of the list, it will show the recommended version of the Raspberry Pi Operating System, as depicted in Figure 3.29.

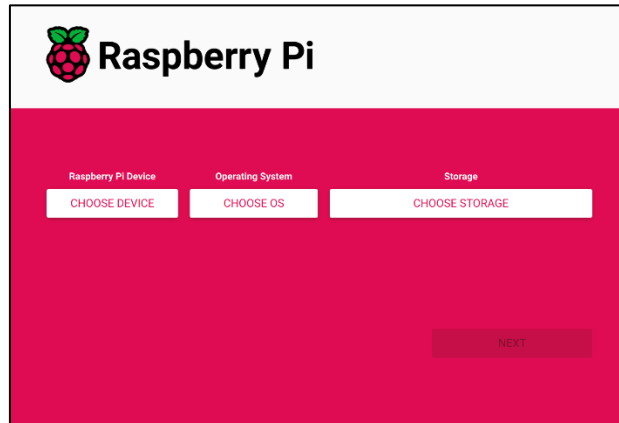


Figure 3.28: Selections Available in Raspberry Pi Imager

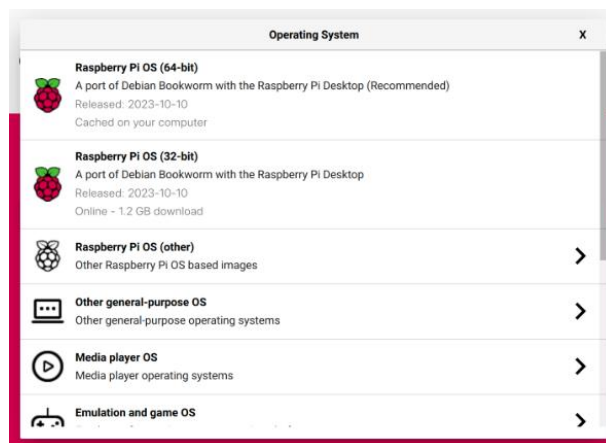


Figure 3.29: Recommended Operating System Shown at the Top

### 3.6.2.1 Headless Connection to Wi-Fi

Before the first boot of the Raspberry Pi, an OS customization menu will appear, allowing users to set up their username, password, Wi-Fi connections, and other advanced settings. Figure 3.30 illustrates the settings for Wi-Fi credentials, where the SSID is filled with the name of the currently connected network and its corresponding password. Once completed, save all the settings. Alternatively, Wi-Fi setup can be done manually using a configuration file. Firstly, a text file named “wpa\_supplicant.conf” is created with the contents illustrated in Figure 3.31. The SSID and password are entered as the Wi-Fi name and password, respectively, while the country code for Malaysia is designated as MY. Afterwards, the text file needs to be copied to the boot directory on

the microSD card before inserting the microSD card into the Raspberry Pi. Once booted, the Raspberry Pi will establish a connection to the Wi-Fi network.

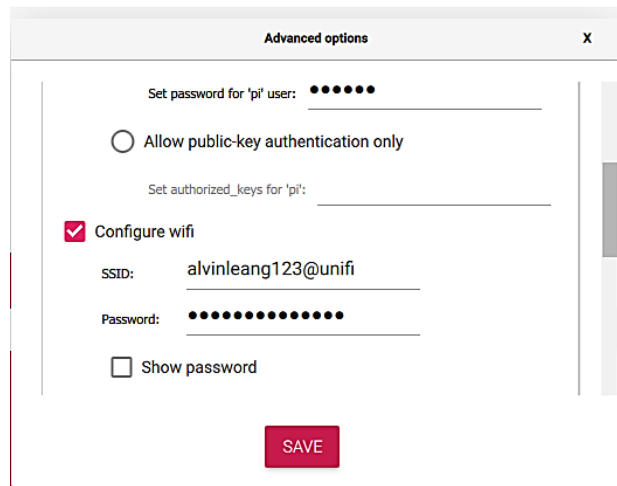


Figure 3.30: Advance Settings for Wi-Fi Credentials

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=<Country Code>

network={
    ssid="<SSID>"
    psk="<PASSWORD>"
    scan_ssid=1
}
```

Figure 3.31: Text File Is Created

### 3.6.2.2 Secure Shell (SSH)

SSH is mainly used for the purpose for remote access. In the Raspberry Pi Imager tool, SSH can be enabled, and a username and password can be created as shown in Figure 3.32.

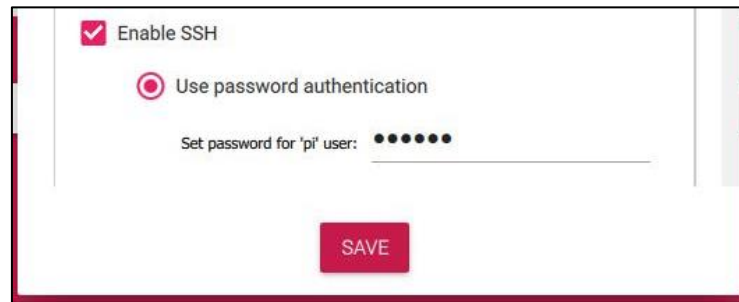


Figure 3.32: Settings for SSH

### 3.6.2.3 Virtual Network Computing (VNC)

To remotely control the Raspberry Pi from another internet-connected device, enabling VNC is essential for secure access to desktop screen sharing on the Raspberry Pi. With this VNC connection, location is no more a concern for the user. As illustrated in Figure 3.33, the VNC server in the Raspberry Pi configuration can be enabled. RealVNC Viewer is installed to facilitate the remote control of the Raspberry Pi. By entering the IP address of the Raspberry Pi into the RealVNC Viewer, users can establish a connection to the VNC server. After that, the RealVNC Viewer grants full access to the graphical desktop interface of the Raspberry Pi. Figure 3.34 shows the home screen of the RealVNC Viewer. By entering the username and password in the authentication window, the connection will be established. The authentication window is illustrated in Figure 3.35.



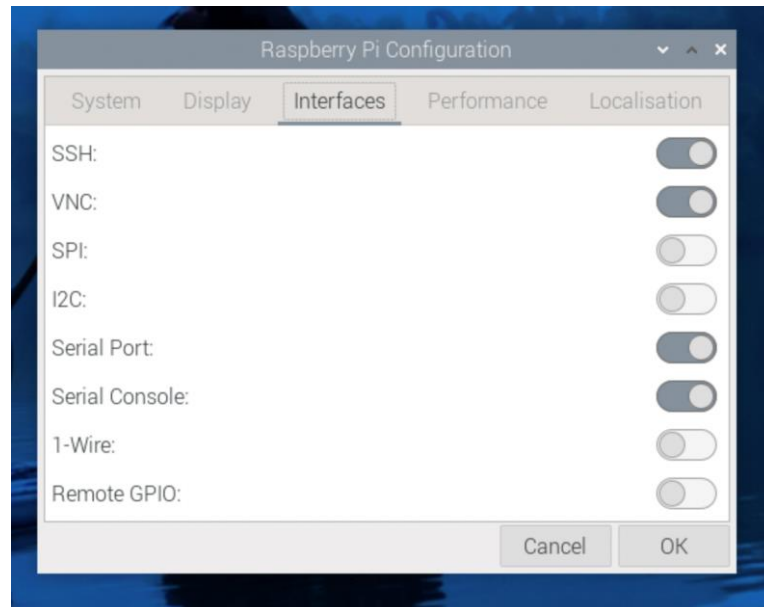


Figure 3.33: VNC Enabled in the Raspberry Pi Configuration

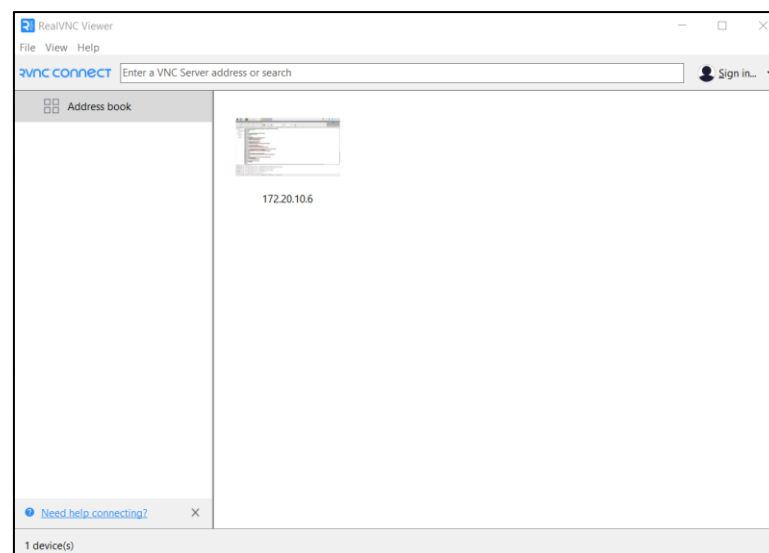


Figure 3.34: RealVNC Viewer Home Screen

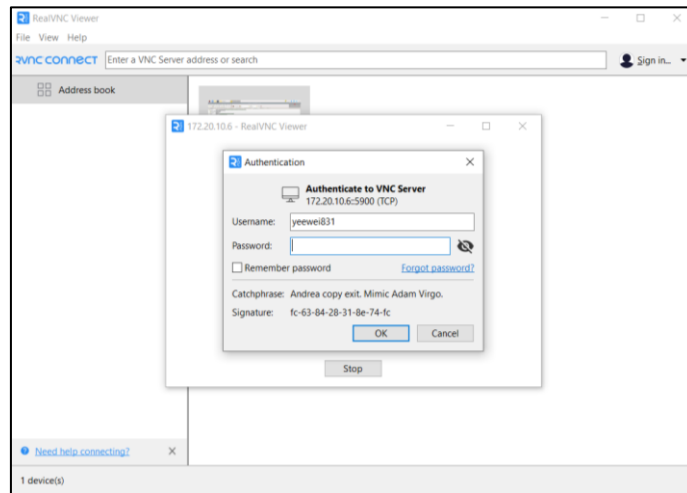


Figure 3.35: Authentication Window in RealVNC Viewer

## 3.7 Pill Detection

### 3.7.1 Object Detection

Object detection is one of the computer vision tasks that identify, classify, and locate objects within an image or video by drawing bounding box around them. Object detection models can be trained to detect multiple objects simultaneously and are widely used in many applications ranging from medical imaging to self-driving cars (Murel, 2024).

### 3.7.2 YOLO (You Only Look Once)

YOLO is a real-time object detection algorithm. As illustrated in Figure 3.36, the YOLO algorithm processes images by resizing them and then passing them through a single Convolutional Neural Network (CNN). CNN will then process the input image by dividing it into a grid, and with each cell predicting multiple bounding boxes along with the class probability. After that, based on the confidence scores, YOLO will threshold these detections and filter out the low-confidence predictions (Redmon *et al.*, 2016). In

addition, the YOLO algorithm is a single-shot object detection algorithm that involves a single pass of the input image to predict the bounding boxes and class of the object. In other words, YOLO can predict the presence of an object and the bounding box in a single pass of the network (Buhl, 2024). Therefore, YOLO is fast in detecting objects, making it suitable for real-time applications.

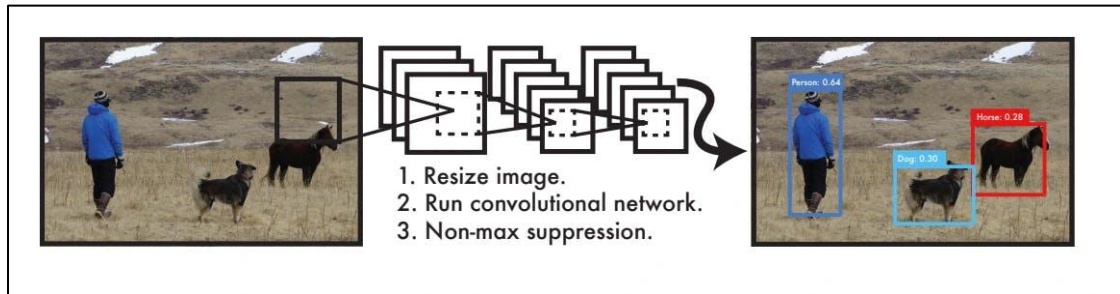


Figure 3.36: Detection System of YOLO (Redmon *et al.*, 2016)

### 3.7.3 YOLO Model Training Workflow

In training a YOLO model on the custom dataset, the process begins with dataset collection. After collecting the datasets, the next step is to annotate them by drawing the bounding boxes and their respective class labels. Then, the dataset is partitioned to split it into training and validation sets. After that, the YOLO model is trained and evaluated. The workflow of the YOLO model training is depicted in Figure 3.37.

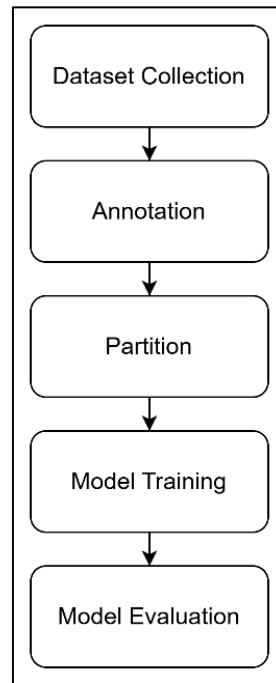


Figure 3.37: YOLO Model Training Workflow

### 3.7.3.1 Dataset Collection

To train the YOLO model for object detection, all the images for the custom dataset are captured using the Raspberry Pi Camera Module V2. By ensuring the consistency of the image captured, the camera is fixed on a stable platform at Home Base, facing inward toward the centre where the Pill Box is placed. Besides that, a Python script is written to capture the images for the dataset. Figure 3.38 presents the output messages during the capturing process.

In this project, the YOLO model is required to detect and classify four object classes, which are Amlodipine, Simvastatin, BoxPresent (the presence of a Pill Box), and Unknown (foreign object that may appear inside the Pill Box). Images are captured with different lighting conditions, orientations, different pill counts, and pill arrangements (single pill, partially overlapping) to vary the conditions. This approach helps the model improve its robustness and reliability in real-world situations. A total of 3200 images are captured and used as the dataset for this system. Figure 3.39 shows the images captured using a Python script.

```

Format: 1640x1232-pBAA
Camera warming up...
Capturing process started...
Captured /home/yeewei831/dataset/v4/pill_0.jpg
Resized (padded): /home/yeewei831/dataset/v4/pill_0.jpg to (640, 640)
Waiting 2 seconds for next capture...
Captured /home/yeewei831/dataset/v4/pill_1.jpg
Resized (padded): /home/yeewei831/dataset/v4/pill_1.jpg to (640, 640)
Waiting 2 seconds for next capture...
Captured /home/yeewei831/dataset/v4/pill_2.jpg
Resized (padded): /home/yeewei831/dataset/v4/pill_2.jpg to (640, 640)
Waiting 2 seconds for next capture...
Captured /home/yeewei831/dataset/v4/pill_3.jpg
Resized (padded): /home/yeewei831/dataset/v4/pill_3.jpg to (640, 640)
Waiting 2 seconds for next capture...
Captured /home/yeewei831/dataset/v4/pill_4.jpg
Resized (padded): /home/yeewei831/dataset/v4/pill_4.jpg to (640, 640)
[INFO] Image captured successfully!
yeewei831@raspberrypi:~ $

```

Figure 3.38: Output Messages When Capturing Images

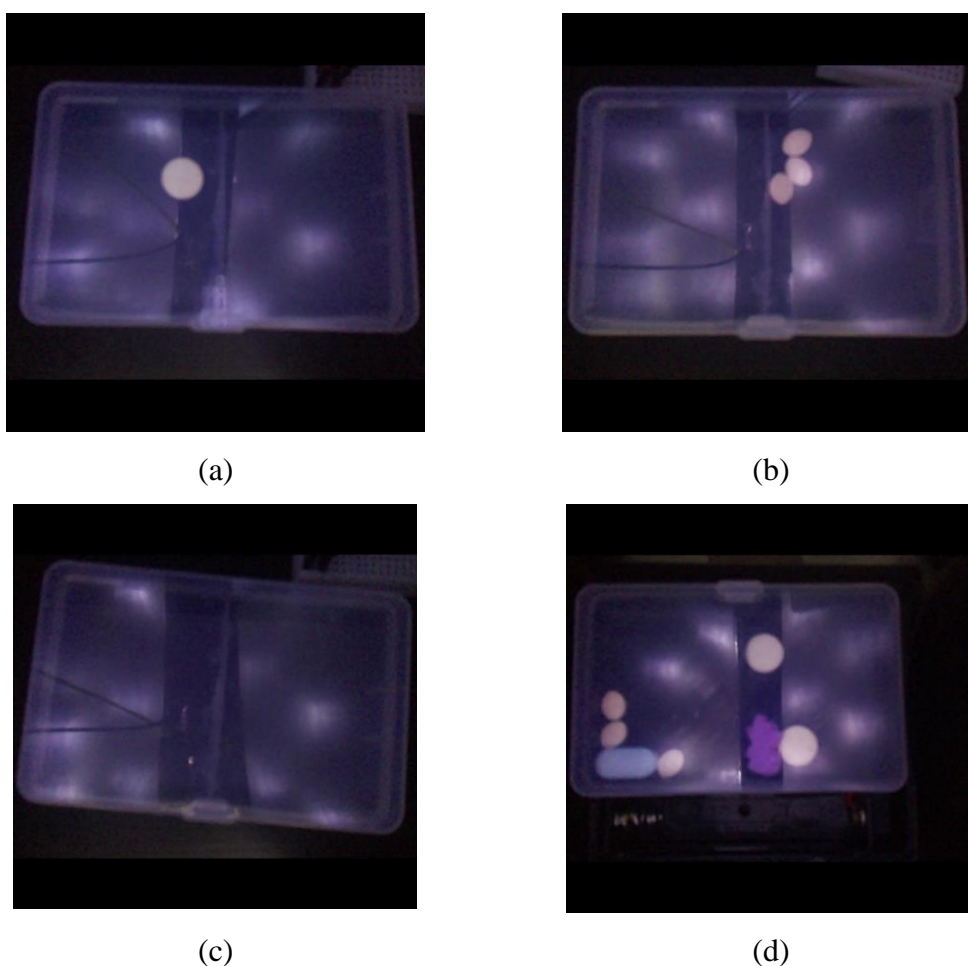


Figure 3.39: Images Captured by the Python Script (a) Amlodipine (b) Simvastatin (c) BoxPresent (d) Unknown

### 3.7.3.2 Annotation

After capturing the images, each image in the dataset is manually annotated by assigning bounding boxes and class labels to each object of interest. Then, the annotations are saved in YOLO format, where the text file contains the bounding box's position and the corresponding object classes. This step is crucial to ensure the model can differentiate between objects and background. The annotation tool used is LabelImg. Figure 3.40 illustrates the annotated images using LabelImg.

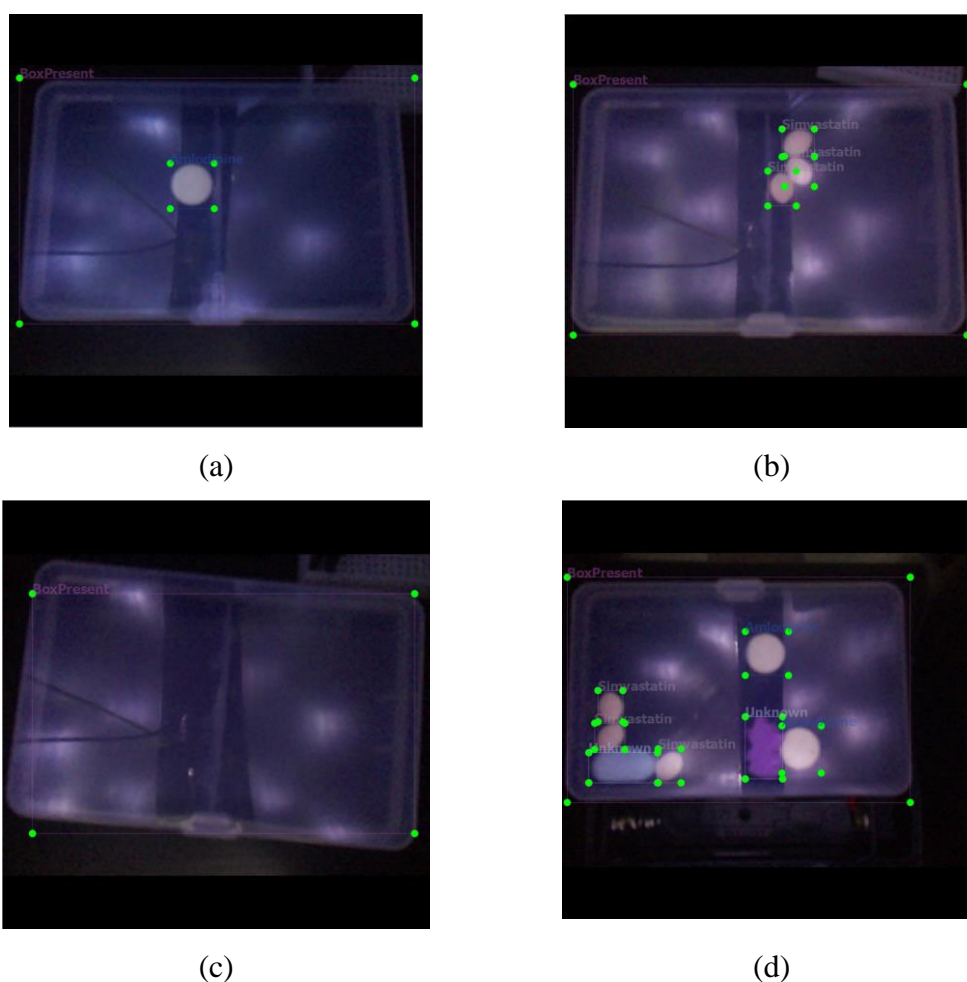


Figure 3.40: Annotated Images (a) Amlodipine (b) Simvastatin (c) BoxPresent (d) Unknown

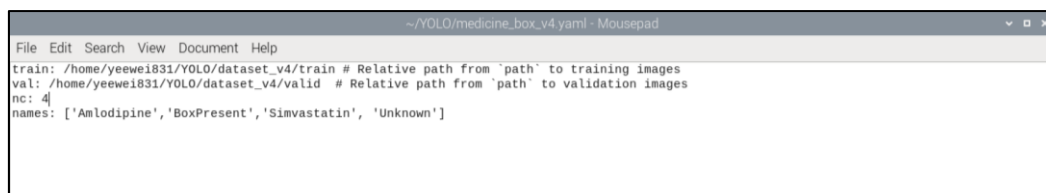
### 3.7.3.3 Partition

Partitioning in the training process involves the splitting of the dataset into 80% for training and 20% for validation. This approach ensures the model has adequate data for training and also data to evaluate its performance.

### 3.7.3.4 Model Training

After splitting the dataset, the object detection model is trained using the YOLOv8 framework on the Raspberry Pi. During this stage, a YAML file and a Python file are created. The path to the train and validation dataset, the number of the object classes, and the name of each class are specified in the `medicine_box_v4.yaml` file as shown in Figure 3.41.

To start the actual training, a Python file named `train.py` is created. In this Python file, the YOLO model is initialized, and some training parameters are also stated, such as the image size, batch size, and number of epochs. YOLOv8 Nano (YOLOv8n) is chosen for this project because of its high speed and lightweight architecture. The number of epochs is set to 20, with the batch size of 2 and image size of  $416 \times 416$ , so that it can accommodate the limited computational capabilities of the Raspberry Pi 4 Model B. To improve training results, the training parameters can be fine-tuned. During training, the terminal displays the progress of each epoch along with some metrics such as loss and accuracy, as illustrated in Figure 3.42. Based on the progress displayed in the terminal, the model's performance can be monitored as shown in Figure 3.43.



```
~/YOLO/medicine_box_v4.yaml - Mousepad
File Edit Search View Document Help
train: /home/yeewe1831/YOLO/dataset_v4/train # Relative path from 'path' to training images
val: /home/yeewe1831/YOLO/dataset_v4/valid # Relative path from 'path' to validation images
nc: 4
names: ['Amlodipine', 'BoxPresent', 'Simvastatin', 'Unknown']
```

Figure 3.41: `Medicine_box_v4.yaml` File Is Created

```

train.py x
1  from ultralytics import YOLO
2  model=YOLO("yolov8n.pt")
3
4
5  #model.predict(source=0, show=True)
6
7  model.train(
8      data ="/home/yeewei831/YOLO/medicine_box_v4.yaml",
9      epochs =20,
10     batch=2,
11     imgsiz=416,
12     workers=2,
13     project = "YOLO_results_v4",
14     name="pill_detector_v4"
15 )
16
17 print("Training completed!")

```

Figure 3.42: Train.py File Is Created

```

val: New cache created: /home/yeewei831/YOLO/dataset_v4/valid/labels.cache
Plotting labels to YOLO_results_v4/pill_detector_v4/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.00125, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 416 train, 416 val
Using 0 dataloader workers
Logging results to YOLO_results_v4/pill_detector_v4
Starting training for 20 epochs...

```

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	mAP50	mAP50-95
1/20	6G	0.8077	1.182	0.9658	20	416: 100% [1:15:23<00:00, 3.53s/it]	0.968	0.758
	Class	Images	Instances	Box(P	R			
	all	640	3369	0.93	0.985			
2/20	6G	0.7564	0.6348	0.9459	30	416: 100% [1:11:14<00:00, 3.34s/it]	0.988	0.777
	Class	Images	Instances	Box(P	R			
	all	640	3369	0.964	0.954			
3/20	6G	0.7596	0.5721	0.9479	3	416: 100% [1:11:12<00:00, 3.34s/it]	0.986	0.789
	Class	Images	Instances	Box(P	R			
	all	640	3369	0.964	0.956			
4/20	6G	0.7282	0.4864	0.9427	17	416: 100% [1:11:09<00:00, 3.34s/it]	0.995	0.888
	Class	Images	Instances	Box(P	R			
	all	640	3369	0.993	0.992			
5/20	6G	0.7137	0.465	0.9359	18	416: 100% [1:11:13<00:00, 3.34s/it]	0.995	0.817
	Class	Images	Instances	Box(P	R			
	all	640	3369	0.991	0.986			

Figure 3.43: Terminal Displays the Progress of Each Epoch



### 3.7.3.5 Model Evaluation

After the completion of training, there are two weight files generated, which are best.pt and last.pt, representing the best epoch and the latest epoch. To ensure optimal performance, the best.pt model is used for real-time pill detection. As shown in Figure 3.44, the terminal will display the message indicating the end of the training, along with the final loss and performance metrics. The detailed model evaluation is provided in Chapter 4.

```
Validating YOLO_results_v4/pill_detector_v42/weights/best.pt...
Ultralytics 8.3.84 Python-3.11.2 torch-2.6.0+cpu CPU (Cortex-A72)
Model summary (fused): 72 layers, 3,006,428 parameters, 0 gradients, 8.1 GFLOPs

```

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
all	640	3369	0.998	0.995	0.995	0.842
Amlodipine	371	1287	0.998	0.998	0.995	0.853
BoxPresent	597	587	0.997	1	0.995	0.973
Simvastatin	372	1377	0.999	0.991	0.995	0.766
Unknown	80	188	0.996	0.991	0.995	0.776

```

Speed: 3.0ms preprocess, 602.7ms inference, 0.0ms loss, 1.3ms postprocess per image
Results saved to YOLO_results_v4/pill_detector_v42
Training completed!
(MY_CV) yee@18318raspberrypi:~/YOLO $

```

Figure 3.44: Model Performance Shows in Terminal

## **CHAPTER 4**

### **RESULTS AND DISCUSSION**

#### **4.1 Mobile Application User Interface (UI)**

The flow of the mobile application UI is depicted in Figure 4.1. New users can create an account on the Register Page, while existing users can log in via the Login Page. The mobile application consists of two pages, which are the Reminder Alarm Page and the Pill Tracker Page. On the Reminder Alarm Page, users can add or delete reminders. When the scheduled time is reached, a notification will appear to remind the user to take their medication, thus improving their medication adherence. On the Pill Tracker page, it displays the current pill count for each medication, allowing users to track the remaining pill supply. When the pill count falls to 20% or below of the original quantity, an alert notification will appear to remind users to refill the medication.

On the Register Page, new users can create an account by filling in their first name and last name, email, and password as presented in Figure 4.2. The account will be created only if all the required fields are completed. For existing users, they can access the app by logging in with their registered email and password. As illustrated in Figure 4.3, the users must provide their email and password to ensure security.

The first page of the mobile application is the Reminder Alarm Page. If no alarms are scheduled, it will display the message “Nothing to show”, as displayed in Figure 4.4. As shown in Figure 4.5, when the user sets a reminder, it will be displayed on the page. Each reminder consists of details such as the quantity and type of pills to

be consumed at the scheduled time to avoid confusion. In addition, each reminder has an on/off switch that allows the user to enable or disable the alarm. There is a cross button located next to the on/off switch that can be used to delete the reminder if it's no longer needed. There is a logout button located at the top right corner that allows the user to log out of the application. The bottom navigation bar features two icons that represent the first and second pages of the mobile application. By tapping these icons, users can easily switch between pages. By clicking the second icon, users are directed to the second page, the Pill Tracker Page, as shown in Figure 4.6. This page displays the types of pills along with their respective remaining amounts, allowing users to monitor and track their current pill count effectively.

By clicking the round button in the bottom-right corner of the Reminder Alarm Page, users can add a new reminder easily. Figure 4.7 illustrates that the add reminder dialog will appear once users click the button. This dialog allows users to select the reminder time using the time picker and enter the pill name and the quantity to be consumed during the scheduled time. The time picker UI component is illustrated in Figure 4.8. The additional information provided by the user is essential for enhancing the clarity of the scheduled reminder. After the reminder is successfully added, a toast message will appear at the bottom of the screen, as shown in Figure 4.9.

On the other hand, users can remove the reminders that are no longer needed by pressing the cross button beside the on/off button. Figure 4.10 shows the delete reminder dialog, which serves as a confirmation prompt before deleting the reminder. The success of this action is confirmed with a toast message, indicating the reminder has been successfully deleted, as illustrated in Figure 4.11.

On the Pill Tracker Page, users are able to update the Pill Tracker by clicking the "Refill Pill" button and manually entering the refill dosages. Figure 4.12 presents the dialog box that appears where users can enter the pill name and dosage they are refilling. As shown in Figure 4.13, a snackbar message will show at the bottom of the screen, indicating the successful pill refill and the creation of a new pill log in Cloud Firestore.

When the scheduled time arrives, a reminder notification will pop up on the screen to remind users to take their medication following the prescribed schedule, as shown in Figure 4.14. The notification displays the pill name and dosage to avoid any errors in medication intake. On top of that, the notification includes a “Mark as Taken” button at the bottom. Pressing this button will automatically update the pill count in the Pill Tracker by deducting the quantity stated in the reminder. This feature can ensure that the Pill Tracker always shows the current pill supply, helping users manage their medication more efficiently. Additionally, Figure 4.15 shows that the system features a refill alert notification. It will activate when the pill quantity falls to 20% or below of the original amount. In this case, users can be notified when it is time to refill their prescriptions, thereby reducing the risk of missed doses. This function is vital for individuals with chronic conditions, ensuring they always have an adequate supply in their pill box.

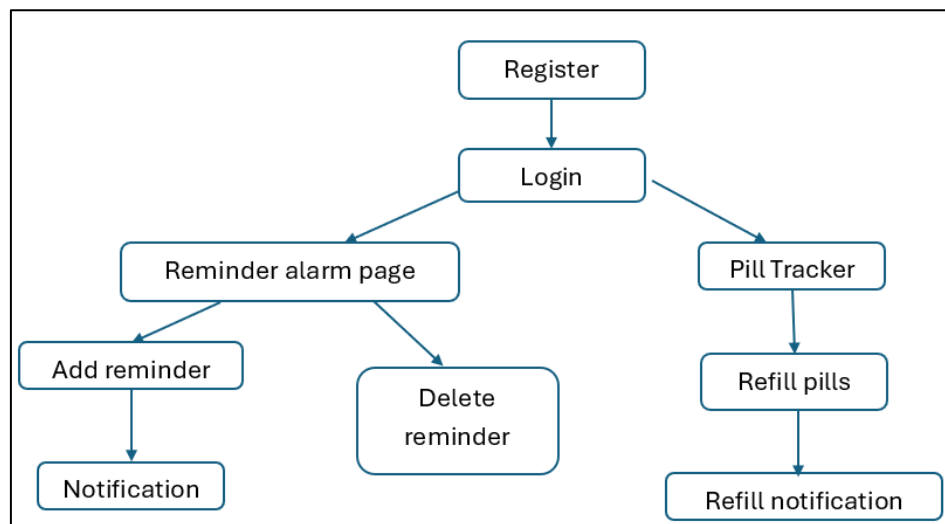
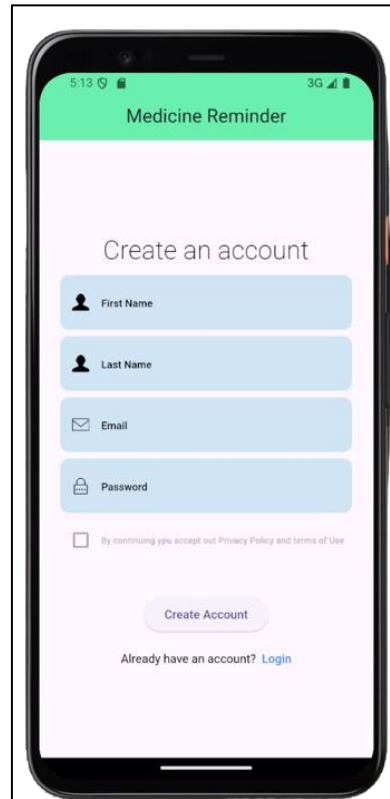
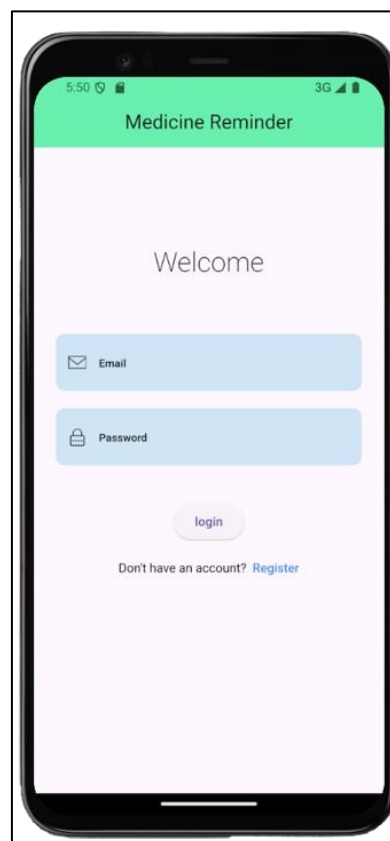


Figure 4.1: Flow of the Mobile Application UI



The image shows a mobile application interface for a "Medicine Reminder" app. The screen has a green header bar with the text "Medicine Reminder". Below the header, the text "Create an account" is displayed. There are four input fields: "First Name", "Last Name", "Email", and "Password", each with a corresponding icon (person, person, envelope, and lock respectively). Below these fields is a checkbox with the text "By continuing you accept our Privacy Policy and terms of Use". At the bottom, there is a "Create Account" button and a link "Already have an account? Login".

Figure 4.2: Register Page



The image shows a mobile application interface for a "Medicine Reminder" app. The screen has a green header bar with the text "Medicine Reminder". Below the header, the text "Welcome" is displayed. There are two input fields: "Email" and "Password", each with a corresponding icon (envelope and lock respectively). Below these fields is a "login" button. At the bottom, there is a link "Don't have an account? Register".

Figure 4.3: Login Page



Figure 4.4: Reminder Alarm Page When No Reminder Is Set

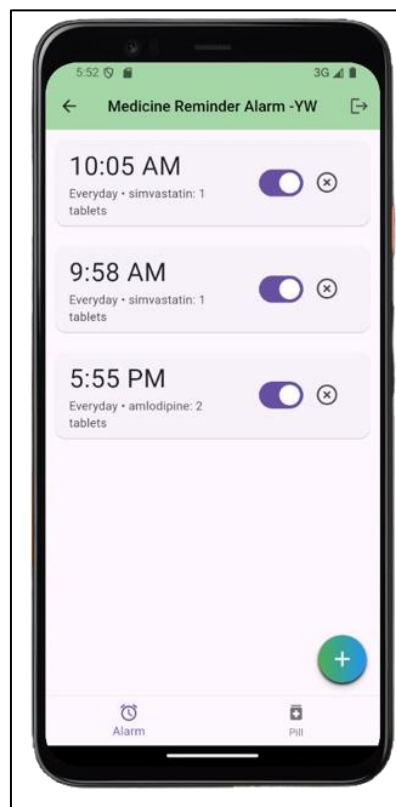


Figure 4.5: Reminder Alarm Page with Set Reminders

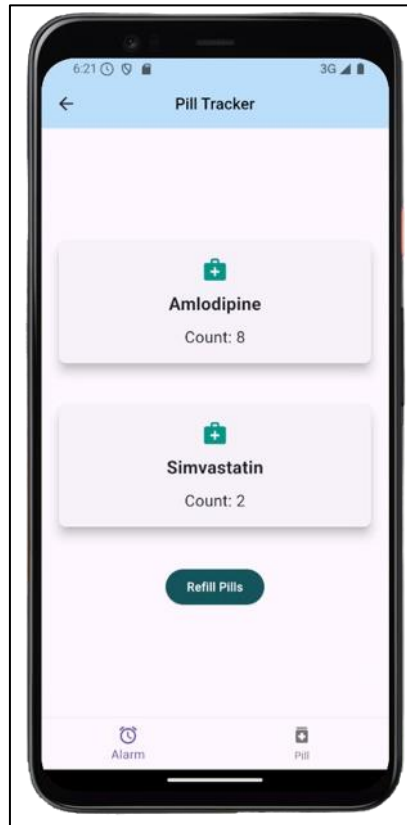


Figure 4.6: Pill Tracker Page

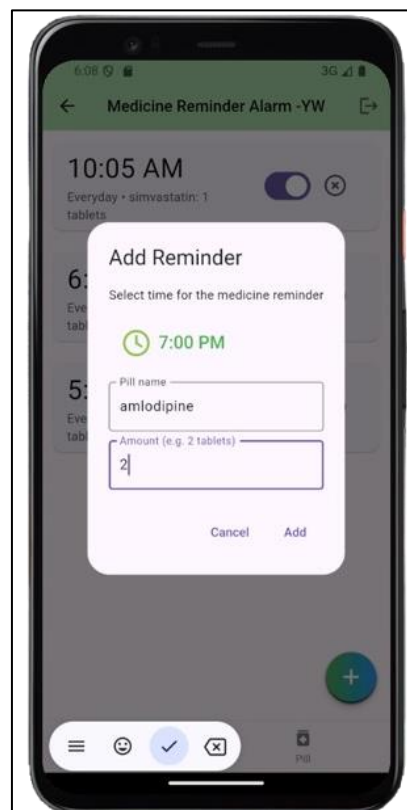


Figure 4.7: Add Reminder Dialog



Figure 4.8: Time Picker

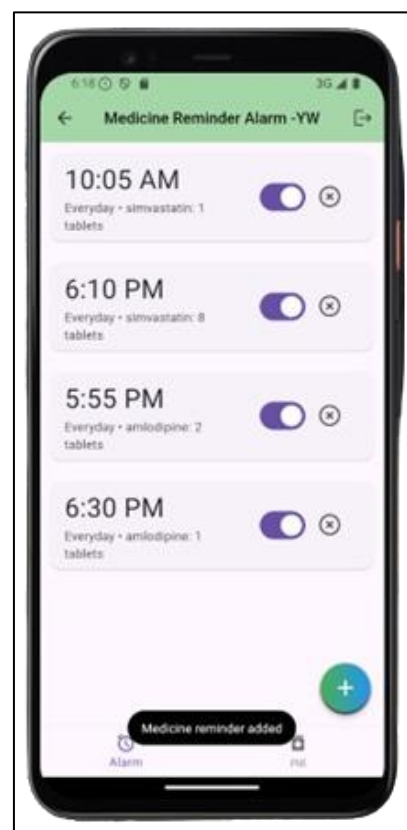


Figure 4.9: Reminder Successfully Added



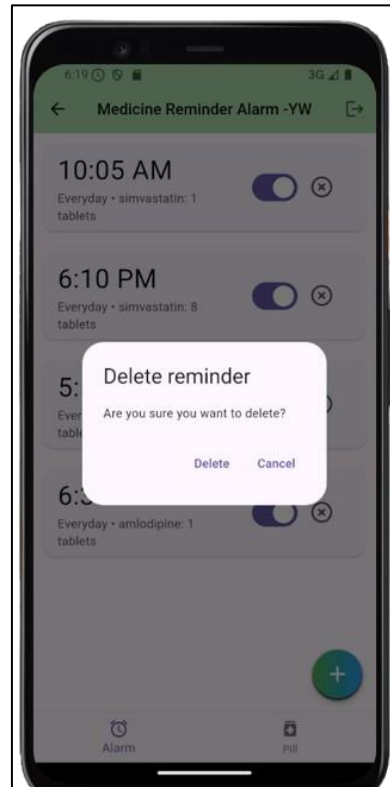


Figure 4.10: Delete Reminder Dialog

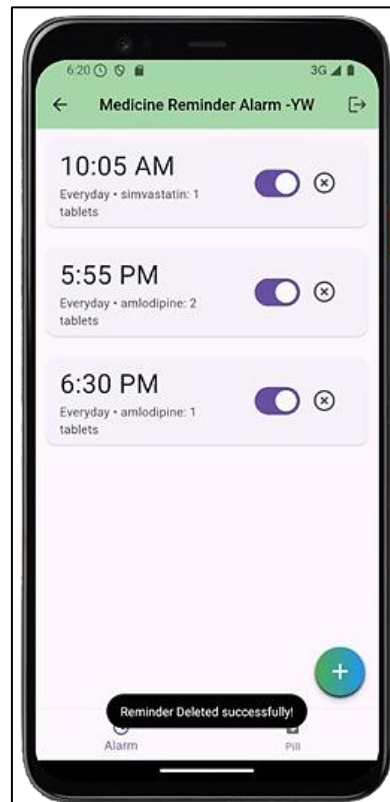


Figure 4.11: Reminder Successfully Deleted

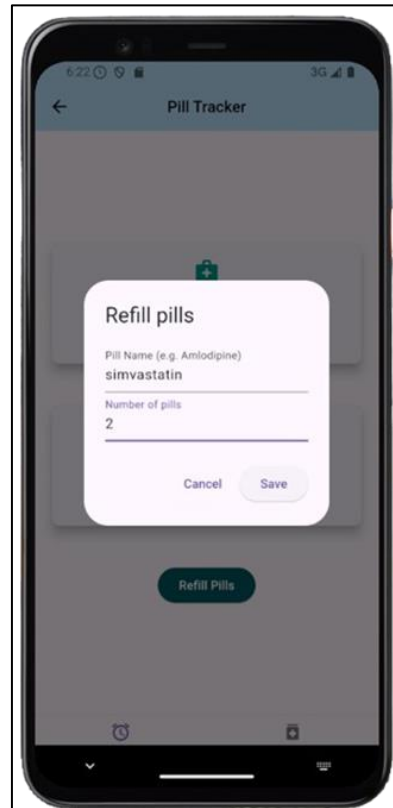


Figure 4.12: Refill Pill Dialog

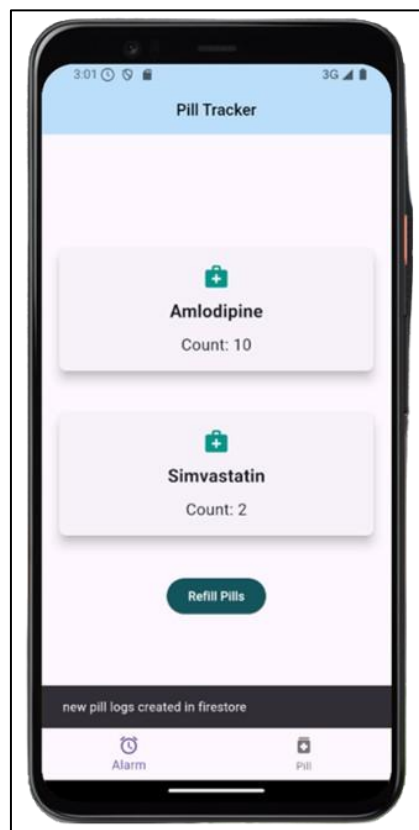


Figure 4.13: Successful Pill Refill



Figure 4.14: Reminder Notification

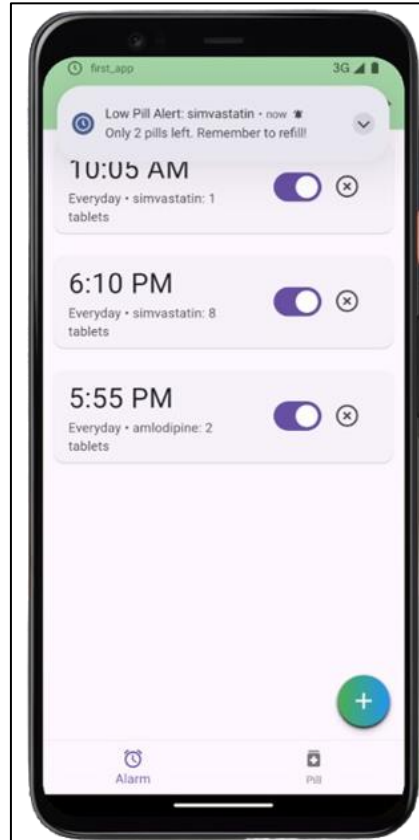


Figure 4.15: Refill Alert Notification

## 4.2 Prototype

Figure 4.16 illustrates the prototype for the Intelligent Medicine Box System, which consists of a portable Pill Box, a Home Base, and its drawer. The compartment in the Pill Box, the Home Base, and its drawer were created using 3D printing technology. The prototype is constructed using this approach because 3D printing is cost-effective, able to provide robust structures, and more environmentally friendly compared to other materials. After printing, the components are sanded and spray-painted to improve their surface quality. Figure 4.17 shows that the Intelligent Medicine Box System detected the presence of the Pill Box. The Pill Box is placed on the drawer and then slid into the Home Base. Inside the Home Base, the Raspberry Pi camera is positioned facing inwards. This setup allows the camera to capture images of the Pill Box and then perform pill detection.

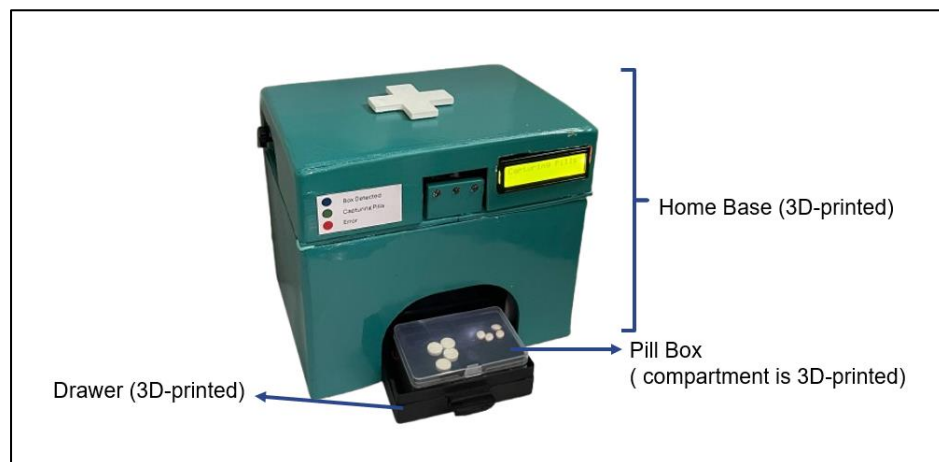


Figure 4.16: Prototype of Intelligent Medicine Box System



Figure 4.17: Pill Box Detected by the System

### 4.2.1 Portable Pill Box

Figure 4.18 presents the front view of the Pill Box. It is a transparent container with a status label attached, so that users can understand the system's current state based on the RGB LED indicators. Different colours shown by the RGB LED represent different statuses. Figure 4.19 illustrates the side views of the Pill Box, showing that the upper section of the Pill Box serves as the compartment for pill storage, while the lower section houses the circuit for the ESP8266 microcontroller that connects to a buzzer and an RGB LED. As observed from the top view of the Pill Box, the pill compartment is divided into two sections, allowing two different types of pills to be stored separately, as shown in Figure 4.20. Besides that, a rechargeable battery with a battery holder is attached to the back of the Pill Box, which supplies power to the ESP8266 microcontroller.



Figure 4.18: Front View of the Pill Box



(a)



(b)

Figure 4.19: Side Perspectives of the Pill Box (a) Right Side View (b) Left Side View



Figure 4.20: Top View of the Pill Box

#### 4.2.2 Home Base

The front view, side view, and top view of the Home Base are shown in Figure 4.21, Figure 4.22, and Figure 4.23, respectively. The Home Base is sprayed with green colour because it is associated with healthcare, providing a calming and cooling effect. In the upper section of the Home Base, three LEDs are positioned at the centre to represent the specific state of the system. The status label is attached next to these LEDs to provide visual information for the user. An LCD is placed on the right-hand side to display the output messages when the program is running on the Raspberry Pi. Figure 4.24 shows the drawer designed for placing the Pill Box.



Figure 4.21: Front View of Home Base



Figure 4.22: Side Perspectives of Home Base (a) Right Side View (b) Left Side View



Figure 4.23: Top View of the Home Base



Figure 4.24: Drawer in Home Base

## 4.3 Pill Box

### 4.3.1 Program of NodeMCU ESP8266 V2

As the ESP8266 does not support the Firebase SDK, it accesses the data from Cloud Firestore via the REST API. REST API refers to Representational State Transfer Application Programming Interface, which allows devices to communicate with the server using Hypertext Transfer Protocol (HTTP) requests, such as GET, POST, and DELETE. Figure 4.25 shows that the Firebase credentials and Firestore API URL are included in the code to allow communication between ESP8266 and Cloud Firestore. In Arduino IDE, the code is written to allow ESP8266 to fetch the reminder data from Cloud Firestore via REST API, parse the payload, and then trigger the buzzer when the reminder time is reached.

The program's operational flow is depicted in Figure 4.26. In the `setup()` function, the program starts by initializing the pins and serial communication. For serial data transmission between ESP8266 and the computer, it is initialized at a baud rate of 9600 bits per second. Next, the ESP8266 starts to connect with Wi-Fi, and the RGB LED will produce white light once it is successfully connected. To get the current time in UTC, the ESP8266 syncs with the Network Time Protocol (NTP) server, a time synchronization protocol. During the process, a red light will turn on to indicate system errors.

With Wi-Fi connected and having an accurate time, the system will enter the loop function that will run continuously. Inside the loop function, the ESP8266 will retrieve the latest reminder documents from the Cloud Firestore. To fetch reminders, the ESP8266 sends the HTTP GET request to Cloud Firestore. If a valid HTTP response code is received (`httpResponseCode` is 200), the RGB LED will show blue light. Once the ESP8266 successfully received the payload (scheduled reminder time) from Cloud Firestore, it is indicated by a green light. On the other hand, failure is indicated by a red light. After that, the payload, which is in JavaScript Object Notation (JSON) format, will be parsed by extracting timestamps and the on/off status. It is proceeding to convert the date and time into Unix timestamps and store them in a reminder array. The maximum number of reminders is 10 in this program.



After that, the ESP8266 will get the current UTC, which is returned as a Unix timestamp. Next, the current time is compared with the scheduled reminder times to determine the triggering of the buzzer. The buzzer will only be triggered when: it is active (onOff is true), the reminder is not yet triggered, current time is within the range of  $\pm 180$  seconds (3 minutes) of the reminder time. By considering the delays from NTP synchronization or Wi-Fi connectivity, the buzzer will be triggered within the 3-minute window around the exact reminder time. In this case, the buzzer will only be triggered once and preventing missing reminders. Figure 4.27 shows the conditions that need to be met to trigger the buzzer.

```
// Firebase project credentials
const String FIREBASE_HOST ="http://medicine-reminder-92853.firebaseio.com";
const String FIREBASE_API_KEY ="AIzaSyDgvm-d-bI_QsNiK0zmArLWX8WxRoeZwxs";
const String FIREBASE_PROJECT_ID ="medicine-reminder-92853";
const String USER_ID ="vfUin8HcMlba6MefkhxpBBk5xxT2";

// Firestore API URL
String firestoreURL = "https://firestore.googleapis.com/v1/projects/" + FIREBASE_PROJECT_ID +
    "/databases/(default)/documents/users/" + USER_ID + "/reminder?key=" + FIREBASE_API_KEY;
```

Figure 4.25: Firebase Credentials and Firestore API URL Included in the Code

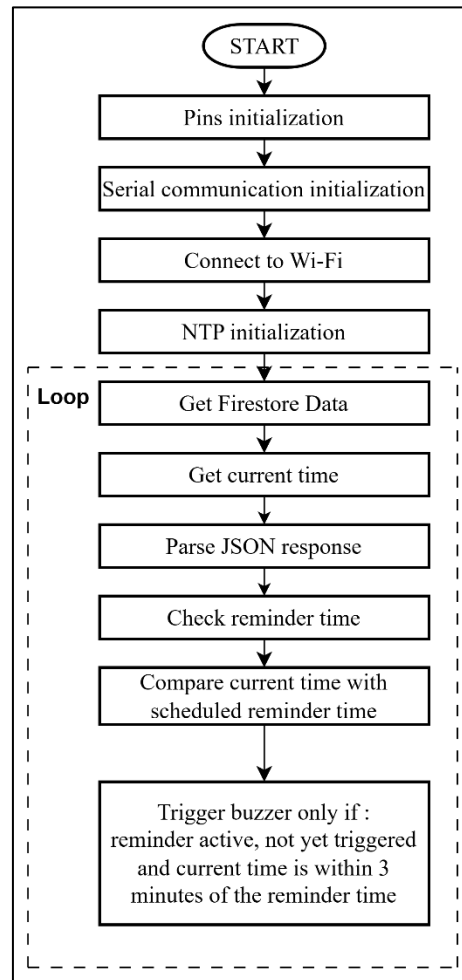


Figure 4.26: Program Flow in ESP8266

```

if (!reminders[i].triggered &&
    (timeDiff >= -180) &&
    (timeDiff <= 180) &&
    reminders[i].onOff) {

    Serial.print("Reminder Time(UTC):");
    Serial.println(reminders[i].reminderTime);
    Serial.println(reminders[i].onOff? "true" : "false");
    Serial.println("Time reached, buzzer triggered!");
    triggerBuzzer();
    reminders[i].triggered = true;
} else {
    Serial.println("Buzzer does not trigger");
}
}
}

```

Figure 4.27: Conditions to Trigger Buzzer

### 4.3.2 Hardware Connection in Pill Box

Figure 4.28 illustrates the connection between the components used to build the circuit inside the Pill Box, including NodeMCU ESP8266 V2, an LC18650 rechargeable battery with holder, an MT3608 step-up power module, an RGB LED, resistors, and a buzzer. For the step-up power module, it is used to boost the voltage from 3.7V (from the lithium-ion battery) to 5V (as the input voltage for ESP8266). All the components are connected on a stripboard as shown in Figure 4.29.

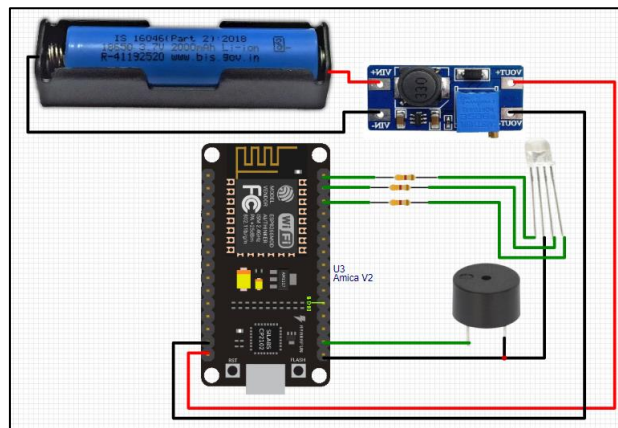


Figure 4.28: Hardware Connection for Circuit Inside Pill Box

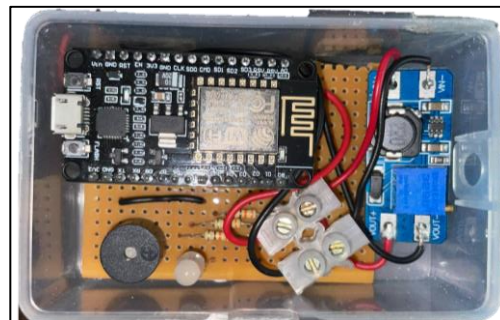
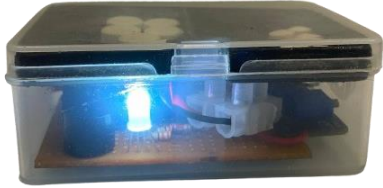

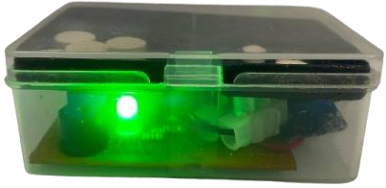
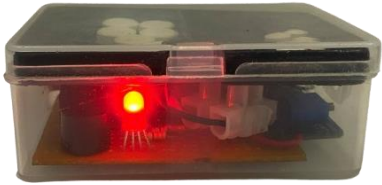



Figure 4.29: Circuit Connection Inside Pill Box

### 4.3.3 Light Indications and Sound Alert

The portable Pill Box features an RGB light that will change based on the task being performed. Table 4.1 illustrates the four types of light indications on the Pill Box when performing different tasks, such as connected to Wi-Fi, data retrieval, successful retrieval, and error occurred. On top of that, there is a sound alert from the buzzer inside the Pill Box to improve the medication adherence of the user.

Table 4.1: Light Indications and Sound Alert on Pill Box

Pill Box	Explanations
	White light from the RGB LED indicates that Wi-Fi is connected.
	Blue light represents that the HTTP response code received is correct and data is being retrieved from Cloud Firestore.
	Green light indicates that the data has been successfully retrieved from the Cloud Firestore.
	Red light indicates a system error.
	The buzzer inside the Pill Box will sound when the scheduled time is reached, reminding the user to take their medication on time.

## 4.4 Home Base

### 4.4.1 YOLOv8 Model Evaluation

After training the YOLOv8 model, the performance of the model can be evaluated based on several metrics, including Precision (P), Recall (R), and Mean Average Precision (mAP). Precision refers to the model's accuracy in correctly detecting objects, while recall measures how many objects are detected in the image by the model (Torres, 2024). For the mAP, it evaluates the accuracy of a model by combining precision and recall. From these metrics, the effectiveness of the model can be easily evaluated. After training the YOLOv8 model, several performance plots such as the normalized confusion matrix, the precision-recall curve, and the F1-confidence curve are automatically generated to visualize how well the model is performing.

Figure 4.30 shows the normalized confusion matrix obtained after training the YOLOv8 model. The normalized confusion matrix illustrates the classification performance for each of the classes. From the normalized confusion matrix, an overview of the comparison between true labels and predicted labels can be observed. Since this is a normalized confusion matrix, the values are represented as percentages, which makes the performance for each class can be compare easily. It can be seen that the four main classes (Amlodipine, Simvastatin, BoxPresent, and Unknown) are achieving 100% accuracy in classification along their respective rows. In other words, four of the classes are correctly predicted without any misclassifications. However, there are misclassifications in the background class. From the plot obtained, the background is wrongly predicted as Amlodipine (12%), BoxPresent (10%), Simvastatin (75%), and Unknown (2%), leading to false positives in the detection. This highlights that the background might struggle to distinguish the background and pills, especially simvastatin. This might be due to the small size of simvastatin, which is visually similar to the LED bulb. In addition, the reflective surface of the transparent pill box lid might introduce the light reflections of the LED that looks like a pill.

The precision-recall curve illustrated in Figure 4.31 highlights the trade-off between precision and recall at various thresholds. The blue line on the precision-recall

curve shows the average precision-recall performance. It remains high precision and recall across different confidence thresholds. From the plot, it indicates that the model is performing well with the mean Average Precision (mAP) of 0.995 at the IoU threshold of 0.5. The Intersection over Union (IoU) threshold of 0.5 means the predicted bounding box is counted as correct if it overlaps the ground truth by at least 50%. Overall, the YOLOv8 model is achieving high performance by showing strong classification accuracy with minimal false positives.

A confidence curve can also be used to evaluate the overall performance of a model. Basically, the F1 score represents the harmonic mean of precision and recall. Based on the confidence curve as shown in Figure 4.32, the F1 curve is the representation of the F1 score across different confidence thresholds. At the confidence threshold of 0.681, the F1 score gets the highest peak of 1.00, highlighting that the model has the optimal performance at this point. This means that this is the point where precision and recall are best balanced. At this confidence threshold, all predicted objects are correct, and all instances are detected. It can be observed that the curve drops after 0.8, indicating that a high confidence threshold will cause miss detections. Based on the F1 curves, the best confidence threshold is 0.681, which can be used for deployment later.

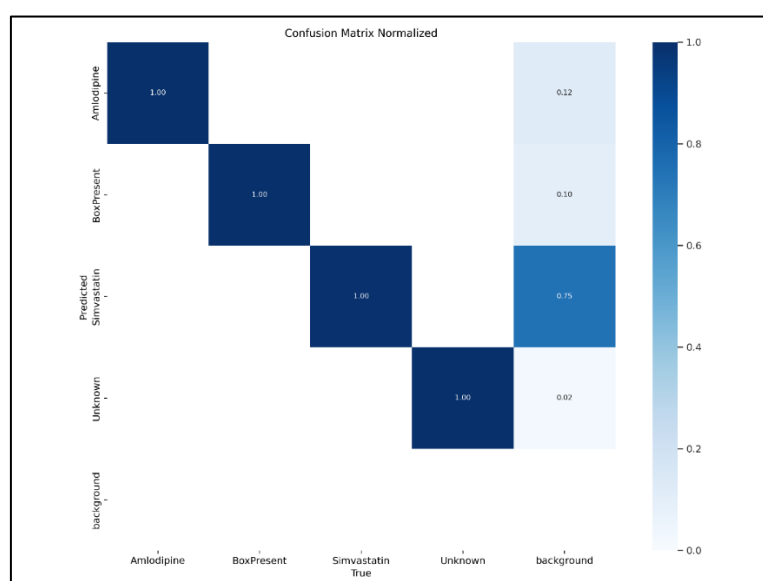


Figure 4.30: Normalized Confusion Matrix

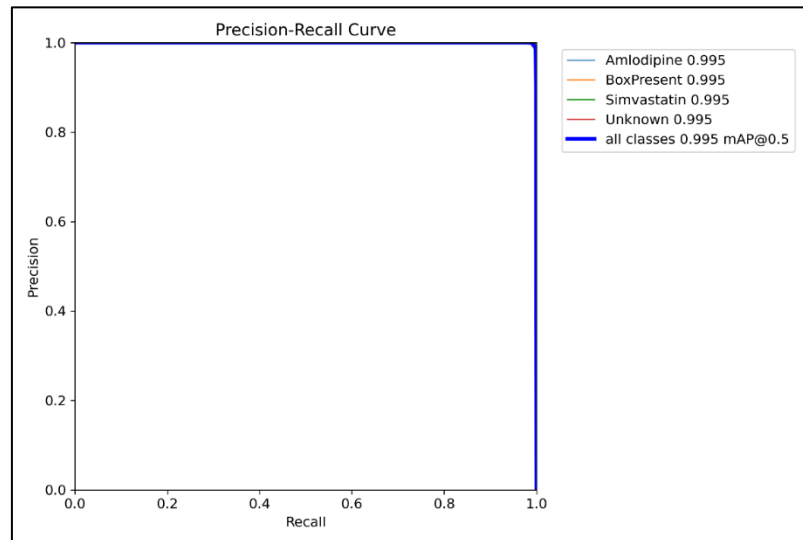


Figure 4.31: Precision-Recall Curve

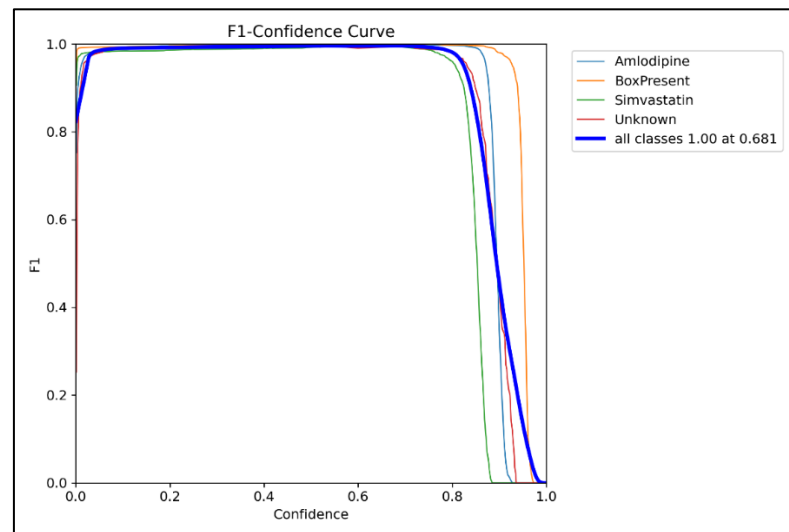


Figure 4.32: F1- Confidence Curve

#### 4.4.2 Operation of the Home Base System

After training the YOLOv8 model, it is deployed on the Raspberry Pi for real-time pill detection. A Python program is written to load the trained model and perform pill detection. By referring to the flowchart in Figure 4.33, the program begins with the initialization of GPIO, LCD, camera, Firebase, and the YOLOv8 model. After that, the YOLOv8 object detection model is loaded so that the program is able to perform object

detection. Once the initialization is complete, the program will enter the main loop, where the blinking green LED indicates that the system is capturing images and capturing pills. The image is captured using the Raspberry Pi Camera Module V2 and then passed into the YOLOv8 model to perform object detection. Based on the model's detection output, the system counts the number of Amlodipine pills, Simvastatin pills, and Unknown objects.

The program verifies the presence of the Pill Box by checking if the YOLO model has detected the BoxPresent object. If no box is detected, the LCD shows the messages "No box detected" and "skipping counting" and turns off the green LED. Then, the program loops back to the image capturing process. If the BoxPresent class is detected by the model, the LCD displays "Box detected" message and the blue LED is activated, allowing the counting results to be displayed on the LCD.

If there is an unknown object is detected, the red LED turns on and the LCD displays an alert message "E: unknown obj!" This alert message and red LED can alert the user to check their Pill Box when the model detects unknown objects in the Pill Box. After that, the system compares the detected pill count (number of Amlodipine and Simvastatin) with the latest pill log that is stored in the Cloud Firestore. If the detected pill count matches the stored data, the LCD displays "Pill Match!" with no alert triggered. Otherwise, it will show "E: Pill Mismatch!" and trigger the red LED. The purpose of comparing the pill count is to determine if a discrepancy exists, which might indicate problems like incorrect dosage or missed doses. When the system detects a pill count that differs from the one stored in the latest pill log, an alert message "E: Pill mismatch!" will be displayed to alert the user, so that they can take appropriate corrective actions. After the comparison, the program will turn off the blue and green LEDs and continue looping.



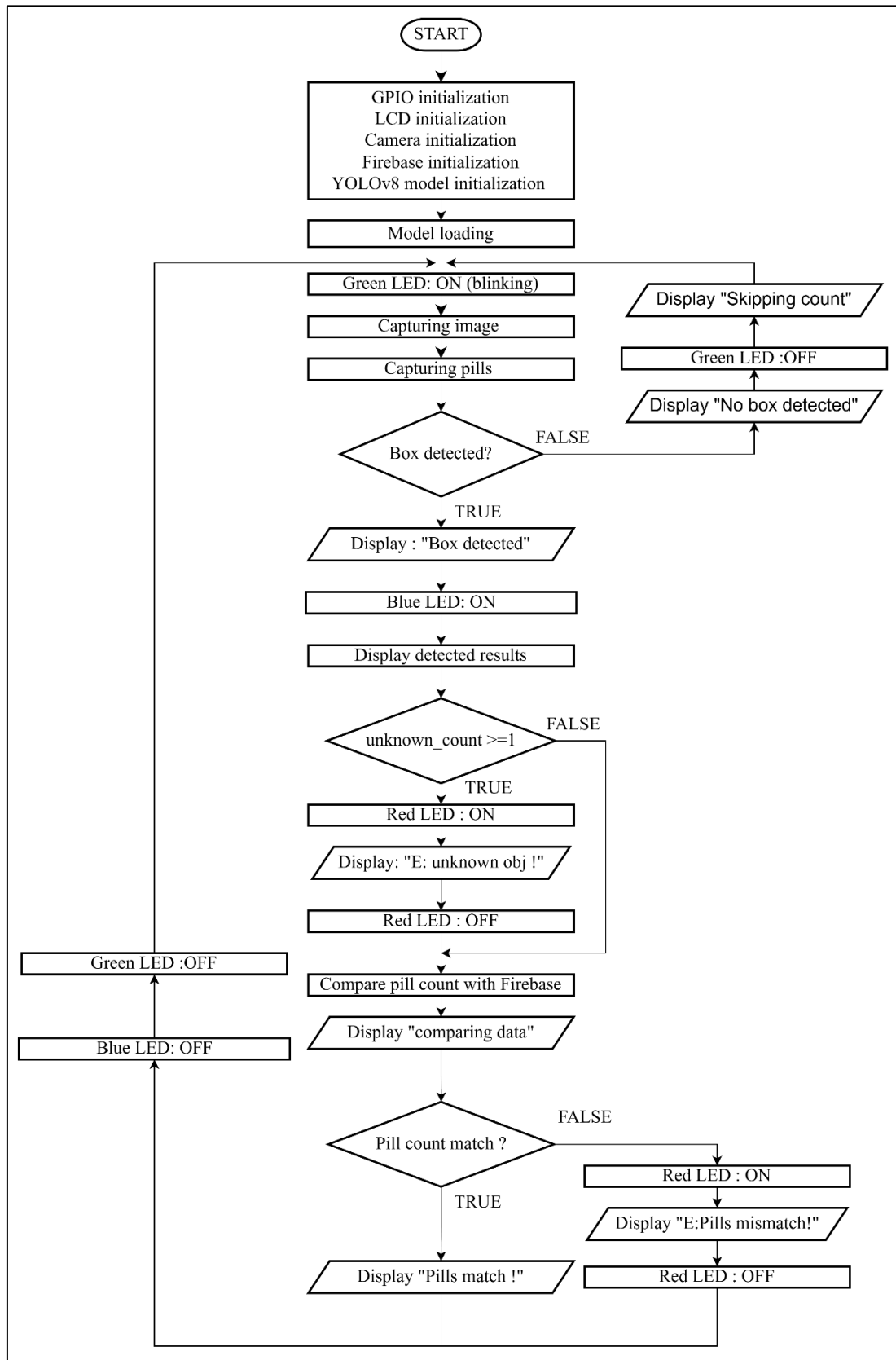


Figure 4.33: Flowchart of the Home Base System

#### 4.4.3 Results of the Deployment of YOLOv8 Model on Raspberry Pi

In the program, the image captured will be passed into the YOLO model to perform object detection. When running the program on the Raspberry Pi, a real-time detection window will pop out to allow the developer to visualize the model's predictions. It shows how the model identifies and detects the object in real time.

Based on the detection window shown in Figure 4.34 and Figure 4.35, each image displays the predicted bounding boxes along with the confidence score and their class labels (Amlodipine, Simvastatin, BoxPresent, or Unknown). Each detected object is surrounded by a coloured bounding box along with its class label and confidence score. The confidence score reflects how confident the model is of this prediction.

Figure 4.35 displays the prediction output when the model detects unknown objects inside the Pill Box. In this project, unknown objects include the unrecognized pill and small non-pill item intentionally placed in the compartment of the Pill Box. This demonstrates the additional function of the system to detect any foreign objects and then alert users to these issues. When the program is executing, the Raspberry Pi's terminal will display the output messages. Figure 4.36 illustrates the output messages from the terminal when no box is detected in Home Base. Figure 4.37, Figure 4.38, and Figure 4.39 show the output messages when the box is detected, and the program will proceed to verify the presence of any unknown objects and compare the pill count with the latest pill log stored in Cloud Firestore.

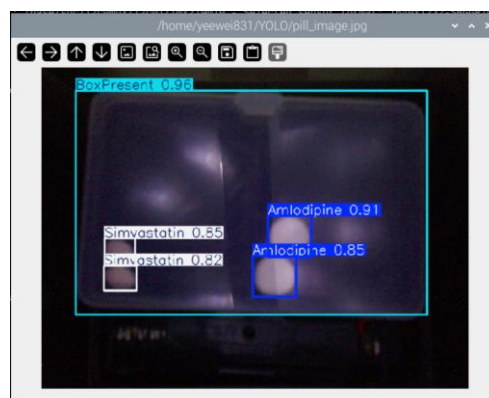


Figure 4.34: Result from Real-Time Detection Window

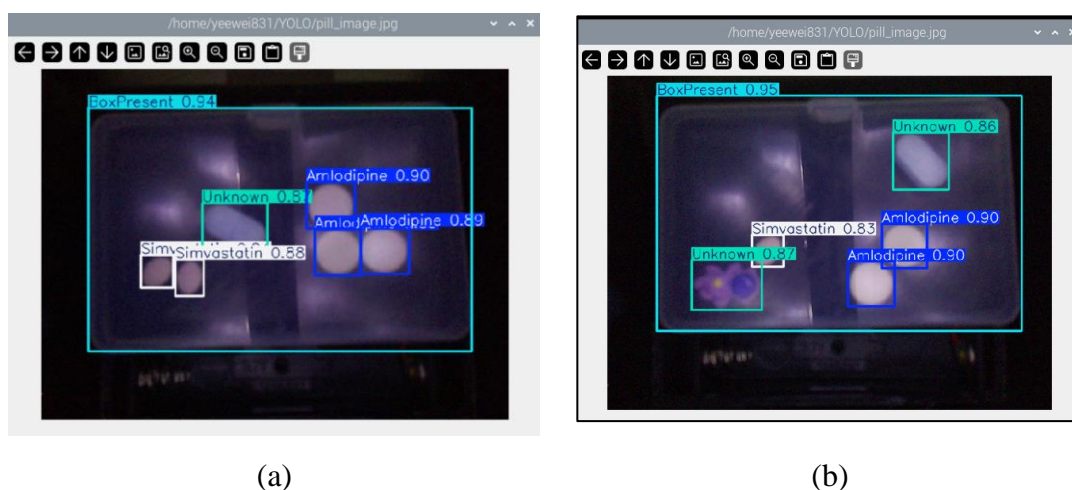


Figure 4.35: Unknown Objects Detected (a) One Unknown Object Detected (b) Two Unknown Objects Detected

```
YOLO detection took 1.14 seconds
{0: 'Amlodipine', 1: 'BoxPresent', 2: 'Simvastatin', 3: 'Unknown'}
Box present: NO
skipping count
(MY_CV) yeewei831@raspberrypi:~/YOLO $
```

Figure 4.36: Output Messages When No Box Detected

```
YOLO detection took 1.07 seconds
{0: 'Amlodipine', 1: 'BoxPresent', 2: 'Simvastatin', 3: 'Unknown'}
Box Present: Yes
Amlodipine Count: 2
Simvastatin Count: 2
Unknown Count: 0
box detected and data are compared
Comparing data with data stored in firestore
[FIRESTORE]Amlodipine:2, Simvastatin:2
[LOCAL] Amlodipine:2, SIMvastatin:2
no alert, pill count match
```

Figure 4.37: Output Messages When Pill Match

```
YOLO detection took 3.38 seconds
{0: 'Amlodipine', 1: 'BoxPresent', 2: 'Simvastatin', 3: 'Unknown'}
Box Present: Yes
Amlodipine Count: 3
Simvastatin Count: 2
Unknown Count: 0
box detected and data are compared
Comparing data with data stored in firestore
[FIRESTORE]Amlodipine:2, Simvastatin:2
[LOCAL] Amlodipine:3, SIMvastatin:2
E:pills mismatch!
```

Figure 4.38: Output Messages When Pill Mismatch

```

YOLO detection took 1.34 seconds
{0: 'Amlodipine', 1: 'BoxPresent', 2: 'Simvastatin', 3: 'Unknown'}
Box Present: Yes
Amlodipine Count: 3
Simvastatin Count: 2
Unknown Count: 1
E: Unknown object detected!
box detected and data are compared
Comparing data with data stored in firestore
[FIRESTORE]Amlodipine:2, Simvastatin:2
[LOCAL] Amlodipine:3, SIMvastatin:2
E:pills mismatch!


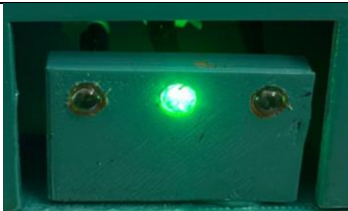






```











Figure 4.39: Output Messages When Unknown Object Detected

#### 4.4.4 LED and LCD Indications

The Home Base prototype is equipped with LEDs and an LCD to provide real-time feedback on the task that is being executed. The LCD shows the detection results and messages for user monitoring, while the LEDs indicate various statuses: a green light indicates that the program is capturing images, a blue LED lights up when the Pill Box is detected, and a red LED is activated when an error occurs. There are two specific conditions that trigger There are two scenarios that trigger the red LED: when the model detects an unknown object or when there is a pill count mismatch, alerting the user to these issues that require attention. Table 4.2 presents all the LED and LCD indicators along with explanations.

Table 4.2: LED and LCD Indications on Home Base

LCD	LED light indication	Explanations
	 <b>Green light (blinking):</b> The pill capturing process is in progress	The system starts with a capturing process to detect the presence of the Pill Box.
<b>Condition 1: No box detected</b>		
	 No LED lights up	No box is detected inside Home Base.
	 No LED lights up	The system does not display the counting results.
<b>Condition 2: Pill Box detected</b>		
	 <b>Blue light:</b> Pill Box detected	Pill Box detected inside Home Base.

	 <p><b>Blue light:</b> Pill Box detected</p>	<p>The counting results are displayed on LCD:</p> <p><b>A</b> – Amlodipine <b>S</b> – Simvastatin <b>U</b> – Unknown</p>
	 <p><b>Blue light:</b> Pill Box detected</p>	<p>The system compares the pill count with the latest pill log stored in Cloud Firestore.</p>
	 <p><b>Blue light:</b> Pill Box presence detected</p>	<p>Pill count matched.</p>
	 <p><b>Red light:</b> Error encountered</p>	<p>Pill count mismatch!</p>
	 <p><b>Red light:</b> Error encountered</p>	<p>Unknown object detected.</p>

## **CHAPTER 5**

### **CONCLUSION AND RECOMMENDATIONS**

#### **5.1 Conclusion**

In conclusion, this project has achieved all the objectives and successfully developed a functional Intelligent Medicine Box System with AI-Powered Pill Detection and IoT Integration. With the user-friendly mobile application, users are able to adhere to their medication regimens and have better medication management through the timely reminders and notifications. Additionally, the pill tracker feature in the mobile application allows user to monitor their current pill quantity and send an alert when a refill is needed. This approach ensures user take their medicine consistently and makes sure an adequate supply of pills is always available. Besides that, the integration of IoT technology into the portable Pill Box can ensure the user receives timely reminders anytime and anywhere. The AI-powered pill detection and classification features in the Intelligent Medicine Box System are achieved through the deployment of the YOLOv8 model on the Raspberry Pi. The system is able to automate the process of detection and counting and subsequently compare these pill counts with the pill log stored in Cloud Firestore to verify that the user is adhering to the treatment plan. Besides that, the system is capable of detecting foreign objects in the Pill Box to avoid contamination or incorrect medication intake. The concept of the system is not only applicable in healthcare but also adaptable to other fields that require inventory management, such as logistics and warehousing.

## **5.2 Future Improvements and Recommendations**

For future improvements, several enhancements can be made to improve the overall functionality of the system. Firstly, a more lightweight pill box with multiple compartments could be designed to increase portability and allow users to store multiple types of medication separately. For the mobile application, it can be designed to display history logs, which allow the user to review when the dose was taken or missed. By having this feature, the user will be more likely to stay consistent with their medication schedule, thus improving their medication adherence. Furthermore, the Pill Box and Home Base can incorporate voice commands to enhance accessibility, especially for users with disabilities. Besides that, the accuracy of the AI model can be continued to improve in order to enhance the detection performance. Additionally, the AI model could be further improved to recognize newly introduced pills automatically by integrating the system with the healthcare database. For instance, integrating the system with the Ministry of Health Malaysia (MOH) to get the latest and updated medication information so that the system can recognize the new pill automatically.



## REFERENCES

Abba, I. (2021) *Firestore – Database Crash course*, *freeCodeCamp.org*. Available at: <https://www.freecodecamp.org/news/firestore-crash-course/> (Accessed: 10 May 2025).

Agarwal, T. (2021) *Buzzer: Working, types, circuit, Advantages & Disadvantages*, *ElProCus*. Available at: <https://www.elprocus.com/buzzer-working-applications/> (Accessed: 01 May 2025).

Azlan, M.A.I. and Yahya, R. (2023) ‘Smart medicine pill box reminder’, *Evolution in Electrical and Electronic Engineering*. Available at: <https://publisher.uthm.edu.my/periodicals/index.php/eeee/article/view/10766> (Accessed: 15 July 2024).

Brown, M.T. and Bussell, J.K. (2011) ‘Medication adherence: Who cares?’, *Mayo Clinic Proceedings*, 86(4), pp. 304–314. doi:10.4065/mcp.2010.0575.

Buhl, N. (2024) *Yolo Object Detection explained: Evolution, algorithm, and applications*, *Encord*. Available at: <https://encord.com/blog/yolo-object-detection-guide/> (Accessed: 11 May 2025).

Cd-Team (2023) *LCD 16x2 Pinout, Commands, and Displaying Custom Character, Electronics for You*. Available at: <https://www.electronicsforu.com/technology-trends/learn-electronics/16x2-lcd-pinout-diagram>.

Chiu, Y.-J. (2024) ‘Automated medication verification system (AMVS): System based on Edge Detection and CNN classification drug on Embedded Systems’, *Heliyon*, 10(9). doi:10.1016/j.heliyon.2024.e30486.

Dayananda, P. and Upadhya, A.G. (2024) ‘Development of smart pill expert system based on IOT’, *Journal of The Institution of Engineers (India): Series B*, 105(3), pp. 457–467. doi:10.1007/s40031-023-00956-2.

Ehsan, M., Diop, M. and Pham, C. (2018) *Introduction to arduino IDE: Arduino Lora IOT Online Tutorial, Introduction to Arduino IDE | Arduino LoRa IoT online tutorial*. Available at: [https://cpham.perso.univ-pau.fr/LORA/HUBIQUITOUS/solution-lab/arduino-lora-tutorial/introduction\\_arduino\\_ide/introduction\\_arduino\\_ide/#:~:text=The%20Arduino%20IDE%20\(Integrated%20Development,reason%20Arduino%20became%20so%20popular](https://cpham.perso.univ-pau.fr/LORA/HUBIQUITOUS/solution-lab/arduino-lora-tutorial/introduction_arduino_ide/introduction_arduino_ide/#:~:text=The%20Arduino%20IDE%20(Integrated%20Development,reason%20Arduino%20became%20so%20popular). (Accessed: 01 May 2025).

Harun, H.N. and Nizam, F. (2024). *More than half a million Malaysian adults have diabetes, hypertension, high cholesterol plus obesity | New Straits Times*. [online] NST Online. Available at: <https://www.nst.com.my/news/nation/2024/05/1051421/more-half-million-malaysian-adults-have-diabetes-hypertension-high> (Accessed 30 Jun. 2024).

Harwani, P. (2024) *What is Android Studio?*, *Scaler Topics*. Available at: <https://www.scaler.com/topics/android/what-is-android-studio/> (Accessed: 11 May 2025).

Institute for Public Health (IPH) (2024) *Key Findings from the National Health and Morbidity Survey (NHMS) 2023: Health & Healthcare Demand, Institute for Public Health*. Available at: <https://iku.gov.my/nhms-2023> (Accessed: 11 May 2025).

Jinbao Plastic (2025) *What makes acrylic sheets ideal for LED light diffusion panels?* - *jinan jinbao plastic co, ltd.*. Available at: <https://www.jinbaoplastic.com/What-Makes-Acrylic-Sheets-Ideal-for-LED-Light-Diffusion-Panels-id45565516.html> (Accessed: 08 May 2025).

Kvarnström, K. *et al.* (2021) 'Factors contributing to medication adherence in patients with a chronic condition: A scoping review of qualitative research', *Pharmaceutics*, 13(7), p. 1100. doi:10.3390/pharmaceutics13071100.

Lee, W.M. (2019) *Introduction to cloud firestore, CODE*. Available at: <https://www.codemag.com/Article/1905071/Introduction-to-Cloud-Firestore> (Accessed: 02 May 2025).

Murel, J. (2024) *What is object detection?*, *IBM*. Available at: <https://www.ibm.com/think/topics/object-detection> (Accessed: 05 May 2025).

Nasir, Z. *et al.* (2023) 'Design of a smart medical box for automatic pill dispensing and health monitoring †', *INTERACT* 2023, 3, p. 7. doi:10.3390/engproc2023032007

Redmon, J. *et al.* (2016) *You only look once: Unified, real-time object detection*, *arXiv.org*. Available at: <https://arxiv.org/abs/1506.02640> (Accessed: 14 May 2025).

Religioni, U. *et al.* (2025) 'Enhancing therapy adherence: Impact on clinical outcomes, healthcare costs, and patient quality of life', *Medicina*, 61(1), p. 153. doi:10.3390/medicina61010153.

Robottronic (2020) *DC-DC boost converter MT3608*, *Instructables*. Available at: <https://www.instructables.com/DC-DC-Boost-Converter-MT3608/> (Accessed: 02 May 2025).

Saddam (2016) *Visitor monitoring system with Raspberry Pi and pi camera*, *Circuit Digest - Electronics Engineering News, Latest Products, Articles and Projects*. Available at: <https://circuitdigest.com/microcontroller-projects/visitor-monitoring-with-raspberry-pi-and-pi-camera> (Accessed: 11 May 2025).

Santos, R. (2019) *How do RGB leds work?*, *Random Nerd Tutorials*. Available at: <https://randomnerdtutorials.com/electronics-basics-how-do-rgb-leds-work/> (Accessed: 12 March 2025).

Scully, T. (2019) *How Does a 5mm LED Work?*, *LED Supply*. Available at: <https://www.ledsupply.com/blog/how-does-a-5mm-led-work/> (Accessed: 25 March 2025).

Singh, A. (2025) *Raspberry pi 4 pinout, specifications and applications*, *Hackatronic*. Available at: <https://www.hackatronic.com/raspberry-pi-4-specifications-pin-diagram-and-description> (Accessed: 25 March 2025).

Torres, J. (2024) *How to evaluate yolov8 model: A comprehensive guide*, *YOLOv8*. Available at: <https://yolov8.org/how-to-evaluate-yolov8-model/> (Accessed: 10 May 2025).

## APPENDICES

### Appendix A: QR Code for Demonstration Video



### Appendix B: C++ Code for ESP8266

```
#include<ESP8266WiFi.h>
#include<WiFiClientSecure.h>
#include<ArduinoJson.h>
#include<ESP8266HTTPClient.h>
#include<WiFiClient.h>
#include<NTPClient.h>
#include<WiFiUdp.h>

const char* ssid = "AndroidAPD67C";
const char* password = "nlba2708";
```

```

// Firebase project credentials
const String FIREBASE_HOST ="http://medicine-reminder-92853.firebaseio.com";
const      String      FIREBASE_API_KEY      ="AIzaSyDgvmd-
bI_QsNiK0zmArLWX8WxRoeZWxs";
const String FIREBASE_PROJECT_ID ="medicine-reminder-92853";
const String USER_ID ="vfUin8HcMlba6MefkhxpBBk5xxT2";

// Firestore API URL
String  firestoreURL  =   "https://firestore.googleapis.com/v1/projects/"   +
FIREBASE_PROJECT_ID +
        "/databases/(default)/documents/users/"      +      USER_ID      +
"/reminder?key=" + FIREBASE_API_KEY;
unsigned long getCurrentTime();
void getFirestoreData();
void parseAndConvertTime(String jsonResponse);
void checkReminderTime(unsigned long currentTime);
void triggerBuzzer();
unsigned long convertToTimestamp(String adjustedReminder);
unsigned long convertToTimestamp(int year, int month, int day, int hour, int minute,
int second);

struct Reminder{
    String adjustedReminder;
    unsigned long reminderTime;
    bool onOff;
    bool triggered;
};

// array size
#define MAX_REM 10

//reminder array

```

```

Reminder reminders[MAX_REM];
int reminderCount =0;

//Define the pin for buzzer and RGB LED
#define BUZZER_PIN D8
#define redPin  D0
#define greenPin D1
#define bluePin D2

//create HTTPClient instance
HTTPClient http;
WiFiClientSecure client;
//WiFiClient wificlient; //SSL/TLS
int yearInt, monthInt, dayInt, hourInt, minuteInt, secondInt;
bool onOff;
String payload = "";

WiFiUDP udp;
NTPClient timeClient(udp, "pool.ntp.org", 0, 3600); // NTP setup

void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

  Serial.begin(9600);
  delay(10);

  // Connect to WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

```

```
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

// white light--> Connected to WIFI
digitalWrite(redPin, HIGH);
digitalWrite(greenPin, HIGH);
digitalWrite(bluePin, HIGH);
delay(1000);

//Initialize HTTPS client
client.setInsecure();

Serial.println("Initializing NTP...");
timeClient.begin();

while (timeClient.getEpochTime() < 1000000000) {
  Serial.println("Waiting for NTP sync...");
  timeClient.update();
  delay(1000);
}

Serial.println("NTP synced successfully!");
}
```

```
void loop() {  
    //get data or send other HTTP requests  
    delay(5000);  
  
    // get payload from firestore  
    getFirestoreData();  
  
    // Get current time from NTP  
    unsigned long currentTime = getCurrentTime();  
  
    // Check the reminder time from Firestore  
    checkReminderTime(currentTime);  
}  
  
// get current time from NTP  
unsigned long getCurrentTime() {  
    timeClient.update(); // Update NTP time  
    unsigned long utcTime = timeClient.getEpochTime(); // Get time in seconds (UTC)  
  
    if(utcTime<10000000000){  
        Serial.println("Error:NTP time not synced properly!");  
        return 0;  
    }  
  
    Serial.println("Raw UTC Time: ");  
    Serial.println(utcTime);  
  
    return utcTime;  
}  
  
void getFirestoreData() {  
    delay(2000);
```



```
int max_tries =3;
int tries =0;
int httpStatusCode =0;

//HTTP request
http.begin(client, url);
http.addHeader("Content-Type", "application/json");

http.setTimeout(8000);

while(tries <max_tries){
// Send GET request
httpStatusCode = http.GET();

if (httpStatusCode > 0) {
  if (httpStatusCode == 200){
    Serial.printf("HTTP Response code: %d\n",httpStatusCode);
    String payload = http.getString();
    Serial.println("Got HTTP response");

    //blue light --> got HTTP response
    digitalWrite(redPin, LOW);
    digitalWrite(greenPin, LOW);
    digitalWrite(bluePin, HIGH);
    delay(1000);

    Serial.println("Response from Firestore:");
    Serial.println(payload);
    if (payload.length() >0){
      Serial.println("Data retrieved successfully!");
      // Green light --> retrieved payload from firestore
      digitalWrite(redPin, LOW);
      digitalWrite(greenPin, HIGH);
```

```
        digitalWrite(bluePin, LOW);
        delay(1000); // Delay 1 second
    }
    parseAndConvertTime(payload);
    break;
} else {
    Serial.println("Error on HTTP request");
    Serial.print("Retrying... ");
    Serial.print("Error code: ");
    Serial.println (httpResponseCode);
    // red light --> indicate error
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, LOW);
    digitalWrite(bluePin, LOW);
    delay(1000); // Delay 1 second
}

} else {
    Serial.println("Error on HTTP request");
    Serial.print("Error code: ");
    Serial.println (httpResponseCode);
    // red light --> indicate error
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, LOW);
    digitalWrite(bluePin, LOW);
    delay(1000); // Delay 1 second
}

    tries++;
}
http.end();
}
```

```
void parseAndConvertTime(String jsonResponse) {
    reminderCount = 0;

    // Create a JSON document object to parse the response
    DynamicJsonDocument doc(16384);
    //Deserialize JSON and check errors
    DeserializationError error = deserializeJson(doc, jsonResponse);

    // Test if parsing succeeds
    if (error) {
        Serial.print(F("deserializeJson() failed: "));
        Serial.println(error.f_str());
        delay(1000);
        return; // Exit the function early if JSON parsing failed
    }

    JsonArray documents = doc["documents"];

    if (documents.size() > 0) {
        for (JsonObject docObj : documents) {
            serializeJsonPretty(docObj, Serial);

            String timestamp = docObj["fields"]["time"]["timestampValue"];
            onOff = docObj["fields"]["onOff"]["booleanValue"];

            if (timestamp != "") {
                Serial.print("UTC Timestamp: ");
                Serial.println(timestamp);

                Serial.print("onOff status: ");
                Serial.println(onOff ? "true" : "false");

                // Convert timestamp
```

```

String dateTimeStr = timestamp; // Example: "2025-01-08T05:50:00Z"
String year = dateTimeStr.substring(0, 4);
String month = dateTimeStr.substring(5, 7);
String day = dateTimeStr.substring(8, 10);
String hour = dateTimeStr.substring(11, 13);
String minute = dateTimeStr.substring(14, 16);
String second = dateTimeStr.substring(17, 19);

// Convert to integers
yearInt = year.toInt();
monthInt = month.toInt();
dayInt = day.toInt();
hourInt = hour.toInt();
minuteInt = minute.toInt();
secondInt = second.toInt();

// Create a string
String adjustedTime = String(yearInt) + "-" + String(monthInt) + "-" +
String(dayInt) +
        " " + String(hourInt) + ":" + String(minuteInt) + ":" +
String(secondInt);

unsigned long reminderTime = convertToTimestamp(yearInt, monthInt, dayInt,
hourInt, minuteInt, secondInt);

if (reminderCount < MAX_REM){
    reminders [reminderCount].adjustedReminder= adjustedTime;
    reminders [reminderCount].reminderTime= reminderTime;
    reminders [reminderCount].onOff = onOff;
    reminderCount++;
}else{
    Serial.println ( "max number of reminders reached");
}
}

```

```

    }
  } else {
    Serial.println("No timestamp found.");
  }
}

void checkReminderTime(unsigned long currentTime) {
  if (currentTime == 0){
    Serial.println("Error: Invalid current time!");
    return;
  }else{
    Serial.print("Current Time (UTC): ");
    Serial.println(currentTime);
  }

  //Loop all the reminders
  for (int i = 0; i<reminderCount; i++){
    long timeDiff = (long)currentTime - (long)reminders[i].reminderTime;

    Serial.print("Reminder Time: ");
    Serial.println(reminders[i].reminderTime);
    Serial.print("Current Time: ");
    Serial.println(currentTime);
    Serial.print("Difference: ");
    Serial.println(timeDiff);

    if (!reminders[i].triggered &&
        (timeDiff >= -180) &&
        (timeDiff <= 180) &&
        reminders[i].onOff) {

      Serial.print("Reminder Time(UTC):");
      Serial.println(reminders[i].reminderTime);
    }
  }
}

```

```

    Serial.println(reminders[i].onOff? "true" : "false");
    Serial.println("Time reached, buzzer triggered!");
    triggerBuzzer();
    reminders[i].triggered = true;
  } else {
    Serial.println("Buzzer does not trigger");
  }
}
}

void triggerBuzzer(){
  digitalWrite(BUZZER_PIN, HIGH);//buzzer connect to D1(digital pin)
  delay(3000); // buzz for 3 second
  digitalWrite(BUZZER_PIN, LOW);// turn off buzzer
}

unsigned long convertToTimestamp(String adjustedReminder) {
  int year = adjustedReminder.substring(0, 4).toInt();
  int month = adjustedReminder.substring(5, 7).toInt();
  int day = adjustedReminder.substring(8, 10).toInt();
  int hour = adjustedReminder.substring(11, 13).toInt();
  int minute = adjustedReminder.substring(14, 16).toInt();
  int second = adjustedReminder.substring(17, 19).toInt();
  return convertToTimestamp(year, month, day, hour, minute, second);
}

unsigned long convertToTimestamp(int year, int month, int day, int hour, int minute,
int second) {
  struct tm tmStruct = {0};
  tmStruct.tm_year = year - 1900;
  tmStruct.tm_mon = month - 1;
  tmStruct.tm_mday = day;
  tmStruct.tm_hour = hour;

```

```

tmStruct.tm_min = minute;
tmStruct.tm_sec = second;

time_t utcTime = mktime(&tmStruct); // Now treated as UTC
return utcTime;
}

```

### Appendix C: Python Code for Capturing Dataset

```

from picamera2 import Picamera2
import cv2
import os
import time
import RPi.GPIO as GPIO
import numpy as np

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)

dataset_folder = "/home/yeewei831/dataset/v4" # Folder to save images

YOLO_IMAGE_SIZE = (640, 640)

picam2 = Picamera2()
camera_config = picam2.create_still_configuration(
    raw={"size": (1640, 1232)},
    main={"format": 'RGB888', "size": (640, 480)}
)

picam2.configure(camera_config)

```

```
picam2.set_controls({
    "Contrast": 1.0,
    "Sharpness": 16.0,
})

def sharpen_image(image):
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
    return cv2.filter2D(image, -1, kernel)

def resize_image(image_path):
    img = cv2.imread(image_path)
    if img is not None:
        img = sharpen_image(img) # sharpening filter
        img = cv2.medianBlur(img, 5) #remove reflection

        # Get original image dimensions
        h, w = img.shape[:2]

        # Calculate padding to make square
        max_dim = max(h, w)
        top = (max_dim - h) // 2
        bottom = max_dim - h - top
        left = (max_dim - w) // 2
        right = max_dim - w - left

        # Padding
        padded_img = cv2.copyMakeBorder(img, top, bottom, left, right,
                                         cv2.BORDER_CONSTANT, value=[0, 0, 0])

        # Resize to 640x640
```



```

        resized = cv2.resize(padded_img, YOLO_IMAGE_SIZE)
        cv2.imwrite(image_path, resized)
        print(f"Resized (padded): {image_path} to {YOLO_IMAGE_SIZE}")
    else:
        print(f" Image not found: {image_path}")

def capture_images(num_images, delay, start_index = 0):
    print("Camera warming up... .")
    time.sleep(2)

    GPIO.output(17, GPIO.HIGH)
    GPIO.output(23, GPIO.HIGH)

    picam2.start()
    print("Capturing process started...")

    for i in range(start_index, start_index+num_images):

        filename = os.path.join(dataset_folder, f"pill_{i}.jpg")
        picam2.capture_file(filename)
        print(f"Captured {filename}")

        resize_image(filename) # Resize image

        if i < start_index+num_images - 1:
            print(f"Waiting {delay} seconds for next capture...")
            time.sleep(delay)

    picam2.stop()
    print("[INFO] Image captured successfully!")
    GPIO.output(17, GPIO.LOW) # Turn LED off
    GPIO.output(23, GPIO.LOW) # Turn LED off
    GPIO.cleanup()

```

```
capture_images(5, 2)
```

#### Appendix D: Python Code for Real-Time Pill Detection

```
from ultralytics import YOLO
import firebase_admin
from firebase_admin import credentials, firestore
from picamera2 import Picamera2
import cv2
import time
import RPi.GPIO as GPIO
import numpy as np
import threading
from picamera2.controls import Controls
from RPLCD import CharLCD

lcd = CharLCD(cols=16, rows=2, pin_rs=26, pin_e=19, pins_data=[13, 6, 5, 11],
              numbering_mode=GPIO.BCM)

# Initialize Firebase
cred = credentials.Certificate("/home/yeewei831/pi/medicine-reminder-92853-
firebase-adminsdk-bnij5-78500064f6.json")
firebase_admin.initialize_app(cred)
db = firestore.client()

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(24, GPIO.OUT)
GPIO.setup(25, GPIO.OUT)
GPIO.setup(16, GPIO.OUT)

# Initialize Camera
```

```

picam2 = Picamera2()
camera_config = picam2.create_preview_configuration(
    raw={"size": (1640, 1232)},
    main={"format": 'RGB888', "size": (640, 480)}
)
picam2.configure(camera_config)

picam2.set_controls({
    "Contrast": 1.0,      # 0.0 to 32.0
    "Sharpness": 16.0,   # 0.0 to 16.0
})

GPIO.output(17, GPIO.HIGH)
GPIO.output(23, GPIO.HIGH)
picam2.start()
# Load YOLO Model
model = YOLO("YOLO_results_v4/pill_detector_v42/weights/best.pt") # trained
model path

def blink_led(pin, stop_event, interval=0.5):
    while not stop_event.is_set():
        GPIO.output(pin, GPIO.HIGH)
        time.sleep(interval)
        GPIO.output(pin, GPIO.LOW)
        time.sleep(interval)

def capture_image():
    print("Capturing Image...")
    picam2.capture_file("pill_image.jpg")
    return "pill_image.jpg"

def capture_pills(image_path):
    print("Capturing Pills...")
    lcd_display("Capturing Pills")

```

```
start_time = time.time()

results = model.predict(source=image_path, conf=0.681, show=True)

end_time = time.time() # End timing
elapsed_time = end_time - start_time
print(f"YOLO detection took {elapsed_time:.2f} seconds")

# Get class name
class_names = model.names
print(model.names)

Amlodipine_count = 0
Simvastatin_count = 0
BoxPresent = False
Unknown_count = 0

for result in results:
    boxes = result.boxes
    for box in boxes:
        class_id = int(box.cls[0]) # Get class ID
        label = class_names[class_id] # Get label name

        if label.lower() == "boxpresent":
            BoxPresent = True
        elif label.lower() == "amlodipine":
            Amlodipine_count += 1
        elif label.lower() == "simvastatin":
            Simvastatin_count += 1
        elif label.lower() == "unknown":
            Unknown_count += 1
```

```

if BoxPresent:
    lcd_display("Box detected")
    time.sleep(2)
    GPIO.output(16, GPIO.HIGH)
    print("Box Present: Yes")
    print(f"Amlodipine Count: {Amlodipine_count}")
    print(f"Simvastatin Count: {Simvastatin_count}")
    print(f"Unknown Count: {Unknown_count}")
    lcd_display_pill_counts(Amlodipine_count, Simvastatin_count,
Unknown_count)
    time.sleep(3)
    if Unknown_count >= 1:
        GPIO.output(24, GPIO.HIGH) # Error
        print(f"E: Unknown object detected!")
        lcd_display("E :unknown obj !")
        time.sleep(3)
        GPIO.output(24, GPIO.LOW)

    print("box detected and data are compared")
    compare_firestore (Amlodipine_count, Simvastatin_count)

else:
    lcd_display("No box detected")
    time.sleep(2)
    lcd_display("skipping count!")
    print ("Box present: NO")
    print("skipping count")
    GPIO.output(16, GPIO.LOW)
    GPIO.output(24, GPIO.LOW)

return BoxPresent, Amlodipine_count, Simvastatin_count, Unknown_count

```

```

    #compare with data stored in firestore
def compare_firestore(amlo_count, simva_count):
    print("Comparing data with data stored in firestore")
    lcd_display("comparing data")

    uid = "vfUin8HcMlba6MefkhxpBBk5xxT2"

    latest_log=(
    db.collection("users")
    .document(uid)
    .collection("pill_logs")
    .order_by("timestamp", direction=firestore.Query.DESCENDING)
    .limit(1)
    .get()
    )

    if latest_log:
        latest_data = latest_log[0].to_dict()
        firestore_amlo = latest_data.get("amlodipine_count",0)
        firestore_simva = latest_data.get("simvastatin_count",0)

        print(f"[FIRESTORE]Amlodipine:{firestore_amlo},
Simvastatin:{firestore_simva}")
        print(f"[LOCAL] Amlodipine:{amlo_count}, SImvastatin:{simva_count}")

        if firestore_amlo != amlo_count or firestore_simva != simva_count:
            print("E:pills mismatch!")
            GPIO.output(24, GPIO.HIGH)
            lcd_display("E:Pills mismatch!")
            time.sleep (3)

```

```

        GPIO.output(24, GPIO.LOW)
        GPIO.output(16, GPIO.LOW)

    else:
        print("no alert, pill count match")
        lcd_display("pills match!")
        GPIO.output(16, GPIO.LOW)

    else:
        print ("no pill log found")

def lcd_display ( message):
    lcd.clear()
    lcd.write_string(message)

def lcd_display_pill_counts(amlo_count, simva_count, unknown_count):
    lcd.clear()
    lcd.write_string(f"A:{amlo_count} S:{simva_count}")
    lcd.crlf()
    lcd.write_string(f"U:{unknown_count}")

def main():
    while True:
        GPIO.output(25, GPIO.HIGH)
        stop_blink = threading.Event()
        blink_thread = threading.Thread(target=blink_led, args=(25, stop_blink))
        blink_thread.start()

        image_path = capture_image()
        BoxPresent, amlo_count, simva_count, Unknown_count =
capture_pills(image_path)

```

```

    stop_blink.set()
    blink_thread.join()
    GPIO.output(25, GPIO.LOW) # Make sure LED ends off

    time.sleep(3)
    if cv2.waitKey(1) & 0xFF == ord('q'): # Press 'q' to exit
        break

    cv2.destroyAllWindows()
    picam2.stop() # Stop the camera
    GPIO.output(17, GPIO.LOW) # Turn LED off
    GPIO.output(23, GPIO.LOW) # Turn LED off
    lcd.clear()
    GPIO.cleanup()

if __name__ == "__main__":
    main()

```

#### Appendix E: Dart Code for Main

```

import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:first_app/firebase_options.dart';
import 'package:first_app/login_screen.dart';
import 'package:first_app/home.dart';
import 'package:flutter/material.dart';

void main() async{
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );

```



```

    runApp (MyApp());
  }
class MyApp extends StatelessWidget {
  final FirebaseAuth _auth = FirebaseAuth.instance;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner:false,
      title:'Medicine Reminder',
      home: _auth.currentUser != null? Alarm() :LoginScreen(),
    );
  }
}

```

#### Appendix F: Dart Code for Home Screen

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:first_app/pill_tracker.dart';
import 'package:first_app/addreminder.dart';
import 'package:first_app/delete_reminder.dart';
import 'package:first_app/login_screen.dart';
import 'package:first_app/services/notification_logic.dart';
import 'package:first_app/switcher.dart';
import 'package:flutter/material.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:intl/intl.dart';

class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

```

```

@override
State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  User? user;
  bool on = true;

  @override
  void initState() {
    super.initState();
    user = FirebaseAuth.instance.currentUser;
    NotificationLogic.init(context, user!.uid);
    listenNotifications();
  }

  void listenNotifications() {
    NotificationLogic.onNotifications.listen((value) {});
  }

  void onClickedNotifications(String? payload) {
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(
        builder: (context) => const HomeScreen(),
      ));
  }

  @override
  Widget build(BuildContext context) {
    return PopScope(
      canPop: false,

```

```

child: Scaffold(
  appBar: AppBar(
    backgroundColor: Colors.green[200],
    centerTitle: true,
    elevation: 0,
    title: const Text(
      "Medicine Reminder Alarm -YW",
      style: TextStyle(
        color: Colors.black,
        fontSize: 18,
        fontWeight: FontWeight.w500,
      ),
    ),
    actions: [
      IconButton(
        icon: const Icon(Icons.logout),
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => const LoginScreen(),
            ));
        },
      ),
    ],
  ),
  floatingActionButton: FloatingActionButton(
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(100),
    ),
    onPressed: () async {
      addReminder(context, user!.uid);
    },
  ),
)

```

```

child: Container(
  decoration: BoxDecoration(
    gradient: const LinearGradient(
      colors: [Colors.green, Colors.blue],
      begin: Alignment.centerLeft,
      end: Alignment.centerRight,
    ),
    borderRadius: BorderRadius.circular(100),
    boxShadow: const [
      BoxShadow(
        color: Colors.black,
        blurRadius: 2,
        offset: Offset(0, 2),
      )
    ],
  ),
  child: const Center(
    child: Icon(
      Icons.add,
      color: Colors.white,
      size: 30,
    ),
  ),
),
),
),
body: StreamBuilder<QuerySnapshot>(
  stream: FirebaseFirestore.instance
    .collection("users")
    .doc(user!.uid)
    .collection('reminder')
    .snapshots(),
  builder:
    (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {

```

```

return const Center(
  child: CircularProgressIndicator(
    valueColor:
      AlwaysStoppedAnimation<Color>(Colors.blueAccent)),
);
}
;

if (snapshot.data!.docs.isEmpty) {
  return const Center(
    child: Text("Nothing to Show"),
  );
};
final data = snapshot.data;
return ListView.builder(
  itemCount: data?.docs.length,
  itemBuilder: (context, index) {
    final doc = data!.docs[index];
    final docData = doc.data() as Map<String, dynamic>;

    Timestamp t = docData['time'] as Timestamp;
    DateTime date =
    DateTime.fromMicrosecondsSinceEpoch(t.microsecondsSinceEpoch);
    String formattedTime = DateFormat.jm().format(date);

    bool on = docData['onOff'] ?? true;
    String pillName = docData['pillName'] ?? "";
    int pillCount = docData['pillCount'] ?? 1;
    int notificationnId = docData['notificationId'] ?? 0;

    if (on) {
      final String uid = user!.uid; // Ensure you extract uid
      final String payload = '$pillName|$uid|$pillCount'; // Correct interpolation

```

```

final int notificationId = notificationnId; // Use a unique ID from Firestore
NotificationLogic.showNotifications(
    dateTime: date,
    id: notificationId,
    pillName: pillName,
    pillCount: pillCount,
    title: "Reminder (YW)",
    body: "Don't forget to take $pillCount your $pillName",
    payload:payload);
}
return SingleChildScrollView(
  child: Column(
    children: [
      Padding(
        padding: EdgeInsets.all(8),
        child: Card(
          child: ListTile(
            title: Text(
              formattedTime,
              style: const TextStyle(fontSize: 30),
            ),
            subtitle: Text("Everyday • $pillName: $pillCount tablets"),
            trailing: Container(
              width: 110,
              child: Row(
                children: [
                  Switcher(on, user!.uid, data.docs[index].id,
                    data.docs[index].get('time')),
                  IconButton(
                    onPressed: () {
                      deleteReminder(context,
                        data.docs[index].id, user!.uid);
                    },

```

```

        icon: const FaIcon(
          FontAwesomeIcons.circleXmark),
      ),
    ],
  ),
),
),
),
),
),
],
),
);
},
);
},
),
),
);
}
}

```

```

class Alarm extends StatefulWidget {
  const Alarm({super.key});

  @override
  State<Alarm> createState() => _AlarmState();
}

```

```

class _AlarmState extends State<Alarm> {
  int currentPage = 0;

  List<Widget> pages = [
    const HomeScreen(),
  ],
}

```

```

const PillTracker(),
];

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: pages[currentPage],
    bottomNavigationBar: BottomNavigationBar(
      onTap: (int value) {
        setState(() {
          currentPage = value; // Update currentPage when tab is tapped
        });
      },
      items: const [
        BottomNavigationBarItem(
          label: 'Alarm',
          icon: Icon(Icons.alarm),
        ),
        BottomNavigationBarItem(
          label: 'Pill',
          icon: Icon(Icons.medication),
        ),
      ],
    ),
  );
}
}

```

### Appendix G: Dart Code for Notification Logic

```

import 'package:first_app/pill_tracker.dart';
import 'package:first_app/home.dart';
import 'package:flutter/material.dart';

```



```

import 'package:flutter_local_notifications/flutter_local_notifications.dart';
import 'package:rxdart/rxdart.dart';
import 'package:timezone/data/latest_all.dart' as tz;
import 'package:timezone/timezone.dart' as tz;
import 'package:cloud_firestore/cloud_firestore.dart';

class NotificationLogic {
  static final _notifications = FlutterLocalNotificationsPlugin();
  static final onNotifications = BehaviorSubject<String?>();

  static Future init(BuildContext context, String uid) async {
    tz.initializeTimeZones();

    const initializationSettingAndroid = AndroidInitializationSettings("clock");
    const initializationSettings = InitializationSettings(android:
initializationSettingAndroid);
    await _notifications.initialize(
      initializationSettings,
      onDidReceiveNotificationResponse: (NotificationResponse response) async {
        final payload = response.payload;
        if (response.actionId == 'mark_taken') {
          print('pill marked as taken via notification');

          await markPillAsTaken (payload);
        } else {
          Navigator.push(context,
            MaterialPageRoute(builder: (context) => const HomeScreen()));
        }
      }
    );
  }

  static Future showNotifcations(

```

```

    {int id = 0,
    String? title,
    String? pillName,
    int? pillCount,
    String? body,
    String? payload,
    required DateTime dateTime}) async {
  if (dateTime.isBefore(DateTime.now())) {
    dateTime = dateTime.add(const Duration(days: 1));
  }

  const AndroidNotificationDetails androidDetails = AndroidNotificationDetails(
    "Schedule Reminder",
    "Don't Forget to eat the pill",
    importance: Importance.max,
    priority: Priority.max,
    actions: <AndroidNotificationAction>[
      AndroidNotificationAction(
        'mark_taken', //action ID
        'Mark as Taken',
        showsUserInterface: true,
        cancelNotification: true,
      ), ], );
  final notificationDetails = const NotificationDetails(android: androidDetails);

  await _notifications.zonedSchedule(
    id,
    title,
    body,
    tz.TZDateTime.from(dateTime, tz.local),
    notificationDetails,
    uiLocalNotificationDateInterpretation:
      UILocalNotificationDateInterpretation.absoluteTime,

```

```

        androidScheduleMode: AndroidScheduleMode.exactAllowWhileIdle,
        matchDateTimeComponents: DateTimeComponents.time,
        payload: payload,
    ); }

static Future<void> showInstantNotification({
    required String title,
    required String body,
    required int id,
}) async {
    const androidDetails = AndroidNotificationDetails(
        'pill_channel',
        'Pill Notifications',
        importance: Importance.max,
        priority: Priority.high,
    );

    const notificationDetails = NotificationDetails(android: androidDetails);

    await _notifications.show(id, title, body, notificationDetails);
}

//after pressing mark as taken button can minus the pillCount amount & updates pill
tracker page
static Future markPillAsTaken(String? payload) async {
    if (payload == null) return;

    print('Received payload: $payload');

    final parts = payload.split('|');
    if (parts.length < 3) return; // 3 parts

    final String pillName = parts[0];
    final String uid = parts[1];
    final int amountToDeduct = int.tryParse(parts[2]) ?? 1;

```

```

final pillLogRef = FirebaseFirestore.instance
    .collection('users')
    .doc(uid)
    .collection('pill_logs');

try {
    final snapshot = await pillLogRef.orderBy('timestamp', descending:
true).limit(1).get();

    if (snapshot.docs.isEmpty) {
        print('No pill logs found.');
```

```

        return;
    }
    final latestDoc = snapshot.docs.first;
    final data = latestDoc.data();
    if (pillName.toLowerCase().contains('amlodipine')) {
        int originalCount = data['amlodipine_count'] ?? 0;
        int updatedCount = originalCount - amountToDeduct;
        await latestDoc.reference.update({'amlodipine_count': updatedCount});
        checkPillThreshold(pillName: pillName, originalCount: originalCount,
currentCount: updatedCount);

    } else if (pillName.toLowerCase().contains('simvastatin')) {
        int originalCount = data['simvastatin_count'] ?? 0;
        int updatedCount = originalCount - amountToDeduct;
        await latestDoc.reference.update({'simvastatin_count': updatedCount});
        checkPillThreshold(pillName: pillName, originalCount: originalCount,
currentCount: updatedCount);

    } else {
        print('Unrecognized pill name.');
```

```

        return;
    }
    print('Pill count updated in latest log.');
```

```

    } catch (e) {
      print('Error marking pill as taken: $e');
    }
  }}

```

## Appendix H: Dart Code for Add Reminder

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:first_app/reminder_model.dart';
import 'package:first_app/services/notification_logic.dart';
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';
import 'package:fluttertoast/fluttertoast.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';

addReminder(BuildContext context, String uid) {
  TimeOfDay time = TimeOfDay.now();
  final TextEditingController pillNameController = TextEditingController();
  final TextEditingController pillCountController = TextEditingController();

  add(String uid, TimeOfDay time) async{
    try {
      DateTime d = DateTime.now();
      DateTime dateTime =
        DateTime(d.year, d.month, d.day, time.hour, time.minute).toUtc();
      Timestamp timestamp = Timestamp.fromDate(dateTime);

      DocumentReference docRef =FirebaseFirestore.instance
        .collection('users')
        .doc(uid)
        .collection('reminder')
        .doc();

```

```

int notifId = DateTime.now().millisecondsSinceEpoch.remainder(100000);

ReminderModel reminderModel = ReminderModel();
reminderModel.timestamp = timestamp;
reminderModel.onOff = true; // turn ON the reminder
reminderModel.pillName = pillNameController.text.trim();
reminderModel.pillCount = int.tryParse(pillCountController.text);
reminderModel.notificationId = notifId;

await docRef.set(reminderModel.toMap());

Fluttertoast.showToast(msg: "Medicine reminder added");
} catch (e) {
  Fluttertoast.showToast(msg: e.toString());
}
}

return showDialog(
  context: context,
  builder: (context) {
    return StatefulBuilder(
      builder:
        (BuildContext context, void Function(void Function()) setState) {
      return AlertDialog(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(25),
        ),
        title: const Text("Add Reminder"),
        content: SingleChildScrollView(
          child: Column(
            children: [
              const Text("Select time for the medicine reminder"),

```

```

const SizedBox(height: 20),
MaterialButton(
  onPressed: () async {
    //create TimePicker
    TimeOfDay? newTime = await showTimePicker(
      context: context, initialTime: TimeOfDay.now());
    if (newTime == null) return;
    setState(() {
      time = newTime;
    },);
  },
  child: Row(
    children: [
      const FaIcon(
        FontAwesomeIcons.clock,
        color: Colors.lightGreen,
        size: 30,
      ),
      const SizedBox(width: 10),
      Text(
        time.format(context).toString(),
        style: const TextStyle(
          color: Colors.green,
          fontSize: 20,
        ),
      )
    ],
  ),
),
const SizedBox (height:20),
TextField(
  controller: pillNameController,
  decoration:

```

```

        const InputDecoration(
          labelText: "Pill name",
          border: OutlineInputBorder(),
        ),
      ),
      const SizedBox(height: 10),

      TextField(

        controller: pillCountController,

        decoration: const InputDecoration(
          labelText: "Amount (e.g. 2 tablets)",
          border: OutlineInputBorder(),
        ),
        keyboardType: TextInputType.text,
      )

    ],
  ),
),
actions: [
  TextButton(
    onPressed: () {
      Navigator.pop(context);
    },
    child: const Text("Cancel"),
  ),
  TextButton(
    onPressed: () {
      final pillName = pillNameController.text.trim();
      final pillCountText = pillCountController.text.trim();

```



```

        final pillCount = int.tryParse(pillCountText);
        if (pillNameController.text.trim().isEmpty) {
            Fluttoast.showToast(msg: "Please enter a pill name.");
            return;
        }
        if (pillCount == null || pillCount <= 0){
            Fluttoast.showToast(msg: "Please enter a valid number");
            return;
        }

        add(uid, time);
        Navigator.pop(context); // close the dialog
    },
    child: const Text("Add"),
),
],
);
},
);
},
);
}

```

## Appendix I: Dart Code for Button

```

import 'package:flutter/material.dart';

class CustomButton extends StatelessWidget {
    const CustomButton({super.key,
        required this.label,
        this.onPressed});
}

```

```

final String label;
final void Function()? onPressed;

@override
Widget build(BuildContext context) {
  return SizedBox(
    width: 180,
    height: 42,
    child: ElevatedButton(
      onPressed: onPressed,
      child: Text(
        label,
        style: const TextStyle(fontSize: 18),
      )),
  );
}

```

#### Appendix J: Dart Code for Delete Reminder

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:fluttertoast/fluttertoast.dart';

deleteReminder(BuildContext context, String id, String uid) {
  return showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(25),
        ),
        title: const Text("Delete reminder"),

```

```

content: const Text("Are you sure you want to delete?"),
actions: [
  TextButton(
    onPressed: () {
      try {
        FirebaseFirestore.instance
          .collection("users")
          .doc(uid)
          .collection("reminder")
          .doc(id)
          .delete();
        Fluttertoast.showToast(
          msg: "Reminder Deleted successfully!");
      } catch (e) {
        Fluttertoast.showToast(msg: e.toString());
      }
      Navigator.pop(context);
    },
    child: const Text("Delete"),
  ),
  TextButton(
    onPressed: () {
      try {} catch (e) {}
      Navigator.pop(context);
    },
    child: const Text("Cancel"),
  ),
];
});
}

```

## Appendix K: Dart Code for Login Screen

```

import 'package:firebase_auth/firebase_auth.dart';
import 'package:first_app/home.dart';
import 'package:first_app/round_text_field.dart';
import 'package:first_app/signup_screen.dart';
import 'package:flutter/material.dart';
class LoginScreen extends StatefulWidget {
  const LoginScreen({super.key});

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final FirebaseAuth _auth = FirebaseAuth.instance;

  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passController = TextEditingController();
  bool _isObscure = true;
  final _formKey = GlobalKey<FormState>();

  Future<User?> _signIn(
    BuildContext context, String email, String password) async {
    try {

      UserCredential usercredential = await _auth.signInWithEmailAndPassword(
        email: email, password: password);
      User? user = usercredential.user;
      ///navigate to Firstpage if the sign in is successful
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => const Alarm(),
        ),

```

```

);
return user;

} catch (e) {
  //show a snackbar if sign-in fails
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('Login Failed')),
  );
  return null;
}
}

@override
void initState() {
  super.initState();
}

@override
Widget build(BuildContext context) {
  var media = MediaQuery.of(context).size;
  return MaterialApp(
    debugShowCheckedModeBanner: false,
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Medicine Reminder'),
        backgroundColor: Colors.greenAccent,
        centerTitle: true,
      ),
      body: SafeArea(
        child: SingleChildScrollView(
          child: Container(
            padding: const EdgeInsets.symmetric(

```

```

        vertical: 15, horizontal: 25),
child: Form(
  key: _formKey,
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    crossAxisAlignment: CrossAxisAlignment.center,
    children: [
      SizedBox(height: media.height * 0.1),
      SizedBox(
        width: media.width,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            SizedBox(
              height: media.width * 0.03,
            ),
            const Text(
              'Welcome',
              textAlign: TextAlign.center,
              style: TextStyle(
                color: Colors.black,
                fontSize: 30,
                fontWeight: FontWeight.w200),
            )
          ],
        ),
      ),
      SizedBox(
        height: media.width * 0.15,
      ),
      RoundTextField(
        textEditingController: _emailController,
        hintText: "Email",

```

```

        icon: "assets/icons/email_icon.png",
        textInputType: TextInputType.emailAddress,
        validator: (value) {
          if (value == null || value.isEmpty) {
            return "Please enter email";
          }
          return null;
        },
      ),
      SizedBox(
        height: media.width * 0.05,
      ),
      RoundTextField(
        textEditingController: _passController,
        hintText: "Password",
        icon: "assets/icons/password_icon.png",
        textInputType: TextInputType.text,
        isObscureText: _isObscure,
        validator: (value) {
          if (value == null || value.isEmpty) {
            return "Please enter password";
          }
          return null;
        },
      ),
      rightIcon: TextButton(
        onPressed: () {
          setState(
            () {
              _isObscure = !_isObscure;
            },
          );
        },
      ),
      child: Container(

```

```

        alignment: Alignment.center,
        height: 20,
        width: 20,
      ),
    )),
    SizedBox(
      height: media.width * 0.1,
    ),
    ElevatedButton(
      onPressed: () {
        if (_formKey.currentState!.validate()) {
          _signIn(context, _emailController.text,
            _passController.text);
        }
      },
      child: const Text("login")),

    SizedBox(
      height: media.width * 0.01,
    ),
    TextButton(
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => const
SignupScreen(),
        ),
      );
    },
    child: RichText(
      textAlign: TextAlign.center,
      text: const TextSpan(
        style: TextStyle(

```



```
        color: Colors.black,
        fontSize: 14,
        fontWeight: FontWeight.w400,
      ),
      children: [
        TextSpan(text: "Don't have an account? "),
        TextSpan(
          text: "Register",
          style:
            TextStyle(
              color: Colors.blueAccent,
              fontSize: 14,
              fontWeight:FontWeight.w500,
            ),
        ),
      ],
    ),
  ),
),
],
),
);
}
}
```

## Appendix L: Dart Code for Pill Tracker

```
import 'package:firebase_core/firebase_core.dart';
import 'package:first_app/firebase_options.dart';
import 'package:first_app/main.dart';
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:intl/intl.dart';
import 'package:first_app/services/notification_logic.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(MyApp());
}

class PillTracker extends StatelessWidget {
  const PillTracker({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    final String uid = FirebaseAuth.instance.currentUser!.uid;
    final TextEditingController nameController = TextEditingController();
    final TextEditingController amountController = TextEditingController();

    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.blue[100],
        centerTitle: true,
        elevation: 0,
        title: const Text(
          'Pill Tracker',

```

```

        style: TextStyle(
          color: Colors.black,
          fontSize: 18,
          fontWeight: FontWeight.w500,
        ),
      ),
    ),
  body: StreamBuilder<QuerySnapshot>(
    stream: FirebaseFirestore.instance
      .collection("users")
      .doc(uid)
      .collection('pill_logs')
      .orderBy('timestamp',
        descending: true) // make sure is the latest first
      .limit(1) // only get the latest one
      .snapshots(),
    builder: (context, snapshot) {
      print('Stream updated!');
      print('Stream triggered at ${DateTime.now()}');

      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      }

      if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
        return const Center(child: Text('No pill logs found.'));
      }

      final latestLog = snapshot.data!.docs.first;
      final data = latestLog.data() as Map<String, dynamic>;

      print('Latest document data: $data');
      print('pill log id: ${latestLog.id}');
    },
  ),
),

```

```

final amlodipine_count = data['amlodipine_count'] ?? 0;
final simvastatin_count = data['simvastatin_count'] ?? 0;
final timestamp = data['timestamp'] as Timestamp?;
final docId = latestLog.id; //Get the document ID
return Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      buildPillCard('Amlodipine', amlodipine_count),
      const SizedBox(height: 40),
      buildPillCard('Simvastatin', simvastatin_count),
      const SizedBox(height: 40),
      ElevatedButton(

        onPressed: () {
          showRefillDialog(context,uid, docId); //pass document ID
        },
        child: const Text('Refill Pills'),
        style:
          ElevatedButton.styleFrom(
            backgroundColor: const Color.fromARGB(255, 25, 85, 94),
            foregroundColor: Colors.white,
          )
        ),
    ],
  ),
);
},
),
);
}

```

```

Widget buildPillCard(String name, int count) {
  return Card(
    elevation: 10,
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),
    child: Container(
      width: double.infinity,
      padding: const EdgeInsets.all(16),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        children: [
          const Icon(Icons.medical_services, size: 30, color: Colors.teal),
          const SizedBox(height: 10),
          Text(name,
            style:
              const TextStyle(fontSize: 20, fontWeight: FontWeight.bold)),
          const SizedBox(height: 10),
          Text('Count: $count', style: const TextStyle(fontSize: 18)),
        ],
      ),
    ),
  );
}

void checkPillThreshold({
  required String pillName,
  required int originalCount,
  required int currentCount,
}) {
  final int threshold = (originalCount * 0.2).ceil();
  print('original count: $originalCount');
  print('Threshold check: $pillName | Current: $currentCount | Threshold: $threshold');
}

```

```

if (currentCount <= threshold) {
    NotificationLogic.showInstantNotification(
        title: 'Low Pill Alert: $pillName',
        body: 'Only $currentCount pills left. Remember to refill!',
        id: DateTime.now().millisecondsSinceEpoch ~/ 1000,
    );
}
}

void showRefillDialog (BuildContext context, String uid, String docId){
    print('Latest document ID: $docId');
    final TextEditingController nameController = TextEditingController();
    final TextEditingController amountController = TextEditingController();
    String pillName = "";
    int refillAmount =0;
    showDialog(
        context: context,
        builder: (context){
            return AlertDialog(
                title:Text('Refill pills'),
                content:Column(
                    mainAxisAlignment:MainAxisSize.min,
                    children: [
                        TextField(
                            controller:nameController,
                            keyboardType:TextInputType.text,
                            decoration:const InputDecoration(labelText:'Pill Name (e.g. Amlodipine)'),
                            onChanged:(value){
                                pillName= value.trim().toLowerCase();
                            },
                        ),
                        TextField(

```

```

        controller:amountController
        keyboardType:TextInputType.number,
        decoration:const InputDecoration(labelText:'Number of pills'),
        onChanged: (value){
            refillAmount = int.tryParse(value)??0;
        }
    ),
],
),
actions:[
    TextButton( child: const Text('Cancel'),
        onPressed: (){
            Navigator.of(context).pop(); // close the dialog
        },
    ),
    ElevatedButton(
        child: const Text('Save'),
        onPressed: () async{
            await refillPills(docId, pillName, refillAmount, context);
        },
    ),
],
);
},
);
}

Future<void> refillPills(String docId, String pillName, int amount, BuildContext
context) async {
    if (pillName.isEmpty || amount <= 0) {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: const Text('Please enter valid pill name and amount.')),
        );
    }
}

```

```

    return;
}

final String uid = FirebaseAuth.instance.currentUser!.uid;
final userRef = FirebaseFirestore.instance.collection("users").doc(uid);
final pillLogsRef = userRef.collection('pill_logs');

try {
    final snapshot= await pillLogsRef.orderBy('timestamp', descending :
true).limit(1).get();

    int currentAmlodipine = 0;
    int currentSimvastatin = 0;

    if ( snapshot.docs.isNotEmpty){
        final latestData = snapshot.docs.first.data();
        currentAmlodipine = latestData['amlodipine_count'] ?? 0;
        currentSimvastatin = latestData ['simvastatin_count'] ?? 0;
    }

    if (pillName.contains('amlodipine')) {
        currentAmlodipine +=amount;
        checkPillThreshold(
            pillName: 'Amlodipine',
            originalCount: currentAmlodipine,
            currentCount: currentAmlodipine,
        );
    }

    }else if (pillName.contains('simvastatin')) {
        currentSimvastatin += amount;

        checkPillThreshold(
            pillName: 'Simvastatin',
            originalCount: currentSimvastatin,
            currentCount: currentSimvastatin,

```



```

);

} else {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Pill name not recognized.')),
  );
  return;
}

//create pill log everytime after refills
await pillLogsRef.add({
  'amlodipine_count': currentAmlodipine,
  'simvastatin_count':currentSimvastatin,
  'timestamp':Timestamp.now(),
  'source':'flutter_app',
});

Navigator.of(context).pop(); // close refill dialog
ScaffoldMessenger.of(context).showSnackBar(
  const SnackBar(content: Text('new pill logs created in firestore')));

} catch (e) {
  print('Error updating pills: $e');
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('Failed to update! ')),
  );
}
}

```

## Appendix M: Dart Code for Reminder Model

```
import 'package:cloud_firestore/cloud_firestore.dart';

class ReminderModel{
  Timestamp? timestamp;
  bool? onOff;
  String?pillName;
  int?pillCount; // amountf of pills user should take
  int? notificationId;

  ReminderModel({this.timestamp,  this.onOff,  this.pillName,  this.pillCount,
this.notificationId});

  //convert into this,data structure that can easily store in firestore
  Map<String, dynamic> toMap() {
    return {
      'time':timestamp,
      'onOff':onOff,
      'pillName':pillName,
      'pillCount':pillCount,
      'notificationId': notificationId,
    };
  }

  factory ReminderModel.fromMap(map) {
    return ReminderModel(
      timestamp:map['time'],
      onOff:map['onOff'],
      pillName:map['pillName'],
      pillCount:map['pillCount'],
      notificationId: map['notificationId'],
    );
  }
}
```

## Appendix N: Dart Code for Round Text Field

```
import 'package:flutter/material.dart';// import Flutter UI components

class RoundTextField extends StatelessWidget {
  final TextEditingController? textEditingController;
  final FormFieldValidator? validator;
  final ValueChanged<String>? onChanged;
  final String hintText;
  final String icon;
  final TextInputType textInputType
  final bool isObscureText;
  final Widget? rightIcon;

  const RoundTextField(
    {super.key, //this passed the key to teh StatelessWidget constructor
    this.textEditingController,
    this.validator,
    this.onChanged,
    required this.hintText,
    required this.icon,
    required this.textInputType,
    this.isObscureText = false,
    this.rightIcon});

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: BoxDecoration(
        color: const Color.fromARGB(255, 207, 228, 244),
        borderRadius: BorderRadius.circular(10),
      ),
      child: TextFormField(
        controller: textEditingController,
```

```
keyboardType: TextInputType,
obscureText: isObscureText,
onChanged: onChanged,
decoration: InputDecoration(
  contentPadding:
    const EdgeInsets.symmetric(
      vertical: 20,
      horizontal: 20),
  enabledBorder: InputBorder.none,
  focusedBorder: InputBorder.none,
  hintText: hintText,
  prefixIcon: Container(
    alignment: Alignment.center,
    width: 20,
    height: 20,
    child: Image.asset(
      icon,
      height: 20,
      width: 20,
      fit: BoxFit.contain,
      color: Colors.black,
    ),
  ),
  suffixIcon: rightIcon,
  hintStyle: const TextStyle(fontSize: 12,color:Colors.black),
),
validator: validator,
));
}
```

## Appendix O: Dart Code for Signup Screen

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:first_app/login_screen.dart';
import 'package:first_app/round_text_field.dart';
import 'package:flutter/material.dart';

class SignupScreen extends StatefulWidget {
  const SignupScreen({super.key});

  @override
  State<SignupScreen> createState() => _SignUpScreenState();
}

class _SignUpScreenState extends State<SignupScreen> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final CollectionReference _users = FirebaseFirestore.instance.collection("users");

  final TextEditingController _firstNameController = TextEditingController();
  final TextEditingController _lastNameController = TextEditingController();
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passController = TextEditingController();
  bool _isObscure = true;
  bool _isChecked = false;
  final _formKey = GlobalKey<FormState>();

  @override
  void initState() {
    super.initState();
    // Initialization code here
  }

  @override
  Widget build(BuildContext context) {
    var media = MediaQuery.of(context).size;
```

```

return MaterialApp(
  debugShowCheckedModeBanner: false,
  home: Scaffold(
    appBar: AppBar(
      title: const Text('Medicine Reminder'),
      backgroundColor: Colors.greenAccent,
      centerTitle: true,
    ),
    body: SafeArea(
      child: SingleChildScrollView(
        child: Container(
          padding: const EdgeInsets.symmetric(
            vertical: 15, horizontal: 25),
          child: Form(
            key: _formKey,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: [
                SizedBox(height: media.height * 0.1),
                SizedBox(
                  width: media.width,
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.center,
                    children: [
                      SizedBox(
                        height: media.width * 0.03,
                      ),
                      const Text(
                        'Create an account',
                        textAlign: TextAlign.center,
                        style: TextStyle(
                          color: Colors.black,

```

```

        fontSize: 30,
        fontWeight: FontWeight.w200),
    ),
  ],
),
),

//enter first name
    SizedBox(
      height: media.width * 0.02,
    ),
    RoundTextField(
      textEditingController: _firstNameController,
      hintText: "First Name",
      icon: "assets/icons/profile_icon.png",
      TextInputType: TextInputType.name,
      validator: (value) {
        if (value == null || value.isEmpty) {
          return "Please enter your first name";
        }
        return null;
      },
    ),

//enter last name
    SizedBox(
      height: media.width * 0.02,
    ),
    RoundTextField(
      textEditingController: _lastNameController,
      hintText: "Last Name",
      icon: "assets/icons/profile_icon.png",
      TextInputType: TextInputType.name,

```

```

        validator: (value) {
          if (value == null || value.isEmpty) {
            return "Please enter your last name";
          }
          return null;
        },
      ),
//enter email
      SizedBox(
        height: media.width * 0.02,
      ),
      RoundTextField(
        textEditingController: _emailController,
        hintText: "Email",
        icon: "assets/icons/email_icon.png",
        TextInputType: TextInputType.emailAddress,
        validator: (value) {
          if (value == null || value.isEmpty) {
            return "Please enter email";
          }
          return null;
        },
      ),
//enter password
      SizedBox(
        height: media.width * 0.02,
      ),
      RoundTextField(
        textEditingController: _passController,
        hintText: "Password",
        icon: "assets/icons/password_icon.png",
        TextInputType: TextInputType.text,
        isObscureText: _isObscure,

```



```

        validator: (value) {
          if (value == null || value.isEmpty) {
            return "Please enter password";
          }
          return null;
        },
        rightIcon: TextButton(
          onPressed: () {
            setState(
              () {
                _isObscure = !_isObscure;
              },
            );
          },
          child: Container(
            alignment: Alignment.center,
            height: 20,
            width: 20,
          ),
        )),
        SizedBox (
          height: media.width*0.02
        ),
        Row(
          mainAxisAlignment: MainAxisAlignment.start,
          children:[
            IconButton(onPressed: () {
              setState(() {
                _isCheck = !_isCheck;
              });
            },
            icon: Icon(
              _isCheck

```

```

        ? Icons.check_box_outlined
        : Icons.check_box_outline_blank,
        color: Colors.grey,
    )),
    const Expanded(child: Text(
        "By continuing you accept our Privacy Policy and terms of Use",
        style: TextStyle(
            color: Colors.grey,
            fontSize: 10,
        ),
    ))
  ],
),
  SizedBox(
    height: media.width * 0.1,
  ),
  ElevatedButton(
    onPressed: () async {
      if(_formKey.currentState!.validate()){
        if(_isChecked){
          try{
            UserCredential userCredential =
              await _auth.createUserWithEmailAndPassword(
                email: _emailController.text,
                password: _passController.text,
              );
            String uid = userCredential.user!.uid;

            await _users.doc(uid).set({
              'email': _emailController.text,
              'firstName': _firstNameController.text,
              'lastName': _lastNameController.text,
            });

```

```

ScaffoldMessenger.of(context).showSnackBar(
  const SnackBar(content: Text("Account created")));
Navigator.push(
  context,
  MaterialPageRoute(
    builder: (context) => LoginScreen()
  );
)catch(e){
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text(e.toString())));
}
}
},
child: const Text("Create Account")),

SizedBox(
  height: media.width * 0.01,
),
TextButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => const LoginScreen(),
      ));},
  child: RichText(
    textAlign: TextAlign.center,
    text: const TextSpan(
      style: TextStyle(
        color: Colors.black,
        fontSize: 14,

```



## Appendix P: Dart Code for Switcher

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:first_app/reminder_model.dart';
import 'package:flutter/material.dart';

class Switcher extends StatefulWidget {
  bool onOff;
  String uid;
  Timestamp timestamp;
  String id;
  Switcher(this.onOff, this.uid, this.id, this.timestamp);
  @override
  State<Switcher> createState() => _SwitcherState();
}

class _SwitcherState extends State<Switcher> {
  @override
  Widget build(BuildContext context) {
    return Switch (
      onChanged:
      (bool value) {
        ReminderModel reminderModel = ReminderModel();
        reminderModel.onOff = value;
        reminderModel.timestamp = widget.timestamp;
        FirebaseFirestore.instance
          .collection('users')
          .doc(widget.uid)
          .collection('reminder')
          .doc(widget.id)
          .update(reminderModel.toMap());
      },
      value: widget.onOff,
    ); }}

```